2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
DECEMBER 6 - 7, 2022

Efficient Loosely-Timed SystemC TLM-2.0 Modeling: A Hands-On Tutorial

Nils Bosbach, RWTH Aachen University

Lukas Jünger, MachineWare GmbH

Rainer Leupers, RWTH Aachen University

MACHINEWARE

Institute for Communication Technologies and Embedded Systems | RWTH AACHEN UNIVERSITY

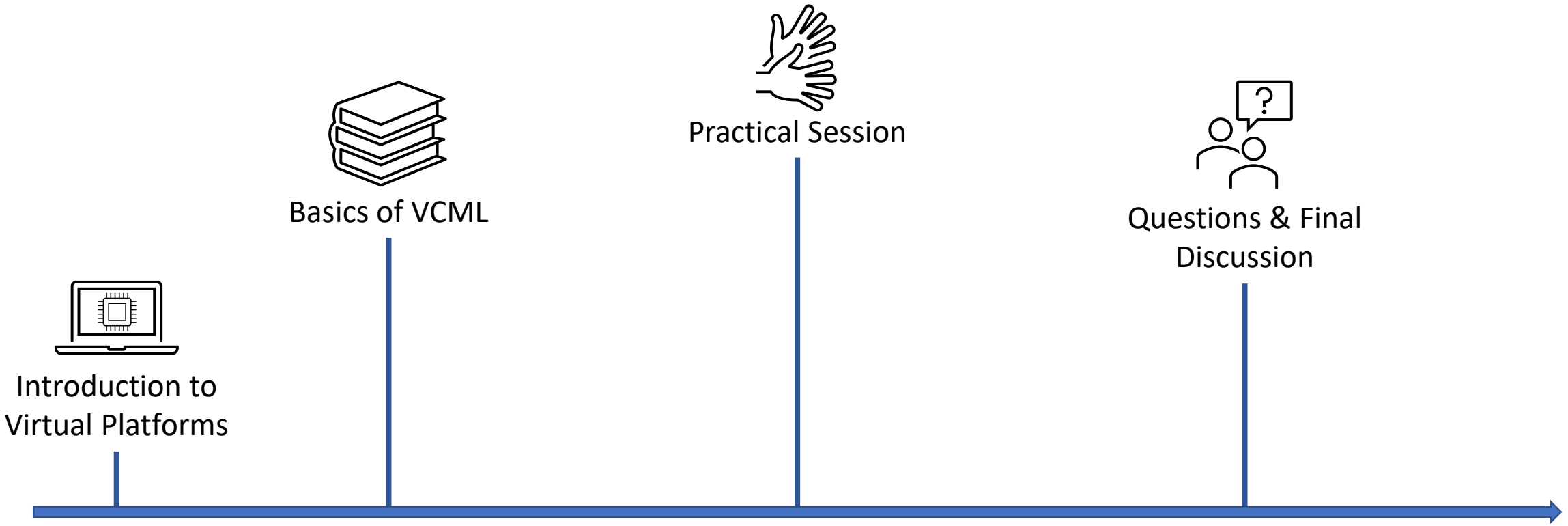
accellera SYSTEMS INITIATIVE

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*

What to expect?



2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY

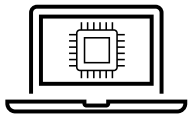
DECEMBER 6 - 7, 2022

Virtual Platform (VP)

Introduction

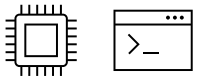
accellera SYSTEMS INITIATIVE

Virtual Platform (VP)

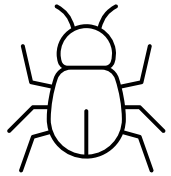


Simulations of Systems-on-Chips (SoCs)

Allow rapid prototyping

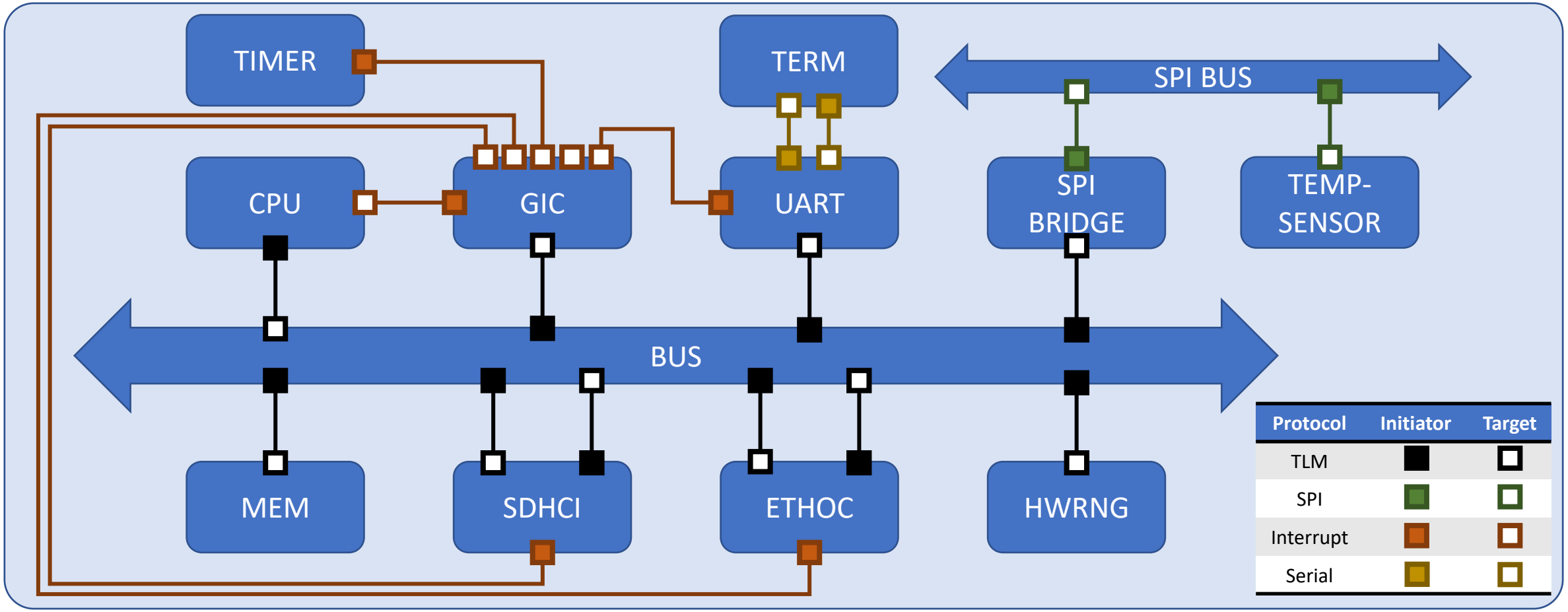


Enabler of hardware/software codesign

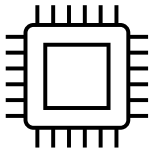


Can be used for software analysis and debugging

Virtual Platform (VP)



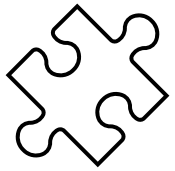
Building Blocks of a VP



Models

- Frequently needed parts (e.g., register model)
- Standard models (e.g., bus models)

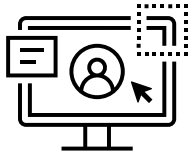
→ Enables **fast development**



Interfaces

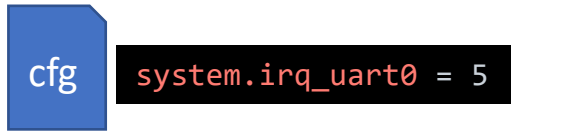
- Defined interfaces
- TLM, SPI, I²C, Serial, Interrupt, etc.

→ Required for **interoperability** between models



Infrastructure

- Configuration options (e.g., config file)
- Logging/tracing
- Remote access protocol
- Visualization




```
vcml::log_error("error %d", err);
```

→ Increases the **useability**

2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY

DECEMBER 6 - 7, 2022

Virtual Components Modeling Library (VCML)

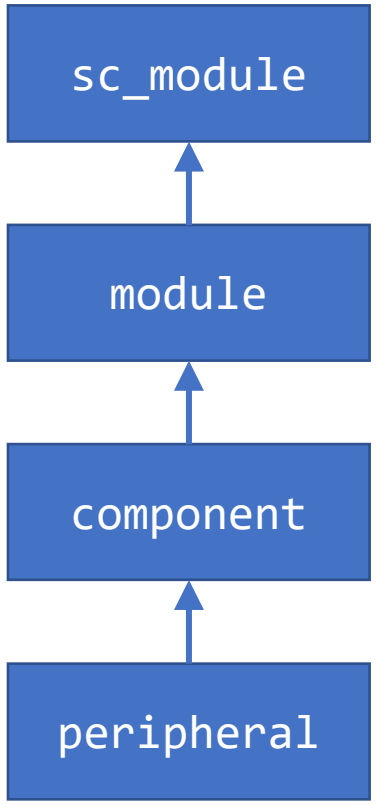
accellera SYSTEMS INITIATIVE

Virtual Components Modeling Library (VCML)

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

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

VCML Building Blocks



- 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 custom I/O peripheral models
- Adds support for registers

VCML Peripheral

Inheritance from `vcml::peripheral`

```
class my_peripheral : public vcml::peripheral {
  public:
    vcml::reg<vcml::u8> my_reg;
    vcml::tlm_target_socket  in;

    explicit my_peripheral(const sc_module_name& mn):
        peripheral(mn),
        my_reg("my_reg", 0x100, 0x00),
        in("in") {
    }
}
```

8-bit register

Socket for register access

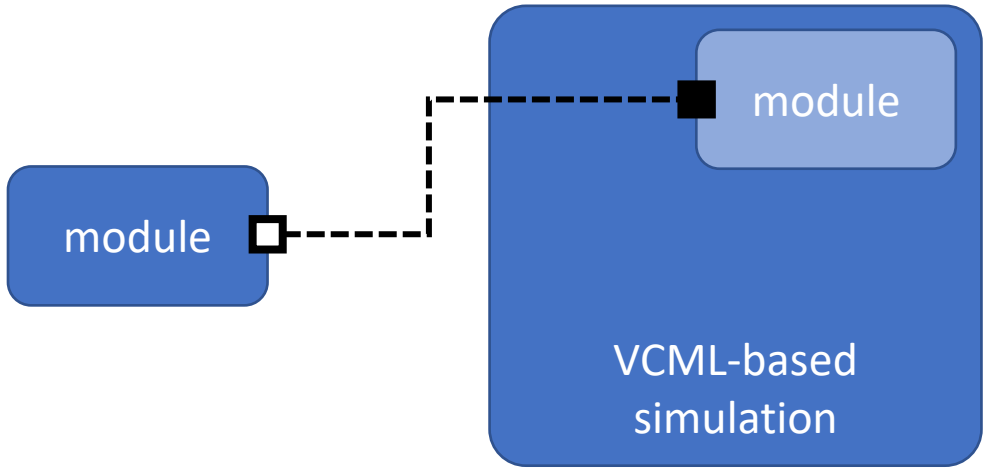
Offset

Initial value

`b_transport` calls of the `vcml::tlm_target_socket` are forwarded to the corresponding register
(address spaces for multiple sockets)

Interfaces/Protocols

- Based on TLM → direct function calls → no delta cycles
- Increases compatibility
- Predefined sockets for standard protocols (I^2C, SPI, GPIO, clock, ...)

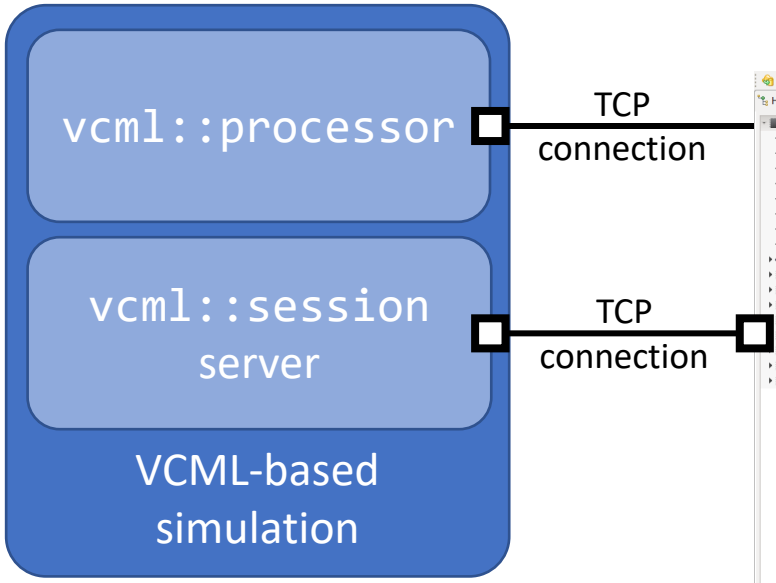


Connect sockets using `bind` function

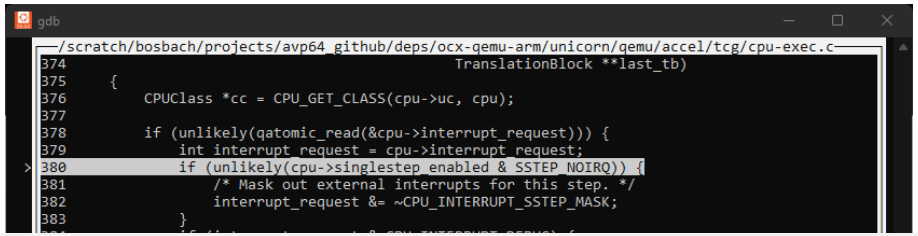
```
mod1.initiator_socket.bind(mod2.target_socket)
```

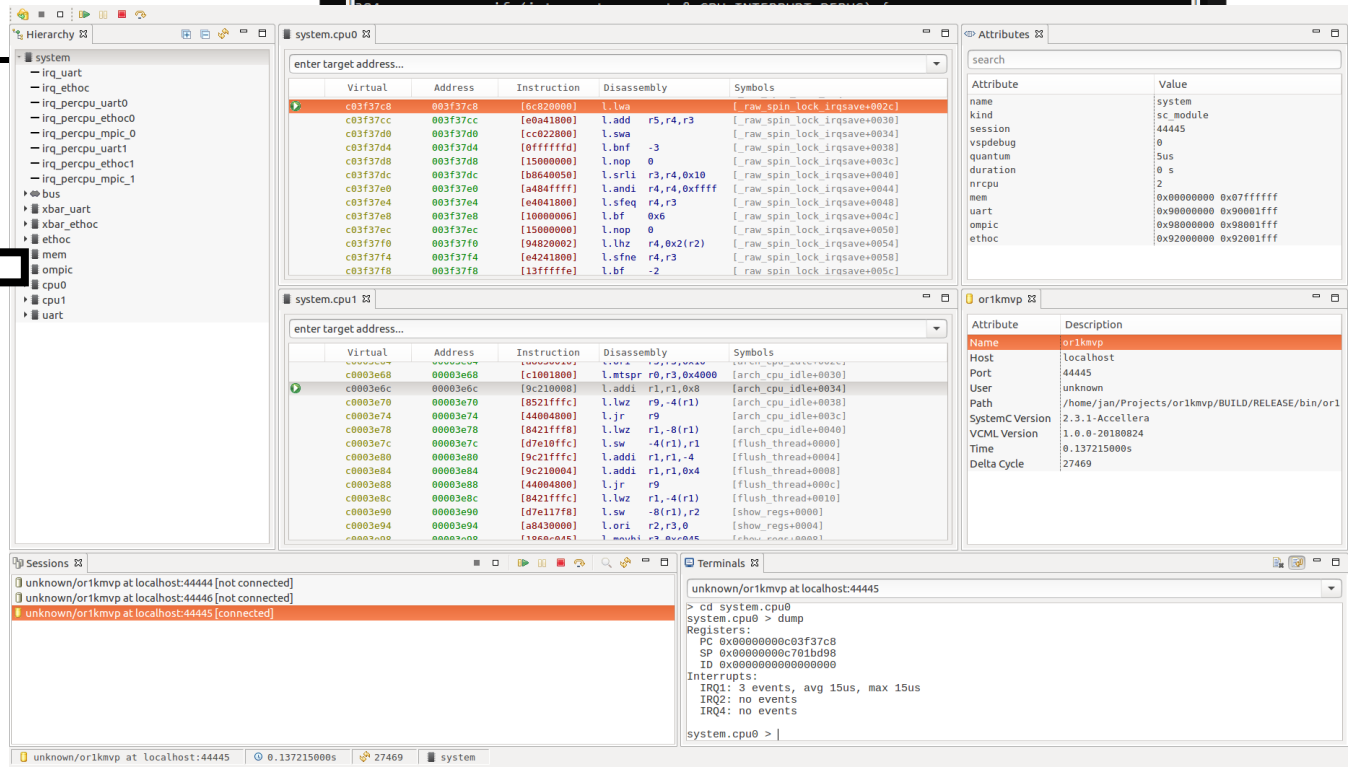
Defined interfaces allow reuse of models

External Interfaces



Remote GDB





Virtual Platform Explorer (VIPER)

VCML Properties

```
class my_module: public vcml::module {
  public:
    vcml::property<uint32_t> my_property;

    explicit my_module(const sc_module_name& mn) :
        module(mn),
        my_property("my_property", 0x00000000) {
        std::cout << "my_property: "
                  << my_property
                  << std::endl;
    }
};
```

Datatype

Name

Default value

- Configuration interface
- Parametrization of modules
- Runtime parameters

Config File

```
system.my_module.my_property = 37
```

Command Line

```
$ ./my_vp -c system.my_module.my_property=1
```

Logging

Logging functions use `printf`-like syntax

```
vcml::log_type(const char* format, ...)
```

Multiple functions for different levels

`vcml::log_error`	Reporting errors, e.g., when an exception is caught
`vcml::log_warning`	Reporting abnormal events, e.g., missing files/erroneous configuration
`vcml::log_info`	Generic info messages, e.g., printing simulation duration/bus memory maps
`vcml::log_debug`	Printing debug messages

Example

```
vcml::log_error("failed getting LR prio for irq %d on cpu %d", irq, cpu);
```

Logging cont.

```
        SystemC 2.3.3-MachineWare GmbH --- Nov  7 2022 09:18:17
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
[I 0.000000000] system.term: listening on port 52010
[D 0.000000000] system.bridge: created slirp ipv4 network 10.0.0.0/24
[D 0.000000000] system.bridge: created slirp ipv6 network fec0::
[I 0.000000000] system: starting infinite simulation using 100 us quantum
[D 0.000000000] system.reset: loading binary file '../config/../sw/arm64/linux/boot.bin' (136 bytes) to offset 0x0
[D 0.000000000] system.reset: loading binary file '../config/../sw/arm64/linux/Image-4.19.4' (10727936 bytes) to offset
0x80000
[D 0.000000000] system.reset: loading binary file '../config/../sw/arm64/linux/linux.dtb' (2528 bytes) to offset
0x7f00000
[D 0.000000000] rsp_52100: gdb architecture aarch64 is supported
[D 0.000000000] rsp_52100: gdb feature org.gnu.gdb.aarch64.core is supported
[D 0.000000000] rsp_52100: gdb feature org.gnu.gdb.aarch64.fpu is not supported
[D 0.000000000] rsp_52100: gdb feature org.gnu.gdb.aarch64.sve is not supported
[D 0.000000000] rsp_52100: gdb feature org.gnu.gdb.aarch64.pauth is not supported
[I 0.000000000] system.arm0: listening for GDB connection on port 52100
[    0.000000] Booting Linux on physical CPU 0x0000000000 [0x410fd083]
…
```

Tracing

- Tracing of TLM transactions
- Available tracers
  - Tracing file: `<simulation> --trace <file>`
  - Terminal: `<simulation> --trace-stdout`
- Activation per model/socket: `system.my_module.[socket.]trace = true`

```
[CLK 0.000000000] system.uart0.clk>> CLK [off->1000000000Hz]
[GPIO 0.000000000] system.uart0.rst >> GPIO+
[GPIO 0.000000000] system.uart0.rst << GPIO+
[GPIO 0.000000000] system.uart0.rst >> GPIO-
[GPIO 0.000000000] system.uart0.rst << GPIO-
[TLM 0.000293169] system.uart0.in  >> WR 0x00000000 [5b] (TLM_INCOMPLETE_RESPONSE)
[TLM 0.000293169] system.uart0.dr  >> WR 0x00000000 [5b] (TLM_INCOMPLETE_RESPONSE)
[SERIAL 0.000293169] system.uart0.serial_tx  >> SERIAL TX [5b] (9600n8)
[[TLM 0.000293169] system.uart0.dr        << WR 0x00000000 [5b] (TLM_OK_RESPONSE)
[TLM 0.000293169] system.uart0.in       << WR 0x00000000 [5b] (TLM_OK_RESPONSE)
```

Construction of a VP using `vcml::system` utility

```
class system : public vcml::system {
  private:
    arm64_cpu m_cpu;
    vcml::generic::clock m_clock;
    ...
  public:
    explicit system(const sc_core::sc_module_name& name) :
        vcml::system(name),
        m_cpu("cpu", 0, 0),
        m_clock("clock", 100000000),
        ... {
            m_clock.clk.bind(m_cpu.clk);
            ...
        }
};
```

Derive from `vcml::system`

Declaration of models

Initialization of models

Connections

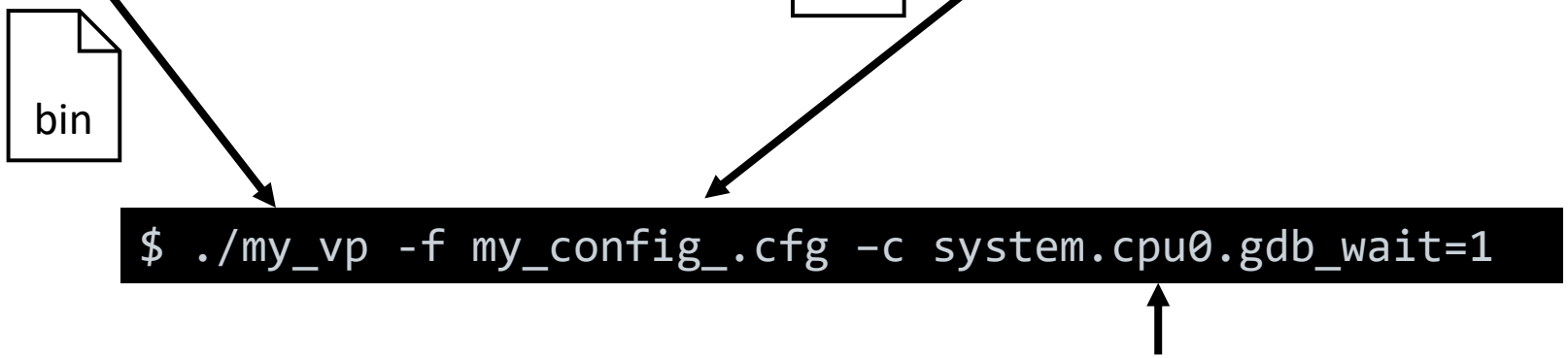
Construction of a VP cont.

SystemC main function

```
extern "C" int sc_main(int argc, char** argv) {
    system system("system");
    return system.run();
}
```

Config file

```
system.clk.hz = 100000000
…
```



2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY

DECEMBER 6 - 7, 2022

Practical Session

Implementation of a PL011 UART model

accellera SYSTEMS INITIATIVE

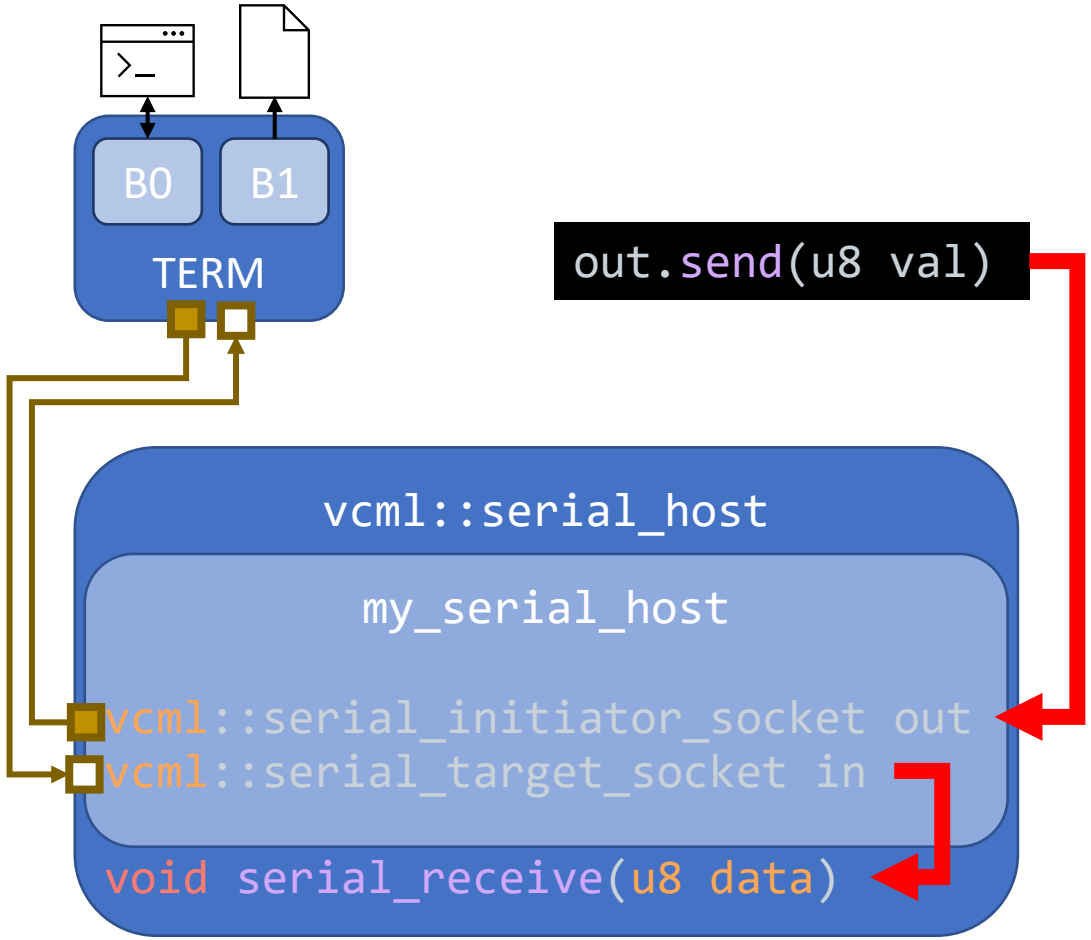
ARM PrimeCell UART (PL011)

- Advanced Microcontroller Bus Architecture (AMBA) compliant SoC
- Developed, tested and licensed by ARM
- Memory-mapped register interface

Data register (16bit) (UARTDR)

- Read: Lower 8 bits contain the received data character
- Write: Lower 8 bits are the character to be transmitted

VCML Terminal & Serial Protocol



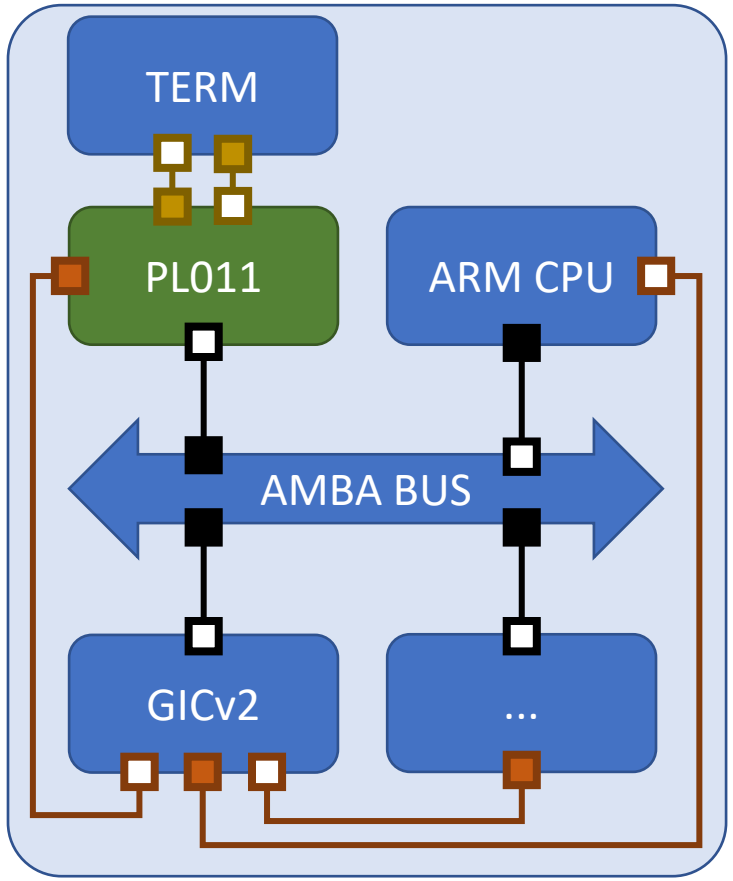
- `vcml::term` handles input/output of data
- Can be used for UART implementation

Backends:

- Read from/write to file: `file:<rx_file>:<tx_file>`
- TCP socket: `tcp:<port>`
- Print output to stderr: `stderr`
- Print output to stdout: `stdout`
- Print output to stdout + read input from stdin: `term`

```
system.term0.backends       = term   tcp:51010
system.term1.backends       = stdout tcp:51011
```

PL011 implementation using VCML



Interfaces

- Memory-mapped registers → `vcml::reg` + `vcml::tlm_target_socket`
- Interrupts → `vcml::gpio_initiator_socket`
- Terminal input → `vcml::serial_target_socket`
- Terminal output → `vcml::serial_initiator_socket`

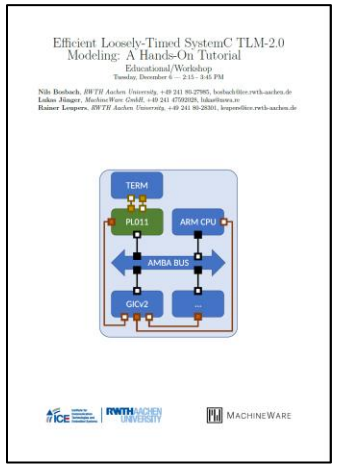
Steps

1. Derive a class from `vcml::peripheral`
2. Add registers + interfaces
3. Add callback functions to the registers → implement functionality
4. Connect the UART model to the VP

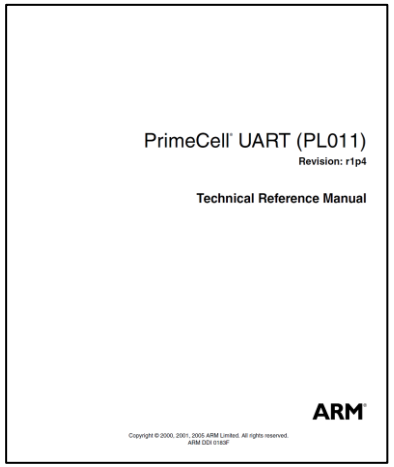
Getting Started

approx. 60 min practical tutorial time

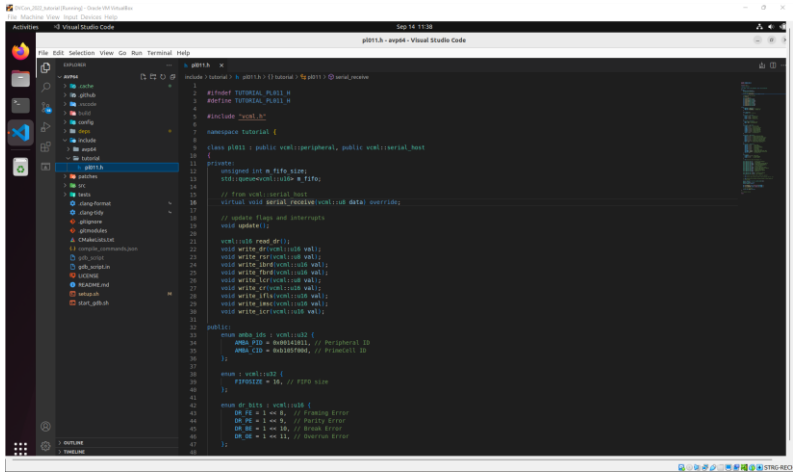
- Download Virtual Box from https://www.virtualbox.org/
- Download the tutorial files from https://mwa.re/dvcon
  - PW: **dvcon2022**



Tutorial Description



PL011
Reference Manual



Virtual Box Image

2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY

DECEMBER 6 - 7, 2022

Implementation

ARM PrimeCell UART (PL011)

accellera SYSTEMS INITIATIVE

Registers

Inheritances

- `vcml::peripheral`
  - Starting point for peripherals
- `vcml::serial_host`
  - Convenient `serial_receive` function

Registers

- Definition of registers corresponding to the datasheet

```
class pl011uart : public peripheral, public serial_host {
  public:
    reg<u16> dr;   // Data Register
    reg<u8> rsr;   // Receive Status Register
    reg<u16> fr;   // Flag Register
    reg<u8> ilpr;  // IrDA Low-Power Counter Register
    reg<u16> ibrd; // Integer Baud Rate Register
    reg<u16> fbrd; // Fractional Baud Rate Register
    reg<u8> lcr;   // Line Control Register
    reg<u16> cr;   // Control Register
    reg<u16> ifls; // Interrupt FIFO Level select
    reg<u16> imsc; // Interrupt Mask Set/Clear Register
    reg<u16> ris;  // Raw Interrupt Status
    reg<u16> mis;  // Masked Interrupt Status
    reg<u16> icr;  // Interrupt Clear Register
    reg<u16> dmac; // DMA Control

    reg<u32, 4> pid; // Peripheral ID Register
    reg<u32, 4> cid; // Cell ID Register
};
```

Registers cont.

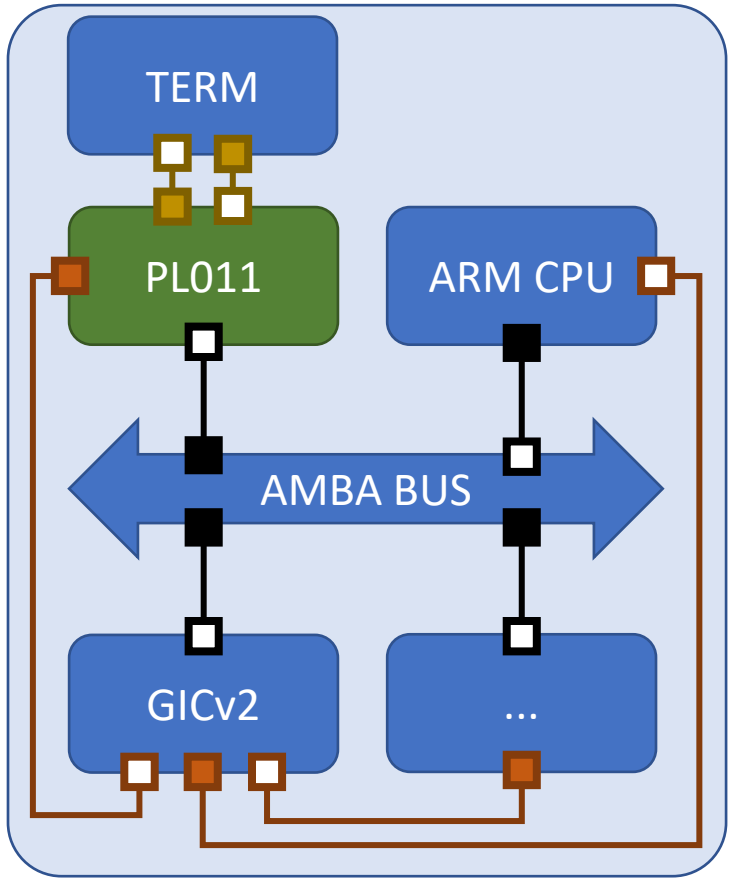
Registers

- Set name, offset, and initial value in member initializer list
- Configure register in the constructor
- Add callbacks to react on read/write operations

```
pl011uart::pl011uart(const sc_module_name& nm):
    peripheral(nm),
    dr("dr", 0x000, 0x0),
    rsr("rsr", 0x004, 0x0) {

    dr.sync_always();
    dr.allow_read_write();
    dr.on_read(&pl011uart::read_dr);
    dr.on_write(&pl011uart::write_dr);
}
```

Interfaces




```
class pl011uart : public peripheral, public serial_host {
  public:
   tlm_target_socket in;
   gpio_initiator_socket irq;

   serial_initiator_socket serial_tx;
   serial_target_socket serial_rx;
};
```

Functionality

```
void pl011::write_dr(vcml::u16 val) {
    if (!is_tx_enabled())
        return;

    dr = val & 0x00ff;

    // set the transmit interrupt bit
    ris |= RIS_TX;

    serial_tx.send(dr);

    // update flags and interrupts
    update();
}
```

Cut the upper 8 bits

Forward the character to the terminal

Functionality

```
vcml::u16 pl011::read_dr() {
    vcml::u16 val = 0;

    if (!m_fifo.empty()) {
        val = m_fifo.front();
        m_fifo.pop();
    }
    dr = val;

    // set flags in Receive Status Register
    rsr = (val >> RSR_O) & RSR_M;

    // update flags and interrupts
    update();
    return val;
}
```

Get the last value from the FIFO

Forward the value to the sender of the read request

Debugging

```
./arch/arm64/include/asm/io.h
 40         static inline void __raw_writeb(u8 val, volatile void __iomem *addr)
 41         {
 42                 asm volatile("strb %w0, [%1]" : : "rZ" (val), "r" (addr));
 43         }
 44
 45         #define __raw_writew __raw_writew
 46         static inline void __raw_writew(u16 val, volatile void __iomem *addr)
 47         {
>48                 asm volatile("strh %w0, [%1]" : : "rZ" (val), "r" (addr));
 49         }
 50
 51         #define __raw_writel __raw_writel
 52         static inline void __raw_writel(u32 val, volatile void __iomem *addr)
 53         {
b+ 54                 asm volatile("str %w0, [%1]" : : "rZ" (val), "r" (addr));
 55         }
 56
 57         #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
```

Kernel Debugging

```
46  void pl011::write_dr(vcml::u16 val) {
47      if (!is_tx_enabled())
48          return;
49
50      dr = val & 0x00ff;
51
52      // set the transmit interrupt bit
53      ris |= RIS_TX;
54
55      serial_tx.send(dr);
56
57      // update flags and interrupts
58      update();
59  }
```

Simulation Debugging

2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

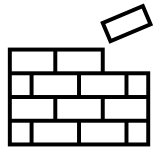
MUNICH, GERMANY

DECEMBER 6 - 7, 2022

Recapitulation

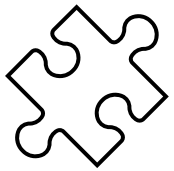
accellera SYSTEMS INITIATIVE

Recapitulation



Building Blocks

- `vcml::peripheral`
- `vcml::registers`
- Logging
- Assembling of VPs

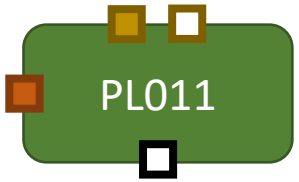


Interconnects

- TLM 2.0-based protocols
- Reuse of models
- Interoperability

Practical Session

- Hands-on experience
- PL011 UART
- Use of predefined models + protocols
- Debugging



Questions?