SoC design platform

Risto Pejašinović

October 3, 2023

CERN

Introduction

- Part of the EP-R&D WP5.2 SoC project
- Radiation Tolerant SoC Ecosystem for HEP ASICS
- Replace custom state machine ASICs with a CPU
  - Reprogrammable, retargetable FE ASICs
  - Reduce design and verification efforts
- Tackle the complexity of SoC designs by:
  - Automatic generation of SoC components
  - Reusability of HW/SW components
- Provide fully open source approach

- People Involved
  - Alessandro Caratelli
  - Anders Lauridsen
  - Anvesh Nookala
  - Kostas Kloukinas
  - Marco Andorno
  - Risto Pejašinović

The Complexity!

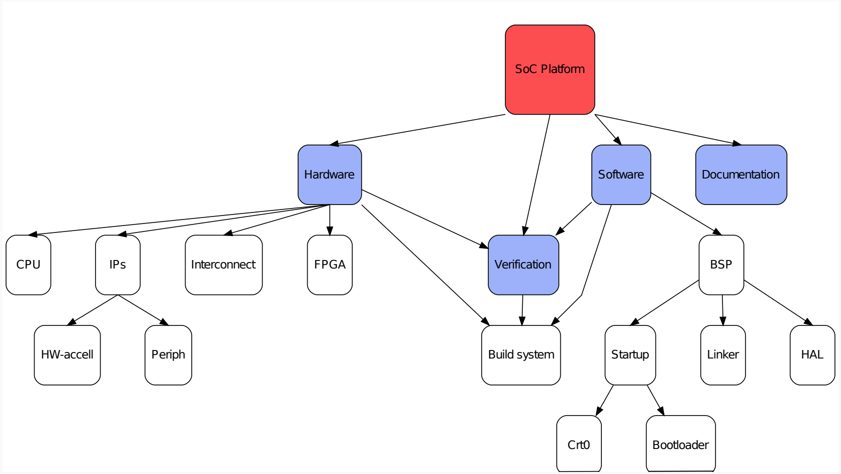


Figure 1: Tasks to build an SoC

Talk Overview

SoCMake

SystemRDL and PeakRDL

Verification

Example SoC designs

Build Systems (software)

- Transform sources into binaries
- Track dependencies and file changes
- Parallel builds
- Reproducible builds
- Testing, packaging, deployment, CI

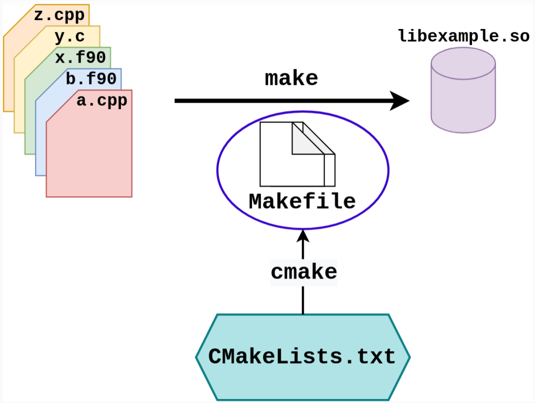
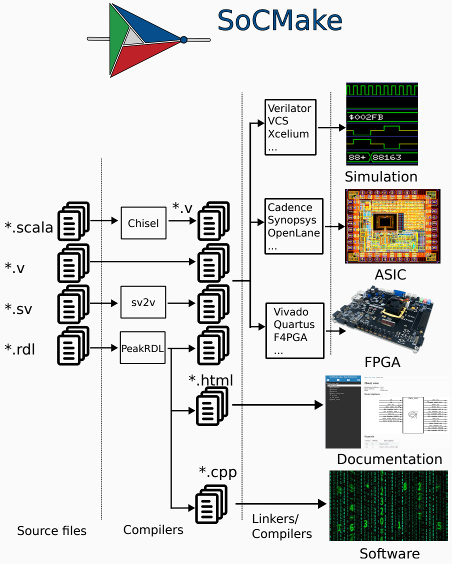


Figure 2: Typical SW build

Build Systems (hardware)

- Code transpilation/generation
- Simulation, Verification
- Synthesis, Implementation
- FPGA Emulation, deploying
- Documentation generation
- IP blocks packaging
- EDA tools abstraction layer



SoCMake

- Based on CMake, popular C++ build system
- Generates Makefiles/Ninja files
- Package manager/versioning with CPM.cmake
- Benefits comparing to other HW build systems
    - Native support for C/C++
    - SystemC/SystemC-UVM, Verilator support
    - Native support for cross-compilation
    - CMake is mature and documented
- EDA tools support

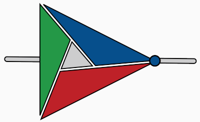


Figure 3: https://socmake.docs.cern.ch

SoCMake example tmrg (Triple Module Redundancy Generator)

```
cmake_minimum_required(VERSION 3.25)
project(tmrg_test NONE)

include("deps.cmake")

add_ip(chip
    VENDOR cern
    LIBRARY ip
    VERSION 0.0.1
    )
ip_sources(chip VERILOG
    ${PROJECT_SOURCE_DIR}/comb03.v)

tmrg(chip REPLACE SED_WOR)
iverilog(chip)
```

Figure 4: Simple CMakeLists.txt

```
$ mkdir build && cd build/
$ cmake ../
-- Downloading CPM.cmake to .../build/cmake/CPM_0.38.1.cmake
-- CPM: Adding package SoCMake@ (newsocgen)
-- Configuring done
-- Generating done
-- Build files have been written to: .../build
```

Figure 5: Configuring CMake project

```
... all (the default if no target is provided)
... clean
... cern__ip__chip__0.0.1_iverilog
... cern__ip__chip__0.0.1_tmrg
... graphviz
... run_cern__ip__chip__0.0.1_iv
```

Figure 6: Listing possible targets with help and graphviz

```
$ make run_cern__ip__chip__0.0.1_iv
[ 50%] Running tmrg on cern__ip__chip__0.0.1
--------------------------------------------------------------------------------
File '..../socmake/comb03.v' has 16 lines
--------------------------------------------------------------------------------
Total number of files parsed: 1
Total number of lines parsed: 16
Total parse time: 0.092 s
--------------------------------------------------------------------------------
Elaboration time : 0.004 s
--------------------------------------------------------------------------------
File '.../socmake/build/tmrg/comb03TMR.v.new' has 59 lines
Total number of triplicated lines: 59
Triplication time : 0.029 s
--------------------------------------------------------------------------------
[ 50%] Built target cern__ip__chip__0.0.1_tmrg
[100%] Running iverilog on cern__ip__chip__0.0.1
[100%] Built target cern__ip__chip__0.0.1_iverilog
Hello from comb03TMR
[100%] Built target run_cern__ip__chip__0.0.1_iv
```

Figure 7: Running simulation

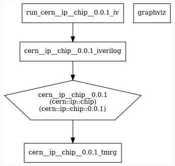

run_cern__ip__chip__0.0.1_iv
graphviz
cern__ip__chip__0.0.1_iverilog
cern__ip__chip__0.0.1
(cern::ip::chip)
(cern::ip::chip::0.0.1)
cern__ip__chip__0.0.1_tmrg

Figure 8: Dependency graph

SoCMake composing IPs

```
cmake_minimum_required(VERSION 3.25)
project(example
    LANGUAGES NONE
    VERSION 0.0.1)

include("deps/deps.cmake")

add_ip(top)
ip_sources(top VERILOG
    ${PROJECT_SOURCE_DIR}/top.v)

add_ip(prim10)
ip_sources(prim10 VERILOG
    ${PROJECT_SOURCE_DIR}/prim10.v)

add_ip(prim11)
ip_sources(prim11 VERILOG
    ${PROJECT_SOURCE_DIR}/prim11.v)

add_ip(prim00)
ip_sources(prim00 VERILOG
    ${PROJECT_SOURCE_DIR}/prim00.v)

add_ip(prim01)
ip_sources(prim01 VERILOG
    ${PROJECT_SOURCE_DIR}/prim01.v)

ip_link(top prim10 prim11)
ip_link(prim10 prim00)
ip_link(prim11 prim00 prim01)
```

Figure 9: Composing RTL libraries

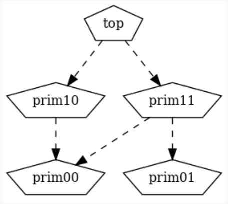


Figure 10: Resulting hierarchy

Talk Overview

SystemRDL

- Accellera open standard
- Describe register maps
- Generate design outputs
- Available commercial (Agnisys) and open-source (PeakRDL) toolchains

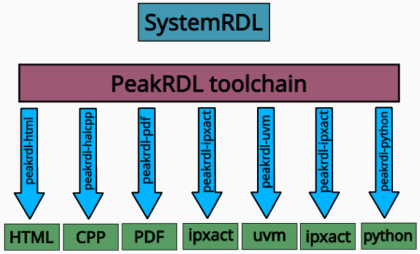


Figure 11: Possible output formats

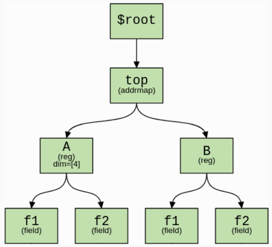


Figure 12: SystemRDL components

SystemRDL compiler and PeakRDL toolchain

- Open-source SystemRDL Compiler
- Python based, ANTLR4 parser
- Easy to create python plugins
- Already available
  - PeakRDL-regblock
  - PeakRDL-html
  - PeakRDL-ipxact
  - PeakRDL-python
  - PeakRDL-uvm
  - PeakRDL-pdf
  - And many more ....
- Developed internally
  - PeakRDL-halcpp
  - PeakRDL-socgen
  - PeakRDL-docusaurus
  - PeakRDL-opentitan
  - Linker script generator
  - PeakRDL-ipblockvg

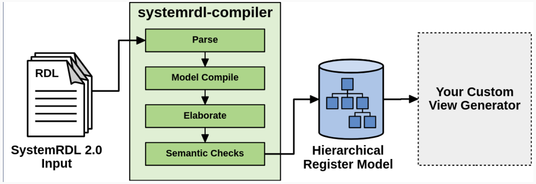


Figure 13: SystemRDL compiler architecture

PeakRDL-regblock

- Generates CSR register file
- Equivalent of Xilinx create AXI IP slave wizard
- Write specification instead of RTL
- Supports multiple protocols
- Write only core logic manually
- Support SW/HW properties like
    - HW/SW read, write
    - rclr/rset, woclr/woset
    - swmod, swacc
    - counter, incr, overflow
    - intr, enable, mask
    - many more, refer to SystemRDL

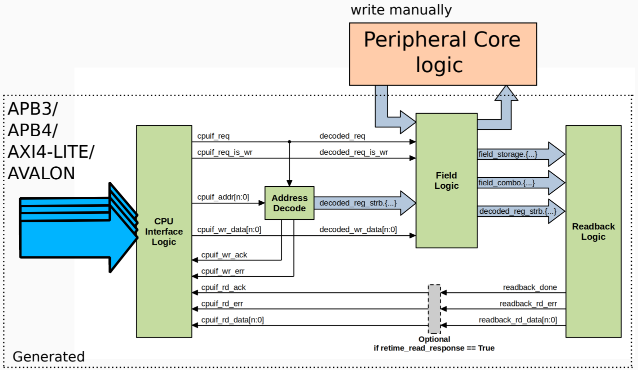


Figure 14: Regblock architecture

PeakRDL-socgen

- Generate top Verilog (2001) file for SoC
- Automatically infer interconnect
    - Crossbar when multiple masters
    - Simple interconnect when one master
- Automatically infer adapters
    - If adapter not available tries to chain
- Support for:
    - APB4
    - AXI4-Full
    - AXI4-Lite
    - NMI (Native Memory Interface of Picorv32)
    - OBI (Interface used in PULP cores)
    - Easy to add more...
- Generate graphviz .dot graph

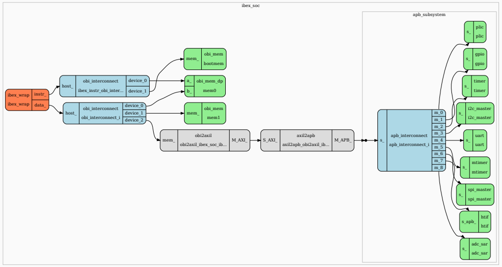
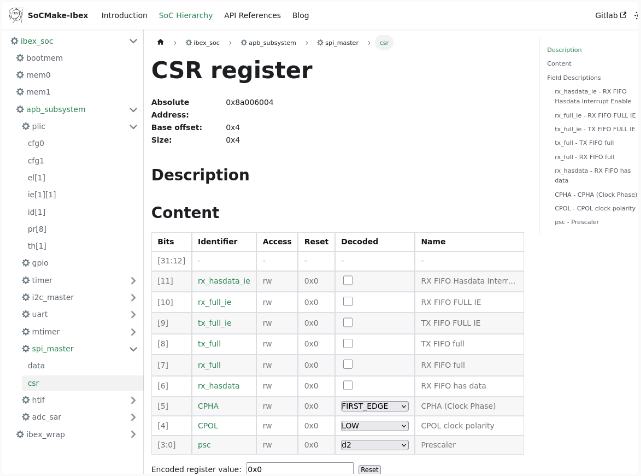


Figure 15: Generated architecture, visit next section

PeakRDL-docusaurus, pdf, markdown, html

- Generates HTML documentation
- Always up to date, no inconsistencies
- Docusaurus (Facebook) website
- Interactive regvalue calculator
- PDF, Markdown, HTML alternatives



ibex_soc
- bootmem
- mem0
- mem1
- apb_subsystem
  - plic
    - cfg0
    - cfg1
    - el[1]
    - ie[1][1]
    - id[1]
    - pr[8]
    - th[1]
  - gpio
  - timer
  - i2c_master
  - uart
  - mtimer
  - spi_master
    - data
    - csr
  - htif
  - adc_sar
- ibex_wrap

ibex_soc › apb_subsystem › spi_master › csr

CSR register

Absolute Address: 0x8a006004
Base offset: 0x4
Size: 0x4

Description

Content

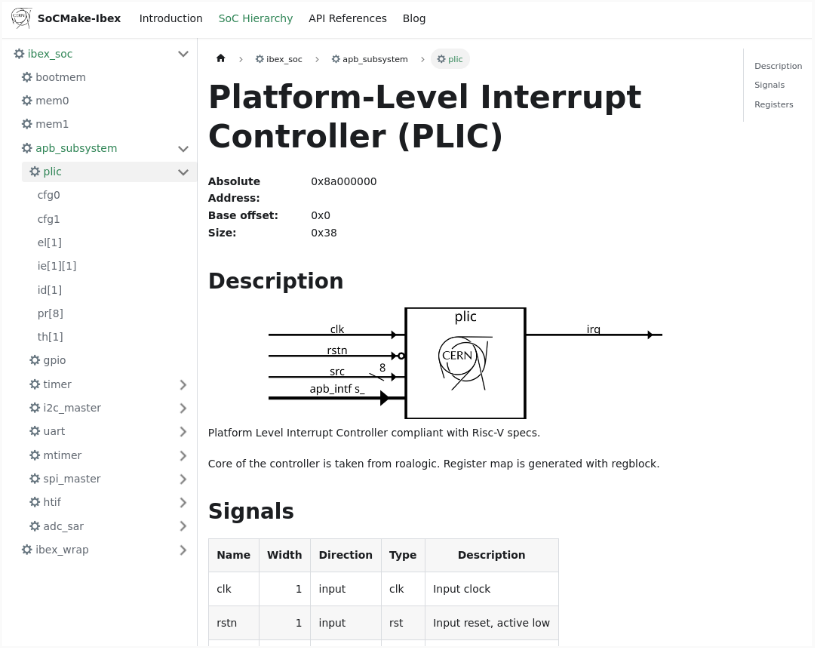
Bits	Identifier	Access	Reset	Decoded	Name
[31:12]	-	-	-	-	-
[11]	rx_hasdata_ie	rw	0x0	☐	RX FIFO Hasdata Interr...
[10]	rx_full_ie	rw	0x0	☐	RX FIFO FULL IE
[9]	tx_full_ie	rw	0x0	☐	TX FIFO FULL IE
[8]	tx_full	rw	0x0	☐	TX FIFO full
[7]	rx_full	rw	0x0	☐	RX FIFO full
[6]	rx_hasdata	rw	0x0	☐	RX FIFO has data
[5]	CPHA	rw	0x0	FIRST_EDGE	CPHA (Clock Phase)
[4]	CPOL	rw	0x0	LOW	CPOL clock polarity
[3:0]	psc	rw	0x0	d2	Prescaler

Encoded register value: 0x0 Reset

Description
Content
Field Descriptions
- rx_hasdata_ie - RX FIFO Hasdata Interrupt Enable
- rx_full_ie - RX FIFO FULL IE
- tx_full_ie - TX FIFO FULL IE
- tx_full - TX FIFO full
- rx_full - RX FIFO full
- rx_hasdata - RX FIFO has data
- CPHA - CPHA (Clock Phase)
- CPOL - CPOL clock polarity
- psc - Prescaler

Figure 16: HTML documentation

PeakRDL-ipblocksvg



SoCMake-Ibex Introduction SoC Hierarchy API References Blog

- ibex_soc
  - bootmem
  - mem0
  - mem1
  - apb_subsystem
    - plic
      - cfg0
      - cfg1
      - el[1]
      - ie[1][1]
      - id[1]
      - pr[8]
      - th[1]
    - gpio
    - timer
    - i2c_master
    - uart
    - mtimer
    - spi_master
    - htif
    - adc_sar
  - ibex_wrap

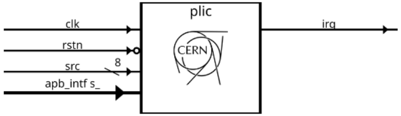
ibex_soc › apb_subsystem › plic

Description
Signals
Registers

Platform-Level Interrupt Controller (PLIC)

Absolute Address:	0x8a000000
Base offset:	0x0
Size:	0x38

Description



Platform Level Interrupt Controller compliant with Risc-V specs.

Core of the controller is taken from roalogic. Register map is generated with regblock.

Signals

Name	Width	Direction	Type	Description
clk	1	input	clk	Input clock
rstn	1	input	rst	Input reset, active low

PeakRDL-halcpp

- C++ 17 HAL (Hardware abstraction layer) driver generator
- Generated SoC hierarchy
- Enhance reuse and readability
  - Enum instead of magic numbers
  - Operator overloads
  - Compile time arrays
- Compile time static template class based
- No increase in binary size compared to C HAL
- Can be used for SystemC-UVM RAL

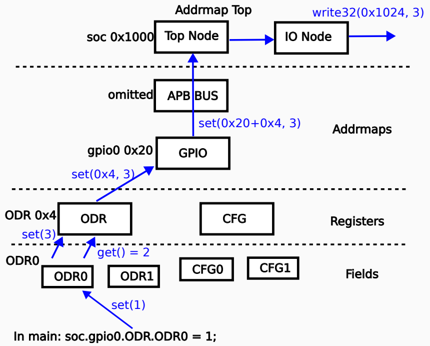


Figure 18: HAL driver architecture

PeakRDL-halcpp

```
int main(){
    IBEX_SOC soc;

    // Opearator overloads
    soc.gpio.odr = 0xf;                 // First
    uint32_t in_data = soc.gpio.idr;    // Second

    // Enumerations and compile time concatenation
    soc.gpio.intcfg = (                 // Third
            Const<28, 0>(),
            irq_edge::RISING_EDGE, irq_en::ENABLED,
            irq_edge::RISING_EDGE, irq_en::DISABLED
            );

    // Compile time array indexing
    soc.plic.pr.at<1>() = 5;            // Fourth
    return 0;
}
```

Figure 19: Operators, Enumerations and arrays

```
// First 1*4 + 2*2 = 8 bytes
lui         a5,0x20001
li         a4,15
sw         a4,4(a5)
// Second 2 bytes
lw         a4,0(a5)

li         a0,0 // Return value for main
// Third, 2*2 = 4 bytes
li         a4,4
sw         a4,12(a5)
// Fourth, access to PLIC change address
// 1*4 + 2*2 = 8bytes
lui         a5,0x20000
li         a4,5
sw         a4,24(a5)

ret // return from main
```

Figure 21: Resulting RiscV assembly (GCC)

```
template<uint32_t BASE, typename PARENT_TYPE=void>
class APB_GPIO : public APB_GPIO_HAL<BASE, PARENT_TYPE> {
public:
  void write_pin(uint16_t pin, bool level) {
      if(level)
          this->BSRR = 1 << pin;
      else
          this->BSRR = 1 << pin << 16;
  }
  APB_GPIO() {}
};
```

Figure 20: Extending generated HAL class

```
Memory region         Used Size  Region Size  %age Used
        bootmem:          0 GB         1 KB      0.00%
           mem0:         286 B        16 KB      1.75%
           mem1:           2 B         4 KB      0.05%
```

Figure 22: Baseline memory usage

```
Memory region         Used Size  Region Size  %age Used
        bootmem:          0 GB         1 KB      0.00%
           mem0:         308 B        16 KB      1.88%
           mem1:           2 B         4 KB      0.05%
```

Figure 23: Memory usage from the given example +22B

Linker script generator

- Generate linker scripts for RiscV or other ISA
- Allow separate data, text, bss, boot, stack sections
- Introduce sections property for memory components in SystemRDL
- *linker_symbol* property to create PROVIDE() symbol for register address

```
mem #(
    .SECTIONS("boot|text"),
    .MEMENTRIES(256)
    ) bootmem @ 0x01100000;

mem_dp #(
    .SECTIONS("text"),
    .MEMENTRIES(4096),
    ) codemem @ 0x01200000;

mem #(
    .SECTIONS("data|bss|stack"),
    .MEMENTRIES(1024),
    ) datamem @ 0x01300000;
```

Figure 24: SystemRDL example with memories using SECTION parameter

Talk Overview

SoCMake

SystemRDL and PeakRDL

Verification

Example SoC designs

Verification

- UVM implementation in SystemC (Accellera)
- Both SystemC, SystemC-UVM and UVM are Accellera standards
- Advantages to SV-UVM
  - Open-source simulator, compile with GCC
  - No license needed for the simulator
  - Write C++ instead of SV
  - Reuse target CPU application code in verif
- Constraint randomization and functional coverage with external libraries

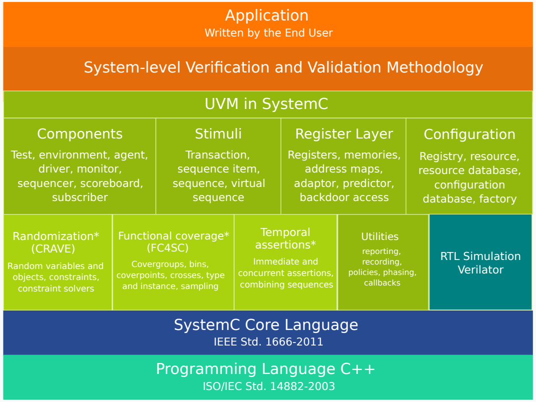
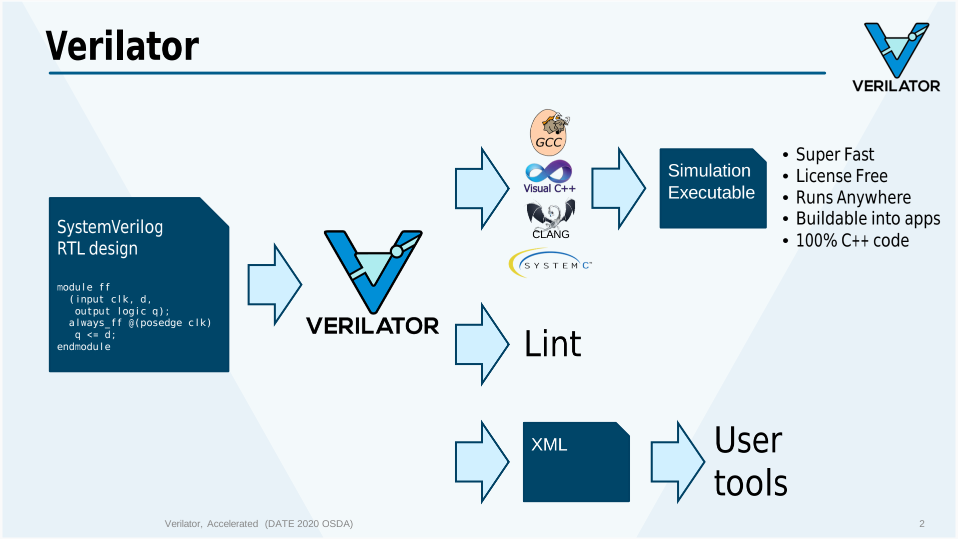

Application
Written by the End User
System-level Verification and Validation Methodology
UVM in SystemC
Components
Test, environment, agent, driver, monitor, sequencer, scoreboard, subscriber
Stimuli
Transaction, sequence item, sequence, virtual sequence
Register Layer
Registers, memories, address maps, adaptor, predictor, backdoor access
Configuration
Registry, resource, resource database, configuration database, factory
Randomization* (CRAVE)
Random variables and objects, constraints, constraint solvers
Functional coverage* (FC4SC)
Covergroups, bins, coverpoints, crosses, type and instance, sampling
Temporal assertions*
Immediate and concurrent assertions, combining sequences
Utilities
reporting, recording, policies, phasing, callbacks
RTL Simulation Verilator
SystemC Core Language
IEEE Std. 1666-2011
Programming Language C++
ISO/IEC Std. 14882-2003

Figure 25: SystemC UVM software stack

Verilator



Verilator

SystemVerilog RTL design

```
module ff
  (input clk, d,
   output logic q);
  always_ff @(posedge clk)
   q <= d;
endmodule
```

- Super Fast
- License Free
- Runs Anywhere
- Buildable into apps
- 100% C++ code

Verilator, Accelerated (DATE 2020 OSDA) 2

SystemC-UVM

- Provided IP blocks and SoCs come with SystemC-UVM env
- Continous integration friendly (no license server)
- Verification environment is tested on an OrangePi-5 SBC (RK3588s ARM SoC 8c)
    - GitLab CI simulation runners on ARM
    - ARM based simulation cluster
- SEU injection framework based on VPI
- Analog simulation with SystemC-AMS supported
- Investigate Zynq FPGA emulation in the future

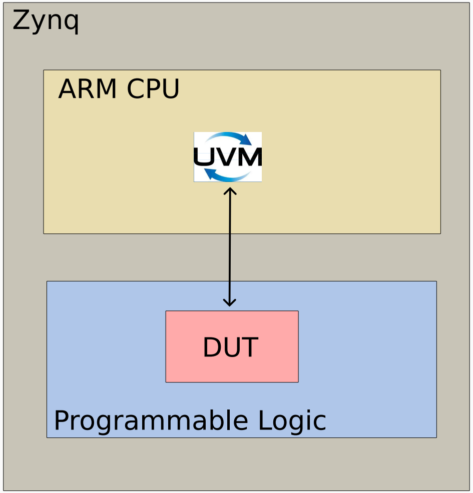


Figure 26: FPGA emulation idea

Talk Overview

Example SoC designs

- Suports 4 CPUs for now
    - LowRISC Ibex
    - ChipsAlliance (Western Digital) EL2
    - Syntacore SCR1
    - Picorv32
- Developed 4 SoC designs as examples
    - SoCMake-Ibex
    - SoCMake-EL2
    - SoCMake-SCR1
    - SoCMake-PicoRV32

SoCMake-Ibex

- Website (socmake-ibex.docs.cern.ch)
- LowRISC Ibex RiscV core
- SoC peripherals (reusable separate git repos)
    - GPIO
    - I2C Master
    - SPI Master
    - RiscV 64bit machine timer
    - RiscV PLIC (Platform Local Interrupt Controller)
    - Timer - General purpose 32bit timer
    - UART
    - ADC-SAR (Successive Approximation Register)

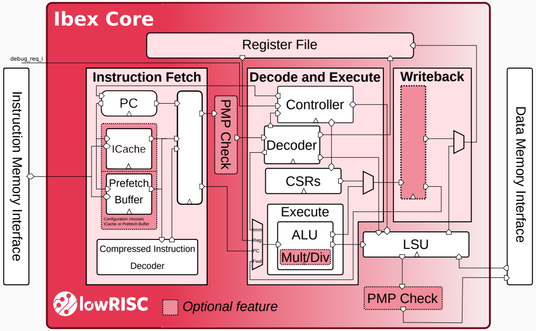
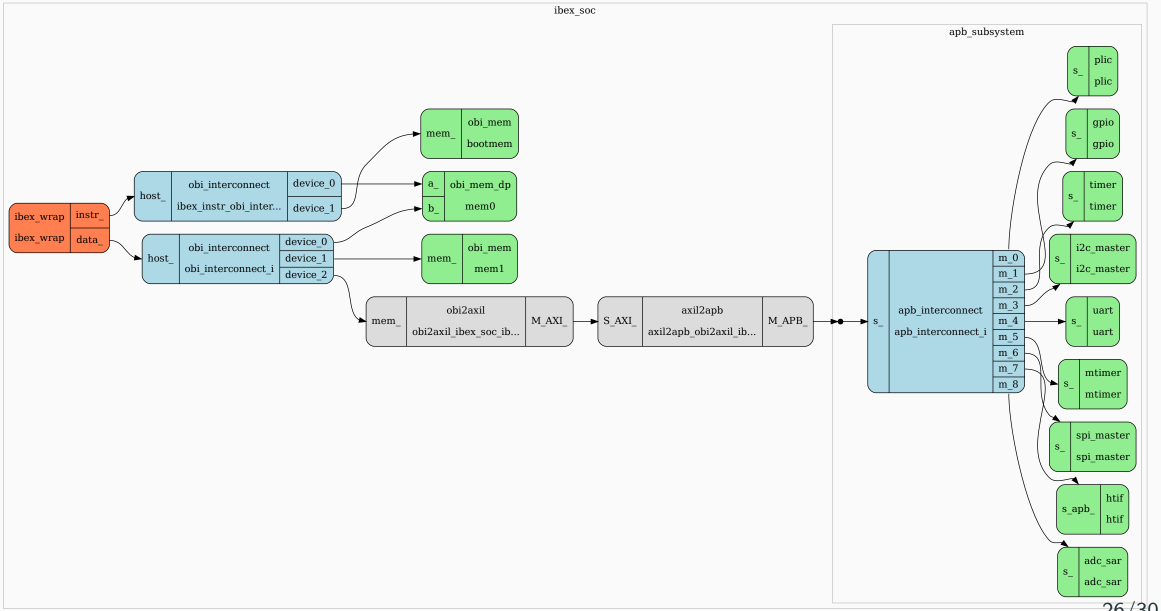


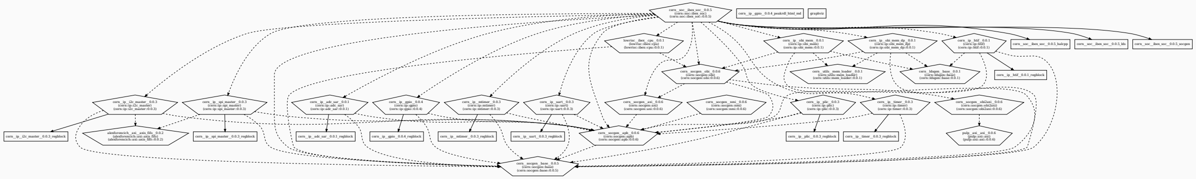
Figure 27: Ibex pipeline architecture

SoCMake-Ibex block diagram (autogenerated)



SoCMake-Ibex SoCMake build graph (autogenerated)

- To generate build graph run: `make graphviz` after project is configured
- Can be useful for analysing missing links


cern__soc__ibex_soc__0.0.5 (cern::soc::ibex_soc) (cern::soc::ibex_soc::0.0.5)
cern__ip__gpio__0.0.4_peakrdl_html_md
graphviz
lowrisc__ibex__cpu__0.0.1 (lowrisc::ibex::cpu) (lowrisc::ibex::cpu::0.0.1)
cern__ip__obi_mem__0.0.1 (cern::ip::obi_mem) (cern::ip::obi_mem::0.0.1)
cern__ip__obi_mem_dp__0.0.1 (cern::ip::obi_mem_dp) (cern::ip::obi_mem_dp::0.0.1)
cern__ip__htif__0.0.1 (cern::ip::htif) (cern::ip::htif::0.0.1)
cern__soc__ibex_soc__0.0.5_halcpp
cern__soc__ibex_soc__0.0.5_lds
cern__soc__ibex_soc__0.0.5_socgen
cern__socgen__obi__0.0.6 (cern::socgen::obi) (cern::socgen::obi::0.0.6)
cern__utils__mem_loader__0.0.1 (cern::utils::mem_loader) (cern::utils::mem_loader::0.0.1)
cern__ldsgen__base__0.0.1 (cern::ldsgen::base) (cern::ldsgen::base::0.0.1)
cern__ip__htif__0.0.1_regblock
cern__ip__i2c_master__0.0.3 (cern::ip::i2c_master) (cern::ip::i2c_master::0.0.3)
cern__ip__spi_master__0.0.3 (cern::ip::spi_master) (cern::ip::spi_master::0.0.3)
cern__ip__adc_sar__0.0.1 (cern::ip::adc_sar) (cern::ip::adc_sar::0.0.1)
cern__ip__gpio__0.0.4 (cern::ip::gpio) (cern::ip::gpio::0.0.4)
cern__ip__mtimer__0.0.3 (cern::ip::mtimer) (cern::ip::mtimer::0.0.3)
cern__ip__uart__0.0.3 (cern::ip::uart) (cern::ip::uart::0.0.3)
cern__socgen__axi__0.0.6 (cern::socgen::axi) (cern::socgen::axi::0.0.6)
cern__socgen__nmi__0.0.6 (cern::socgen::nmi) (cern::socgen::nmi::0.0.6)
cern__ip__plic__0.0.3 (cern::ip::plic) (cern::ip::plic::0.0.3)
cern__ip__timer__0.0.3 (cern::ip::timer) (cern::ip::timer::0.0.3)
cern__socgen__obi2axi__0.0.6 (cern::socgen::obi2axi) (cern::socgen::obi2axi::0.0.6)
cern__ip__i2c_master__0.0.3_regblock
alexforencich__axi__axis_fifo__0.0.2 (alexforencich::axi::axis_fifo) (alexforencich::axi::axis_fifo::0.0.2)
cern__ip__spi_master__0.0.3_regblock
cern__ip__adc_sar__0.0.1_regblock
cern__ip__gpio__0.0.4_regblock
cern__ip__mtimer__0.0.3_regblock
cern__ip__uart__0.0.3_regblock
cern__socgen__apb__0.0.6 (cern::socgen::apb) (cern::socgen::apb::0.0.6)
cern__ip__plic__0.0.3_regblock
cern__ip__timer__0.0.3_regblock
pulp__axi__axi__0.0.6 (pulp::axi::axi) (pulp::axi::axi::0.0.6)
cern__socgen__base__0.0.5 (cern::socgen::base) (cern::socgen::base::0.0.5)

FPGA demo

- FPGA implementation to demonstrate working system
- Tested on Genesys2 and Zybo boards
- UART bootloader for loading the application code from the PC
- SPI master driving OLED display

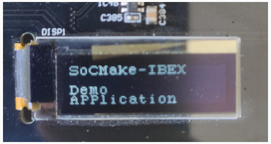


Figure 28: SoCMake-Ibex booting on FPGA

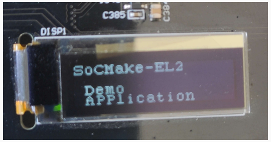


Figure 29: SoCMake-EL2 booting on FPGA

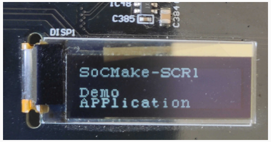


Figure 30: SoCMake-SCR1 booting on FPGA

Summary

- Hardware Build System SoCMake
- SystemRDL tooling for automatic generation of SoC components
- Open Source Verification environment
- SoC examples

Future Work

- Find a real application, on-detector ASIC chip
- Add more CPU cores (Alibaba T-head, OpenHW group...)
- SoCMake support more EDA tools (Implementation flows)
- Multiprocessor support
- SystemC-UVM environment generation
- In process with KT (CERN Knowledge Transfer) to open source

Questions?