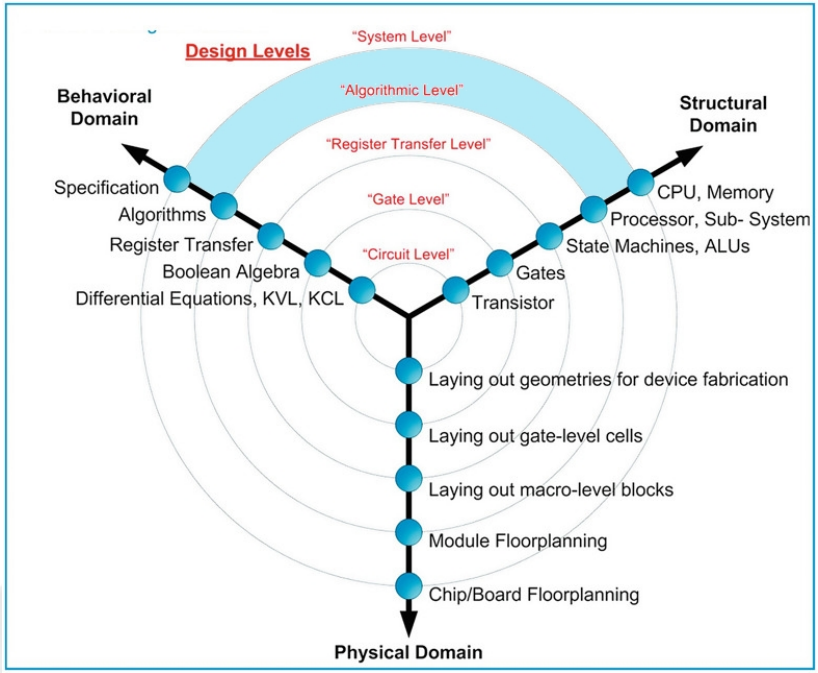
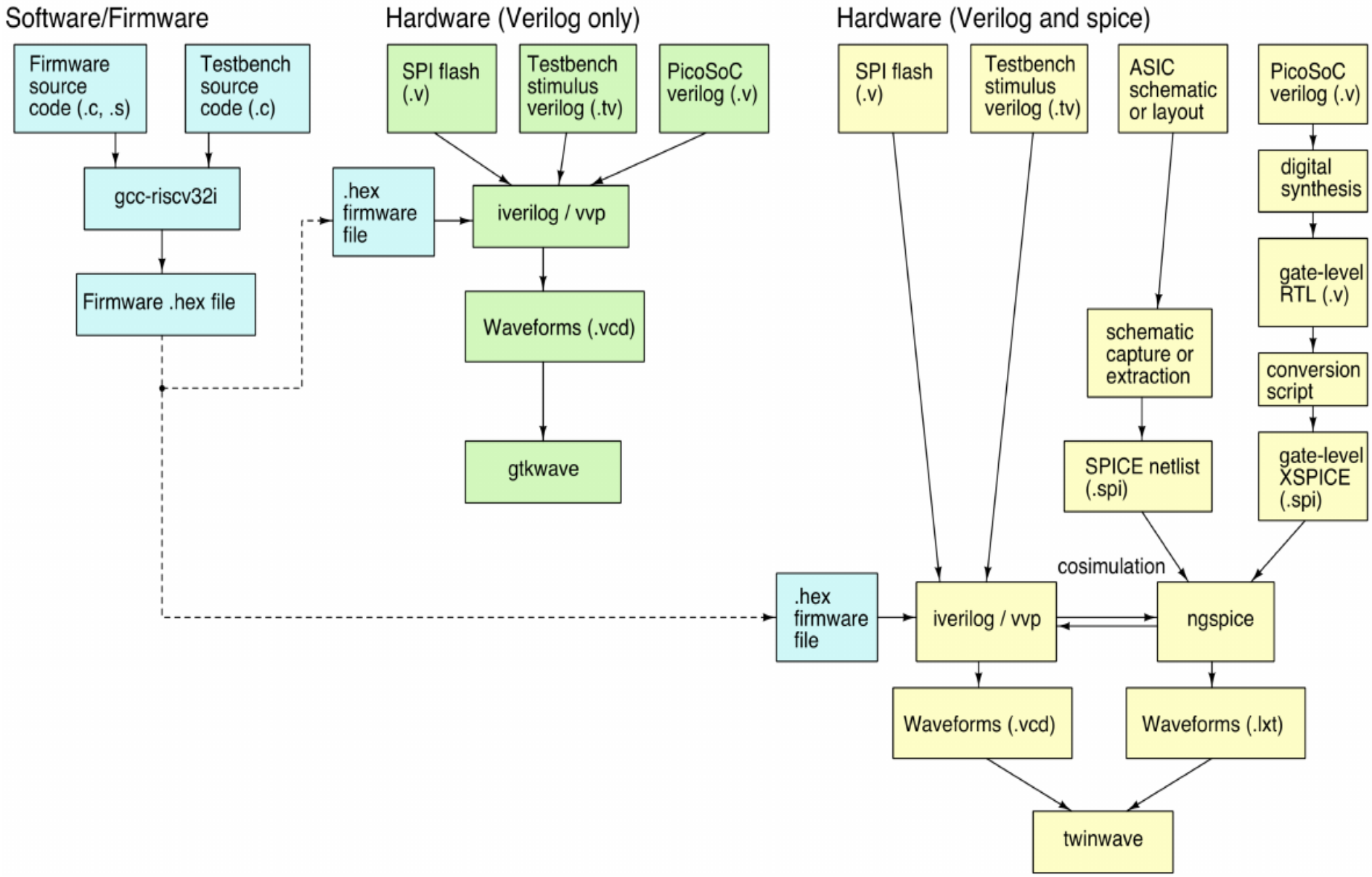
Basics of Processor based Systems

Dr. Tassadaq Hussain




Software/Firmware
Hardware (Verilog only)
Hardware (Verilog and spice)
Firmware source code (.c, .s)
Testbench source code (.c)
gcc-riscv32i
Firmware .hex file
SPI flash (.v)
Testbench stimulus verilog (.tv)
PicoSoC verilog (.v)
.hex firmware file
iverilog / vvp
Waveforms (.vcd)
gtkwave
SPI flash (.v)
Testbench stimulus verilog (.tv)
ASIC schematic or layout
PicoSoC verilog (.v)
digital synthesis
gate-level RTL (.v)
conversion script
gate-level XSPICE (.spi)
schematic capture or extraction
SPICE netlist (.spi)
cosimulation
.hex firmware file
iverilog / vvp
ngspice
Waveforms (.vcd)
Waveforms (.lxt)
twinwave

Topics

- **Symmetric Processor Architecture**

  GPP, Many modern multi-core processors for desktops, servers, and HPC systems.

- Application Specific Processor

  Accelerators, DSP/Vector/GPU

- Asymmetric Processor Architecture:

  Heterogeneous Cores, Smartphones and Tablet

- Frameworks for customizable RISC-V System
- RISCV System Toolchain

General Purpose Processing

General-purpose processor: Have multiple cores, each core can execute a wide range of instructions and run various types of software applications.

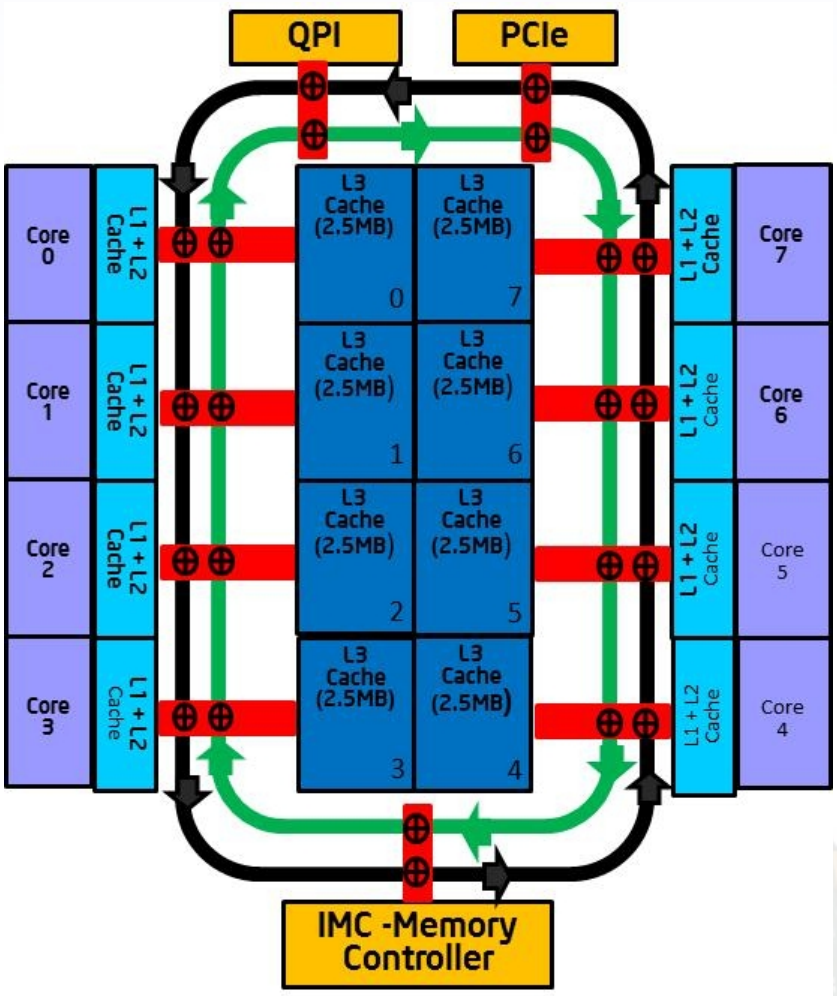
(1,500 to 2,000 Instructions)

Versatility: Bus System is designed for versatility and can handle a broad range of peripheral and components.

Operating Systems: CPUs are well-suited for running general-purpose operating systems like Windows, Linux, and macOS. They can execute a wide variety of software, from web browsers and office applications to scientific simulations and gaming.

Performance: High-performance CPUs are designed to deliver excellent single-threaded performance and are optimized for tasks that do not require extensive parallel processing.

Many-Core CPUs: Modern CPUs often have multiple cores to improve overall performance, allowing them to handle parallel workloads more effectively.



Topics

- Symmetric Processor Architecture

  GPP, Many modern multi-core processors for desktops, servers, and HPC systems.

- **Application Specific Processor**

  Accelerators, DSP/Vector/GPU

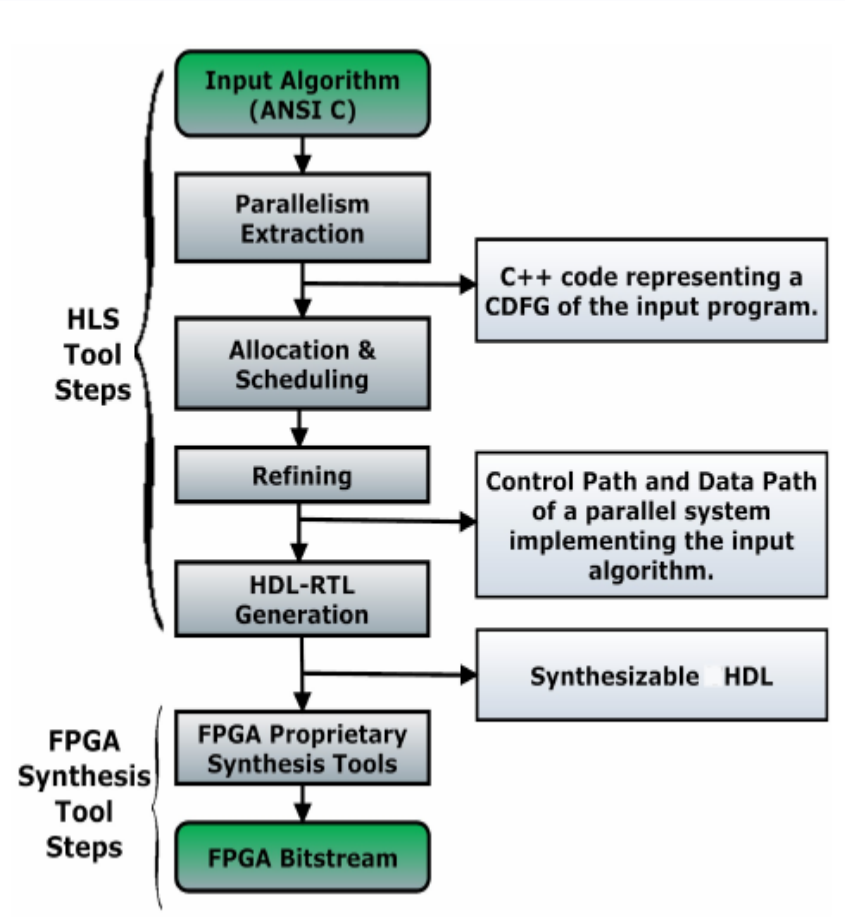
- Asymmetric Processor Architecture:

  Heterogeneous Cores, Smartphones and Tablet

- Frameworks for customizable RISC-V System
- RISCV System Toolchain

Application Specific Processor:
High Level Synthesis

- High-level synthesis (HLS) is a design process used primarily in digital hardware design for FPGAs and, to some extent, for ASICs in VLSI design.
- **Reducing design and verification efforts**
- **More effective reuse**
- **Investing R&D resources where it really matters**
- **Testing and verifications**



HLS Components

HLS systems restrict the target hard-ware.

HLS synthesis tools have their own peculiarities; but most systems generate synchronous hardware and build it with the following parts:

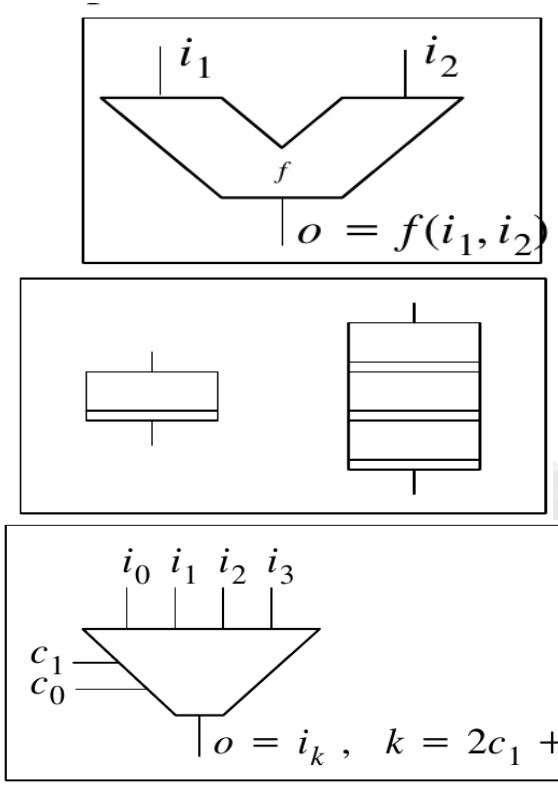
ALU

Registers

MUX

Buses

Three State Driver (Controller)



HLS Hardware Concept

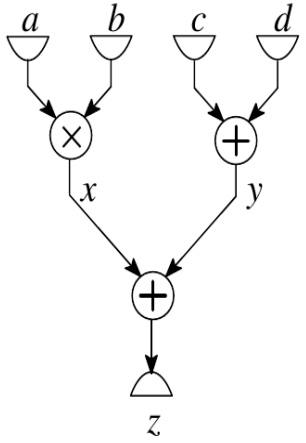
Data Path + Control Structure

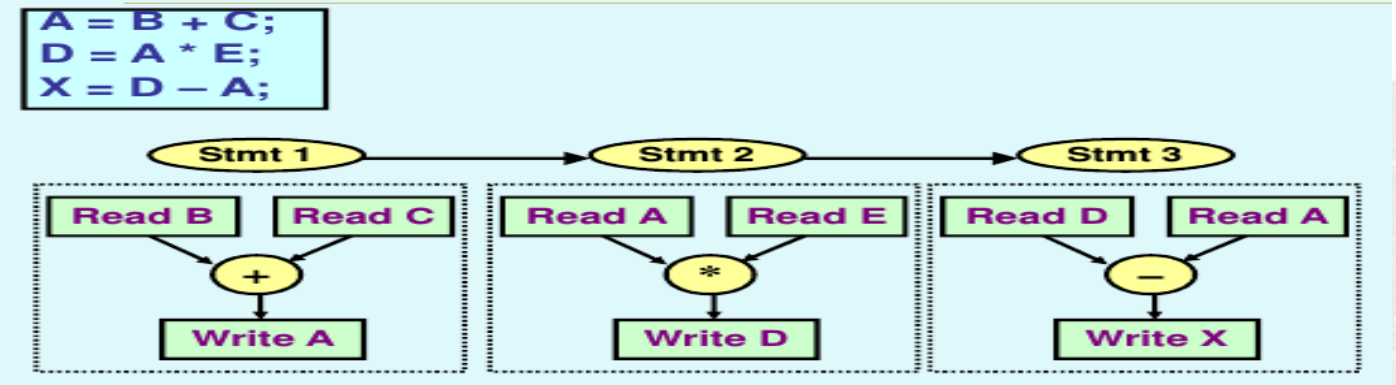
The data path: a network of functional units, registers, multiplexers and buses. The actual ‘‘computation’’ takes place in the data path.

Control: the part of the hardware that takes care of having the data present at the right place at a specific time, of presenting the right instructions to a programmable unit, etc.

Optimization

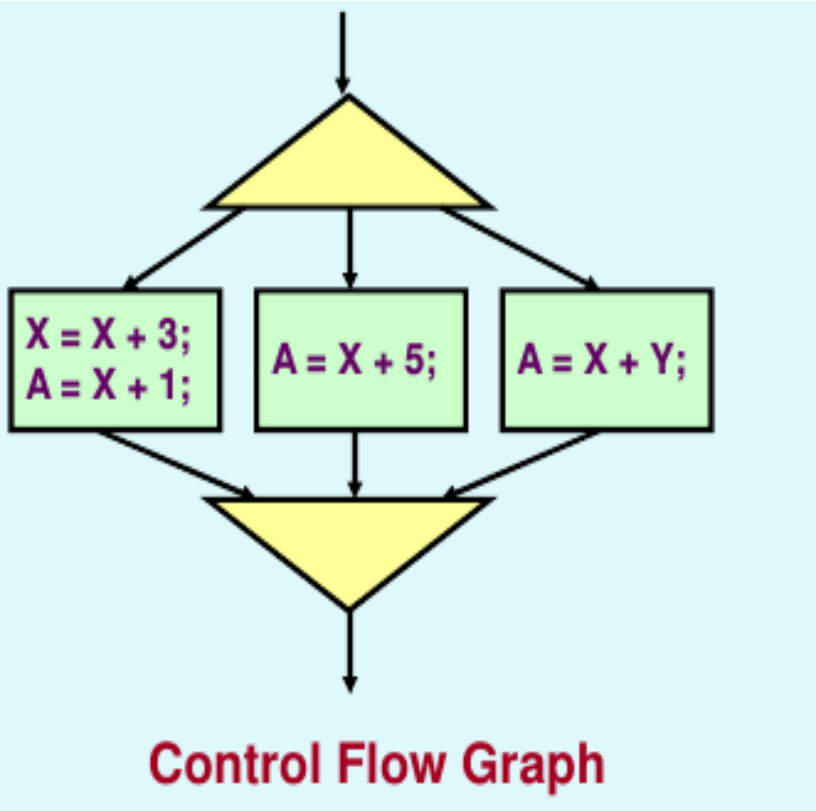
```
x := a * b; y := c + d;
z := x + y;
```





Control-flow Graph

```
case (C)
1:
begin
X = X + 3;
A = X + 1;
end
2:
A = X + 5;
default:
A = X + Y;
endcase
```

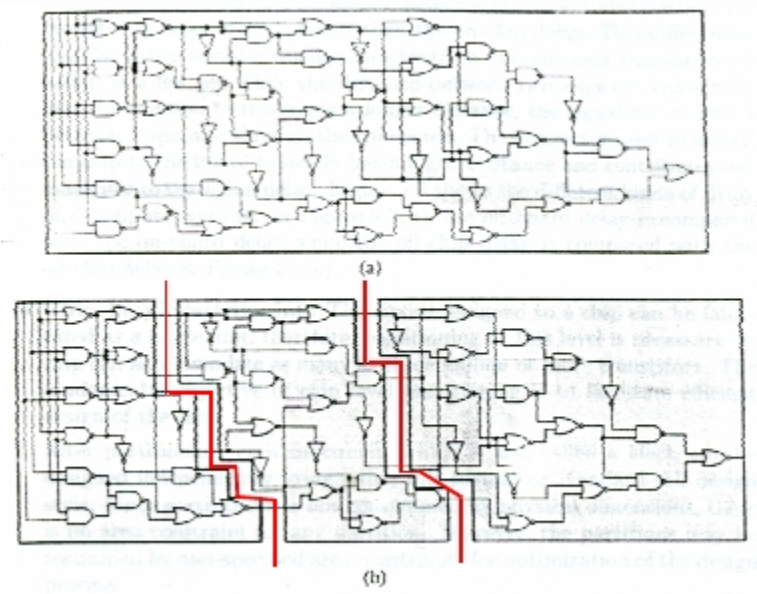


HLS Compiler Transformation

- Constant folding
- Redundant operator elimination
- Tree height transformation
- Control flattening
- Logic level transformation
- Register-Transfer level transformation
- Loop Unrooling

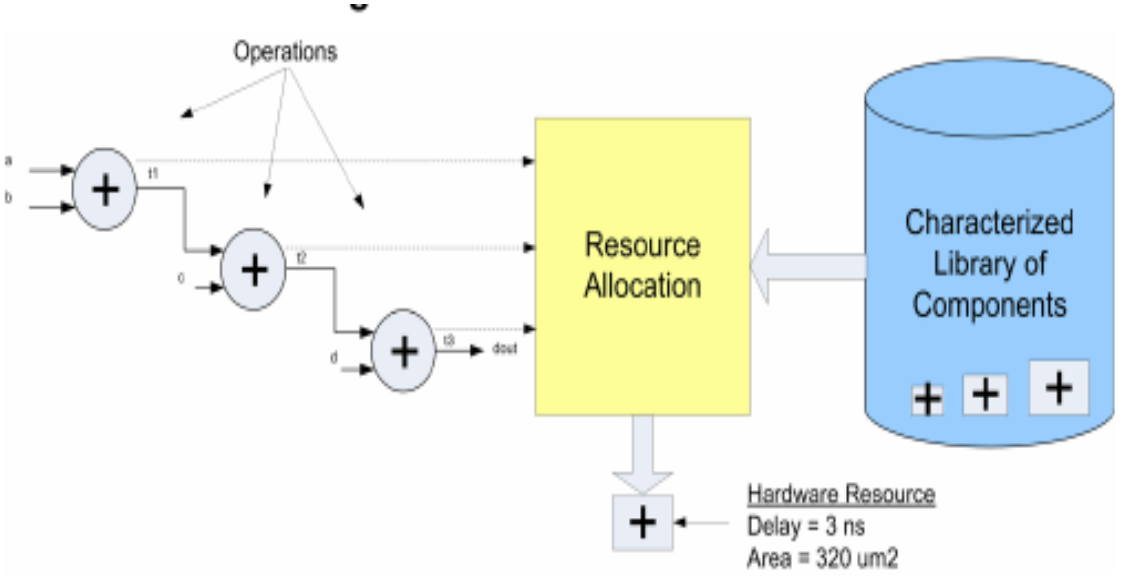
Partitioning

- Scheduling
- Allocation
- Unit selection



Resource Allocation

Once the DFG has been assembled, each operation is mapped onto a hardware resource which is then used during scheduling.



Topics

- Symmetric Processor Architecture

  GPP, Many modern multi-core processors for desktops, servers, and HPC systems.

- Application Specific Processor

  Accelerators, DSP/Vector/GPU

- **Asymmetric Processor Architecture:**

  Heterogeneous Cores, Smartphones and Tablet, **System on Chip Architecture**

- Frameworks for customizable RISC-V System
- RISCV System Toolchain

System on Chip

Integration: An SoC is an integrated circuit that combines multiple hardware components onto a single chip. These components typically include one or more CPU cores, memory, peripherals (e.g., USB controllers, networking interfaces), and often specialized hardware blocks (e.g., GPU, DSP, hardware accelerators).

ARM 100 to 300 – RISCV 50 ISA

Dedicated Tasks: SoCs are designed for specific applications or tasks. They are often used in embedded systems, mobile devices, IoT devices, and other applications where a combination of processing, memory, and I/O capabilities are needed on a single chip.

Customization: SoCs can be customized to meet the specific requirements of an application. Designers can select the CPU cores, peripherals, and specialized hardware blocks that best suit the intended use case.

Efficiency: SoCs are often designed for power efficiency, which makes them suitable for battery-powered devices and embedded systems.

Parallel Processing: Some SoCs may include multiple CPU cores or specialized hardware for parallel processing tasks, which can enhance performance for certain workloads.

Processor Based System

- Hardware
  - Processor
  - Bus
  - Memory
  - Peripherals
- Software
  - System Software
  - ToolChain and Libraries
  - User Software

Clock

Bit Width

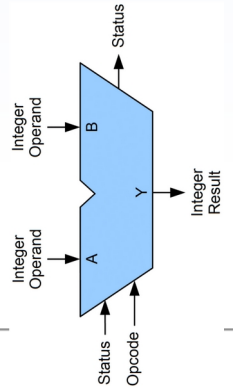
ALUs

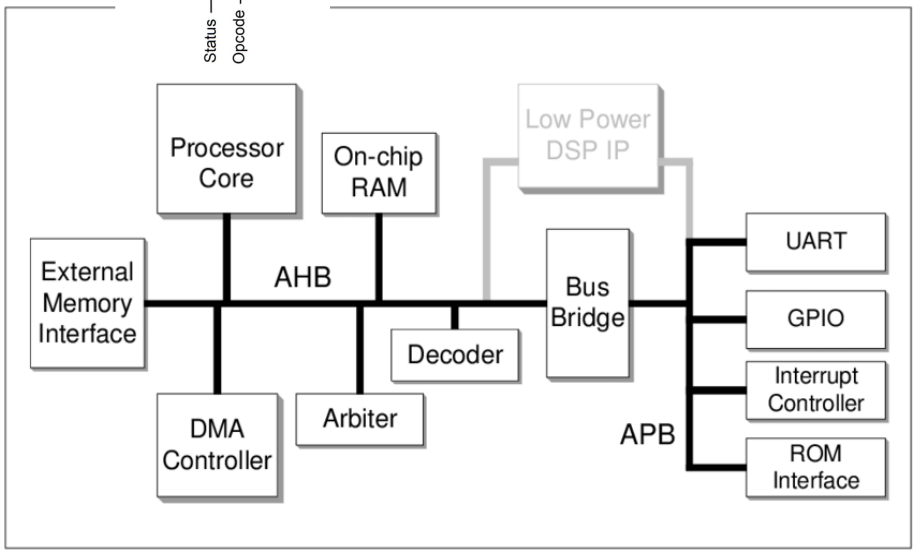
Pipeline Stage

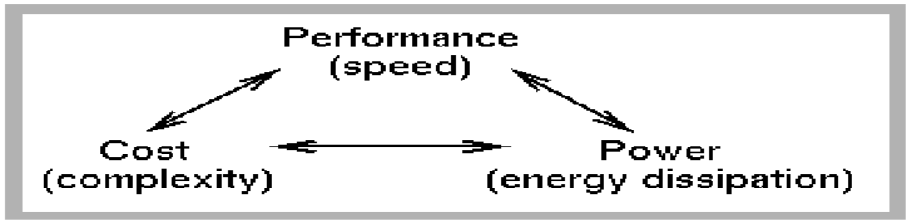
Execution (OoO, SuperScalar) etc.

Performance, Power and Cost

Security and Privacy







System on Chip Design

A collection of all kinds of components and/or subsystems that are appropriately interconnected to perform the specified functions for end users.

Customization Modular Approach: Design “product creation process” which

“Starts at identifying the end-user needs and Ends at delivering a product with enough functional satisfaction.

Benefits:

Reduced size, Reduced overall system cost, Lower power consumption, Increased performance

Time to Market and Design Complexity (DSP, uP, HW/SW, RTOS).

SoC Main Parts

Heterogeneous Core

IPs, Memory Mapped Bus Platform

Interconnection Platform

Partitioning Functionality based HW and SW

Modeling, Testing and Verification

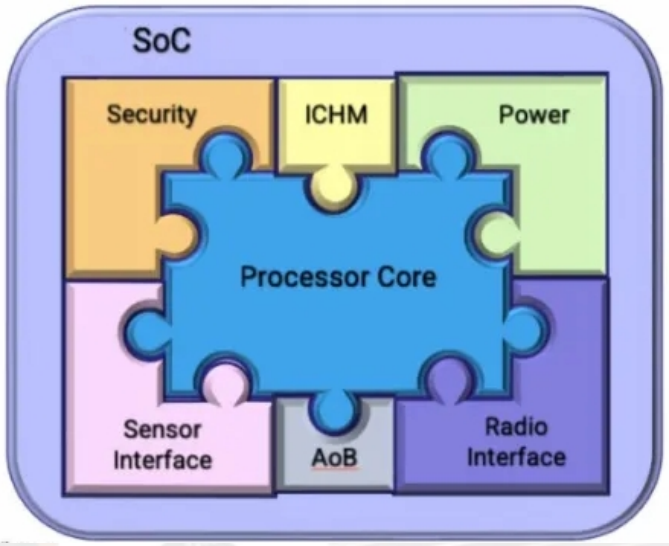
IP Core

An intellectual property core, or IP core, is a block of logic or data that is used in the creation of a semiconductor chip. And is the Intellectual Property of a person or company.

A predefined, designed/verified, reusable building block for System-on-Chip

Software IP, Silicon IP (Soft IP, Hard IP, ...)

IP cores in a System-on-Chip (SoC) are designed to be accessed through memory addresses, just like regular memory locations.



Heterogeneous IP: Selection

Power, performance, area, cost

Flexibility

Hardness (hard IP vs. soft IP)

Available system software

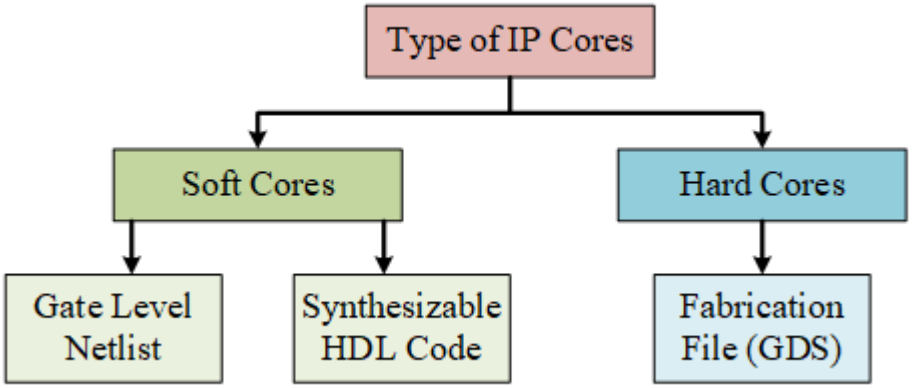
Development environment

Simulation model

Support library

Support OS

Inter-operability with other IP’s



IP Cores: SoC Requirement

- **IP Integration**
  - To support use of heterogeneous commercial IP
- **Hard IP Transition**
  - Better physical design tool
- **IP Standards**
  - Facilitate use of IP from multiple sources
- **IP security**
  - Support various business model

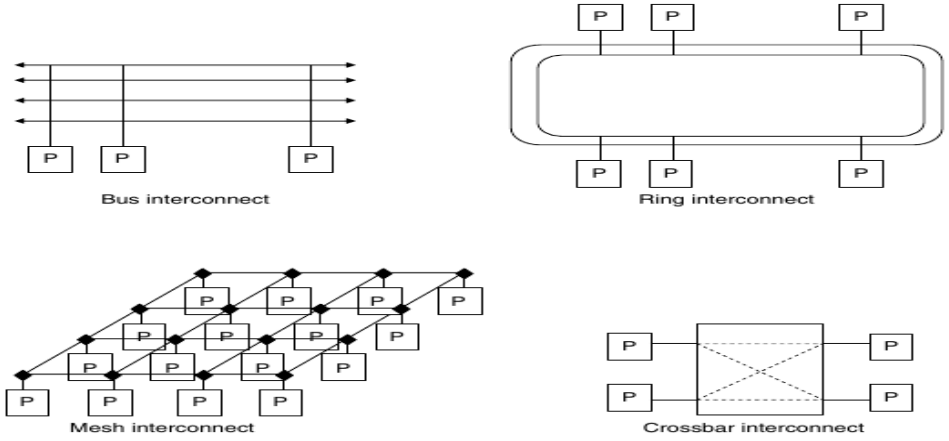
Interconnection Platform

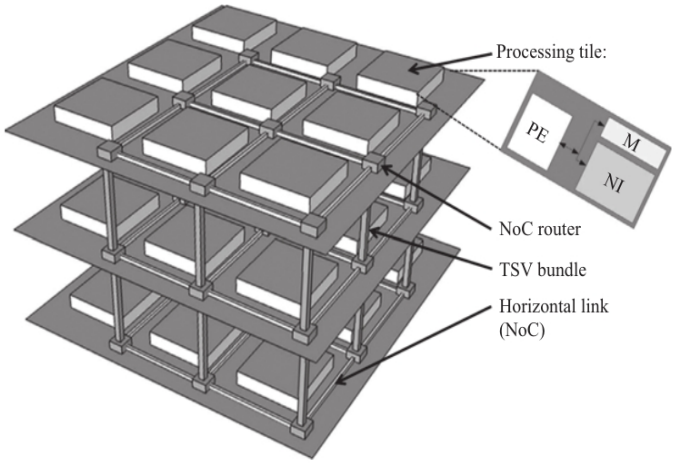
A fully defined bus structure and a collection of IP blocks

A design methodology to support the feature of “Plugging and Playing”

The definition of a platform is the result of a trade-off process involving re-usability (programmability and configure-ability), cost and performance optimization.

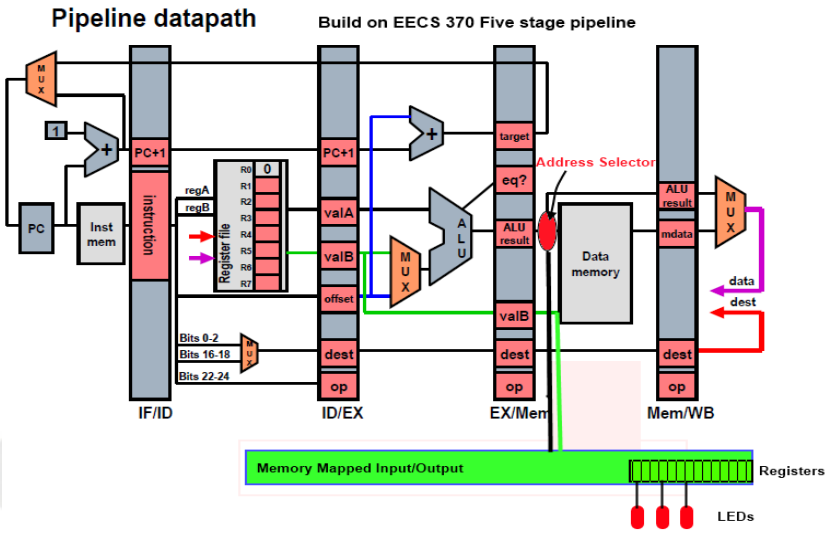
Enhance the differentiation

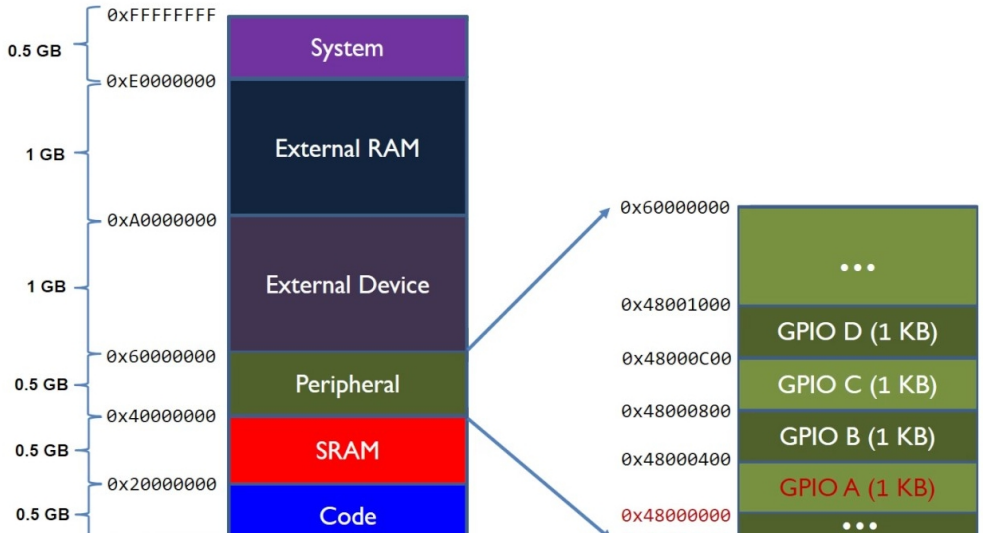




Memory/Address Mapping

A memory map is the bridge between a system-on-chip (SoC) and the firmware and software that is executed on it.





Memory Address Space

All cores are assigned specific memory addresses within the address space of the SoC. Each IP core's control registers, status registers, and data buffers are associated with these addresses.

Control registers, Status registers, and Data buffers Interrupt Handling: Memory-mapped IP cores can generate interrupts to notify the CPU of specific events or conditions. The CPU can check interrupt status registers to determine which IP core generated an interrupt and take appropriate action.

Synchronization: Memory-mapped I/O is often synchronized with memory access instructions provided by the CPU. This means that the CPU can use load and store instructions to read from and write to the memory-mapped addresses, making it a natural and efficient method for controlling and communicating with IP cores.

SoC: Design and Development

HDL

Test Bench

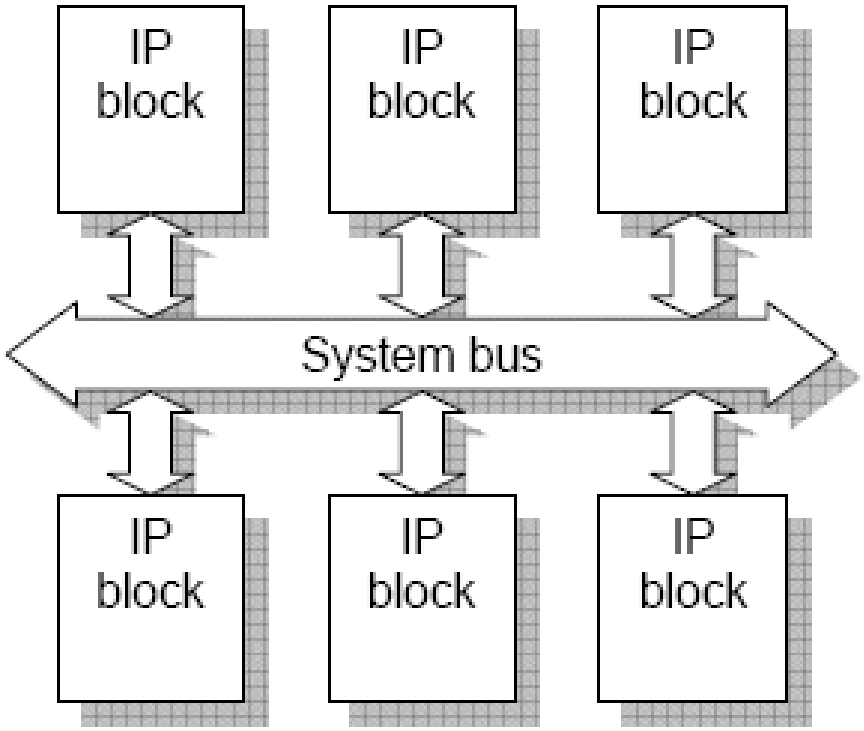
Iverilog

Verilator

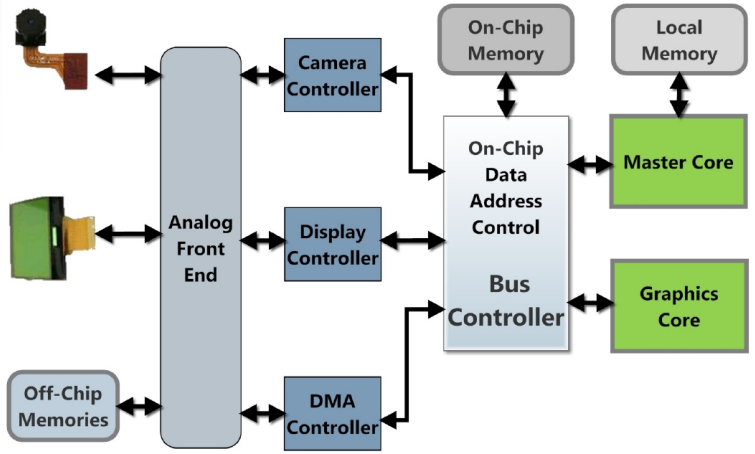
CocoTB

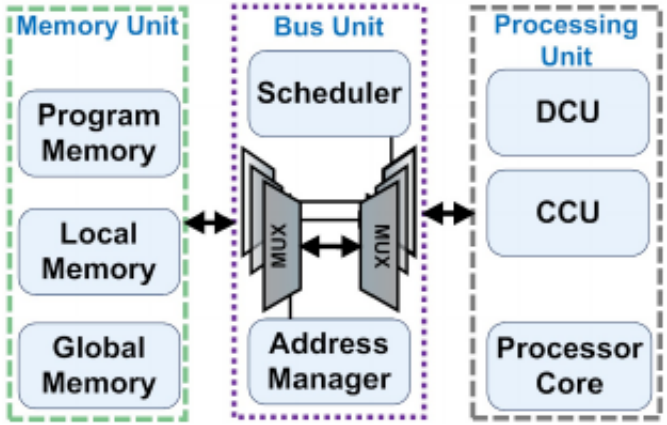
MyHDL

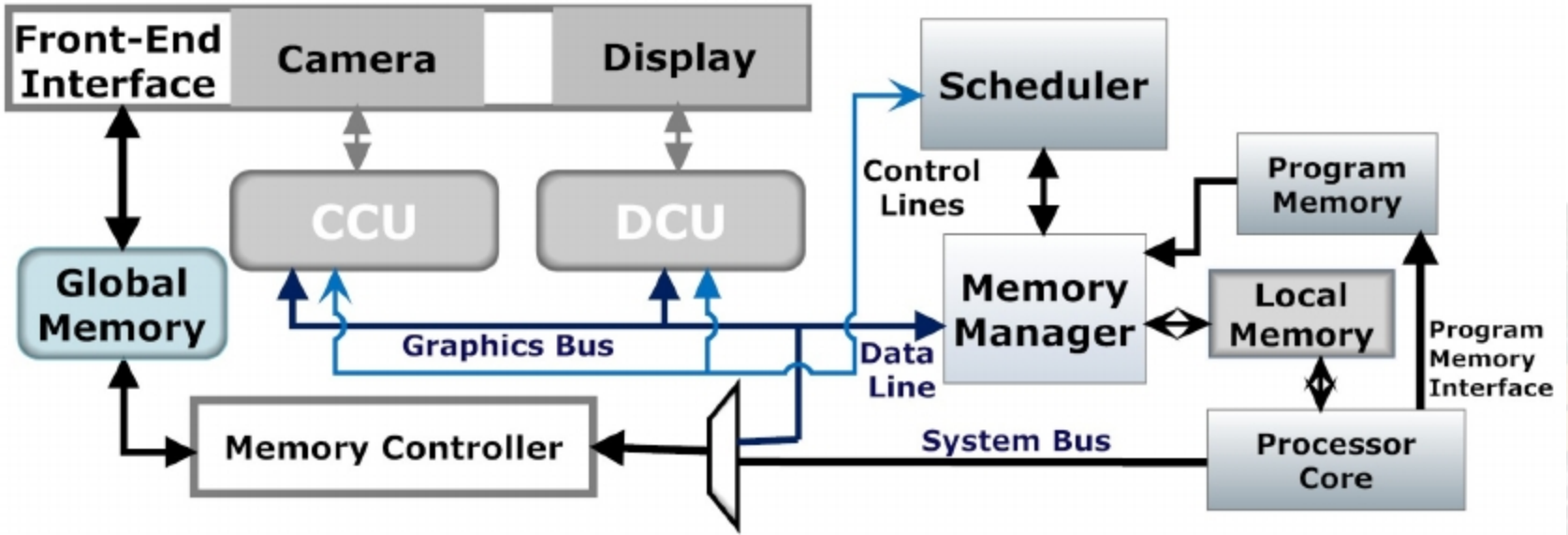
GTKWave vpp



Architecture Design (Infineon XGOLD)







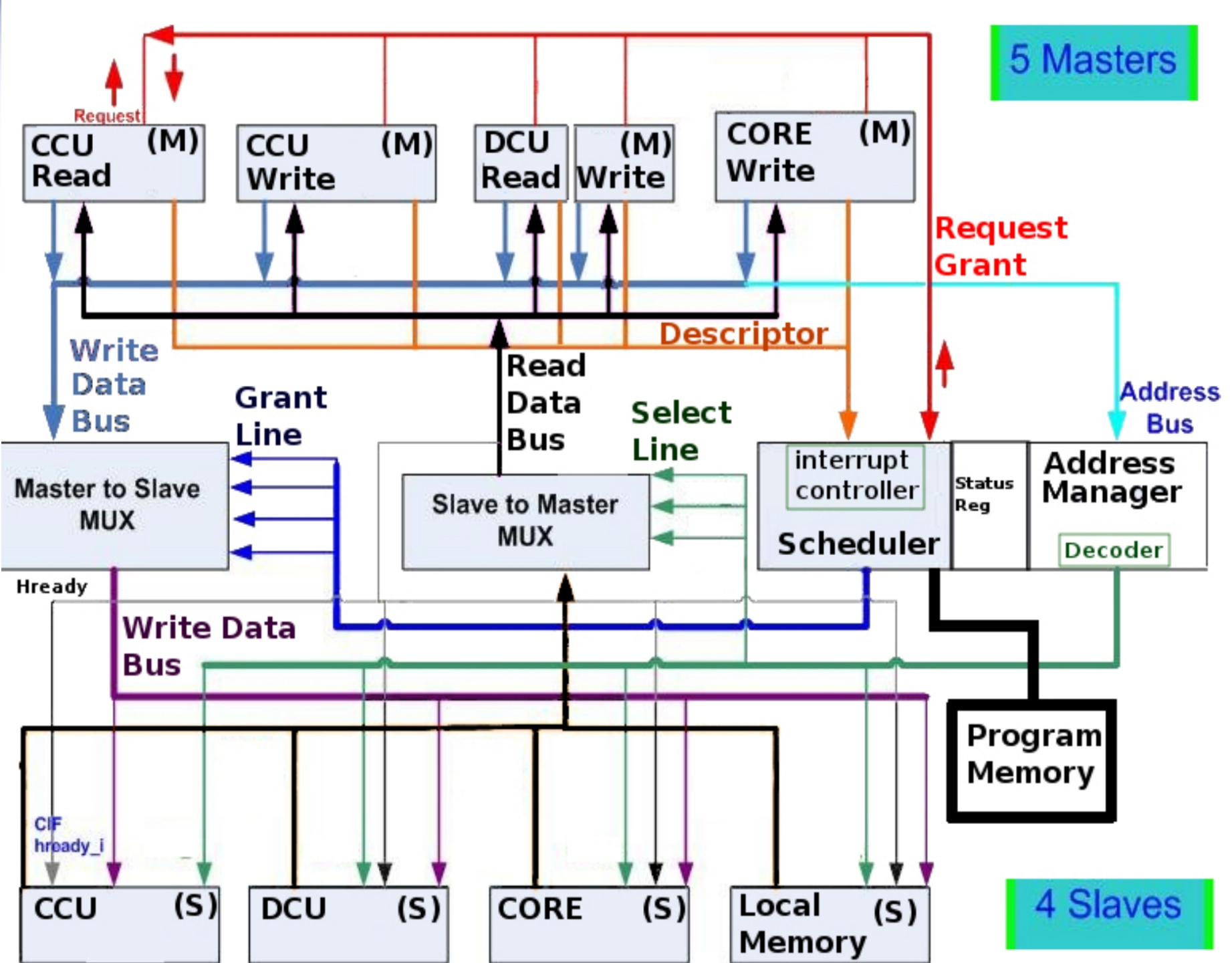
Cadence: Camera Control Unit

Module (Name)	Memory bits	Gate Count (k gates)
Image Signal Processor	135128	179
Color Processing	41600	7
Main Resize	83200	34
Memory Interface	0	113
JPEG Encoder	338944	83
Total	598872	416

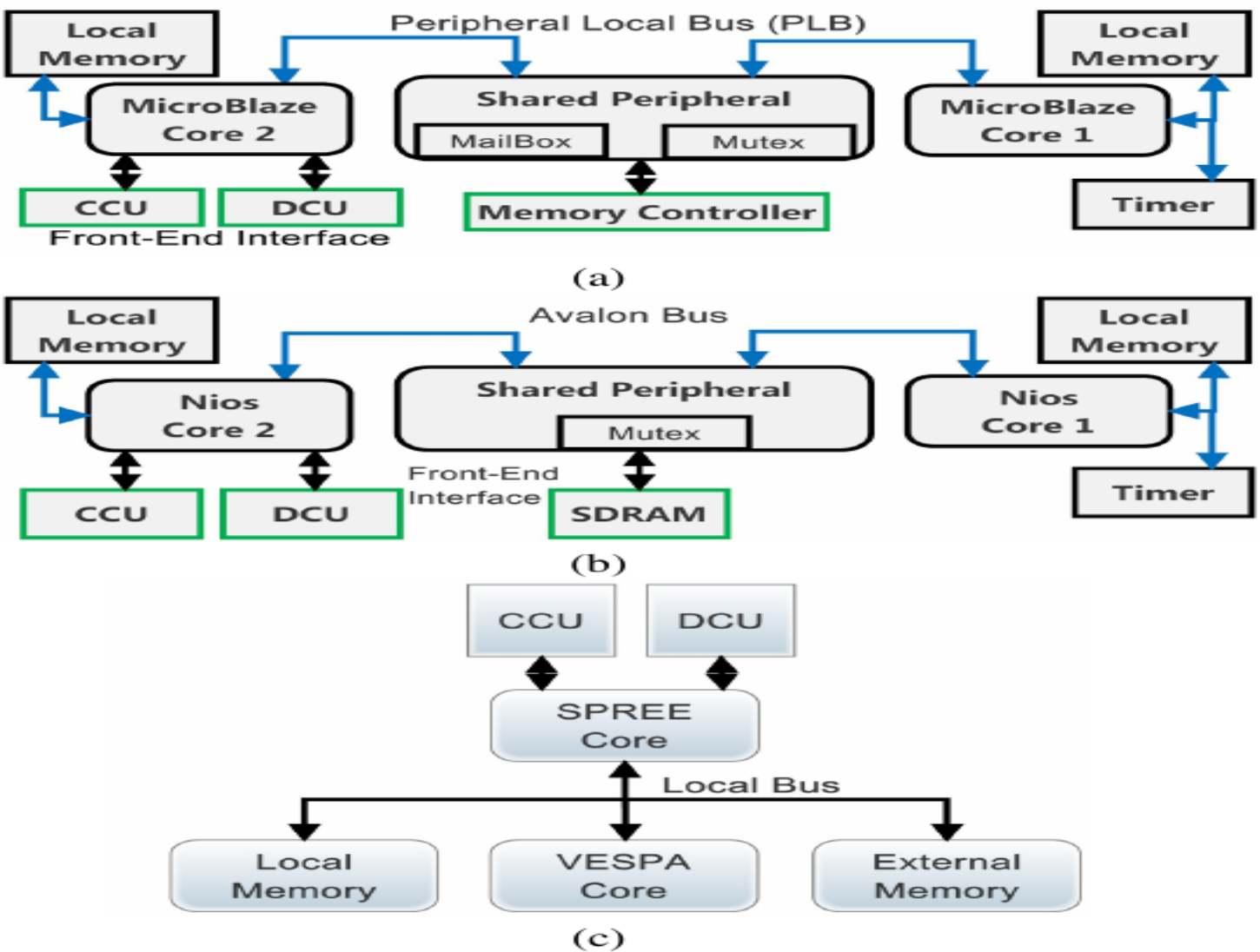
(a)

Module (Name)	Base Address	End Address	Range Hex
Image Signal Processor	E200 0400	E200 0418	1C
Color Processing	E200 0500	E200 0510	14
Main Resize	E200 0600	E200 064C	50
Memory Interface	E200 0700	0200 0810	114
JPEG Encoder	E200 0900	E200 098C	90
Total	-	-	224

(b)

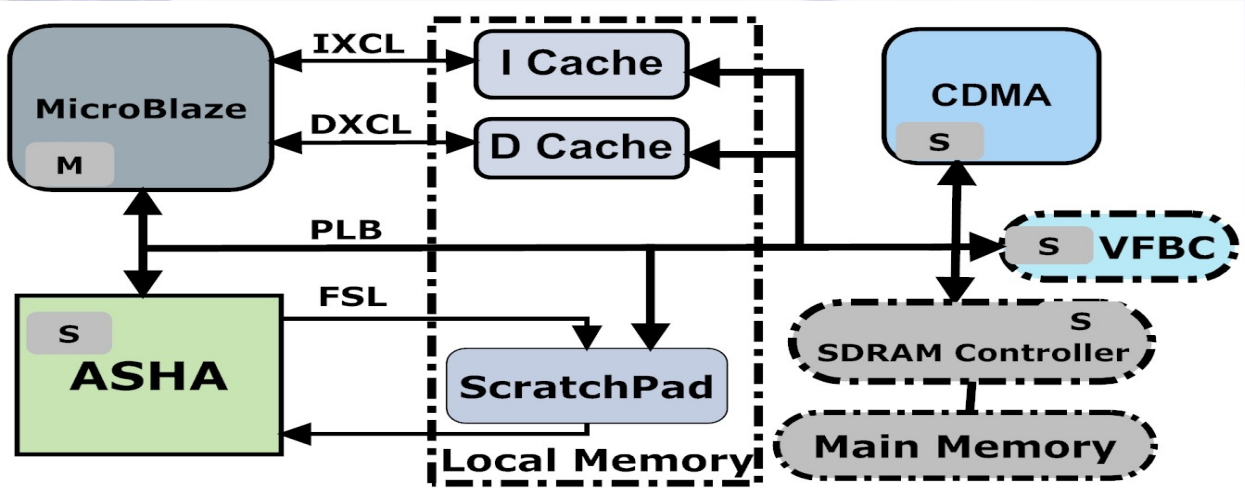

5 Masters
Request
CCU Read (M)
CCU Write (M)
DCU Read
(M) Write
CORE Write (M)
Request
Grant
Descriptor
Write Data Bus
Read Data Bus
Grant Line
Select Line
Address Bus
Master to Slave MUX
Slave to Master MUX
interrupt controller
Scheduler
Status Reg
Address Manager
Decoder
Hready
Write Data Bus
Program Memory
CIF hready_i
CCU (S)
DCU (S)
CORE (S)
Local Memory (S)
4 Slaves

SoC Design Using FPGAs



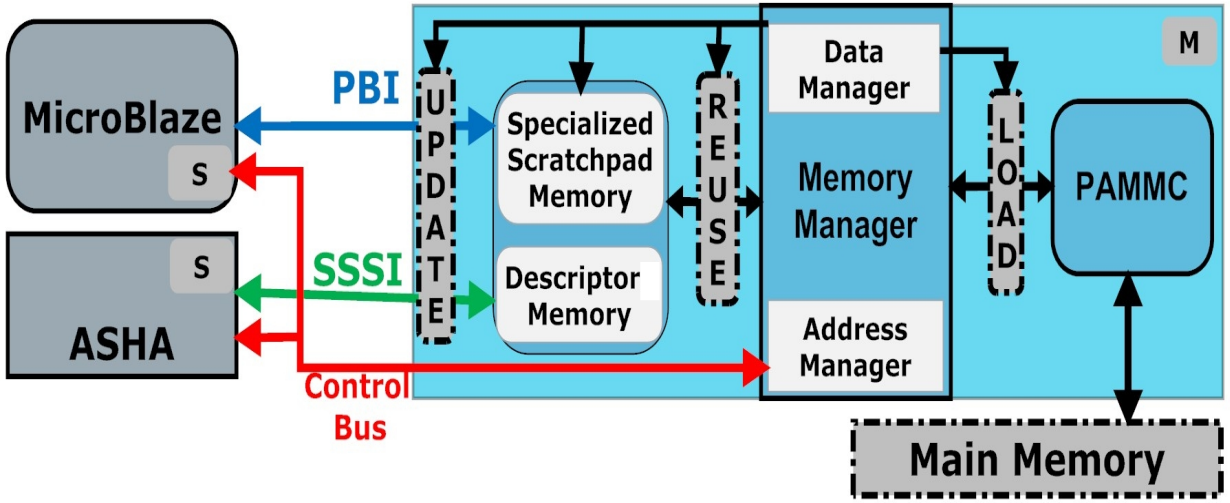
HW/SW Co-Design System Architectures

Baseline system



APMC based system

- Xilinx ML505 evaluation platform with a 65 nm Virtex-5 lx FPGA
- System clock 100 MHz



Topics

- Symmetric Processor Architecture

  GPP, Many modern multi-core processors for desktops, servers, and HPC systems.

- Application Specific Processor

  Accelerators, DSP/Vector/GPU

- Asymmetric Processor Architecture:

  Heterogeneous Cores, Smartphones and Tablet

- **Frameworks for customizable RISC-V System**
- RISCV System Toolchain

FOSS: SoC Design and Modeling

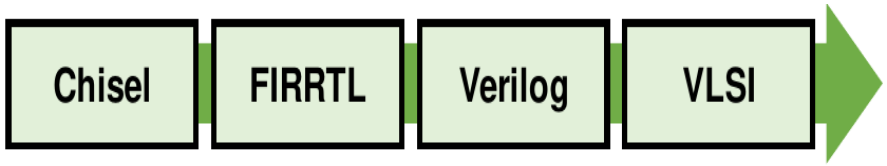
Tool	LiteX	Chipyard	Rocket Chip	FuseSoC	OpenPiton
Overview	A highly modular and configurable SoC builder with support for various FPGA platforms, including RISC-V.	An open-source framework for designing RISC-V-based SoCs with a focus on configurability and customization.	A generator for RISC-V-based SoCs with a focus on processor cores and a well-defined memory subsystem.	An open-source package manager and build tool for managing and creating digital design projects.	An open-source, general-purpose, multithreaded processor platform for scalable multicore SoC designs.
Modularity	Highly modular and configurable. Allows selection of CPU cores, peripherals, and interconnects.	Highly modular, configurable, and customizable. Supports various cores, accelerators, and peripherals.	Modular design with a focus on processor cores and cache hierarchies.	Provides a structured project directory layout and supports modular design.	Supports scalable designs with a variable number of cores.
Supported CPUs	RISC-V, LM32, OpenRISC, ZPU, and others.	Primarily RISC-V cores, including BOOM and Rocket.	RISC-V cores, including Rocket and BOOM.	Supports various IP cores and processor architectures, including RISC-V.	Multithreaded processor platform.
Memory Hierarchy	Customizable memory hierarchy and cache configurations.	Supports various memory hierarchies, including cache configurations.	Well-defined memory hierarchy.	Configurable memory hierarchies.	Comprehensive memory hierarchy.
IP Integration	Supports integration of custom IP cores and third-party IP.	Supports integration of custom IP cores and third-party IP.	Focus on processor cores; limited support for custom IP integration.	Simplifies IP core management and offers pre-packaged IP cores.	Extensive IP core library.
FPGA Platforms	Supports a wide range of FPGA platforms, including Xilinx, Intel, Lattice, and others.	Supports Xilinx FPGAs with Zynq and UltraScale+ MPSoC support.	Primarily targeted at FPGAs, including Xilinx and Intel.	Supports various FPGA platforms, including Xilinx and Intel.	Platform-agnostic.
Ease of Use	May have a steeper learning curve due to extensive configurability.	Offers a range of pre-configured templates for ease of use.	Requires familiarity with the Rocket Chip generator and Chisel hardware description language.	Simplifies project management, module integration, and building.	Simplifies project management and architectural exploration.
Community	Active community and growing user base.	Active development and community contributions.	Developed at UC Berkeley with an active research community.	Active community and contributions.	Active community and development.
Documentation	Documentation available, but may vary depending on the FPGA platform.	Extensive documentation and tutorials available.	Documentation available, primarily for research and academic purposes.	Comprehensive documentation and tutorials.	Comprehensive documentation and resources.
Commercial Support	Limited	Limited	Limited	Limited	Limited
Support for VLSI Tools	Limited VLSI tool integration.	Limited VLSI tool integration.	Limited VLSI tool integration.	Supports integration with VLSI tools like OpenLane, OpenROAD, and Qflow for ASIC development.	Limited VLSI tool integration.

Chisel

Chisel – Hardware Construction Language built on Scala

What Chisel IS NOT:

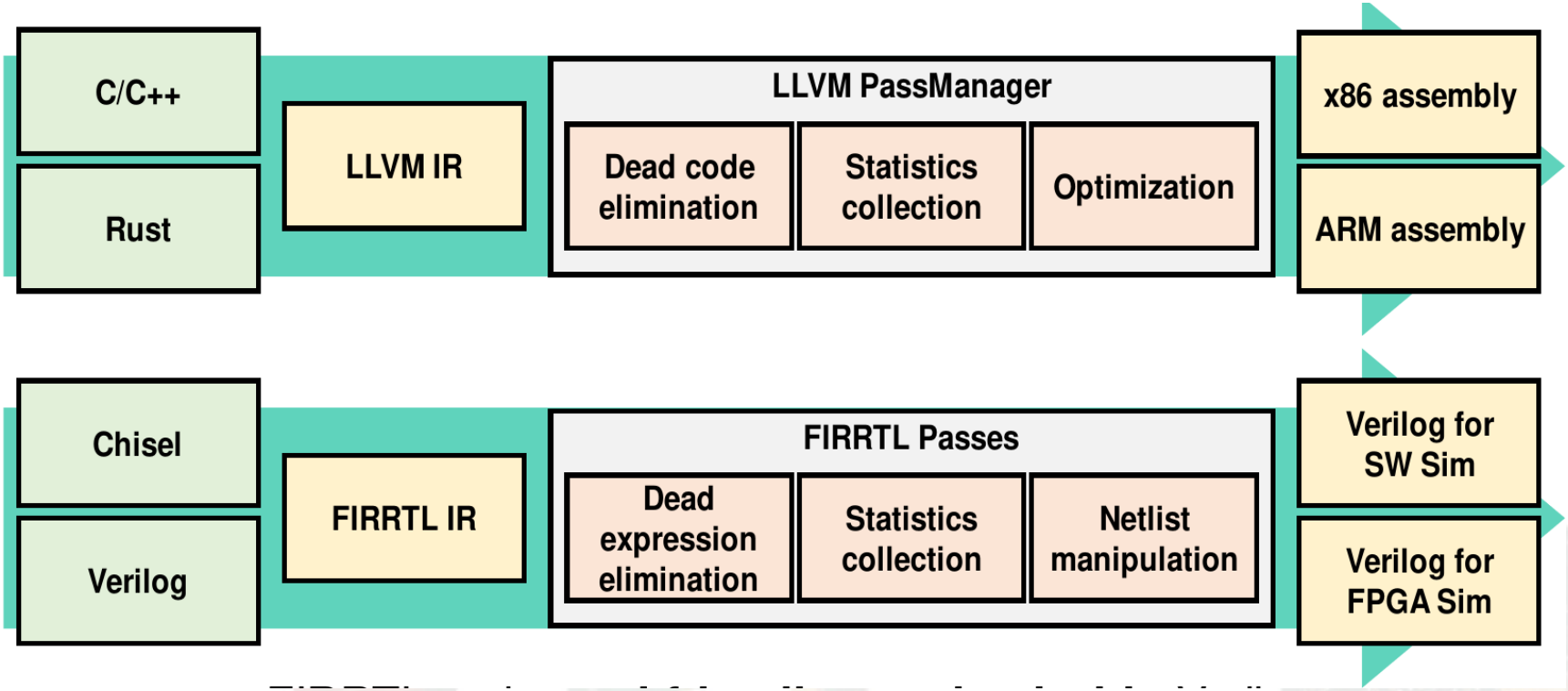
- NOT Scala-to-gates
- NOT HLS
- NOT tool-oriented language



What Chisel IS:

- Productive language for generating hardware
- Leverage OOP/Functional programming paradigms
- Enables design of parameterized generators
- Designer-friendly: low barrier-to-entry, high reward
- Backwards-compatible: integrates with Verilog black-boxes

Flexible Intermediate Representation for RTL



RocketChip SoC Tool

Open-source hardware construction language

Highly parameterized generator

Hierarchical + object oriented + functional construction

Generates Verilog and C model

- Not HLS (high-level synthesis)

Based on Scala

Functional programming

Strong static type system

Compiled to Java bytecode

Scripts Based:

Scripts and configuration files that are used for building, configuring, and managing the RocketChip-based hardware designs and related software tools.

Scala Script of Simple SoC

Configs Scala Scripts plays a important role in the RocketChip design flow.

It allows engineers to define and customize the parameters and components of a RocketChip-based SoC that helps them to create hardware designs tailored to their specific needs.

```
import freechips.rocketchip.config.{Config, Parameters}
import freechips.rocketchip.subsystem._

class MyConfig extends Config(
  new WithNBigCores(4) ++        // Configure 4 Rocket CPU cores
  new WithL1ICacheSize(32) ++    // Set L1 instruction cache size to
  32 KB
  new WithL1DCacheSize(32) ++    // Set L1 data cache size to 32 KB
  new WithMemorySize(256 * 1024) // Configure main memory size
  (256 MB)
)

class MyConfigProject extends Config(
  (pname, site, here) => pname match {
    case PeripheryKey => List(
      // Add peripheral devices here
      UARTParams(address = 0x10000)
    )
    case _ => throw new CDEMatchError
  }
)

// Define your custom configurations
val myConfig = new MyConfig
val myConfigProject = new MyConfigProject
```

rocket-chip/src/main/scala

Parametrization in Rocket-chip

- Core
  - 32-bit or 64-bit
  - With or without FPU
  - With or without virtual memory
  - With or without BTB
  - Multiplier-divider cycles
- L1 cache (I$ & D$)
  - Num of sets
  - Block size (set size)
  - Ways
  - Num of MSHR
  - Num of TLB entries
- Cross clock domain
  - Sync or async

Structure of a Rocket Chip SoC

Tiles: unit of replication for a core

- CPU
- L1 Caches
- Page-table walker

L2 banks:

- Receive memory requests

FrontBus:

- Connects to DMA devices

ControlBus:

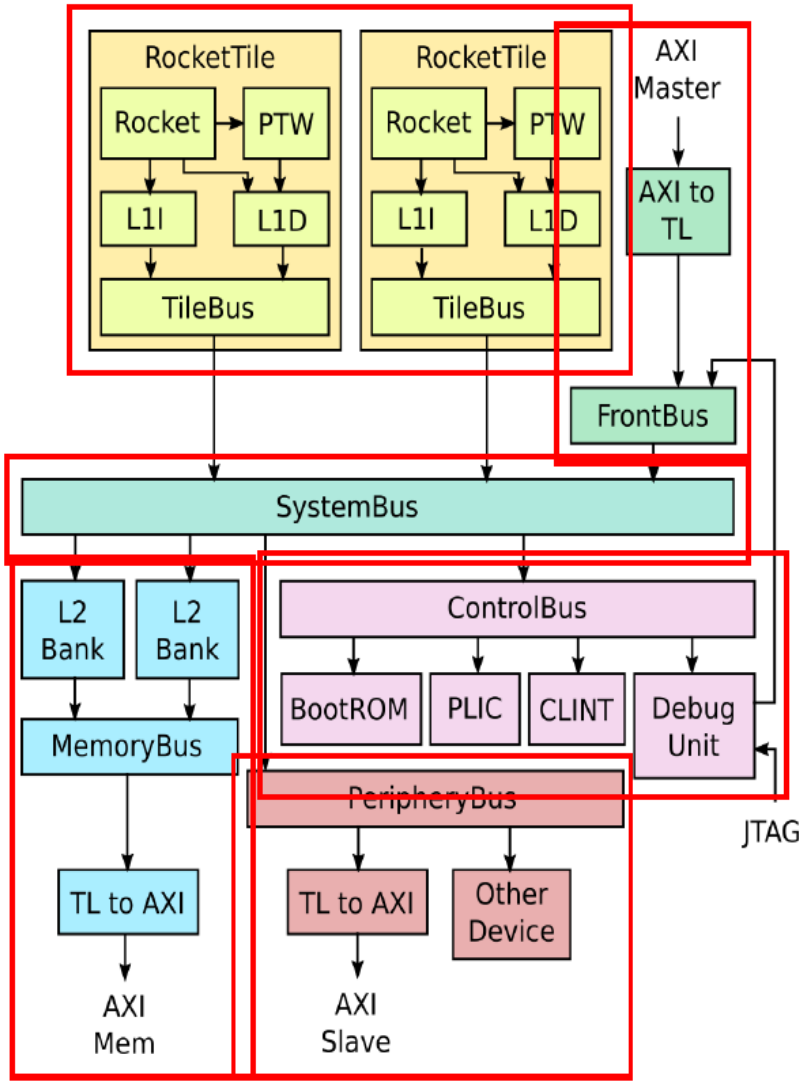
- Connects to core-complex devices

PeripheryBus:

- Connects to other devices

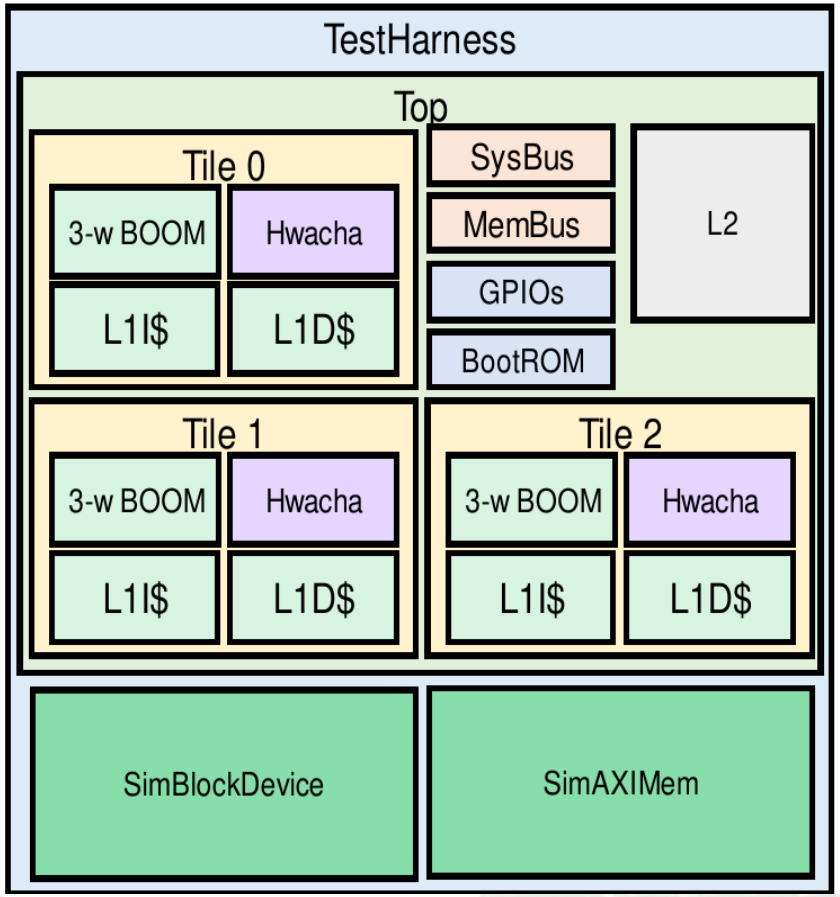
SystemBus:

- Ties everything together



Rocketchip SoC Configuration Script

```
class MyCustomConfig extends Config(
  new WithExtMemSize((1<<30) * 2L)          ++
  new WithBlockDevice                       ++
  new WithGPIO                              ++
  new WithBootROM                           ++
  new hwacha.DefaultHwachaConfig            ++
  new WithInclusiveCache(capacityKB=1024)   ++
  new boom.common.WithLargeBooms            ++
  new boom.system.WithNBoomCores(3)         ++
  new WithNormalBoomRocketTop               ++
  new rocketchip.system.BaseConfig)
```



Memory Configuration:

WithExtMemSize: This configuration sets the external memory size to 2 GB (2^30 * 2L bytes).

Peripheral Components:

WithBlockDevice: Includes support for a block device, which can be used for storage.

WithGPIO: Adds support for GPIO (General-Purpose Input/Output) pins.

WithJtagDTM: Configures the JTAG Debug Transport Module (DTM) for debugging.

WithBootROM: Includes a boot ROM in the design.

RISC-V Core Configuration:

WithRationalBoomTiles and WithRationalRocketTiles: Configures both BOOM (Berkeley Out-of-Order Machine) and **Rocket cores.**

WithMultiRoCCConvAccel, WithMultiRoCCSha3, WithMultiRoCCHwacha: Configures multiple custom Rocket Core Control (RoCC) accelerators.

WithInclusiveCache: Configures an inclusive cache with a capacity of 1024 KB (1 MB).

BOOM Core Configuration:

WithLargeBooms: Configures large BOOM cores.

Number of Cores:

WithNBoomCores(2): Specifies the number of BOOM cores as 2.

Rocket Core Configuration:

WithRV32: Configures the Rocket cores as RV32 (32-bit) cores.

WithNBigCores(1): Specifies the number of big (Rocket) cores as 1.

Top-Level Configuration:

WithNormalBoomRocketTop: Specifies the top-level configuration with a combination of BOOM and Rocket cores.

Base Configuration:

BaseConfig: This is the base configuration for the Chipyard project.

Developming SoC

Export RISCV=/opt/riscv/

~/src/main/scala/system/Configs.scala

~/src/main/scala/subsystem/Configs.scala

cd emulator

make

cd vsim

make verilog

make -j4 run # run all tests

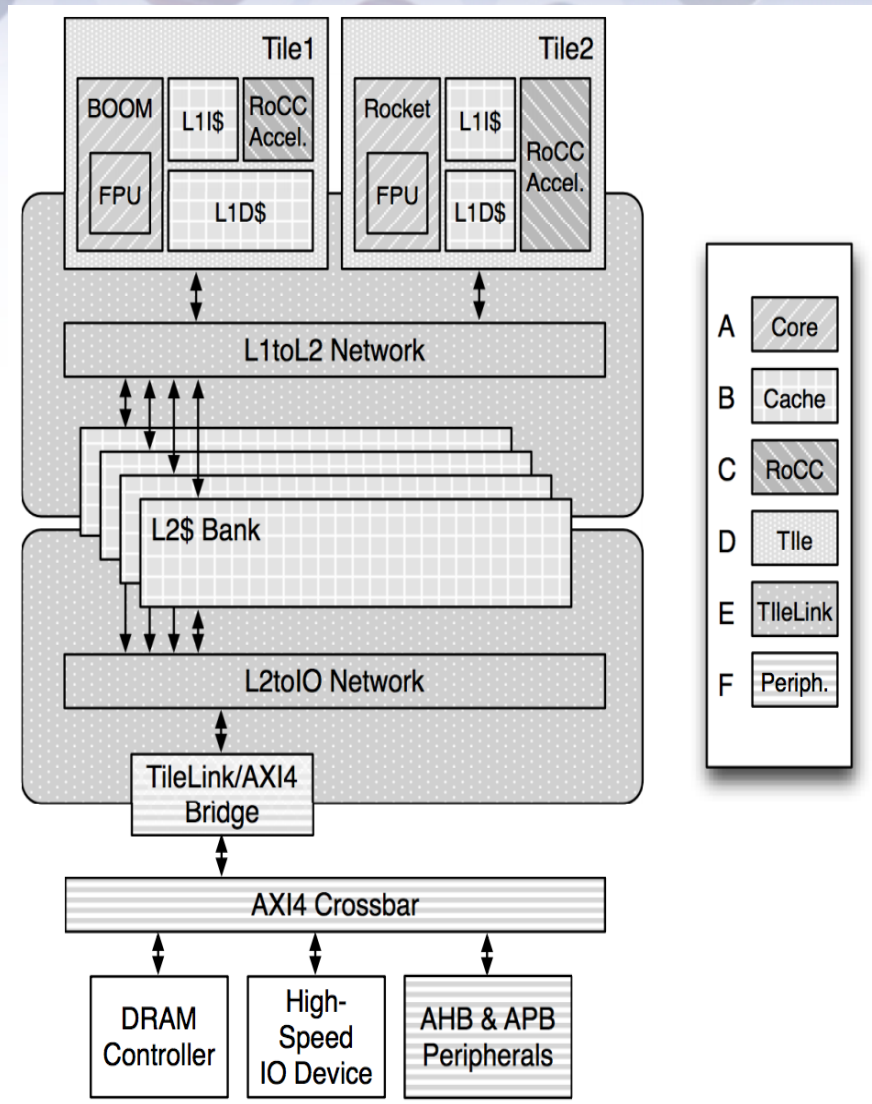
make output/dhrystone.riscv.out

find the passed ones

grep PASSED output/*.out

find the failed ones

grep FAILED output/*.out



Emulator: make

Building RocketChip and Software: Compiling the RocketChip hardware design described in Chisel or another hardware description language (HDL).

Building the software components that run on the RocketChip, which may include a boot loader, operating system, and application programs.

Generating Emulator Binaries:

Simulating the RocketChip Design:

Generating Output Files:

Running Software Tests:

Chip Yard

A highly parameterizable and modular SoC generator

- Replace default Rocket core w/ your own core
- Add your own coprocessor
- Add your own SoC IP to uncore

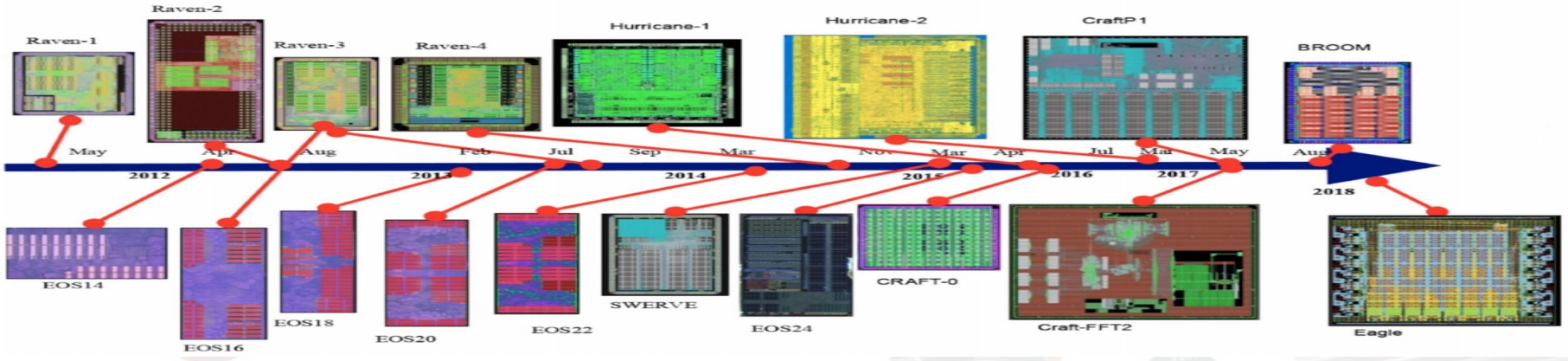
A library of reusable SoC components

- Memory protocol converters
- Arbiters and Crossbar generators
- Clock-crossings and asynchronous queues

The largest open-source Chisel codebase

Developed at Berkeley, now maintained by many

- SiFive, ChipsAlliance, Berkeley



Supported Cores

Rocket: In-Order Core

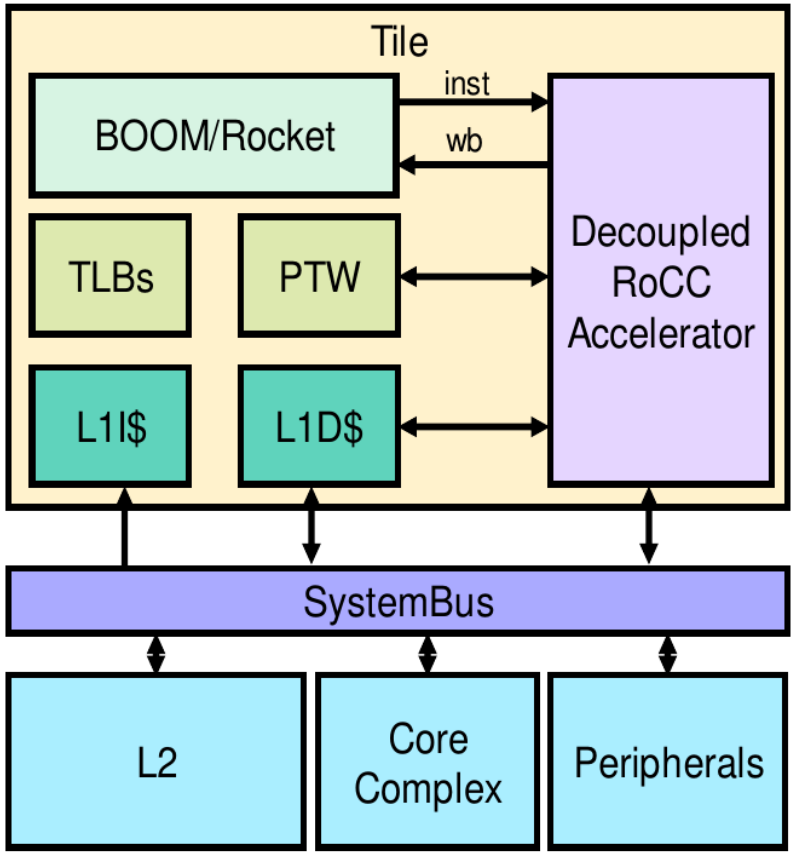
- In-order, single-issue RV64GC core
- Boots Linux

BOOM: Berkeley Out-of-Order

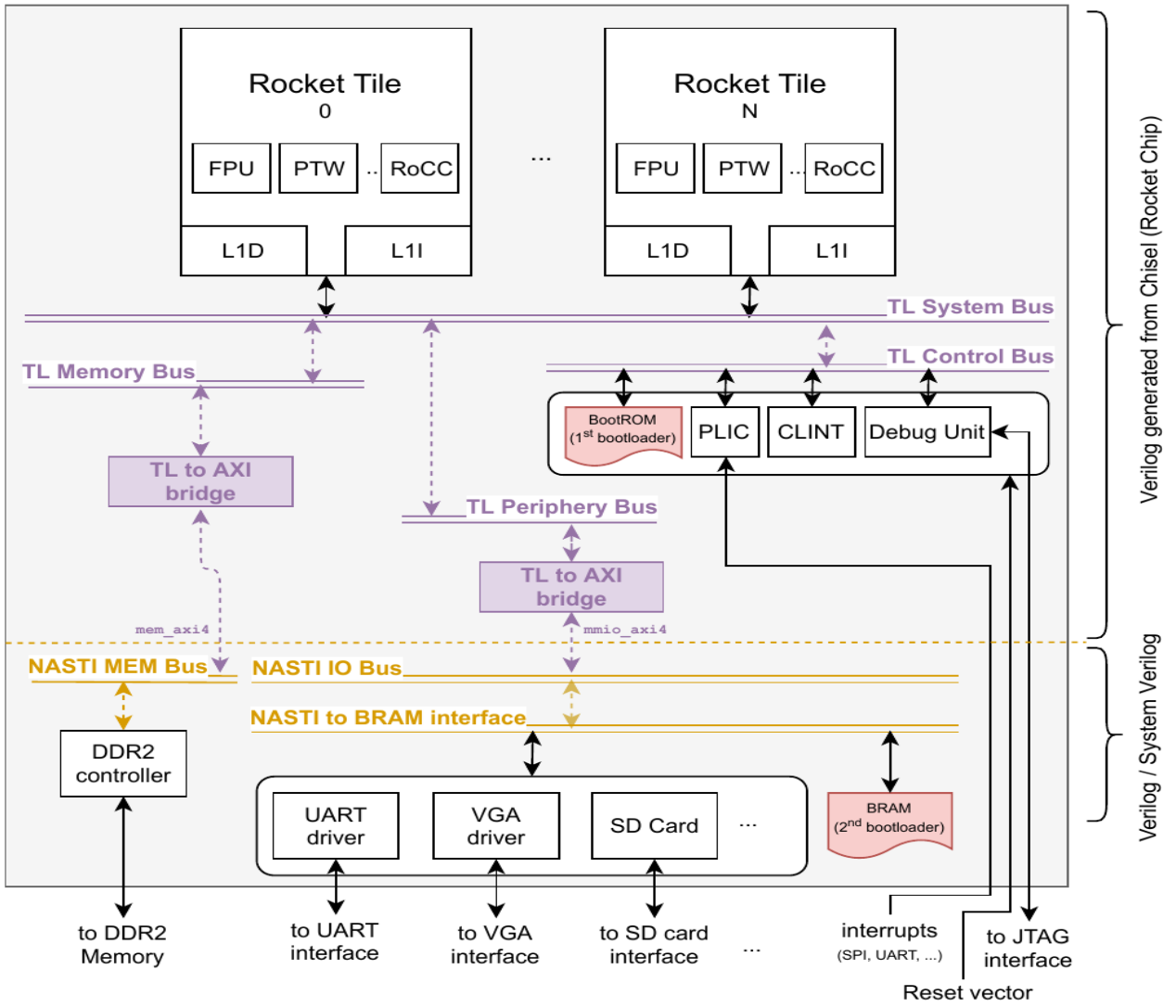
- Superscalar RISC-V OoO core
- Fully integrated in Rocket Chip ecosystem
- Open-source
- Described in Chisel
- Parameterizable generator

RoCC Accelerators

- Vector accelerators
- Memcpy accelerator
- Machine-learning accelerators



LowRISC


Rocket Tile
0
FPU
PTW
RoCC
L1D
L1I
...
Rocket Tile
N
FPU
PTW
RoCC
L1D
L1I
TL System Bus
TL Control Bus
TL Memory Bus
BootROM
(1st bootloader)
PLIC
CLINT
Debug Unit
TL to AXI
bridge
TL Periphery Bus
TL to AXI
bridge
mem_axi4
mmio_axi4
NASTI MEM Bus
NASTI IO Bus
NASTI to BRAM interface
DDR2
controller
UART
driver
VGA
driver
SD Card
...
BRAM
(2nd bootloader)
to DDR2
Memory
to UART
interface
to VGA
interface
to SD card
interface
...
interrupts
(SPI, UART, ...)
to JTAG
interface
Reset vector
Verilog generated from Chisel (Rocket Chip)
Verilog / System Verilog

Topics

- Symmetric Processor Architecture

  GPP, Many modern multi-core processors for desktops, servers, and HPC systems.

- Application Specific Processor

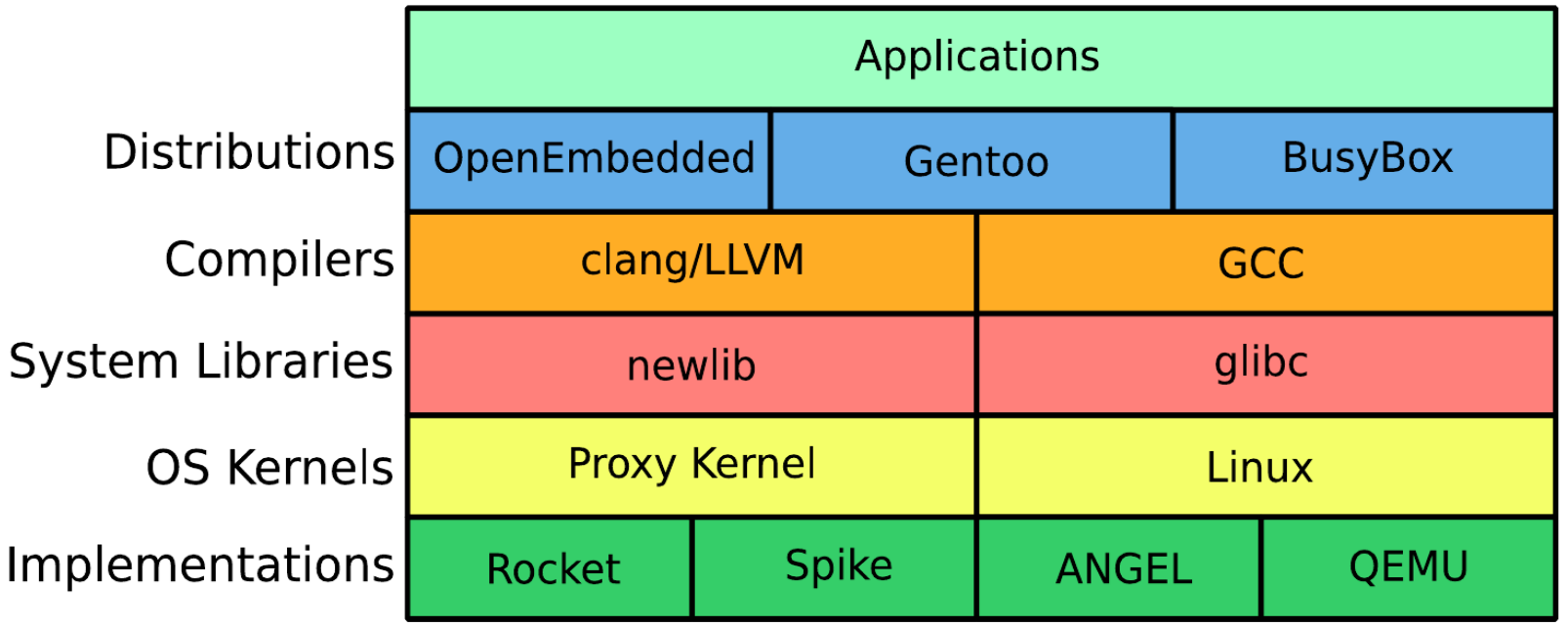
  Accelerators, DSP/Vector/GPU

- Asymmetric Processor Architecture:

  Heterogeneous Cores, Smartphones and Tablet

- Frameworks for Customizable RISC-V System
- **RISCV System Toolchain**

RISC--V Software Stack



RISCV Compiler

This is the GNU Compiler Collection (GCC) for RISC-V. It includes a suite of compilers, assemblers, and other tools for compiling and developing software for RISC-V architectures.

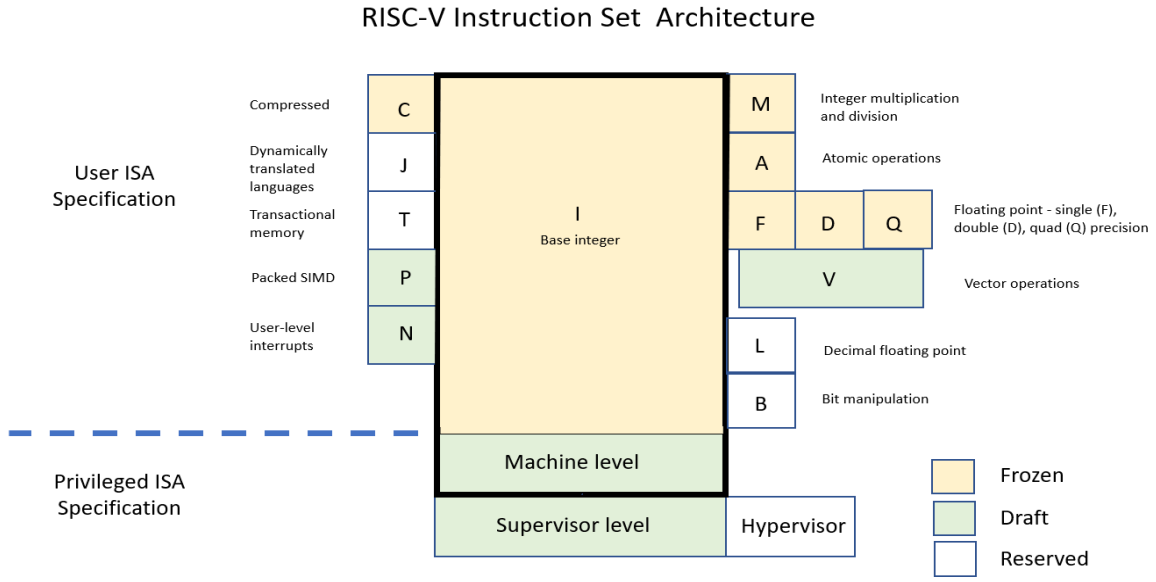
riscv64--unknown--elf--gcc

riscv64--unknown--linux--gnu--gcc

LLVM is another compiler infrastructure that supports RISC-V. It includes a compiler (clang) and other tools for software development.

OpCodes (ISA)

Add the opcode and opcode mask to riscv/opcodes.h. Alternatively, add it to the riscv-opcodes package, and it will do so for you



Front-End Server, Simulator and Proxy Kernel

The Front-End Server (FESVR) is used for interfacing between RISC-V simulators and the host computer. It helps in loading and running RISC-V programs on simulators.

Also known as "spike," (**riscv-isa-sim**) this is a RISC-V ISA simulator. It provides a functional simulation of RISC-V processors and is often used for architecture exploration and software development.

The Proxy Kernel (PK) is a lightweight, bootable RISC-V kernel that can be used with RISC-V simulators to run user-level RISC-V programs.

Emulator

QEMU is an open-source emulator that supports various instruction set architectures, including RISC-V. It allows you to run RISC-V binaries on your host system.

RISC-V Linux refers to the effort to port the Linux operating system to the RISC-V architecture. It includes the kernel and related components necessary to run Linux on RISC-V hardware.

Test

This RISCV-Tests provides a comprehensive suite of RISC-V assembly tests to verify the correctness of RISC-V processors and simulators.

Application Development

SEE: Supervisor Execution Environment

Privileged operations, Control over HAL and Device Drivers, System Calls etc.

SBI: System Binary Interface

Part of OS kernel stack, Management Applications

ABI: Application Binary Interface

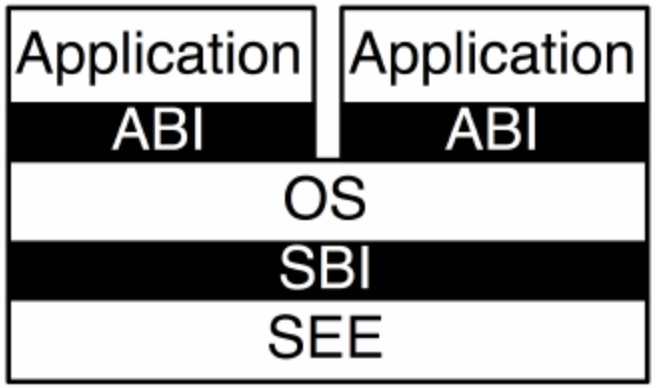
Executable Linkable File, calling conventions, registers usage, data types, handlers, system calls, dynamic linkages

Proxy Kernel with fesvr

ABI

Linux

ABI+SBI

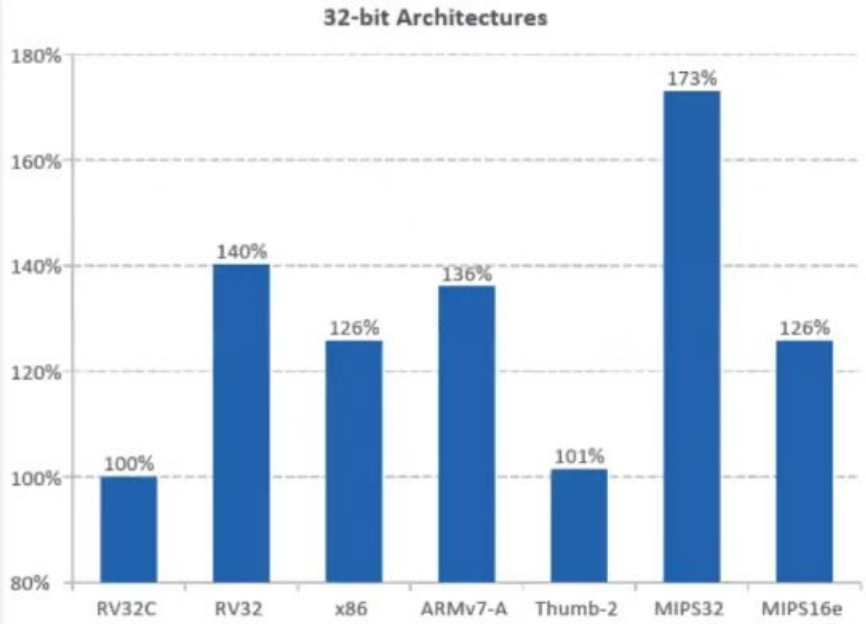


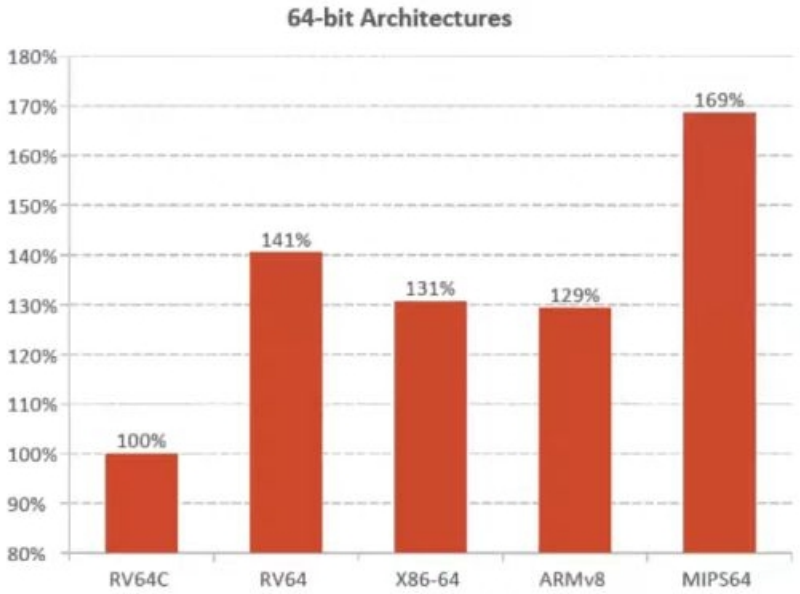
Compiling Programs

riscv64--unknown--elf--gcc (newlib) (Light weight Libraries)
Proxy Kernel + Spike

riscv64--unknown--linux--gnu--gcc (glibc) + POSIX support
Linux + QEMU

Code Size Comparison





- **RISC-V is smallest ISA for 32- and 64-bit processors in SpecInt2k6**
- **All results with same GCC compiler and options**

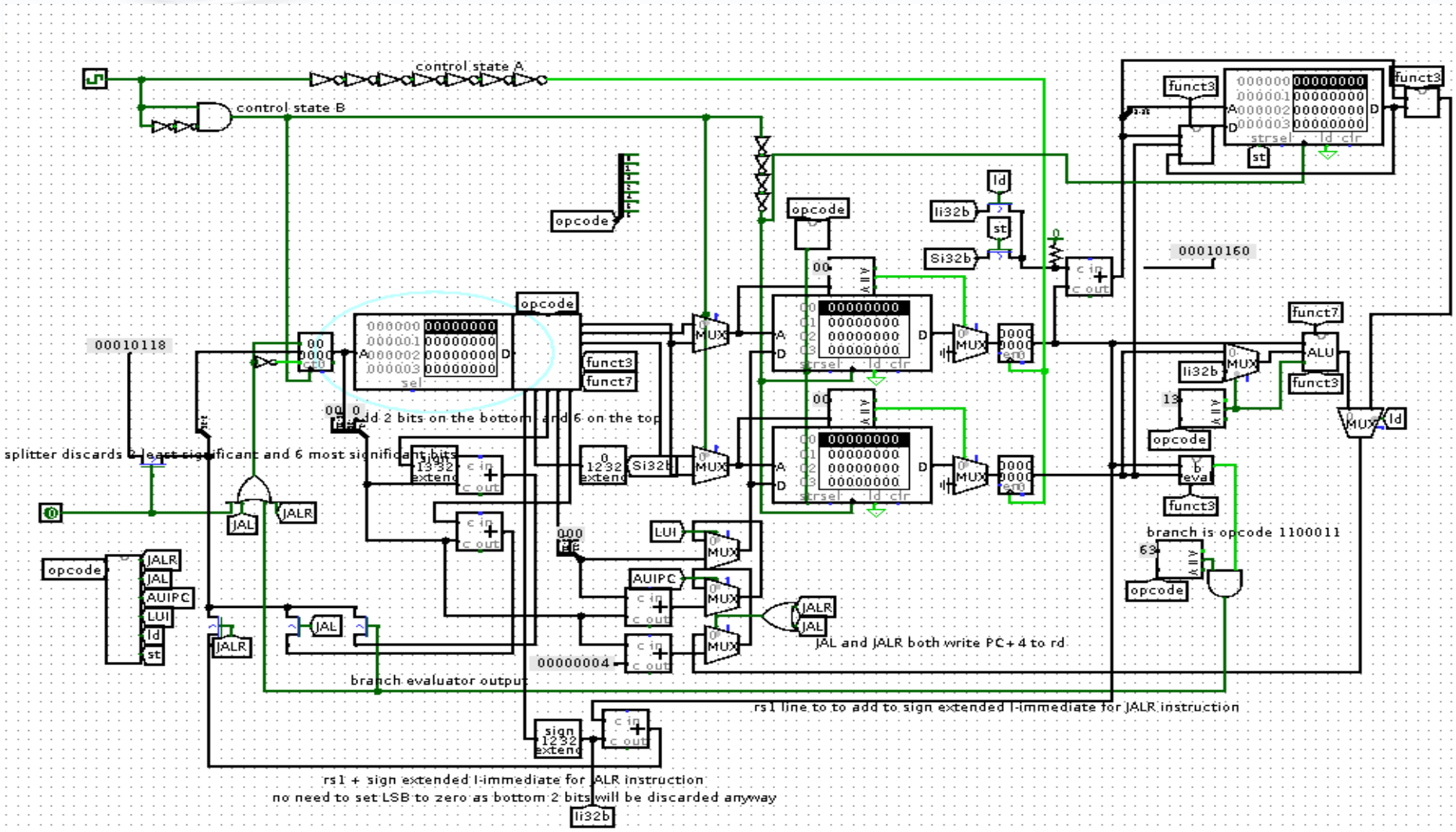
Performance Comparison RISV64 VS x86_64

Timer Start

```
#include <stdio.h>
int main(void) {
printf("Hello World\n");
return 0;
}
```

Timer Stop

Logisim Testing



Code Compiling

Compiling code

riscv32-unknown-elf-gcc -march=rv32i -O3 -o timer.o ./timer.c

Converting into rom file

riscv32-unknown-elf-objdump -s timer | java -jar
../../BinaryConverter/dist/BinaryConverter.jar > rom.image

reading elf file

riscv32-unknown-elf-readelf -a hello | grep " _start"

QEMU + Linux

```
./riscv-gnu-toolchain/qemu/build/qemu-system-riscv64 -
   machine virt -nographic \
     -m 16384 -smp 8 \
     -bios
   /usr/lib/riscv64-linux-gnu/opensbi/generic/fw_jump.elf \
     -kernel /usr/lib/u-boot/qemu-riscv64_smode/uboot.elf \
     -device virtio-net-device,netdev=enp0s31f6 \
     -netdev user,id=enp0s31f6,hostfwd=tcp::5555-:22 \
     -drive file=ubuntu.img,format=raw,if=virtio
```

Formally Certified RTL