RISC-V based HW-SW System for Edge AI

Feng Li (李枫)
hkli2013@126.com
Aug 26, 2022

Agenda

I. Background

- Edge AI
- Tech Stack

II. eFPGA-oriented HW-SW System for Edge AI

- Core-V
- AIRISC
- Miscs
- eFPGA Framework

III. Project CFU-Playground

- Overview
- F4PGA
- Developing with Renode

IV. Software stack for Edge AI

- VEDLIoT
- microTVM
- Ray

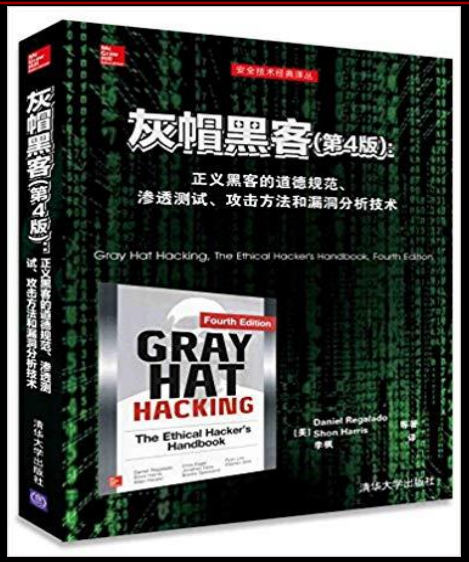
V. Rethinking

- PYNQ
- Python-centric one-stop toolchain

VI. Wrap-up

Who Am I

- The main translator of the book «Gray Hat Hacking The Ethical Hacker's Handbook, Fourth Edition» (ISBN：9787302428671) & «Linux Hardening in Hostile Networks, First Edition» (ISBN: 9787115544384)



- Pure software developmement for ~15 years(~11 years on Mobile Dev)
- Actively participate in various activities of the open source community
- https://github.com/XianBeiTuoBaFeng2015/MySlides/tree/master/Conf
  https://github.com/XianBeiTuoBaFeng2015/MySlides/tree/master/LTS
- Recently, focus on infrastructure of Cloud/Edge Computing, AI, Virtualization, Program Runtimes, Network, 5G, RISC-V, EDA…

I. Background

1) Edge AI

- https://www.techtarget.com/searchenterpriseai/definition/edge-AI
- https://www.advian.fi/en/what-is-edge-ai
- https://www.micro.ai/blog/edge-ai-what-is-it-and-how-does-it-work
- …

Edge Computing

- https://en.wikipedia.org/wiki/Edge_computing

Edge computing is a distributed computing paradigm that brings computation and data storage closer to the sources of data. This is expected to improve response times and save bandwidth.[1] It is an architecture rather than a specific technology.[2] It is a topology- and location-sensitive form of distributed computing.

The origins of edge computing lie in content distributed networks that were created in the late 1990s to serve web and video content from edge servers that were deployed close to users.[3] In the early 2000s, these networks evolved to host applications and application components at the edge servers,[4] resulting in the first commercial edge computing services[5] that hosted applications such as dealer locators, shopping carts, real-time data aggregators, and ad insertion engines.[4]

Internet of things (IoT) is an example of edge computing. A common misconception is that edge and IoT are synonymous.[6]

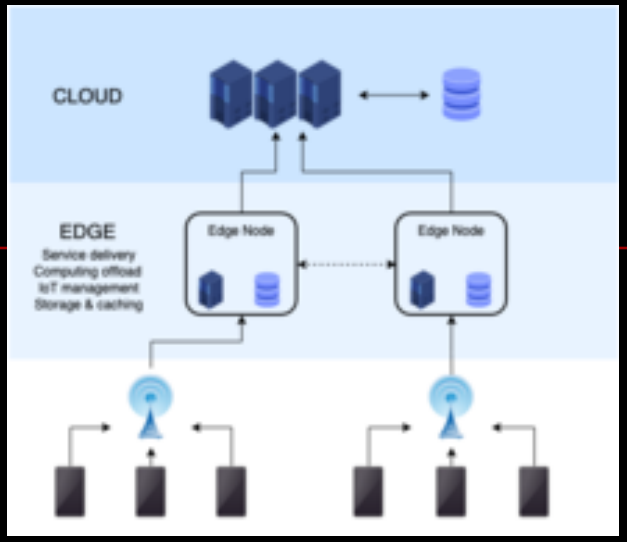
One definition of edge computing is any type of computer program that delivers low latency nearer to the requests. Karim Arabi, in an IEEE DAC 2014 Keynote [7] and subsequently in an invited talk at MIT's MTL Seminar in 2015,[8] defined edge computing broadly as all computing outside the cloud happening at the edge of the network, and more specifically in applications where real-time processing of data is required. In his definition, cloud computing operates on big data while edge computing operates on "instant data" that is real-time data generated by sensors or users.

The term is often used synonymously with fog computing.[9]

According to *The State of the Edge* report, edge computing concentrates on servers "in proximity to the last mile network."[*citation needed*] Alex Reznik, Chair of the ETSI MEC ISG standards committee loosely defines the term: "anything that's not a traditional data center could be the 'edge' to somebody."[10]

Edge nodes used for game streaming are known as *gamelets*,[11] which are usually one or two hops away from the client.[12] Per Anand and Edwin say "the edge node is mostly one or two hops away from the mobile client to meet the response time constraints for real-time games' in the cloud gaming context."[12]

Edge computing may employ virtualization technology to make it easier to deploy and run a wide range of applications on edge servers.[*citation needed*]



Applications [edit]

Edge application services reduce the volumes of data that must be moved, the consequent traffic, and the distance that data must travel. That provides lower latency and reduces transmission costs. Computation offloading for real-time applications, such as facial recognition algorithms, showed considerable improvements in response times, as demonstrated in early research.[25] Further research showed that using resource-rich machines called cloudlets or micro data centers near mobile users, which offer services typically found in the cloud, provided improvements in execution time when some of the tasks are offloaded to the edge node.[26] On the other hand, offloading every task may result in a slowdown due to transfer times between device and nodes, so depending on the workload, an optimal configuration can be defined.

Another use of the architecture is cloud gaming, where some aspects of a game could run in the cloud, while the rendered video is transferred to lightweight clients running on devices such as mobile phones, VR glasses, etc. This type of streaming is also known as *pixel streaming*.[11]

Other notable applications include connected cars, autonomous cars,[27] smart cities,[28] Industry 4.0 (smart industry), and home automation systems.[29]

- **https://en.wikipedia.org/wiki/Edge_device**
- **…**

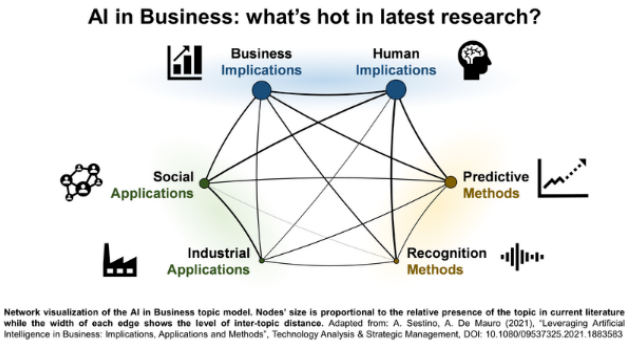
AIoT

- https://en.wikipedia.org/wiki/Artificial_intelligence_of_things

The Artificial Intelligence of Things (AIoT) is the combination of Artificial intelligence (AI) technologies with the Internet of things (IoT) infrastructure to achieve more efficient IoT operations, improve human-machine interactions and enhance data management and analytics.[1][2][3]

In 2018, KPMG published a foresight study on the future of AI including scenarios until 2040.[4] The analysts describe a scenario in detail where a community of things would see each device also contain its own AI that could link autonomously to other AIs to, together, perform tasks intelligently. Value creation would be controlled and executed in real-time using swarm intelligence. Many industries could be transformed with the application of swarm intelligence, including: automotive, cloud, medical, military, research, and technology.

In the AIoT an important facet is AI being done on some Thing. In its purest form this involves performing the AI on the device, i.e. at the edge or Edge Computing, with no need for external connections. There is no need for an Internet in AIoT, it is an evolution of the concept of the IoT and that is where the comparison ends.



Network visualization of the AI in Business topic model. Nodes' size is proportional to the relative presence of the topic in current literature while the width of each edge shows the level of inter-topic distance. Adapted from: A. Sestino, A. De Mauro (2021), "Leveraging Artificial Intelligence in Business: Implications, Applications and Methods", Technology Analysis & Strategic Management, DOI: 10.1080/09537325.2021.1883583

The combined power of AI and IoT, promises to unlock unrealized customer value in a broad swath of industry verticals such as edge analytics, autonomous vehicles, personalized fitness, remote healthcare, precision agriculture, smart retail, predictive maintenance, and industrial automation.[5]

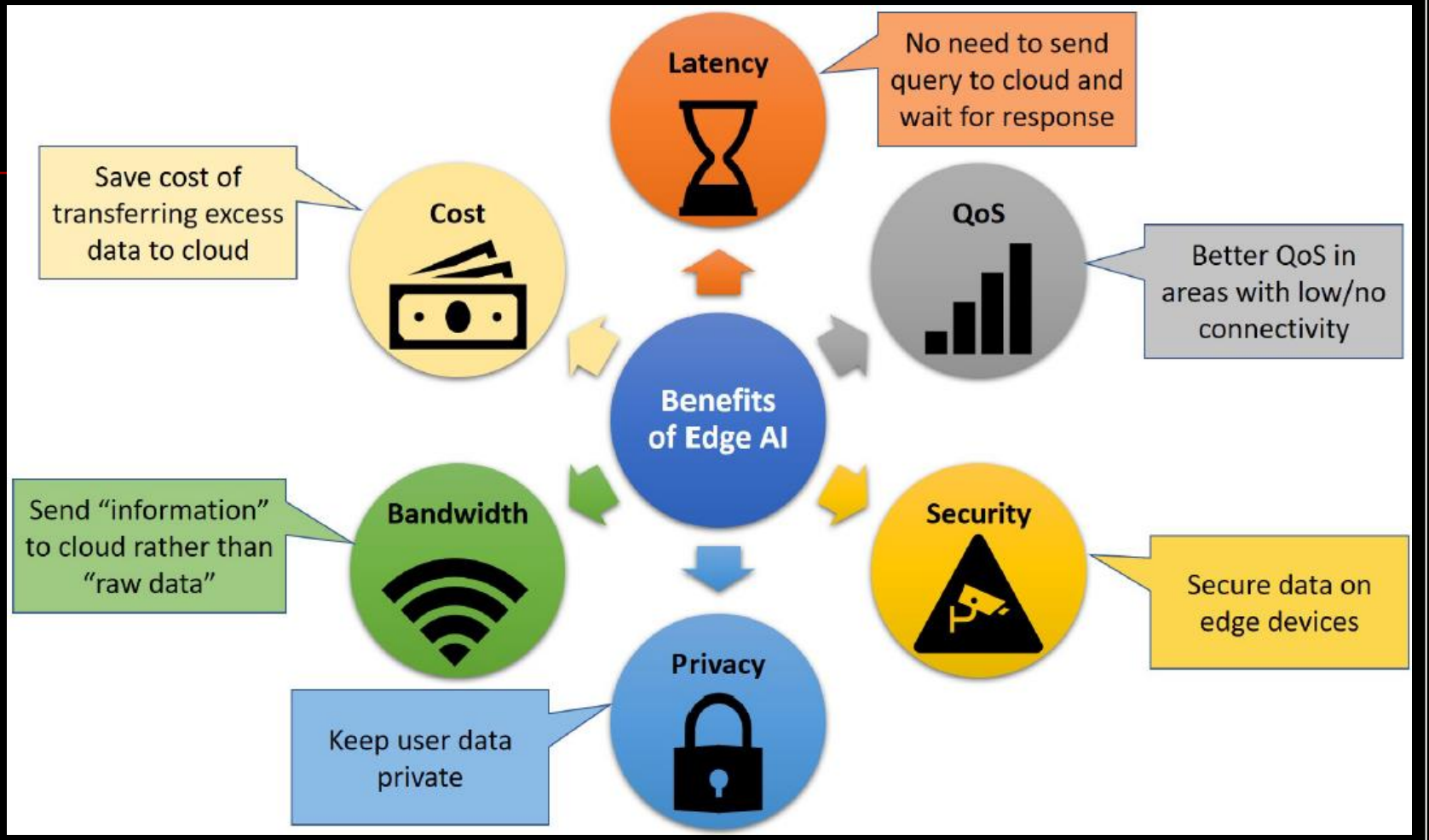
- https://aiotug.org/

Why AIoT? And why now?

Artificial intelligence (AI) and the Internet of Things (IoT) are key pillar stones of digital transformation, especially for manufacturers and operators of physical asset. The combination of AI and IoT (=AIoT) allows to build truly smart, connected products and solutions. First examples utilizing asset intelligence and swarm intelligence in industries such as mobility, healthcare and energy have been successfully deployed. We believe this is only the start - AIoT has the potential to re-shape many industries in the coming decade.

- …

Benefits of Edge AI



Source: "AI-RISC: Scalable RISC-V Processor for IoT Edge AI Applications", Vaibhav Verma, PhD thesis 2022

- …

1.1 HW perspective

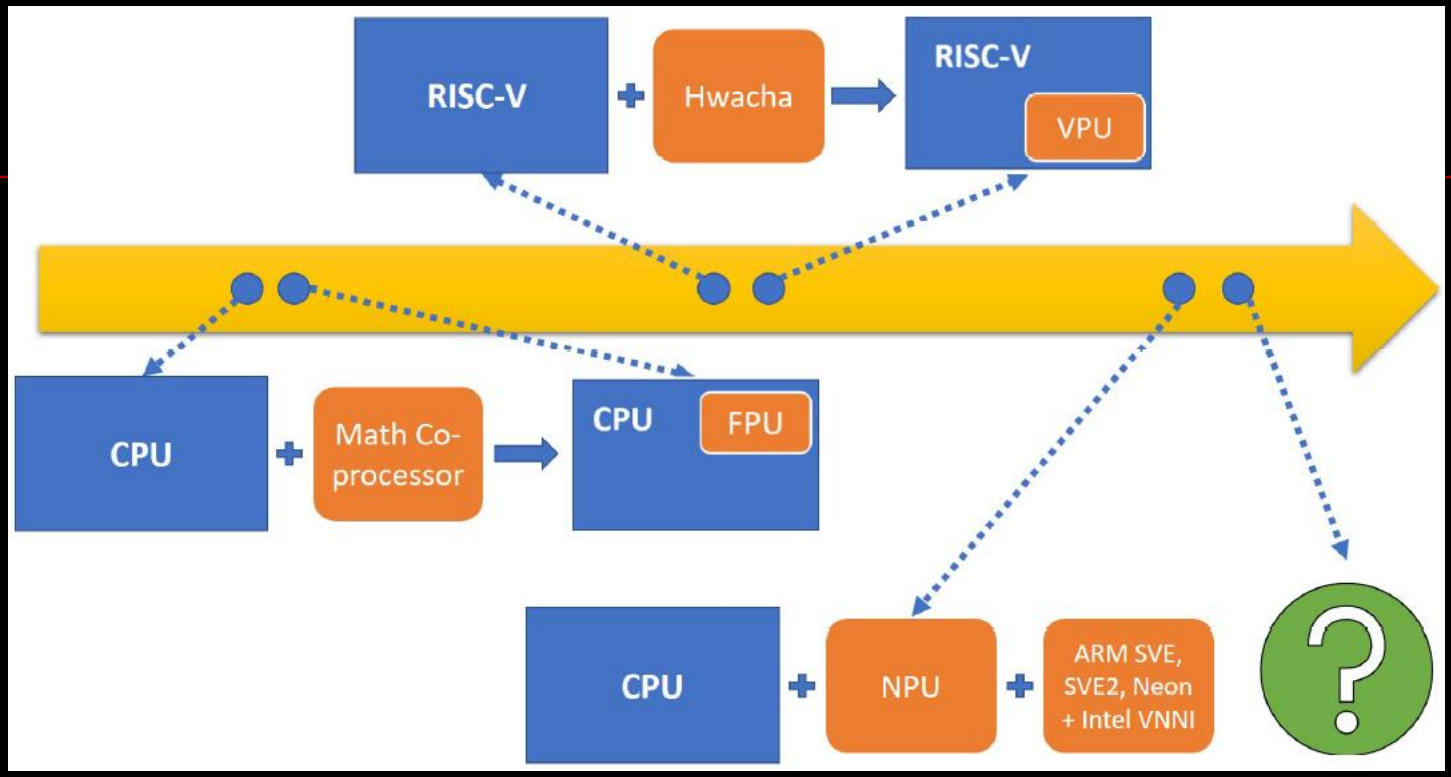


Figure 1.3: Historical timeline showing the evolution of computer architecture during the rise of mathematical floating-point workloads and vector workloads. We believe that as AI workloads become standard over time, the computer architecture will eventually evolve to include the AI functional units within the processor pipeline and AI instructions as part of the processor ISA. This hypothesis serves as a motivation for this dissertation to develop a processor architecture and design methodology to enable research towards this predicted next step in computer architecture evolution.

Source: "AI-RISC: Scalable RISC-V Processor for IoT Edge AI Applications", Vaibhav Verma, PhD thesis 2022

Current AI Accelerator Architectures

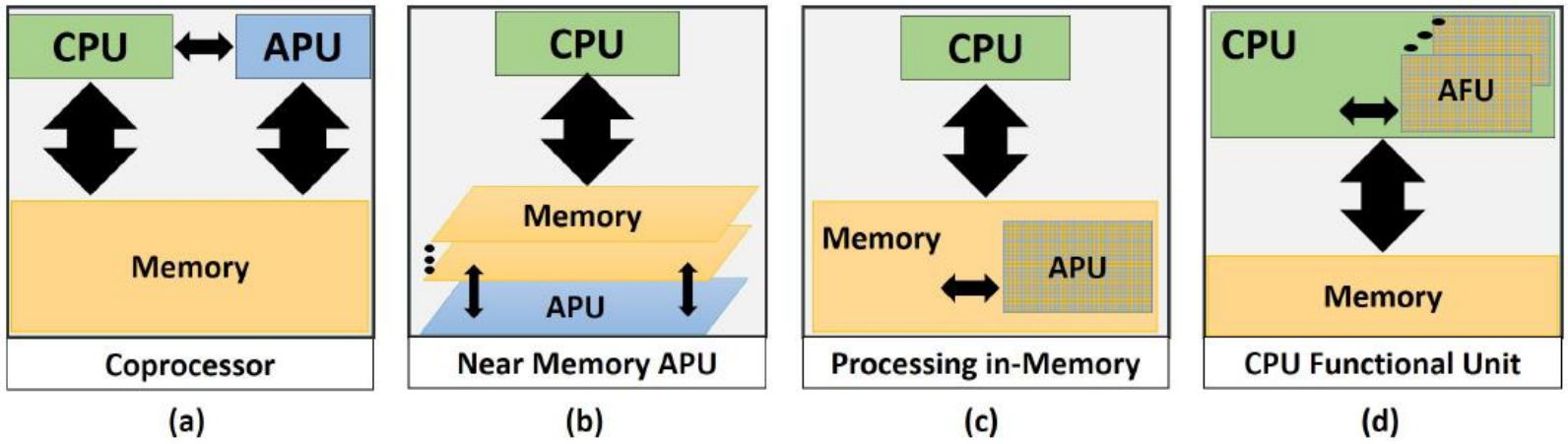


Figure 1.4: Different AI Processing Unit (APU) types and their integration with the CPU hardware stack. (a) Standard loose integration of APU as a co-processor communicating with the CPU over an external bus. (b) Near 3D memory processing solutions embed AI processing functionality into the base logic die of 3D stacked memory. (c) Processing-in-memory solutions convert a part of the memory hierarchy (SRAM, DRAM or NVM) into a dedicated AI Processing Unit (APU). (d) AI-RISC tightly integrates optimum sized accelerators as AI Functional Units (AFU) inside the CPU pipeline (adapted from [1]).

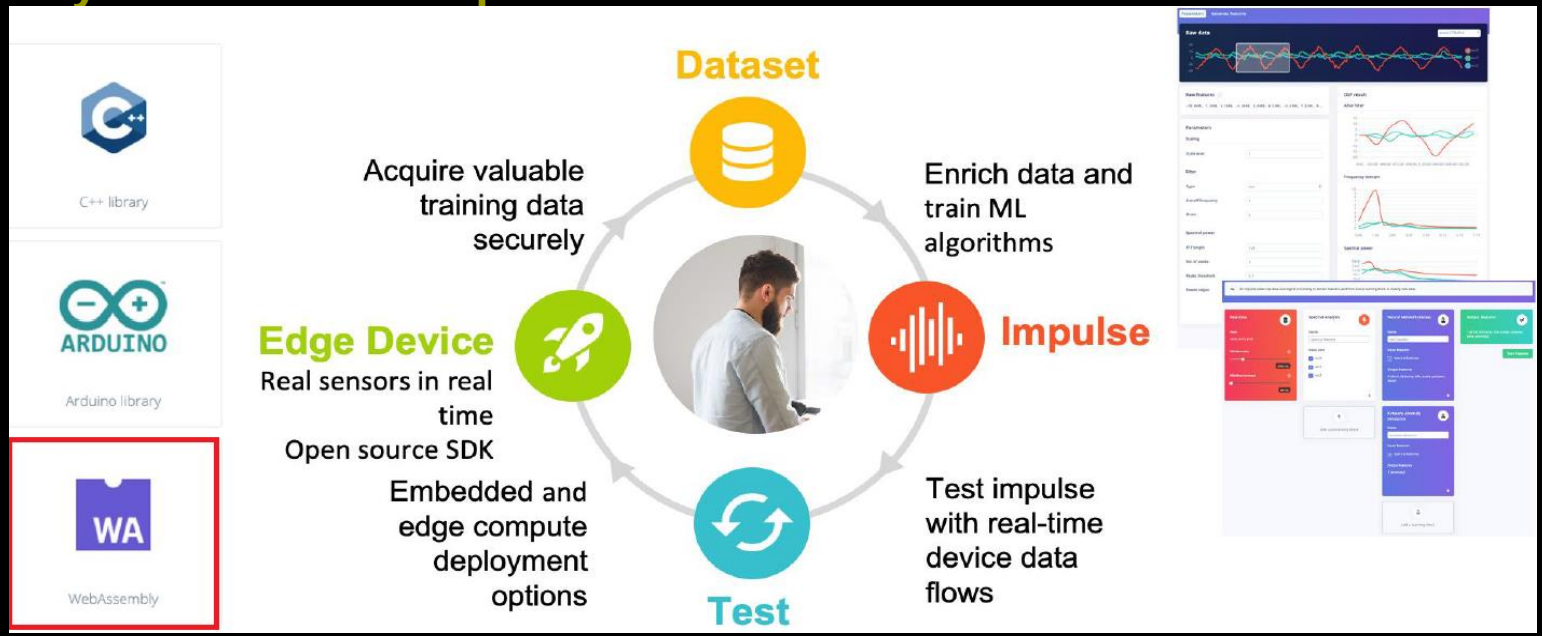
Source: "AI-RISC: Scalable RISC-V Processor for IoT Edge AI Applications", Vaibhav Verma, PhD thesis 2022

1.2 TinyML

- https://www.tinyml.org/

Tiny machine learning is broadly defined as a fast growing field of machine learning technologies and applications including hardware, algorithms and software capable of performing on-device sensor data analytics at extremely low power, typically in the mW range and below, and hence enabling a variety of always-on use-cases and targeting battery operated devices.

- https://semiengineering.com/why-tinyml-is-such-a-big-deal/
- https://www.arm.com/blogs/blueprint/tinyml
- TinyML for all developers



Source: https://cms.tinyml.org/wp-content/uploads/talks2022/tinyML_Talks_Pierre_Gembaczka_211201.pdf

1.2.1 TinyML on ARM

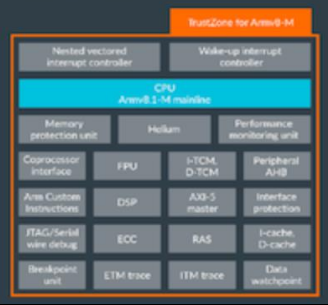
- https://www.arm.com/blogs/blueprint/tinyml
- Cortex-M55

https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m55

First Helium and Custom Instruction capable CPU core.

Cortex-M55: The Most AI-capable Cortex-M Processor

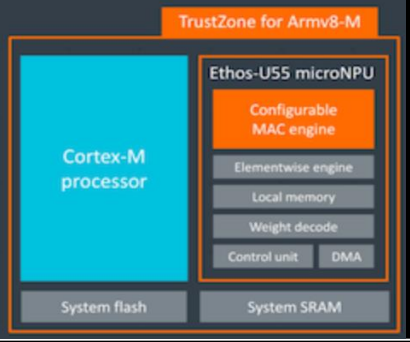
- ✓ First CPU based on Arm Helium technology
  - Energy-efficient and configurable with vector processing capabilities
  - Delivers up to 5x DSP performance and up to 15x ML performance*
  - Versatile capability for both classical ML and NN inference
- ✓ Advanced memory interfaces for fast access to ML data and weights
- ✓ Arm TrustZone security, accelerating the route to PSA Certified



https://www.arm.com/product-filter?families=ethos%20npus&showall=true

Ethos-U55: The First microNPU for Cortex-M

- ✓ Highest efficiency and small memory footprint
- ✓ 32, 64, 128, or 256 unit multiply-accumulate (MAC) engine
- ✓ Weight decoder and DMA for on-the-fly weight decompression
- ✓ Tooling available for offline optimization
- ✓ Works with a range of Cortex-M processors:
  - Cortex-M55
  - Cortex-M7
  - Cortex-M33
  - Cortex-M4



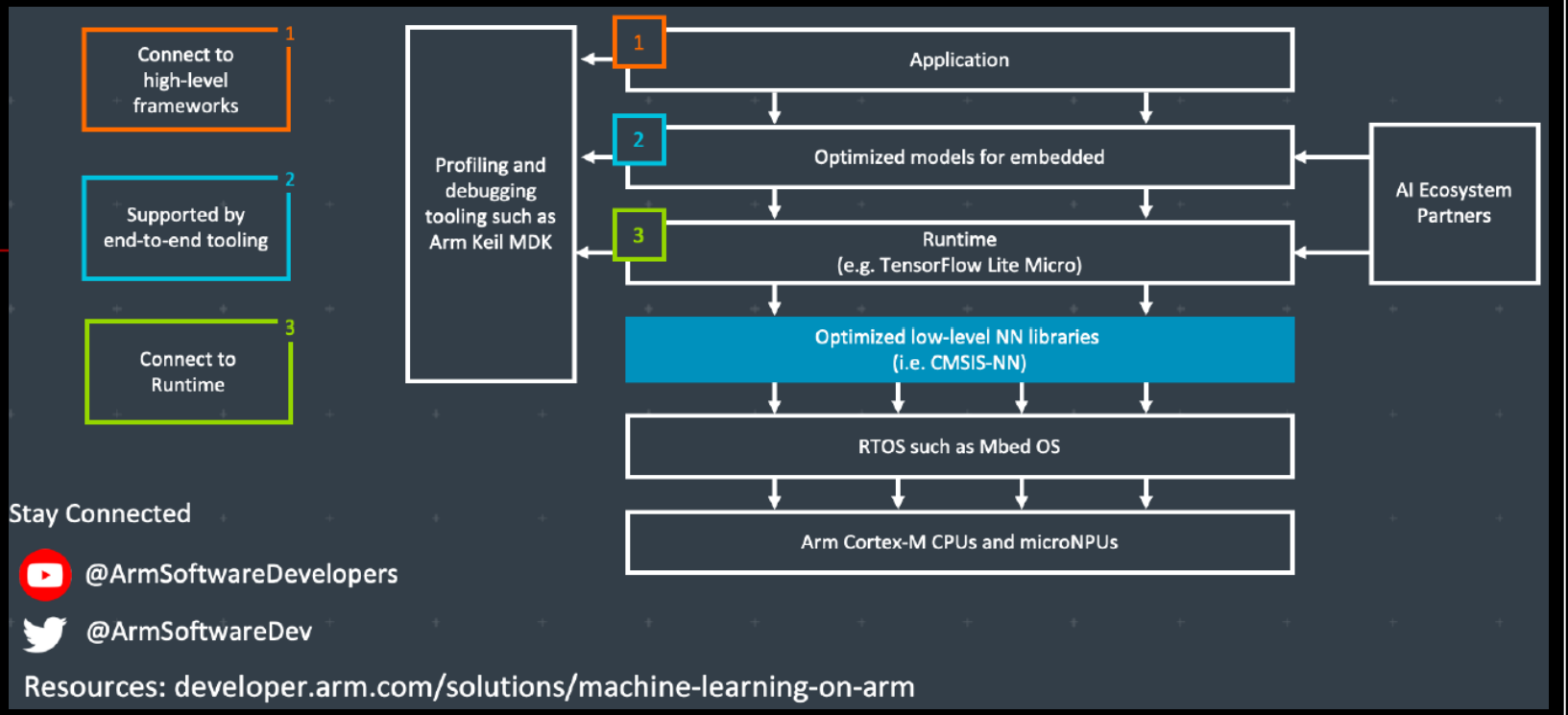
Comparison

https://www.bilibili.com/video/BV1Qu411k7mN

Feature	Cortex-M0	Cortex-M0+	Cortex-M1	Cortex-M23	Cortex-M3	Cortex-M4	Cortex-M33	Cortex-M35P	Cortex-M55	Cortex-M7	Cortex-M85
Instruction Set Architecture	Armv6-M	Armv6-M	Armv6-M	Armv8-M Baseline	Armv7-M	Armv7-M	Armv8-M Mainline	Armv8-M Mainline	Armv8.1-M Mainline	Armv7-M	Armv8.1-M Mainline
TrustZone for Armv8-M	No	No	No	Yes (option)	No	No	Yes (option)	Yes (option)	Yes (option)	No	Yes
Helium (M-Profile Vector Extension)	No	No	No	No	No	No	No	No	Yes (option)	No	Yes (option)
PACBTI Extension	No	No	No	No	No	No	No	No	No	No	Yes (option)
Floating-Point Unit (FPU)	No	No	No	No	No	SP (option)	SP (option)	SP (option)	HP, SP, DP (option)	SP, DP (option)	HP, SP, DP (option)
Digital Signal Processing (DSP) Extension	No	No	No	No	No	Yes	Yes (option)	Yes (option)	Yes	Yes	Yes
Hardware Divide	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Arm Custom Instructions	No	No	No	No	No	No	Yes (option)	No	Yes (option)	No	Yes (available in 2022)
Coprocessor Interface	No	No	No	No	No	No	Yes (option)	Yes (option)	Yes (option)	No	Yes (option)
DMIPS/MHz*	0.96	0.99	0.88	1.03	1.24	1.26	1.54	1.50	1.69	2.31	3.13
CoreMark®/MHz*	2.33	2.46	1.83	2.64	3.45	3.54	4.10	4.10	4.40	5.29	6.28
Maximum # External Interrupts	32	32	32	240	240	240	480	480	480	240	480
Maximum MPU Regions	0	8	0	16	8	8	16	16	16	16	16
Main Bus	AHB Lite (32-bit)	AHB Lite (32-bit)	AHB Lite (32-bit)	AHB (32-bit)	AHB Lite (32-bit)	AHB Lite (32-bit)	AHB (32-bit)	AHB (32-bit)	AXI (64-bit)	AXI (64-bit)	AXI (64-bit)
Instruction Cache	No	No	No	No	No	No	No	2-16kB	0-64kB	0-64kB	0-64kB
Data Cache	No	No	No	No	No	No	No	No	0-64kB	0-64kB	0-64kB
Instruction TCM	No	No	0-1MB	No	No	No	No	No	0-16MB	0-16MB	0-16MB
Data TCM	No	No	0-1MB	No	No	No	No	No	0-16MB	0-16MB	0-16MB
Dual Core Lock-Step (DCLS) Configuration	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes (available in 2022)
Common Criteria Certification	No	No	No	No	No	No	Yes	Yes	No	No	No
Reference Package/System Example	Corstone-101	Corstone-101	-	Corstone-102	Corstone-101	-	Corstone-201	-	Corstone-300	-	Corstone-310

*See individual Cortex-M product pages for further information.
SP = Single-Precision
DP = Double-Precision
HP = Half-Precision

For more information, contact your Arm account manager today or explore the processors in more detail here: developer.arm.com/ip-products/processors/cortex-m



Source: https://cms.tinyml.org/wp-content/uploads/talks2022/tinyML_Talks_Pierre_Gembaczka_211201.pdf

2) Tech Stack

Runtime

- https://en.wikipedia.org/wiki/Runtime_system
- https://en.wikipedia.org/wiki/Register_machine
- https://en.wikipedia.org/wiki/Stack_machine
- https://en.wikipedia.org/wiki/Intermediate_representation
- https://en.wikipedia.org/wiki/Bytecode
- https://en.wikipedia.org/wiki/Compiler
- https://en.wikipedia.org/wiki/Just-in-time_compilation
- https://en.wikipedia.org/wiki/Ahead-of-time_compilation
- https://en.wikipedia.org/wiki/Source-to-source_compiler
- https://en.wikipedia.org/wiki/Interpreter_(computing)
- …
- https://en.wikipedia.org/wiki/Polyglot_(computing)

  Polyglot — use the best tool for the right jobs: high performance, scripting, web, functional programming, etc

  The most popular Polyglot Runtimes: GraalVM, .Net, Wasm…

FPGA

- https://en.wikipedia.org/wiki/Field-programmable_gate_array

2.1 HW/SW Co-design

University of Victoria

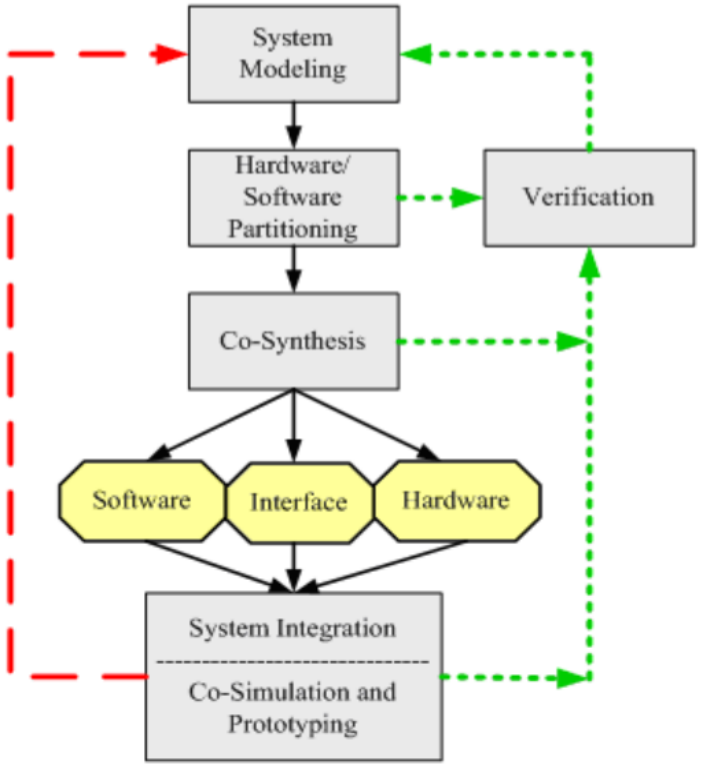
- https://webhome.cs.uvic.ca/~mserra/HScodesign.html
-

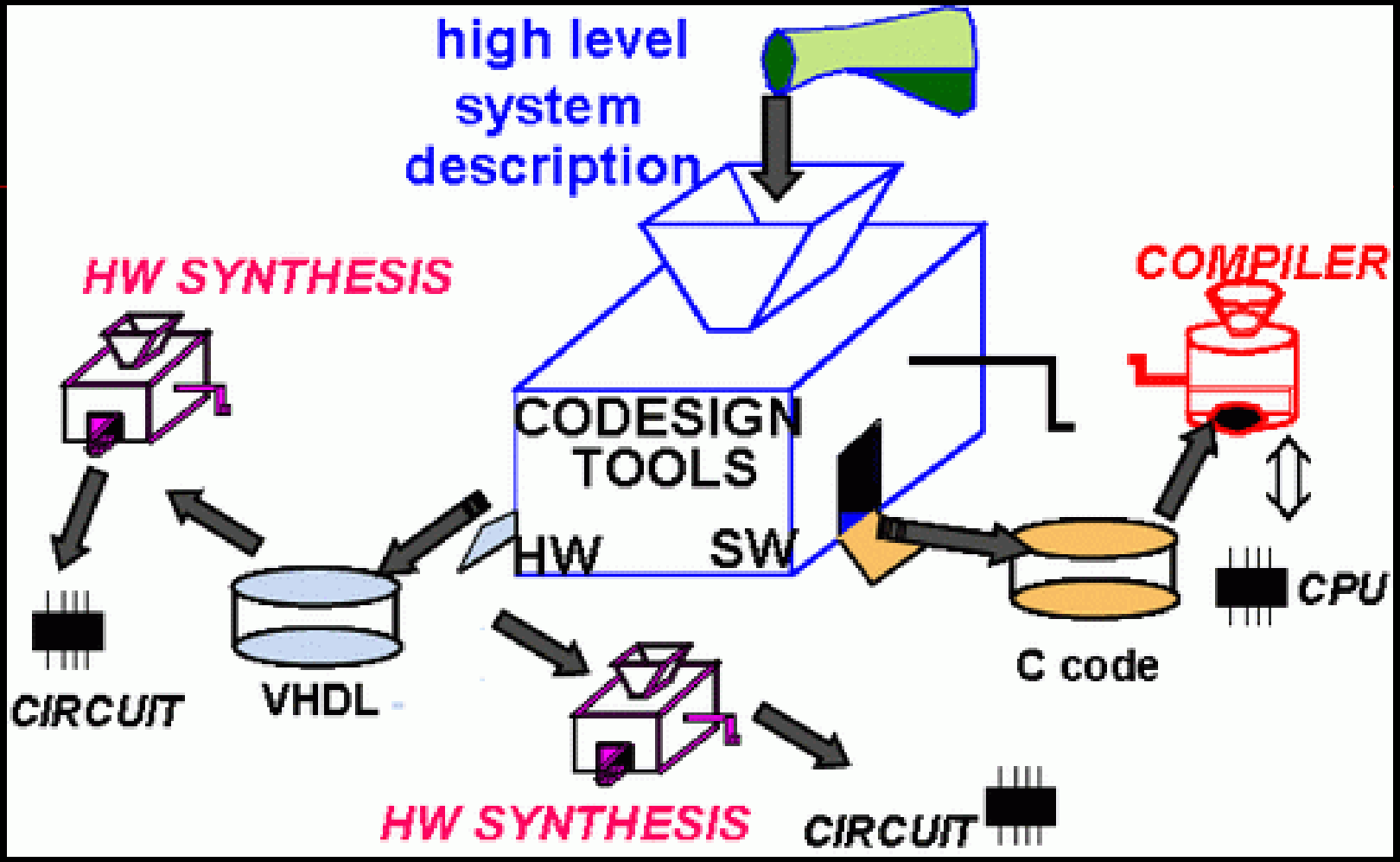
The co-design of HW/SW systems may be viewed as composed of four main phases as illustrated in in the side diagram:

1. Modeling
2. Partitioning
3. Co-Synthesis
4. C-Simulation

Modeling involves specifying the concepts and the constraints of the system to obtain a refined specification. This phase of the design also specifies a software and hardware model. The first problem is to find a suitable specification methodology for a target system. Some researchers favour a formal language which can yield provably-correct code. But most prefer a hardware-type language (e.g., VHDL, Verilog), a software-type language (C, C++, Handel-C, SystemC), or other formalism lacking a hardware or software bias (such as Codesign Finite State Machines). There are three different paths the modeling process can take, considering its starting point:

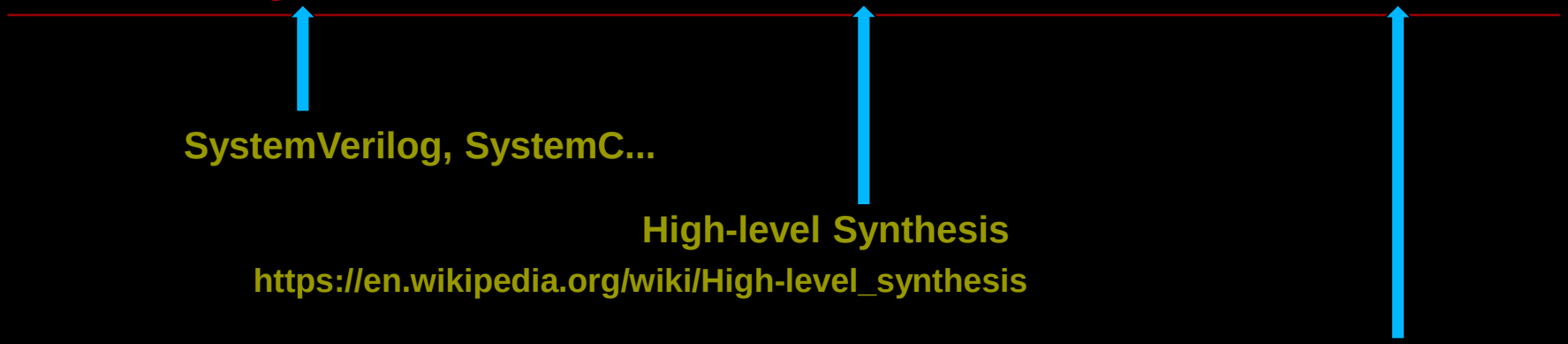
- An existing software implementation of the problem.
- An existing hardware, for example a chip, is present.
- None of the above is given, only specifications leaving an open choice for a model.




high level
system
description
HW SYNTHESIS
COMPILER
CODESIGN
TOOLS
HW
SW
CPU
C code
CIRCUIT
VHDL
HW SYNTHESIS
CIRCUIT

2.1.1 Evolution of HDLs

- https://en.wikipedia.org/wiki/Hardware_description_language
- https://hdl.github.io/awesome/items/
- Verilog/VHDL → HLS → eDSL



Typical eDSLs

- Haskell as host
  Bluespec, Clash...
- Scala as host
  Chisel, SpinalHDL...
- Python as host
  Amaranth/FHDL, PyGears...
- Finally convert to Verilog/VHDL
  https://en.wikipedia.org/wiki/Source-to-source_compiler

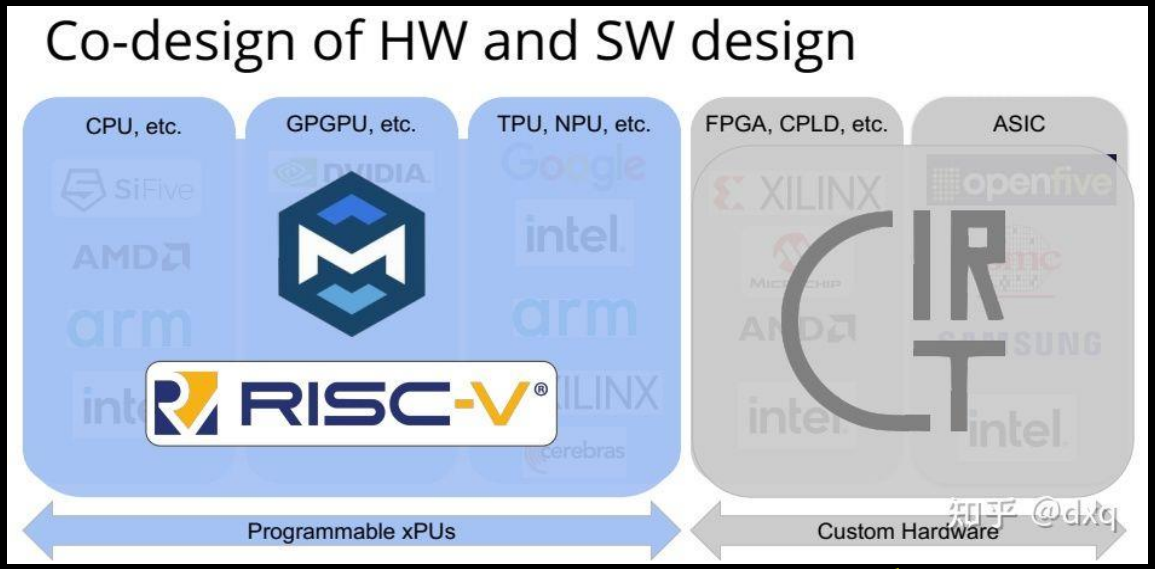
2.1.2 A New Golden Age for Computer Architecture

- https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

"The Linux of the chip world"

- https://www.zdnet.com/article/risc-v-the-linux-of-the-chip-world-is-starting-to-produce-technological-breakthroughs/

2.1.3 The Golden Age of Compiler (in an Era of HW/SW Co-design)



- https://zhuanlan.zhihu.com/p/367035973 //看Chris Lattner在ASPLOS演讲有感

MLIR

- https://mlir.llvm.org/
  Multi-Level Intermediate Representation

The MLIR project is a novel approach to building reusable and extensible compiler infrastructure. MLIR aims to address software fragmentation, improve compilation for heterogeneous hardware, significantly reduce the cost of building domain specific compilers, and aid in connecting existing compilers together.

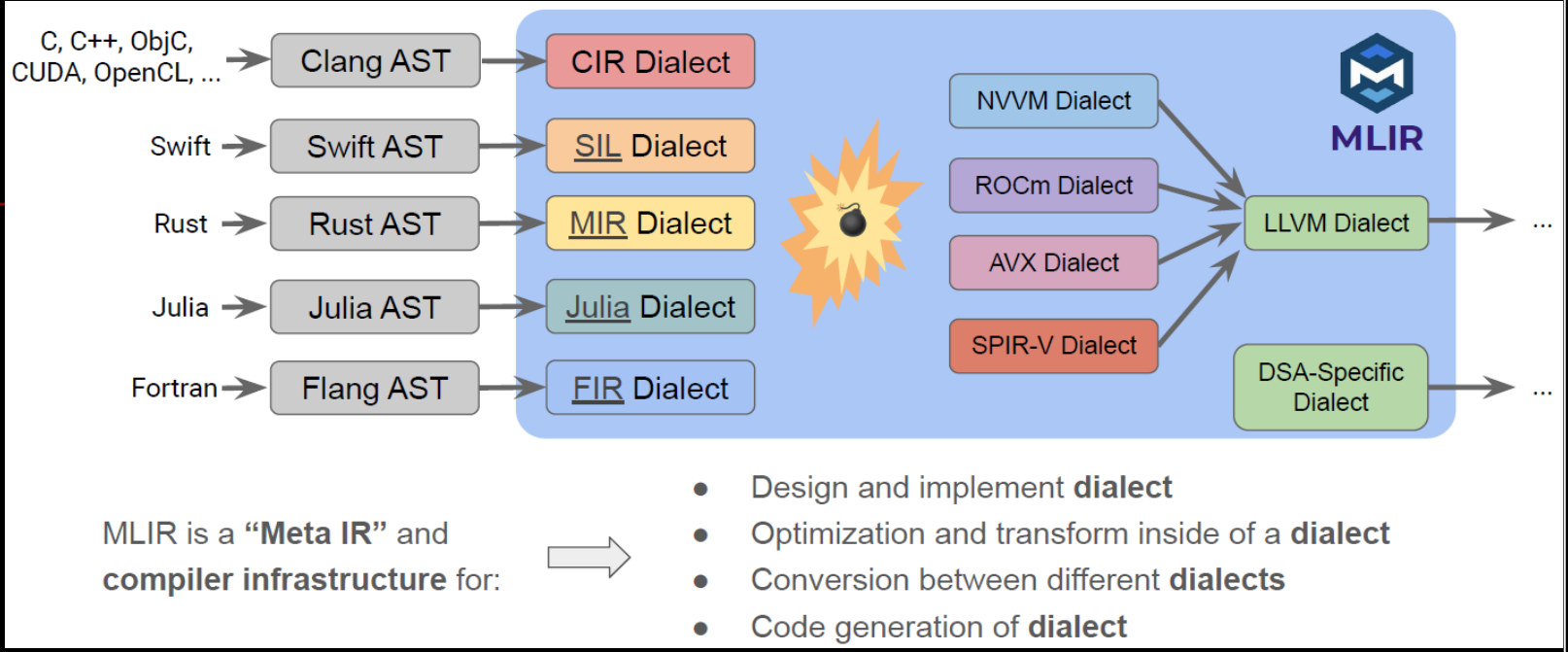
- Motivation

MLIR is intended to be a hybrid IR which can support multiple different requirements in a unified infrastructure. For example, this includes:

- The ability to represent dataflow graphs (such as in TensorFlow), including dynamic shapes, the user-extensible op ecosystem, TensorFlow variables, etc.
- Optimizations and transformations typically done on such graphs (e.g. in Grappler).
- Ability to host high-performance-computing-style loop optimizations across kernels (fusion, loop interchange, tiling, etc.), and to transform memory layouts of data.
- Code generation “lowering” transformations such as DMA insertion, explicit cache management, memory tiling, and vectorization for 1D and 2D register architectures.
- Ability to represent target-specific operations, e.g. accelerator-specific high-level operations.
- Quantization and other graph transformations done on a Deep-Learning graph.
- Polyhedral primitives.
- Hardware Synthesis Tools / HLS.

- ...

MLIR: “Meta IR” and Compiler Infrastructure



Source: https://hanchenye.com/assets/pdf/Gatech_CIRCT_Slides_Hanchen.pdf

CIRCT

- https://circt.llvm.org/
  Circuit IR Compilers and Tools.
- Motivation

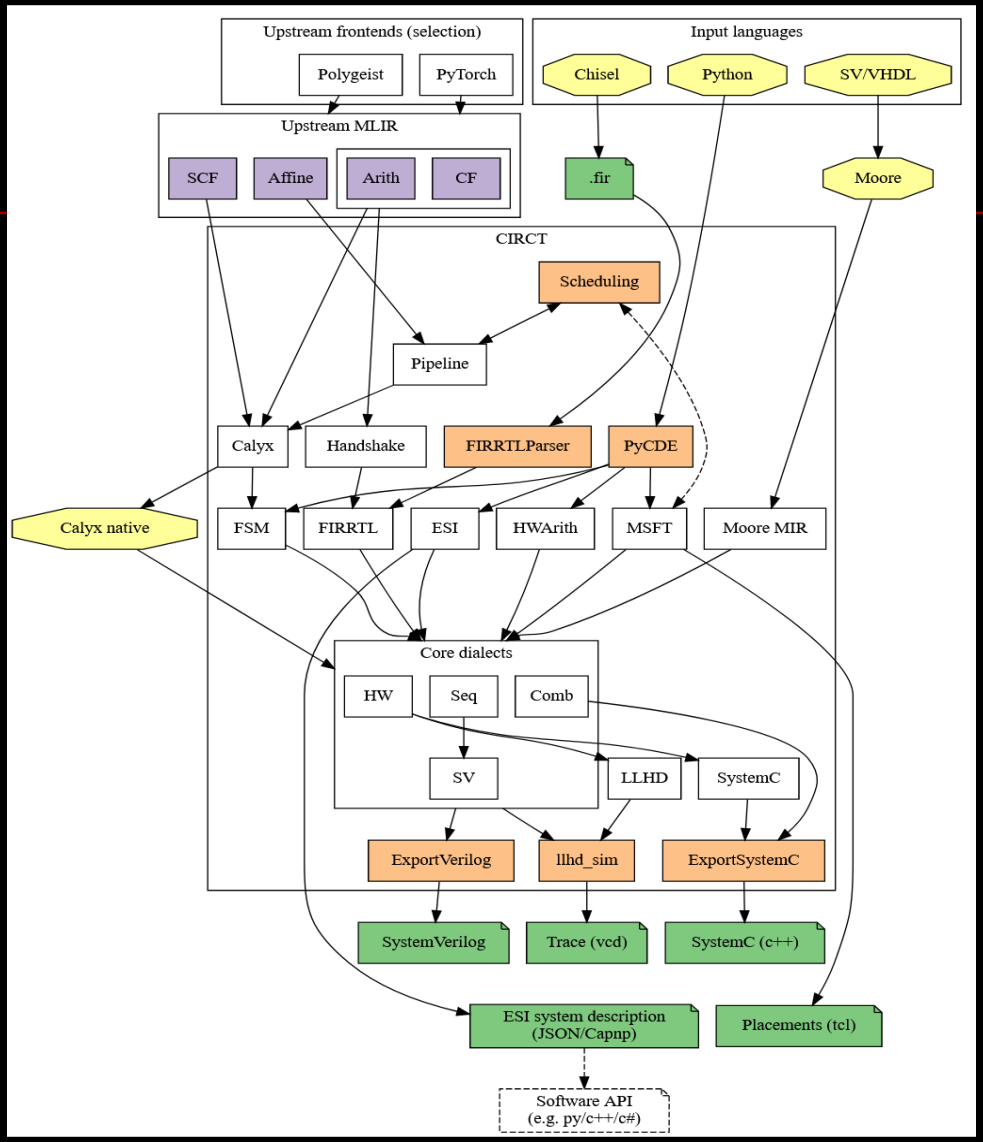
The EDA industry has well-known and widely used proprietary and open source tools. However, these tools are inconsistent, have usability concerns, and were not designed together into a common platform. Furthermore these tools are generally built with Verilog (also VHDL) as the IRs that they interchange. Verilog has well known design issues, and limitations, e.g. suffering from poor location tracking support.

The CIRCT project is an (experimental!) effort looking to apply MLIR and the LLVM development methodology to the domain of hardware design tools. Many of us dream of having reusable infrastructure that is modular, uses library-based design techniques, is more consistent, and builds on the best practices in compiler infrastructure and compiler design techniques.

By working together, we hope that we can build a new center of gravity to draw contributions from the small (but enthusiastic!) community of people who work on open hardware tooling. In turn we hope this will propel open tools forward, enables new higher-level abstractions for hardware design, and perhaps some pieces may even be adopted by proprietary tools in time.

- https://circt.llvm.org/docs/Charter/
- https://mlir.llvm.org/
- https://circt.org/perf/
- …

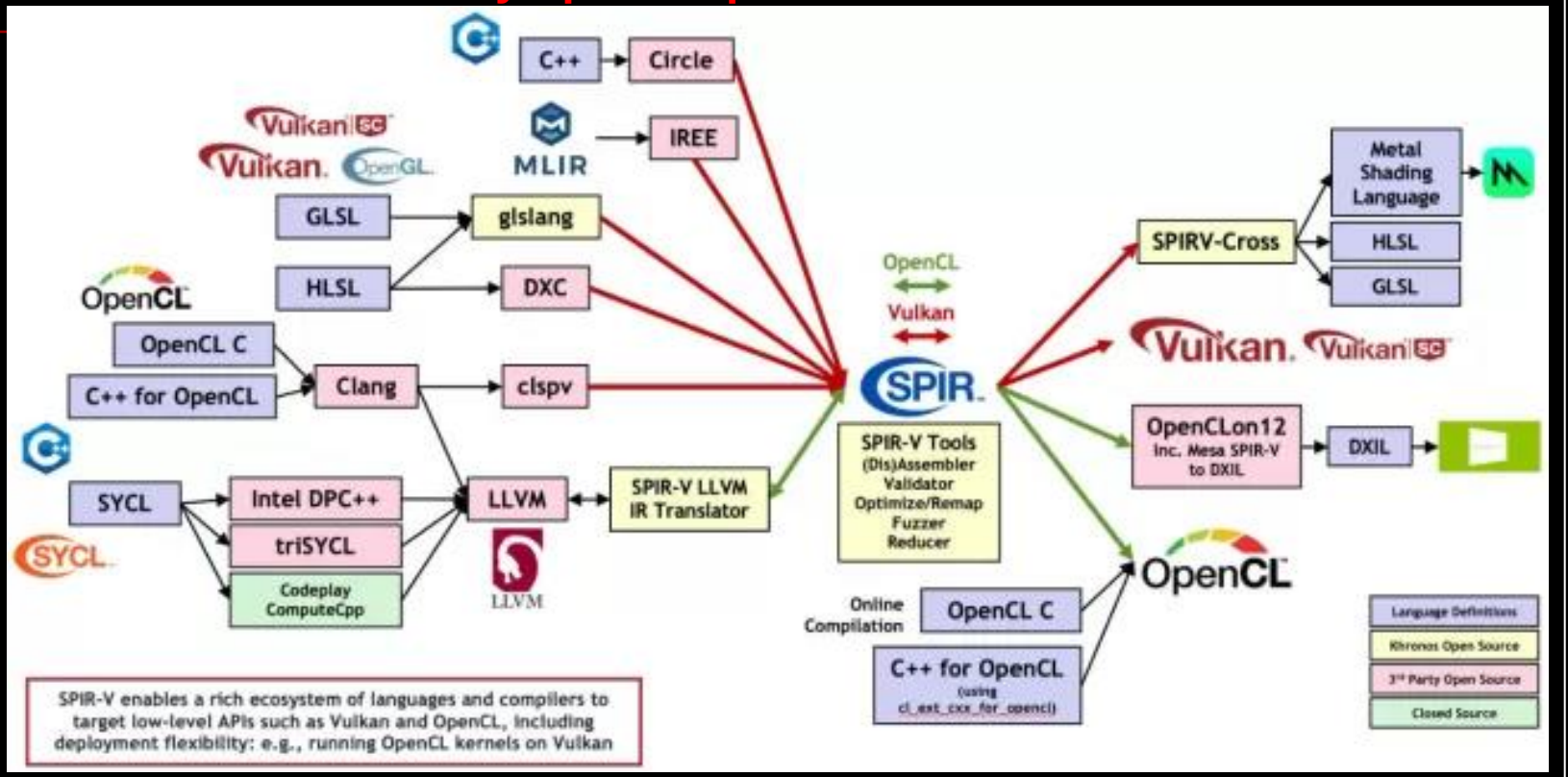
dialects and how they interact



Source: https://circt.llvm.org/includes/img/dialects.svg

Initial SPIR-V Backend Code Lands In LLVM 15

- https://www.phoronix.com/news/LLVM-15-SPIR-V-Backend
  SPIR-V is at the heart of standards/software and with a mainline LLVM back-end can ultimately open it up to even more.



- https://github.com/llvm/llvm-project/commit/7fd4622d4801cc14823ecde678d55b6f3a106eb9
- https://github.com/KhronosGroup/LLVM-SPIRV-Backend

2.1.4 Virtual Prototype/Virtual Platform

2.1.4.1 ARM Virtual Hardware

Enabling Software-Hardware Co-Development for IoT and ML

- **Total Solutions SDK**

https://github.com/ARM-software/open-iot-sdk

Welcome to Arm's Open-IoT-SDK, where you have access to a wide variety of IoT targeted software applications and components ready to run on Arm based platforms. This software framework accelerates intelligent, connected IoT product design by allowing developers to focus on what really matters - innovation and differentiation.

The Open-IoT-SDK contains Total-Solutions applications that allow the user to explore and evaluate not only Arm IP, and the Open-CMSIS-CDI API's, but several other Arm tools as well. It is strongly recommended to explore these pages thoroughly as you will find information on the code itself, the different ways to build and execute the applications as well as the different tools that are available to you, such as Arm Virtual Hardware.

The Open-IoT-SDK located within **GitHub** is a read-only downstream Arm project mirror of the upstream project located in GitLab:Open-IoT-SDK. Any enhancement requests or bugs identified for this repository will need to be submitted to the upstream GitLab:Open-IoT-SDK project.

Arm IoT Total Solutions provides a complete solution designed for specific use-cases containing everything needed to streamline the design process and accelerate product development. To accelerate your software development needs, each Total Solutions brings together:

- Hardware IP
- CSP cloud connectivity middleware
- Real-time OS support
- Machine learning (ML) models
- Advanced tools such as Arm Virtual Hardware
- Application specific reference code
- 3rd Party support from the world's largest IoT ecosystem

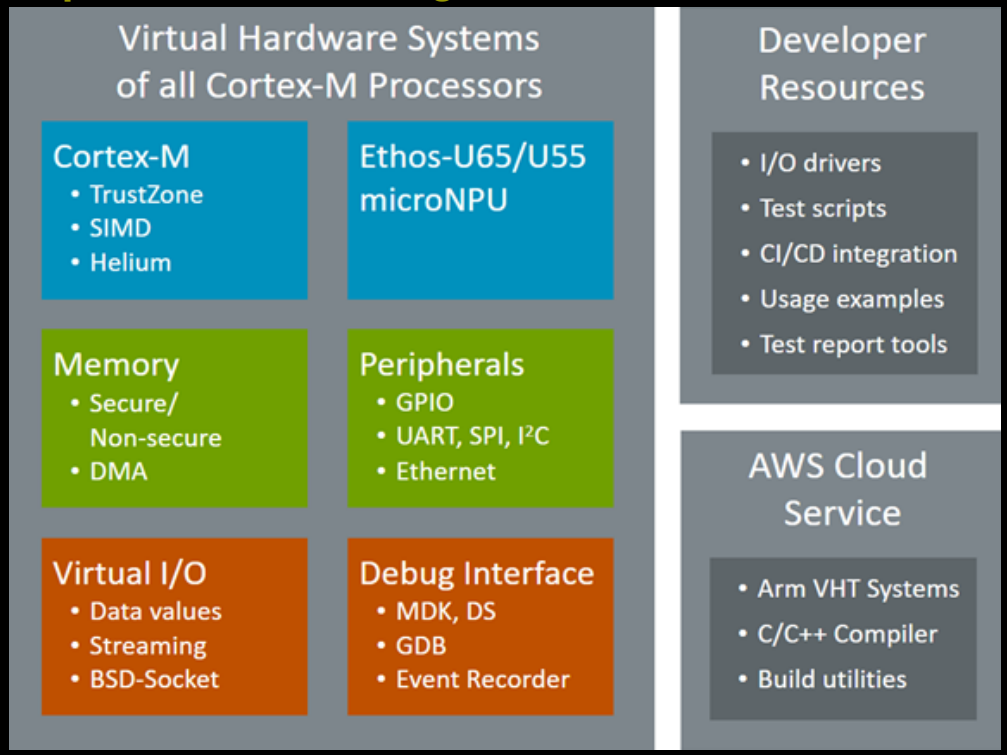
The FreeRTOS kernel is already included in all application's and over time more RTOS kernel abstractions will be added demonstrating the usability of the Open-CMSIS-CDI RTOS interfaces.

AVH

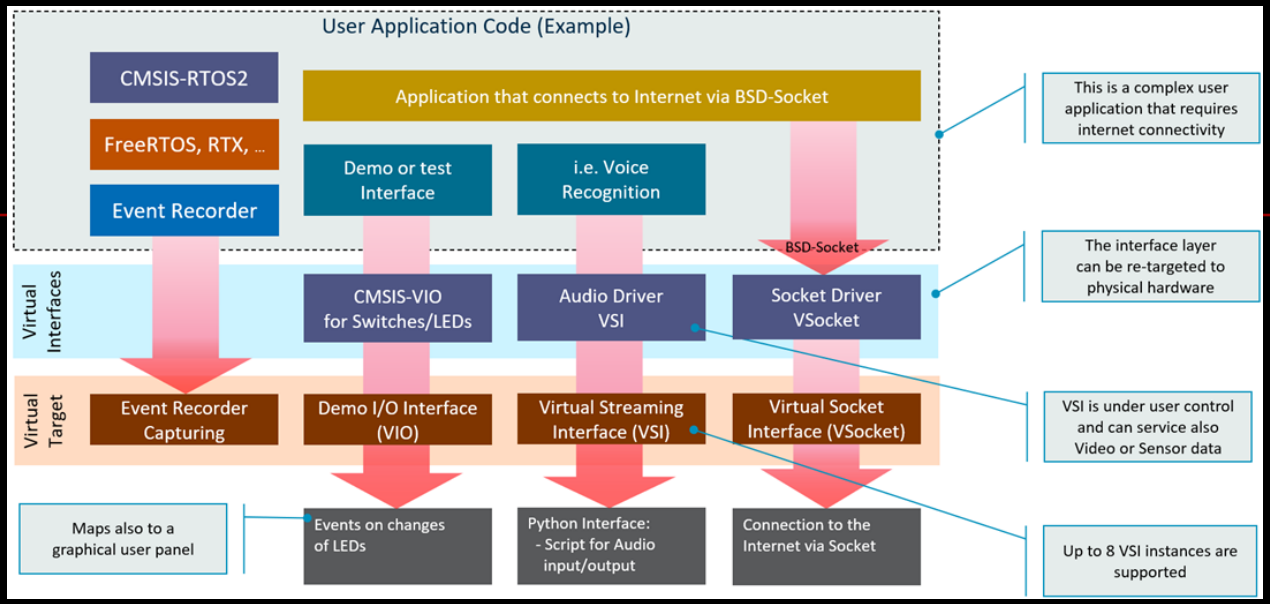
https://avh.arm.com/

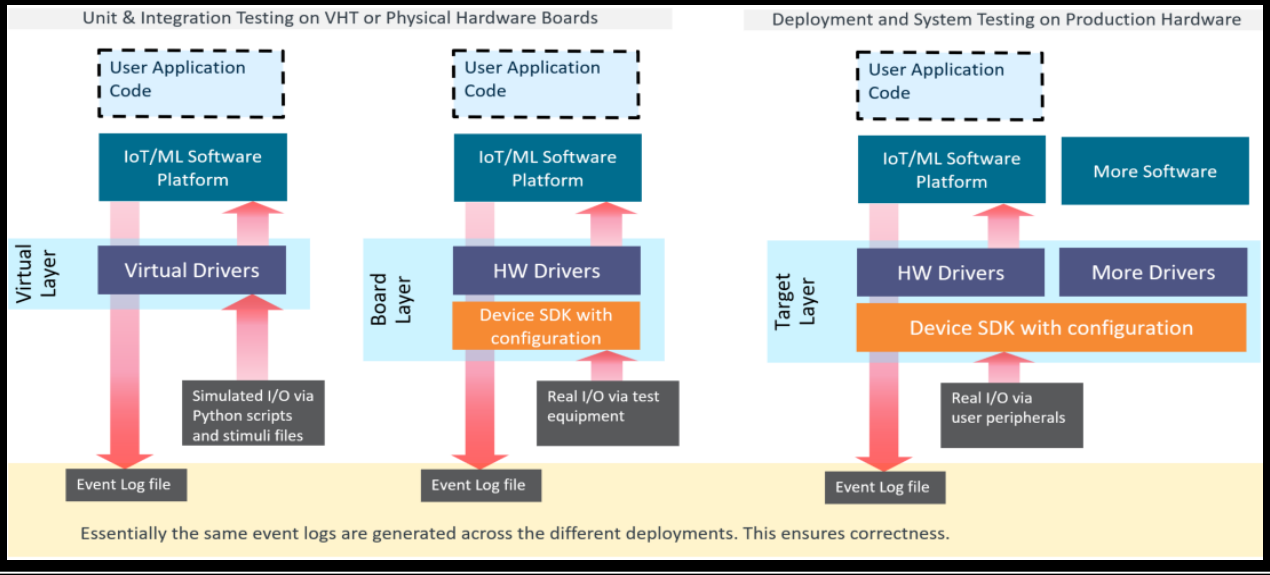
Arm Virtual Hardware (AVH) provides a step change in the evolution of Arm's modeling technology. Arm Virtual Hardware delivers test platforms for developers to verify and validate embedded and IoT applications during the complete software design cycle. Multiple modeling technologies are provided that remove the complexity of building and configuring board farms. This enables modern, agile, cloud native software development practices, such as continuous integration and continuous development CI/CD (DevOps) and MLOps workflows.

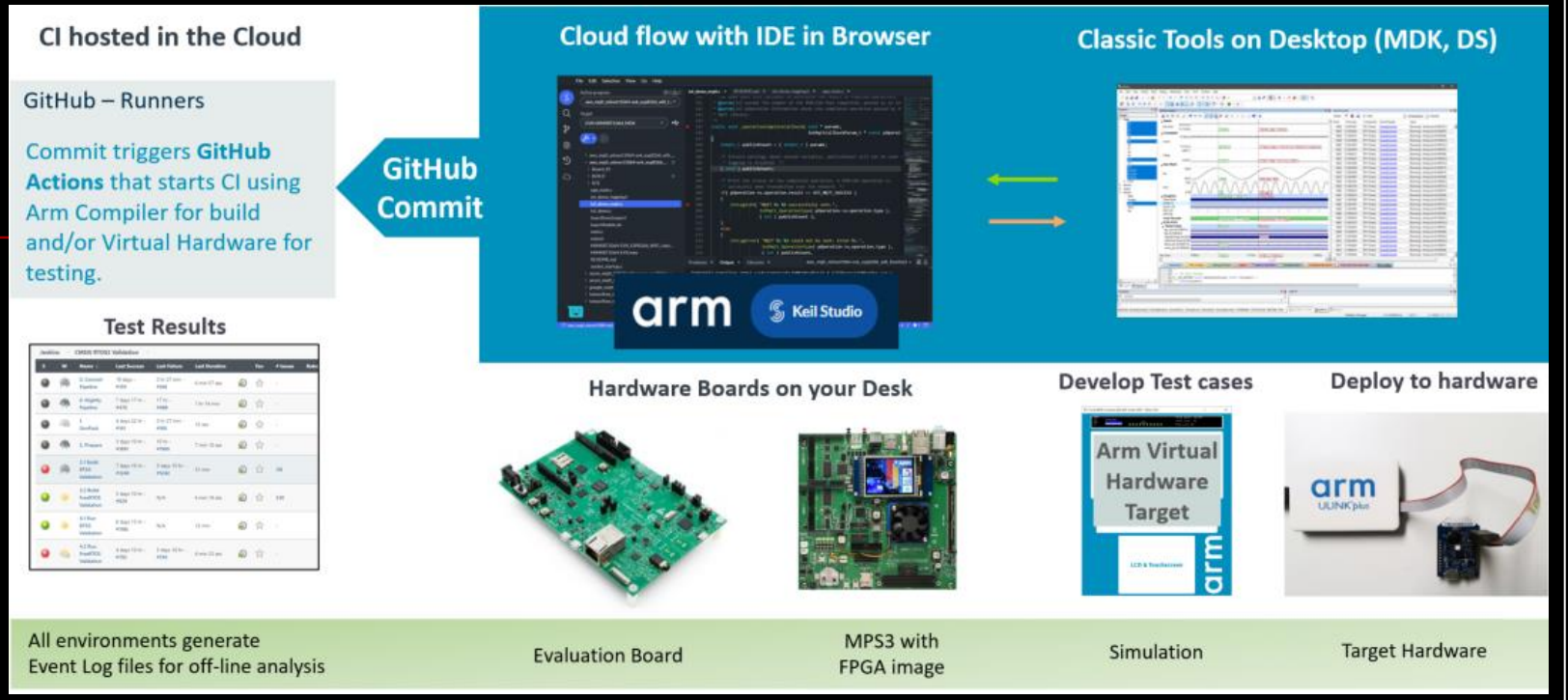
https://arm-software.github.io/AVH/main/overview/html/index.html



https://arm-software.github.io/AVH/main/simulation/html/index.html







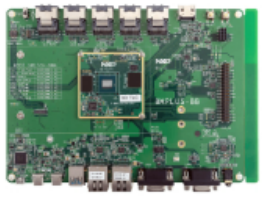
https://github.com/ARM-software/AVH
https://github.com/ARM-software/AVH-GetStarted
https://github.com/ARM-software/AVH-SystemModeling
https://github.com/ARM-software/avh-api
https://github.com/Arm-Software/CMSIS-RTOS2_Validation
https://github.com/Arm-Software/AVH-TFLmicrospeech
https://github.com/Arm-Software/AVH-AWS_MQTT_Demo
https://github.com/Arm-Software/open-iot-sdk/tree/main/examples/ats-keyword
...

3rd Party Hardware

Arm Virtual Hardware hypervisor technology runs on Arm-based Neoverse datacenter CPUs and executes most instructions natively at a much higher speed than is possible with traditional modeling technologies.

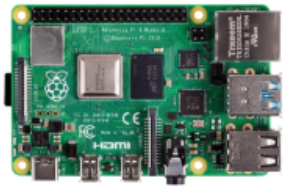
These models virtualize the functional behavior of complete SoCs and development kits, including peripherals, sensors, and other board components. This technology executes the same binaries as on real hardware and leverages the board manufacturers' or Arm silicon partners' SDKs and software code examples.

This vital replacement of board farms utilize cloud-based servers and avoids the complexity and cost of maintaining physical hardware installations, and provides a natural and easy to use API for today's software developers.



i.MX 8M Arm Cortex Complex

AVH model for STM32U5 Discovery



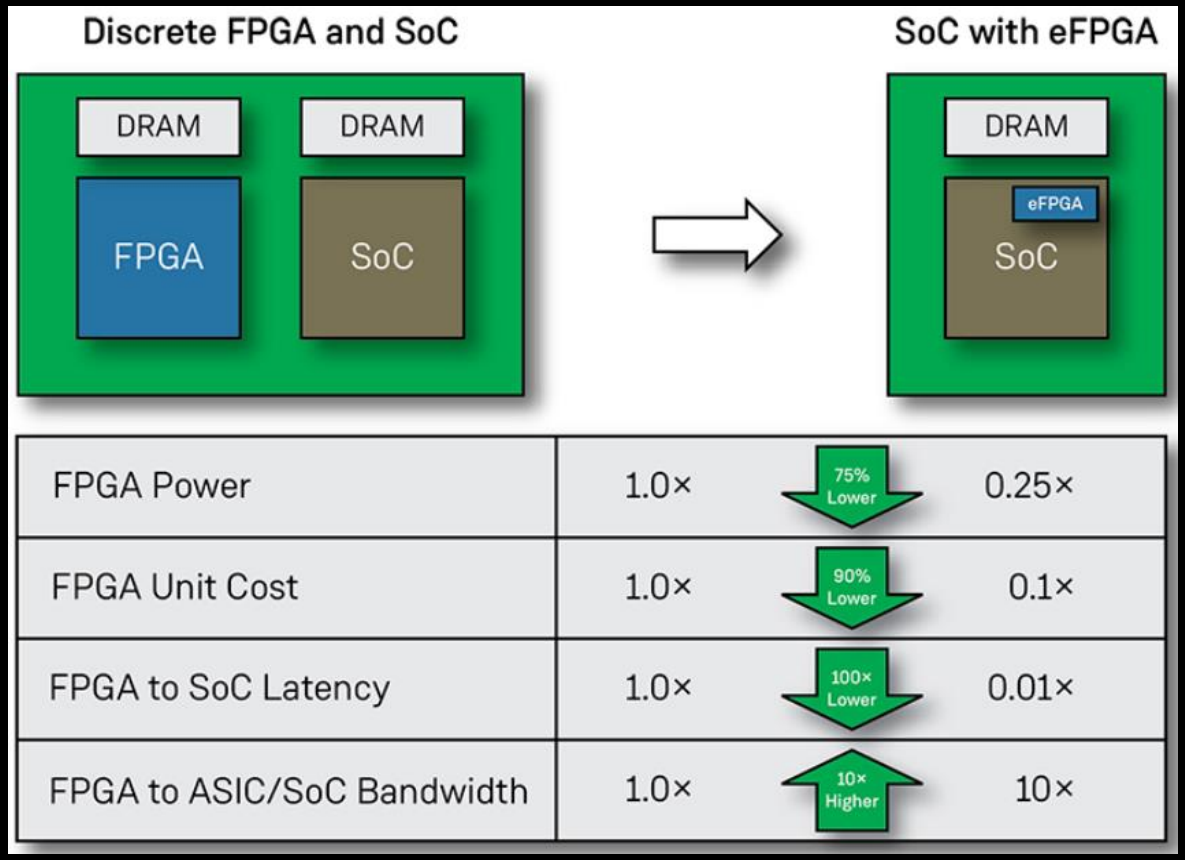
Raspberry Pi Model 4

https://dev.to/aws-builders/welcome-to-the-virtual-raspberry-pi-4-running-on-aws-graviton-processors-2o8e

…

2.2 eFPGA

- embedded Field Programmable Grid Arrays
- https://www.edn.com/embedded-fpga-efpga-technology-past-present-and-future/
- https://flex-logix.com/efpga/what-is-efpga.html
- https://www.achronix.com/product/speedcore



- ...

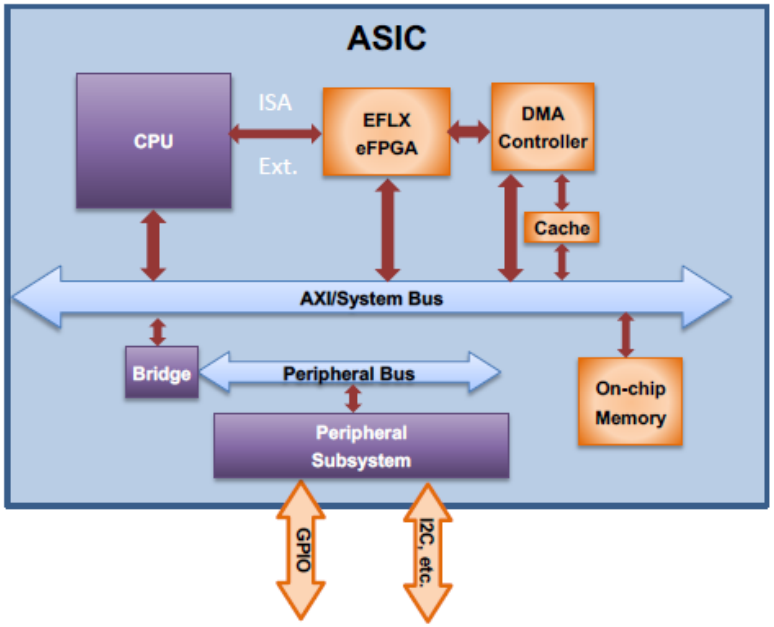
Flexlogix EFLX

Execution

- Accelerators are loaded into local cache
- CPU receives the pragma identifying a specific accelerator
- Kicks off a DMA transaction to load the the bitstream into the eFPGA in background as it continues to execute instructions
- CPU receives interrupt when bitstream loaded
- Sends data to eFPGA to execute like any traditional coprocessor transaction
- Process is repeated as new pragma calls are used by the software

- Configuration time dependent on:
  - Width of configuration bus
  - Configuration bus speed
  - Number of EFLX cores

16ffc and potential max speeds



Configuration time for a single EFLX core

EFLX1K: 482 usec today. (<5 usec potential)

EFLX4K: 1.64 msec today. (< 17 usec potential)

flexlogix AI + eFPGA

EFLX eFPGA - Your Reconfigurable Computing Solution

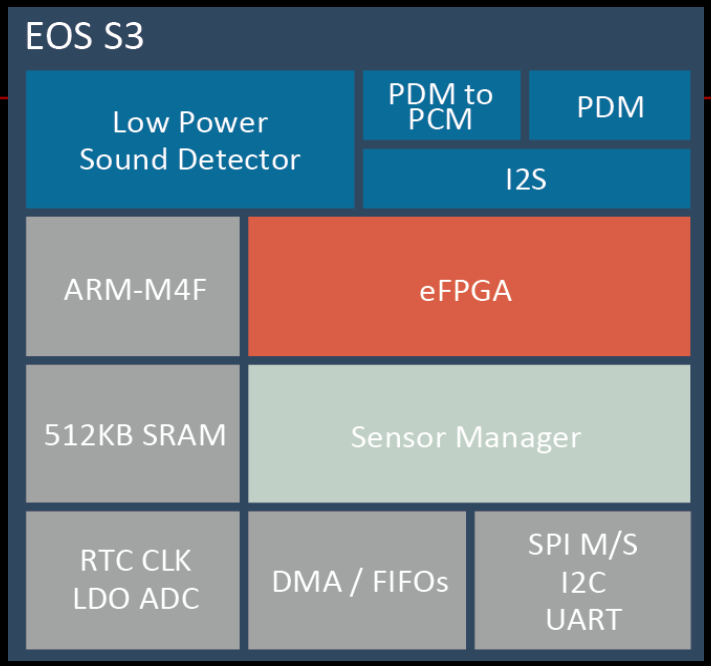
Source: https://d1o0i0v5q5lp8h.cloudfront.net/flexlogix/stage/assets/images/2021-04-06-Dynamically-Reconfigurable-eFPGA.pdf

...

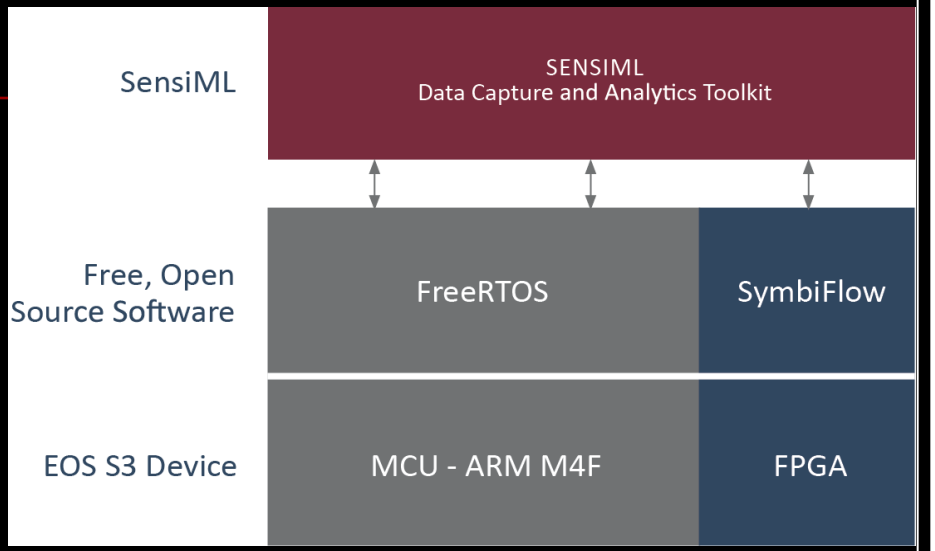
FPGA embedded microcontroller

- https://www.quicklogic.com/products/soc/eos-s3-microcontroller/

AI - Block Diagram

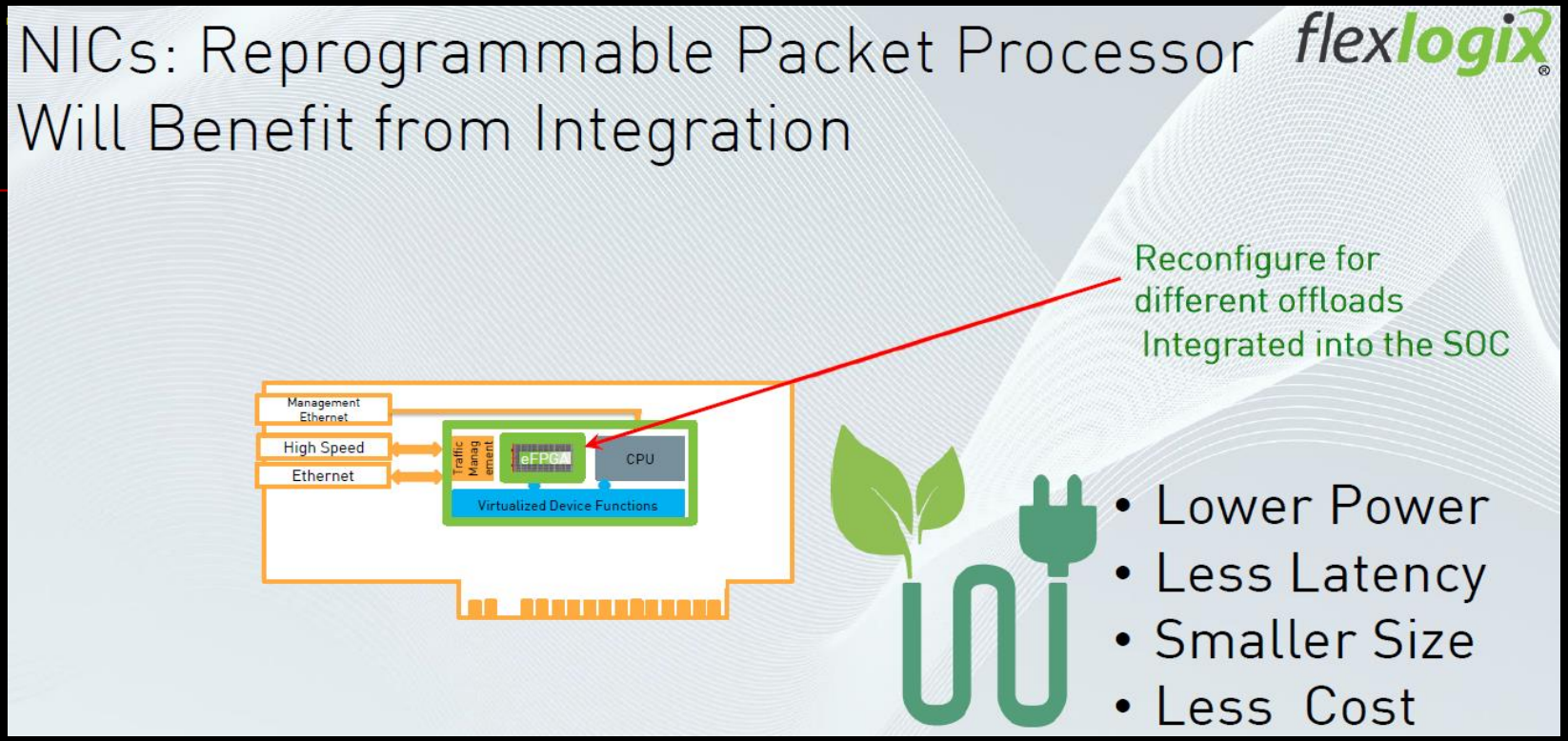


Software Topology



- ...

eFPGA in SmartNIC/DPU



Source: https://smartnicssummit.com/proceeding_files/a0q5f000000lcln/20220427_B-101_Grundler.PDF

- …

eFPGA and Chiplet

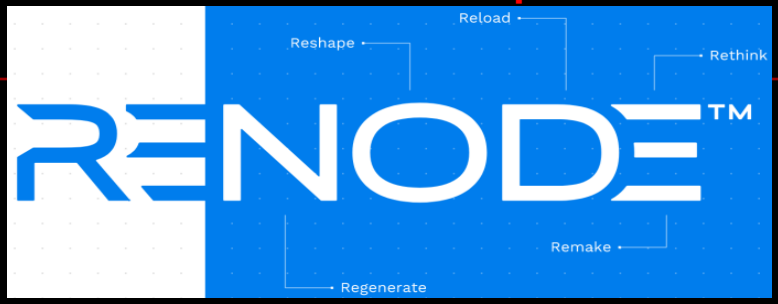
- https://www.quicklogic.com/2022/06/23/efpga-and-chiplet-technology-a-natural-combination/

- https://ir.quicklogic.com/press-releases/detail/637/quicklogic-and-etopus-announce-disaggregated-flexible
- …

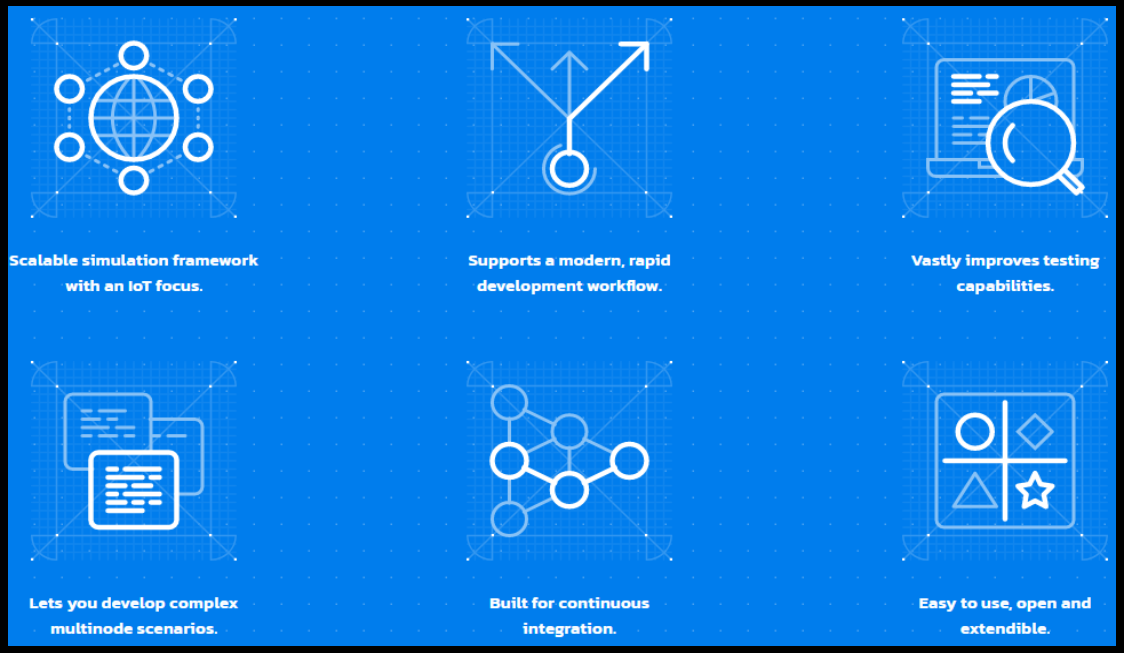
2.3 Renode

- https://renode.io/

  Antmicro's virtual development framework for complex embedded systems.



- Key Features



- https://renode.io/about/

Strength

Open source hardware-simulation framework for:

- Development of complex software for embedded and IoT systems
- Architectural exploration and research
- Pre-silicon prototyping and HW-SW co-development

Features in brief:

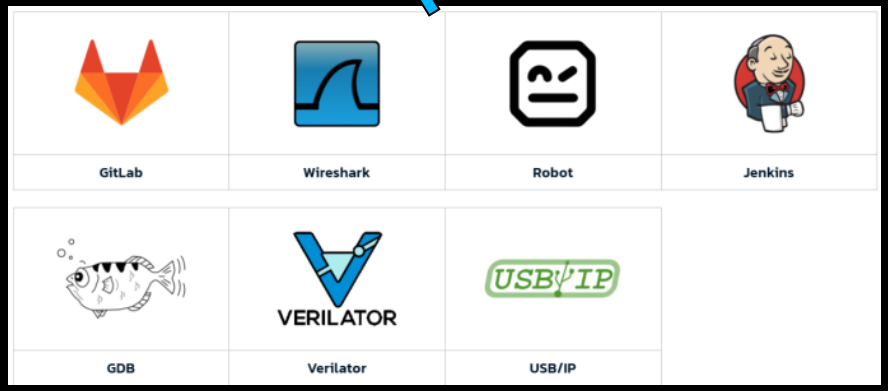
- Plug-and-play building blocks
- Simulates system on many levels - CPUs, SoCs, peripherals, sensors, wired/wireless connection
- Flexible, deterministic and software-agnostic
- Continuous Integration-oriented
- “Batteries included” - lots of demos and binaries

- **full determinism** of execution, shared virtual time
- **transparent & robust debugging**, tracing, analysis, even in multi-node setups
- **easy integration** with your everyday tools, plugins
- **rich model abstractions** with additional functionality "for free" + modular platform description format
- **automated tests and CI integrations**, other collaboration features

Linux-Capable Platforms & Demos:

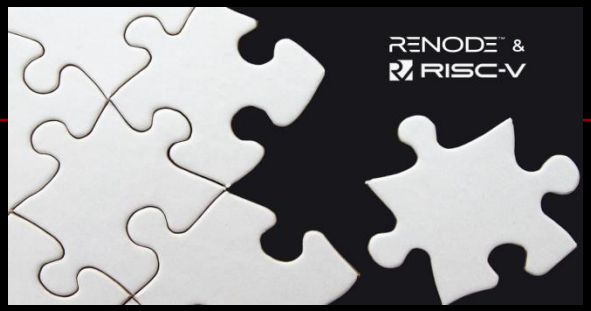
- Kendryte (Linux-nommu)
- LiteX/VexRiscv
- HiFive Unleashed (buildroot)
- PolarFire SoC Icicle Kit (Yocto/buildroot)
- Zedboard
- Vybrid
- Versatile
- Versatile Express
- Nvidia Tegra 3

...



2.3.1 Renode for RISC-V development

Official



RENODE™ RISC-V®

- https://riscv.org/announcements/2021/12/risc-v-celebrates-incredible-year-of-growth-and-progress-ratifying-multiple-technical-specifications-launching-new-education-programs-and-accelerating-broad-industry-adoption/

Notable achievements of RISC-V adoption in 2021 include:

- Antmicro added support for the RVV 1.0 vector instructions to its open source Renode simulation framework, allowing users to enhance their machine learning development experience in a purely virtual environment. RISC-V Vector ISA support will be one of the highlights of the upcoming Renode 1.13 release.

- https://riscv.org/blog/2021/06/antmicro-open-source-portal-launched/
- https://riscv.org/blog/2021/03/bringing-the-benefits-of-risc-v-and-renode-to-the-very-efficient-deep-learning-in-iot-project/
- …

2.3.2 LiteX

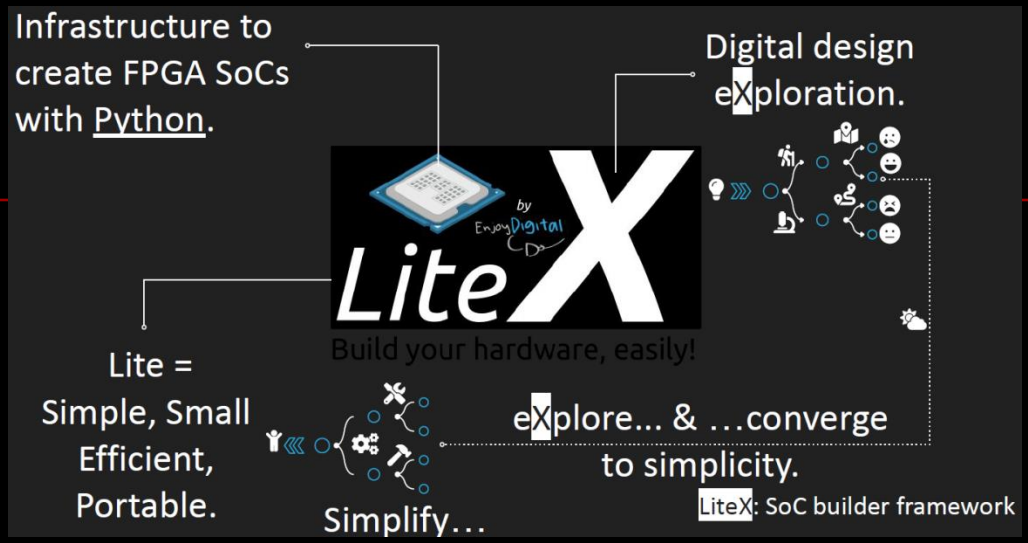
- http://www.enjoy-digital.fr/
  SoC builder framework…
- https://github.com/enjoy-digital/litex

LiteX provides all the common components required to easily create an FPGA Core/SoC:

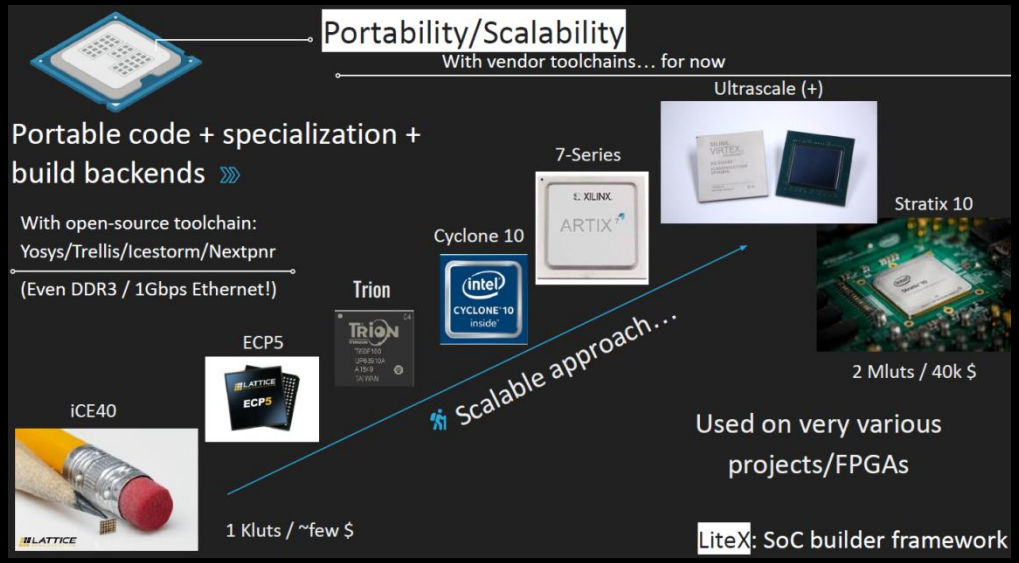
- ✔ Buses and Streams (Wishbone, AXI, Avalon-ST) and their interconnect.
- ✔ Simple cores: RAM, ROM, Timer, UART, JTAG, etc....
- ✔ Complex cores through the ecosystem of cores: LiteDRAM, LitePCIe, LiteEth, LiteSATA, etc...
- ✔ Various CPUs & ISAs: RISC-V, OpenRISC, LM32, Zynq, X86 (through a PCIe), etc...
- ✔ Mixed languages support with VHDL/Verilog/(n)Migen/Spinal-HDL/etc... integration capabilities.
- ✔ Powerful debug infrastructure through the various bridges and Litescope.
- ✔ Direct/Fast simulation through Verilator.
- ✔ Build backends for open-source and vendors toolchains.
- ✔ And a lot more... :)

- https://github.com/litex-hub
- https://github.com/timvideos/litex-buildenv
- https://m-labs.hk/migen/manual/fhdl.html
- https://github.com/litex-hub/litex-boards

Design philosophy

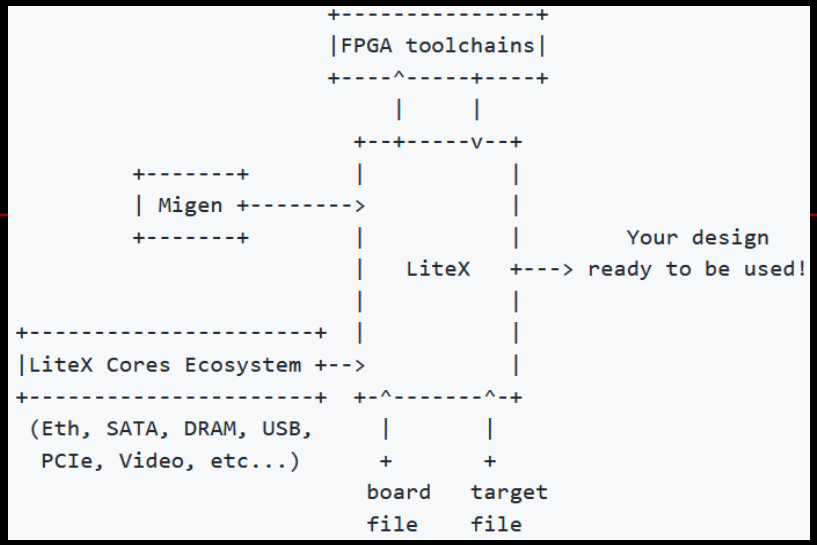


Source: "LiteX: SoC builder framework", Florent Kermarrec

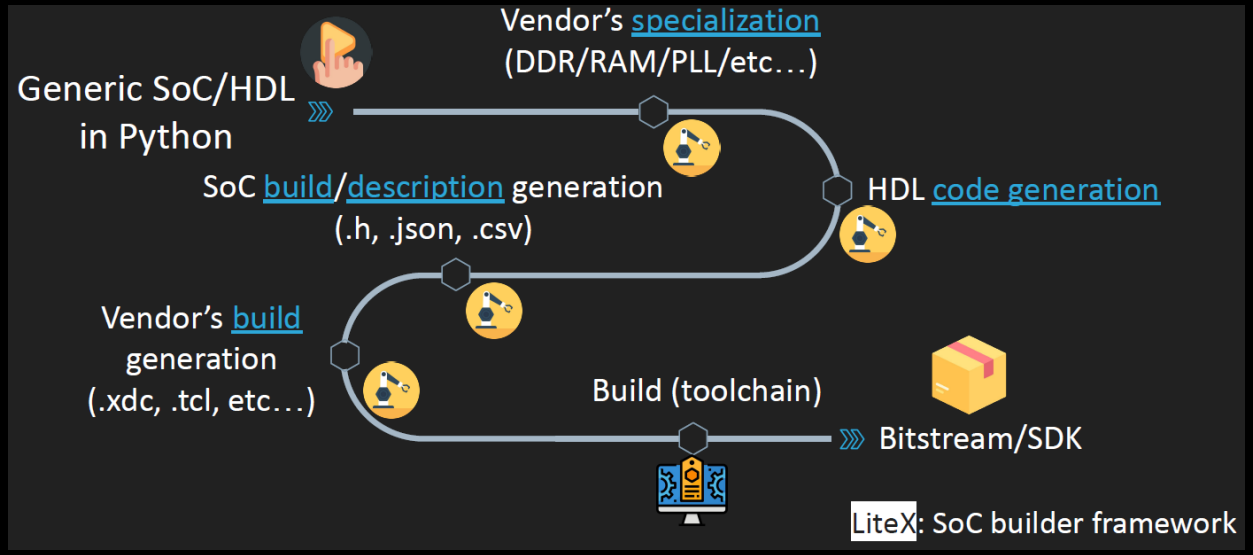


Source: "LiteX: SoC builder framework", Florent Kermarrec

Design flow



Source: https://github.com/enjoy-digital/litex



Source: "LiteX: SoC builder framework", Florent Kermarrec

Simulate in Renode

- https://github.com/enjoy-digital/litex/wiki/Generate-Renode-simulation

  LiteX provides a script for automatic generation of Renode scripts from the json file containing configuration of the LiteX SoC:

  https://github.com/enjoy-digital/litex/blob/master/litex/tools/litex_json2renode.py

- Usage

First, build your LiteX platform with `--csr-json csr.json` switch, e.g.:

```
python3 litex/boards/targets/arty.py --cpu-type vexriscv --with-ethernet --csr-json csr.json
```

Now, use the generated configuration file as an input for `litex_json2renode.py`:

```
./litex_json2renode.py csr.json \
    --resc litex.resc \
    --repl litex.repl
    --bios-binary soc_ethernetsoc_arty/software/bios/bios.bin
```

This will generate two files:

- `litex.repl` - platform definition file, containing information about all the peripherals and their configuration,
- `litex.resc` - Renode script file, allowing to easily run the simulation of the generated platform.

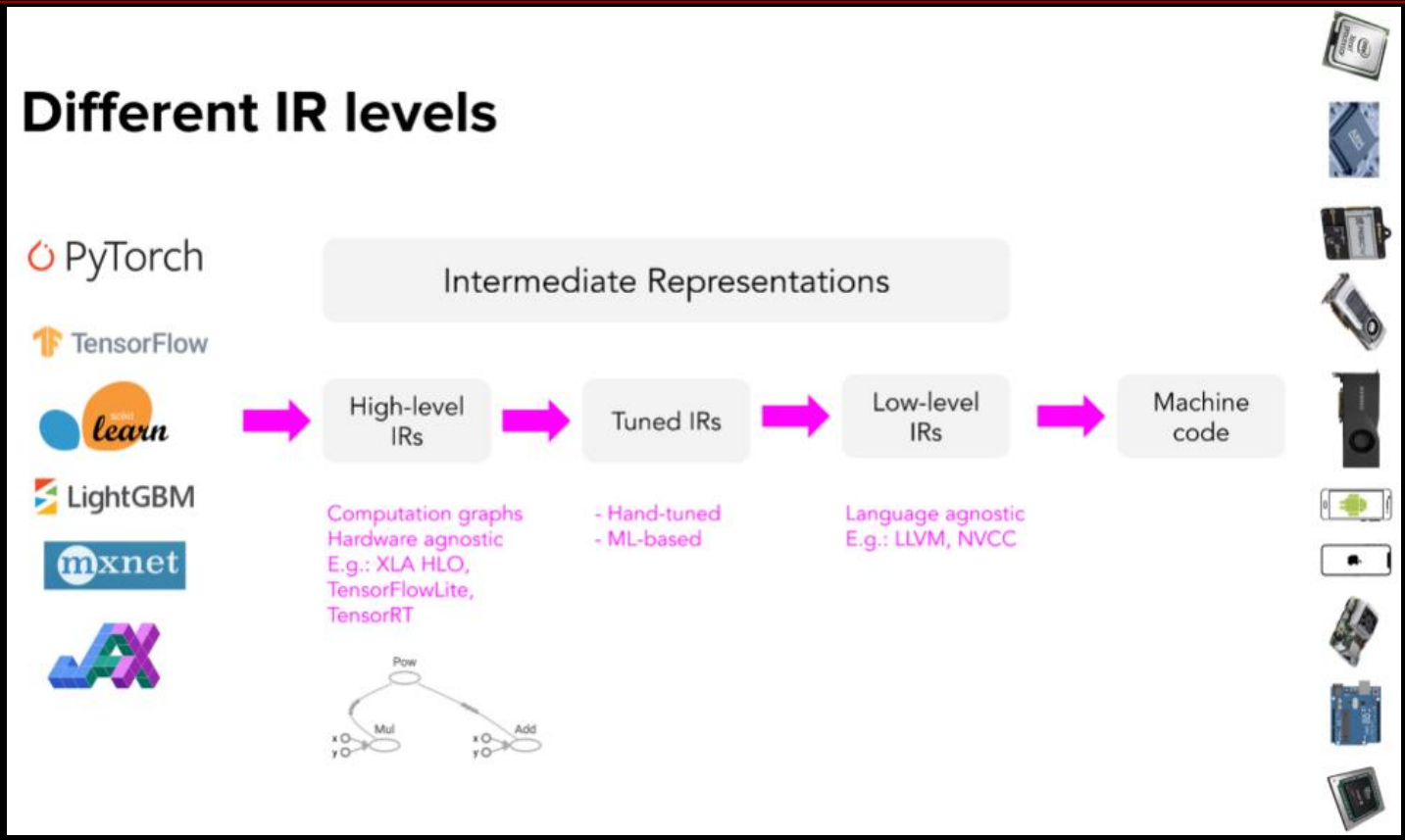
Finally, you can run the simulation by executing the command:

```
renode litex.resc
```

- Abandoned: https://github.com/litex-hub/litex-renode

2.4 AI Compiler

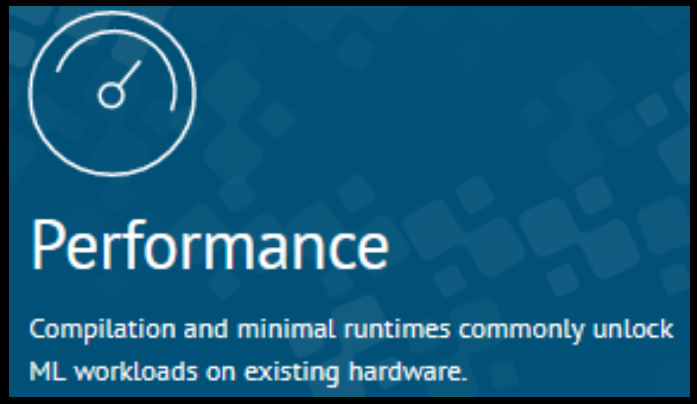
- …
- https://huyenchip.com/2021/09/07/a-friendly-introduction-to-machine-learning-compilers-and-optimizers.html

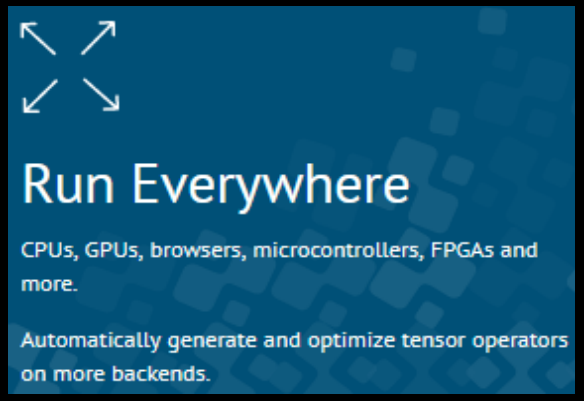


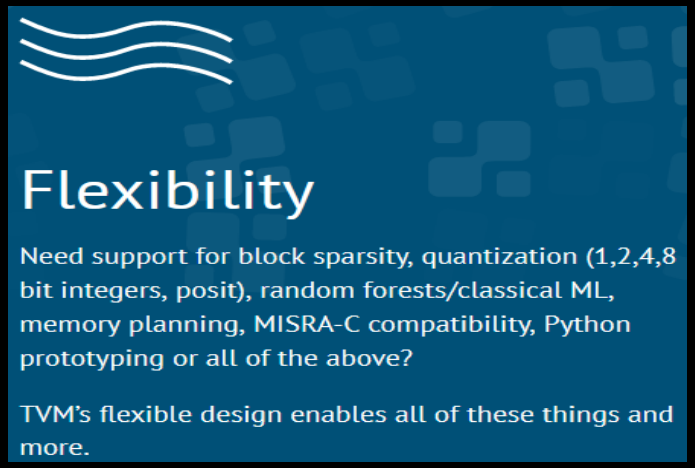
- https://autokernel-docs-en.readthedocs.io/en/latest/blog/ai_compiler%20overview.html

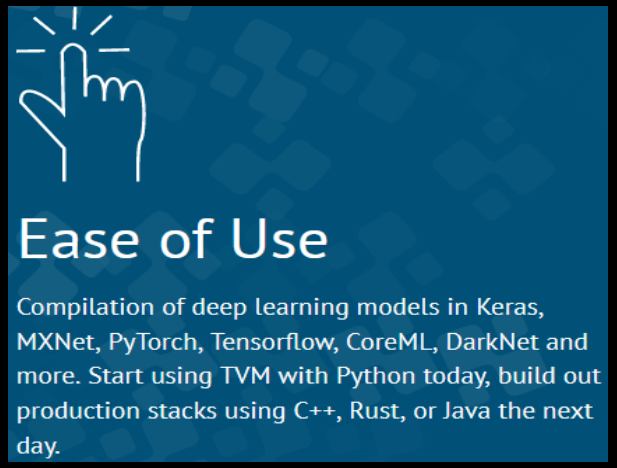
2.4.1 TVM

- https://tvm.apache.org/
  An End to End Machine Learning Compiler Framework for CPUs, GPUs and accelerators
- Key Features & Capabilities



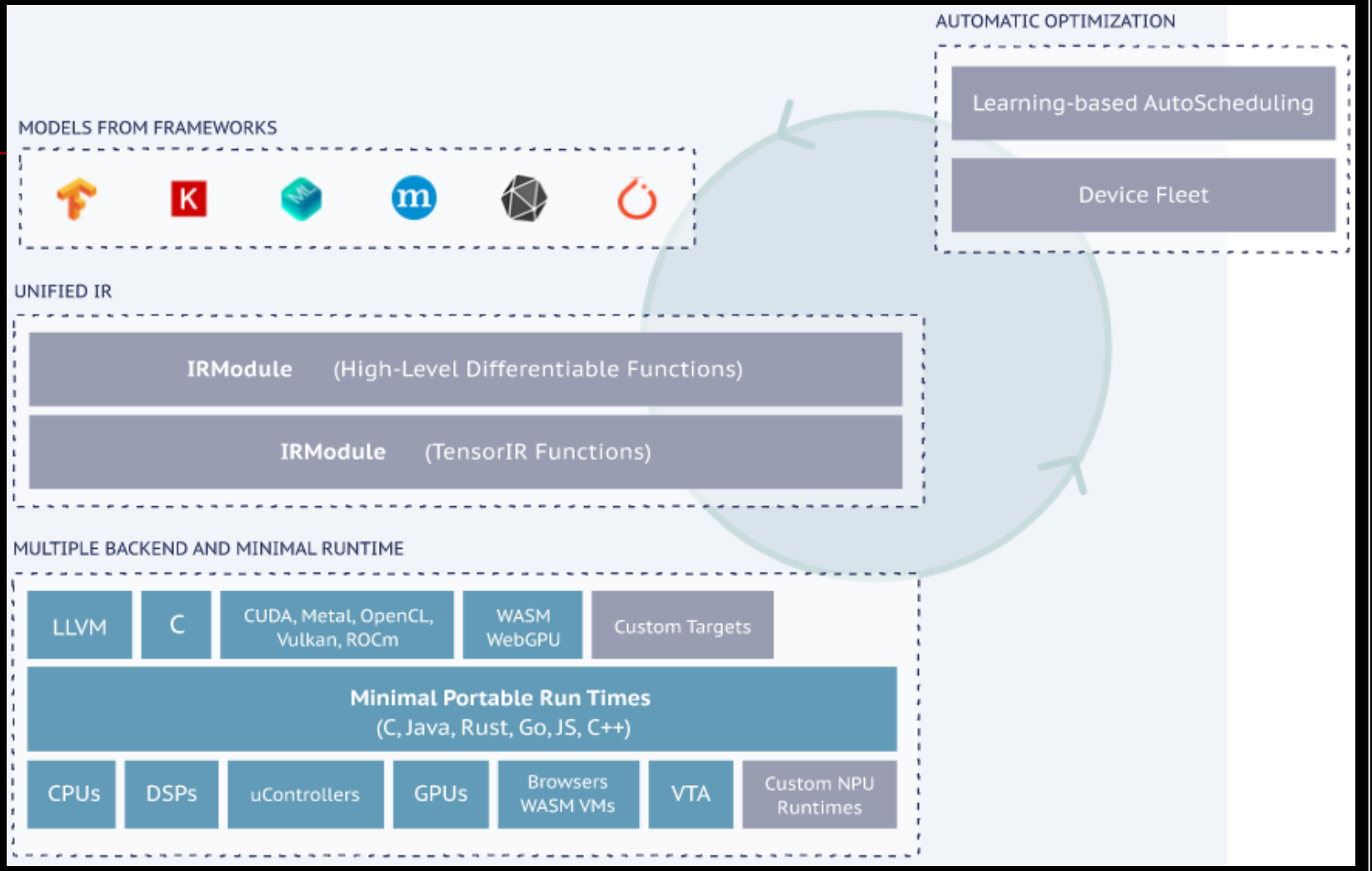






- https://github.com/apache/tvm

Architecture & Design



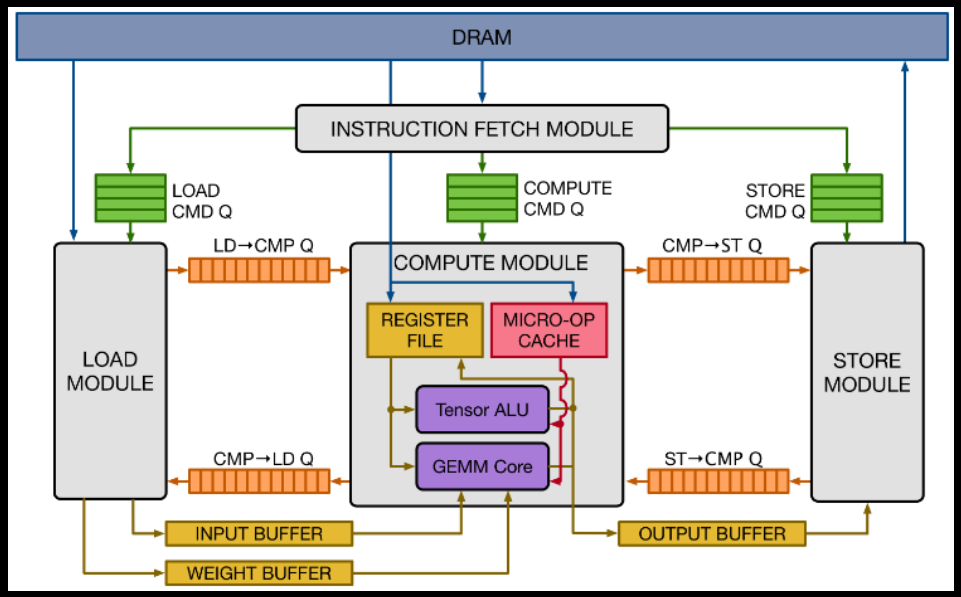
- **Python-first API**

2.4.1.1 VTA

- https://tvm.apache.org/docs/topic/vta/index.html

Versatile Tensor Accelerator

The Versatile Tensor Accelerator (VTA) is an open, generic, and customizable deep learning accelerator with a complete TVM-based compiler stack. We designed VTA to expose the most salient and common characteristics of mainstream deep learning accelerators. Together TVM and VTA form an end-to-end hardware-software deep learning system stack that includes hardware design, drivers, a JIT runtime, and an optimizing compiler stack based on TVM.



Key Features

- Generic, modular, open-source hardware.
- Streamlined workflow to deploy to FPGAs.
- Simulator support to prototype compilation passes on regular workstations.
- Pynq-based driver and JIT runtime for both simulated and FPGA hardware back-end.
- End to end TVM stack integration.

2.4.1.2 microTVM (uTVM)

- https://tvm.apache.org/docs/topic/microtvm/index.html
  Runs TVM models on bare-metal (i.e. IoT) devices. microTVM depends only on the C standard library, and doesn't require an operating system to execute. microTVM is currently under heavy development.
- Features

microTVM is:

- an extension to TVM's compiler to allow it to target microcontrollers
- a way to run the TVM RPC server on-device, to allow autotuning
- a minimal C runtime that supports standalone model inference on bare metal devices.

Supported Hardware

microTVM currently tests against Cortex-M microcontrollers with the Zephyr RTOS; however, it is flexible and portable to other processors such as RISC-V and does not require Zephyr. The current demos run against QEMU and the following hardware:

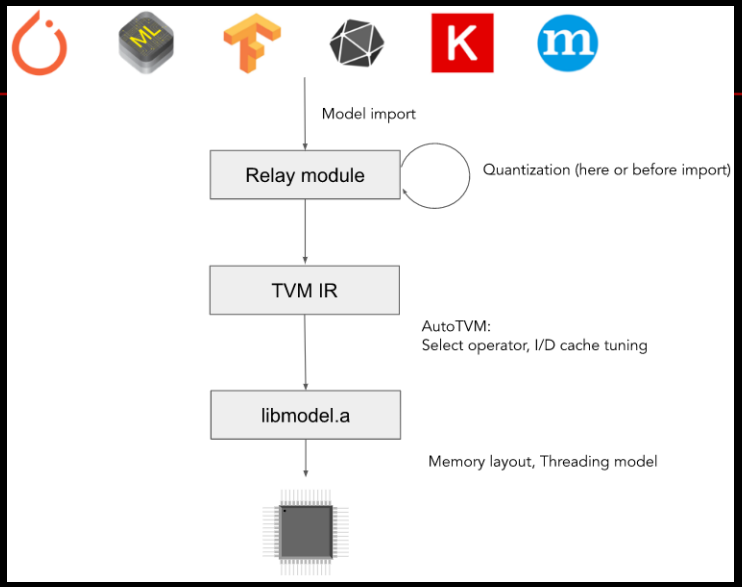
- STM Nucleo-F746ZG
- STM STM32F746 Discovery
- nRF 5340 Development Kit

- Design Goals

1. **Portable Code**. microTVM can translate any Relay model into C code that can compile with only a C standard library.
2. **Minimal Overhead**. microTVM generates target-specific, highly optimized code. As much overhead from the runtime should be removed.
3. **Accessible Code**. microTVM considers C source code as a first-class output mechanism so that it is easier for a firmware engineer to understand and tweak.

Architecture & Design

- https://tvm.apache.org/docs/arch/microtvm_design.html#microtvm-design



- **Executing Models**

The TVM compiler traditionally outputs three pieces:

1. Model operator implementations, as discussed above;
2. A model execution graph, encoded as JSON; and
3. Simplified parameters.

To correctly execute the model, a Graph Executor needs to reconstruct the graph in memory, load the parameters, and then invoke the operator implementations in the correct order.

microTVM supports two ways to do this:

1. **Host-Driven**. The Graph Executor can run on the host and carry out execution by issuing commands to the device using an RPC link with a UART-like transport.
2. **Standalone**. A C Graph Executor is available to be compiled on-device, but it is not particularly memory efficient. This way enables standalone execution without any attached host.

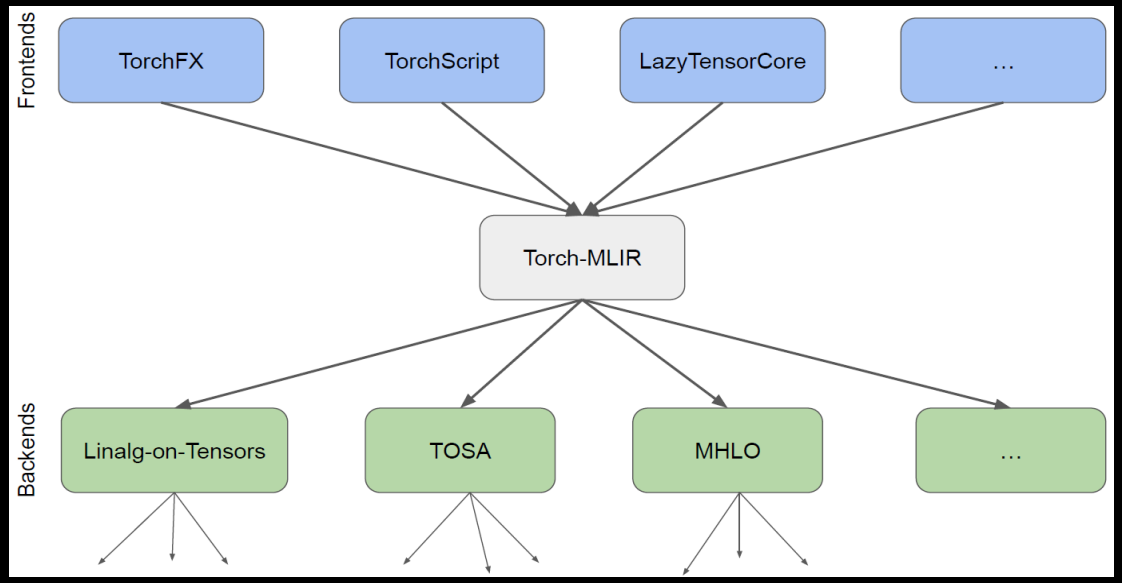
2.4.2 Torch-MLIR

- https://github.com/llvm/torch-mlir

  To provide first class support from the PyTorch ecosystem to the MLIR ecosystem.

Torch-MLIR Multiple Vendors use MLIR as the middle layer, mapping from platform frameworks like PyTorch, JAX, and TensorFlow into MLIR and then progressively lowering down to their target hardware. We have seen half a dozen custom lowerings from PyTorch to MLIR. Having canonical lowerings from the PyTorch ecosystem to the MLIR ecosystem would provide much needed relief to hardware vendors to focus on their unique value rather than implementing yet another PyTorch frontend for MLIR. The goal is to be similar to current hardware vendors adding LLVM target support instead of each one also implementing Clang / a C++ frontend.

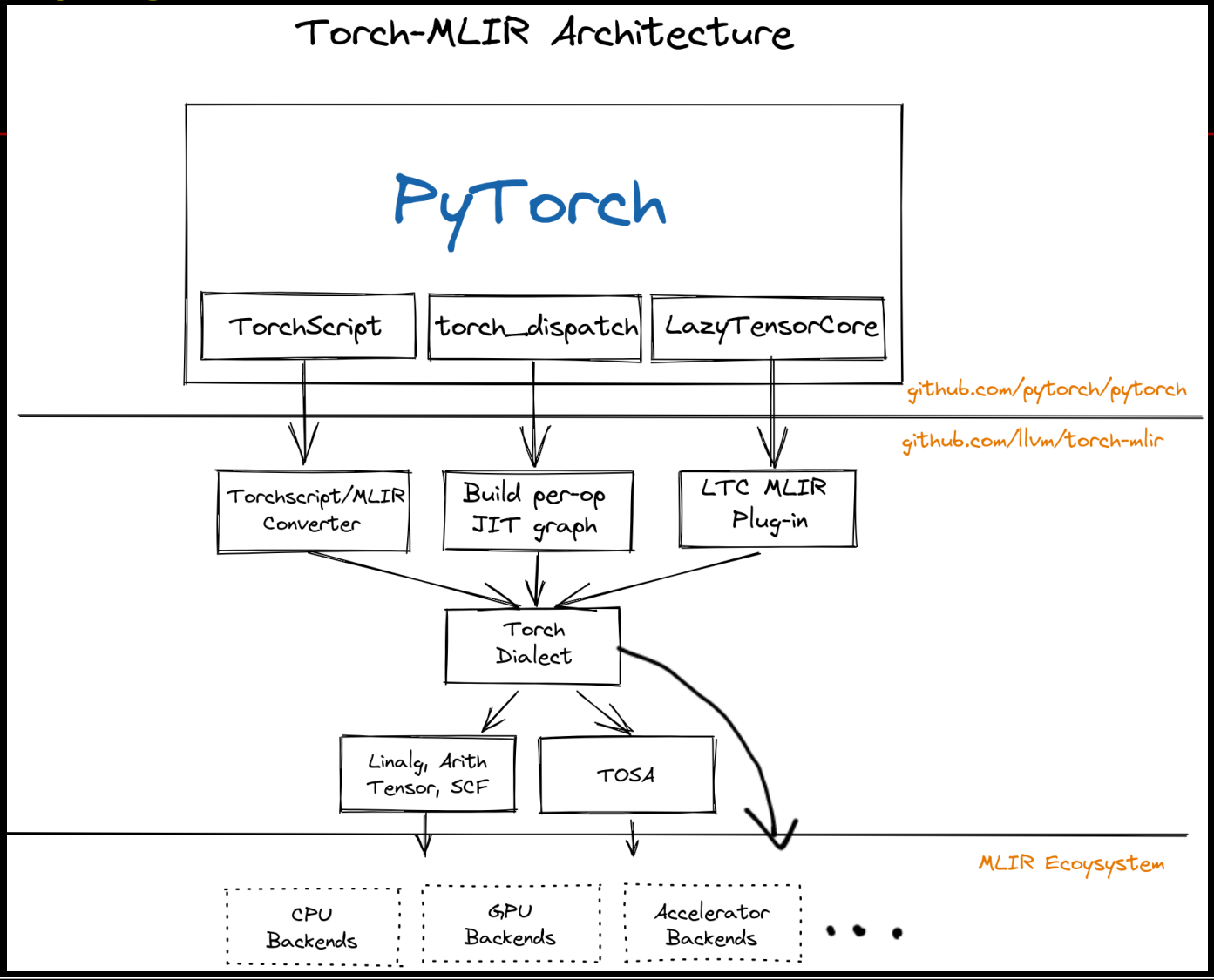
- **Frontends/Backends**



Source: https://mlir.llvm.org/OpenMeetings/2021-10-07-The-Torch-MLIR-project.pdf

Architecture & Design

- https://github.com/llvm/torch-mlir



II. eFPGA-oriented HW-SW Co-design

1) Core-V

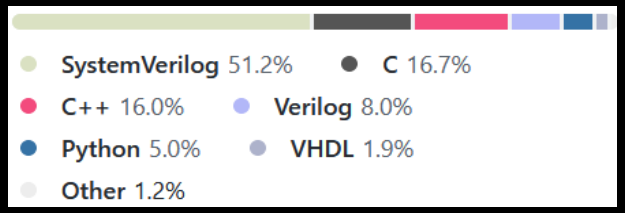
1.1 Overview

- https://github.com/openhwgroup/core-v-mcu

CORE-V MCU originated from PULPissimo [1], [2]. and is now a stand-alone project within OpenHW Group independent from PULPIssimo.

In case you should be interested to join the project please feel free to open an issue, or involve yourself in any open issues/discussions.

Languages



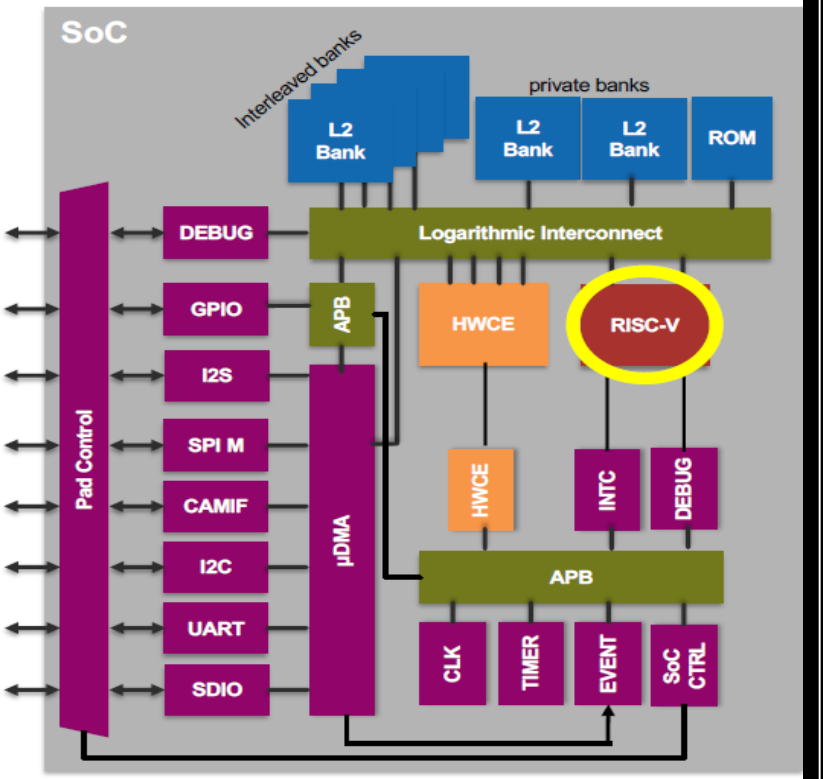
CV32E40P core

- https://github.com/openhwgroup/cv32e40p
  An in-order 4-stage RISC-V RV32IMFCXpulp CPU based on RI5CY from PULP-Platform
- https://cv32e40p.readthedocs.io/en/latest/
- …

PULPissimo Architecture

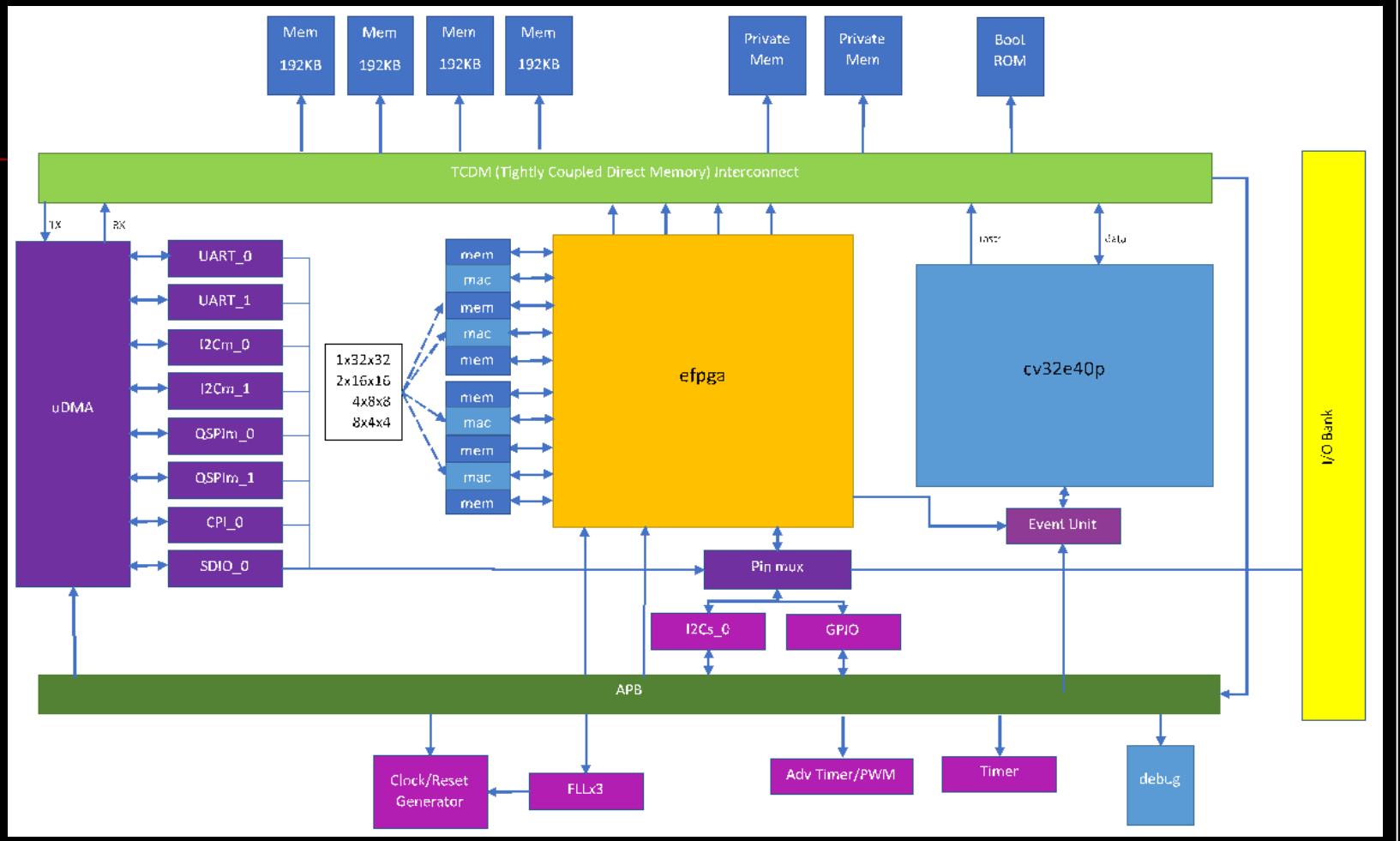
https://github.com/pulp-platform/pulpissimo

- RISC-V based advanced microcontroller
  - 512kB of L2 Memory
  - 16kB of energy efficient latch-based memory (L2 SCM BANK)
- Rich set of peripherals:
  - QSPI (up to 280 Mbps)
  - Camera Interface (up to 320x240@60fps)
  - I2C, I2S (up to 4 digital microphones)
  - JTAG (Debug), GPIOs,
  - Interrupt controller, Bootup ROM
- Autonomous IO DMA Subsystem (µDMA)
- Power management
  - 2 low-power FLLs (IO, SoC)



Source: https://pulp-platform.org/docs/riscv_workshop_zurich/schiavone_wosh2019_tutorial.pdf

Architecture & Design



Source: https://docs.openhwgroup.org/projects/core-v-mcu/doc-src/overview.html

Development

- …

Xilinx Vivado

Vivado is required if you plan to generate you own FPGA bitmaps or load them via the "shared UART/JTAG USB port" (see below). If you will be using the pre-built bitmaps and loading them from a USB drive then you will not need Vivado. The free-to-use Vivado WebPACK is sufficient for working with the Nexys A7 and can be downloaded from the Xilinx here.

Eclipse IDE

The open-source Eclipse IDE supports CORE-V-MCU.

Hardware Requirements:

- Digilent Nexys A7-100T evaulation board.
- USB to MicroUSB cable (typically supplied with the Nexys board).
- Digilent JTAG-H2 Pmod.
- Ashling Opella-LD (if not using Digilent HS2)
- 6-pin Header Gender Changer.
- USB drive.

Optional hardware includes:

- 5V Power supply. Note that the Nexys can typically be powered by the MicroUSB port.
- NOR Flash Pmod.

Source: https://github.com/openhwgroup/core-v-mcu/blob/master/emulation/quickstart/README.md

- **Renode**

https://antmicro.com/blog/2020/12/open-source-fpga-tools-and-renode-for-core-v/

…

1.2 SW
1.2.1 PULP-NN

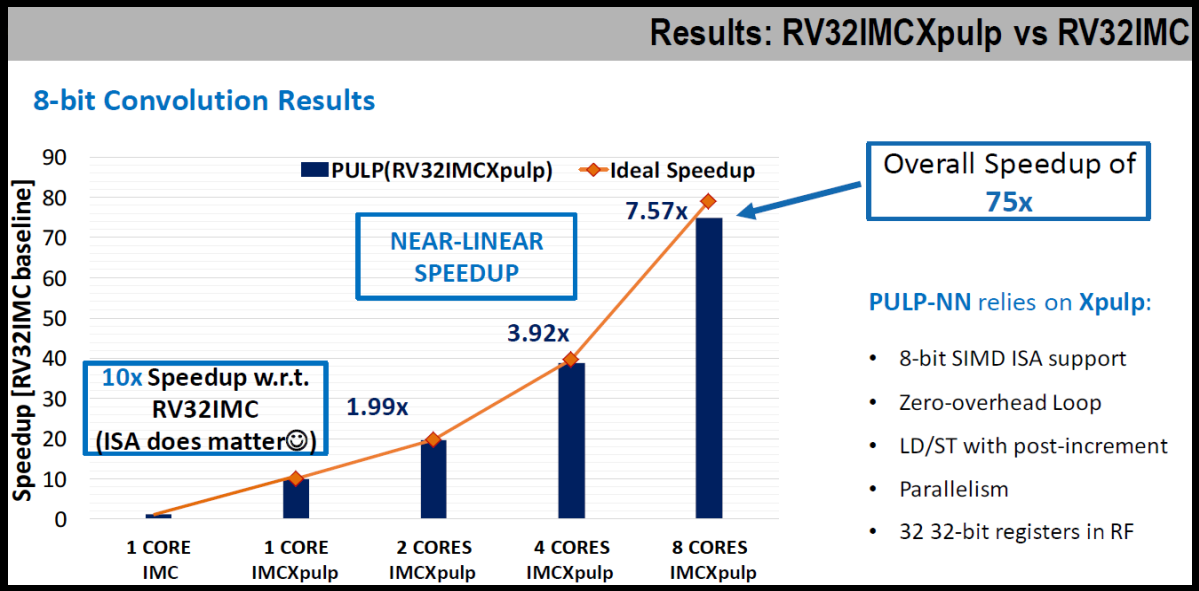
- https://github.com/pulp-platform/pulp-nn

Enabling QNN inference on PULP.

PULP_NN is a multicore computing library for QNN inference on Parallel-Ultra-Low-Power (PULP) Clusters of RISC-V based processors. It adopts the Height-Width-Channel (HWC) layout to store NN weights and activations and the implementation of the convolution-based kernels as a Matrix Multiplication operation, as proposed by ARM's CMSIS-NN open source library. It fully exploits the Xpulp ISA extension and the cluster's parallelism to achieve high performance and high energy efficiency on PULP-based devices.

Languages

C 100.0%



Source: https://pulp-platform.org/docs/riscv_workshop_zurich/RISCV_wokshop_AGarofalo_PULPNN.pdf

1.2.2 PULP-TrainLib

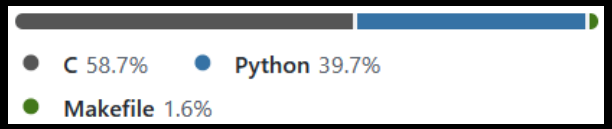
- https://github.com/pulp-platform/pulp-trainlib

PULP library for FP32, FP16.

PULP-TrainLib is the first Deep Neural Network training library for the PULP Platform. PULP-TrainLib features a wide set of performance-tunable DNN layer primitives for training, together with optimizers, losses and activation functions. To enable on-device training, PULP-TrainLib is equipped with AutoTuner, a pre-deployment tool to select the fastest configuration for each DNN layer, according to the training step to be performed and the shapes of the layer tensors. To facilitate the deployment of training tasks on the target PULP device, PULP-TrainLib is equipped with the TrainLib Deployer, a code generator capable of generating a project folder containing all the files and the code to run a DNN training task on PULP.

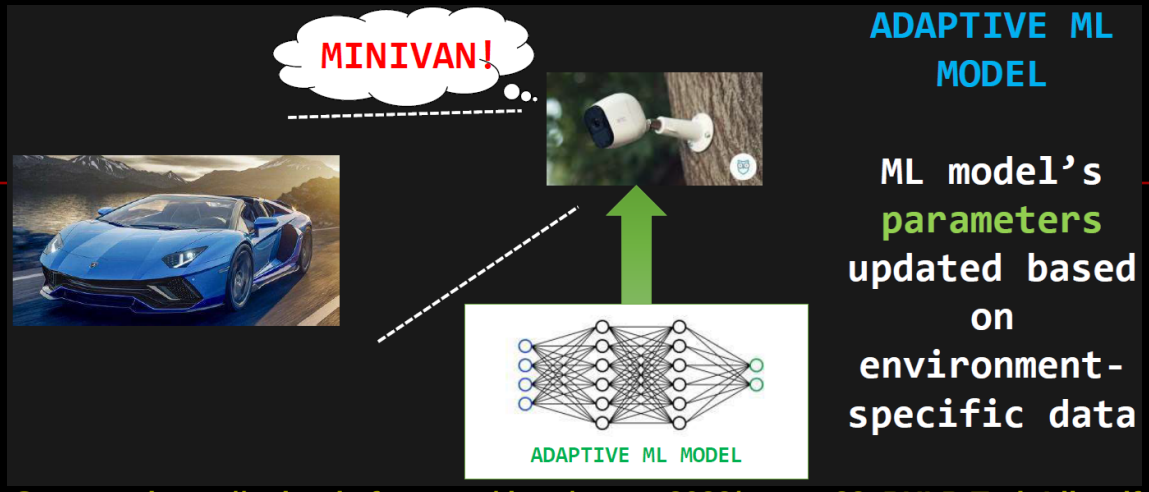
PULP-TrainLib: Enabling On-Device Training for RISC-V Multi-Core MCUs through Performance-Driven Autotuning.

- Src



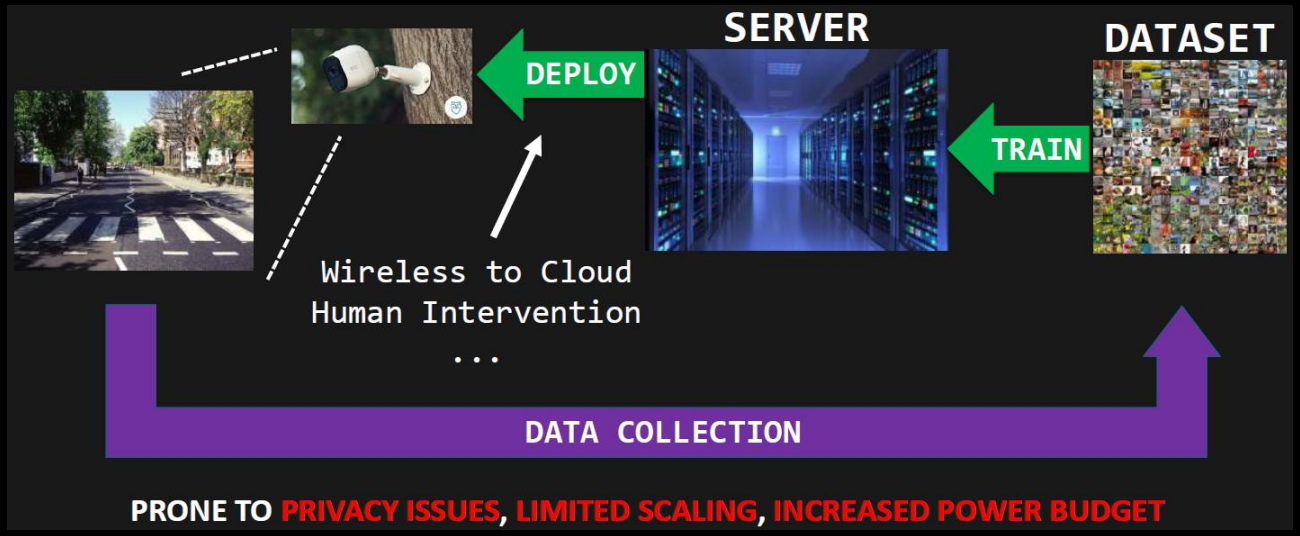

```
[mydev@fedora pulp-trainlib]$ tree -L 2 .
.
├── lib
│   ├── include
│   ├── README.md
│   └── sources
├── LICENSE
├── README.md
├── tests
│   ├── mm_manager_list_fp16.txt
│   ├── mm_manager_list.txt
│   ├── README.md
│   ├── test_act
│   ├── test_conv2d
│   ├── test_conv2d_fp16
│   ├── test_conv_pw_dw
│   ├── test_conv_pw_dw_fp16
│   ├── test_linear
│   ├── test_linear_fp16
│   ├── test_losses
│   ├── test_matmul
│   └── test_pooling
└── tools
    ├── AutoTuner
    ├── memory_footprint_tool
    ├── README.md
    └── TrainLib_Deployer
```

Motivation



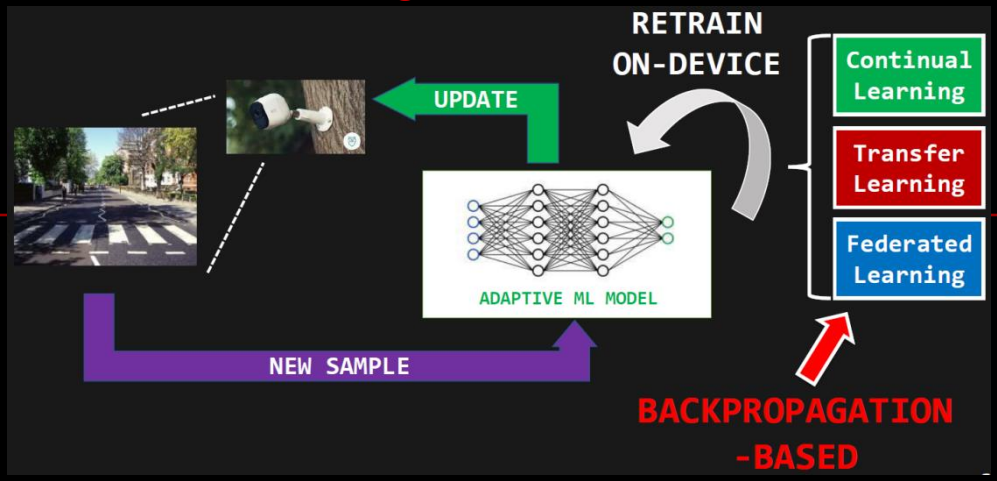
Source: https://pulp-platform.org/docs/samos2022/samos22_PULP-TrainLib.pdf

Standard DNN training



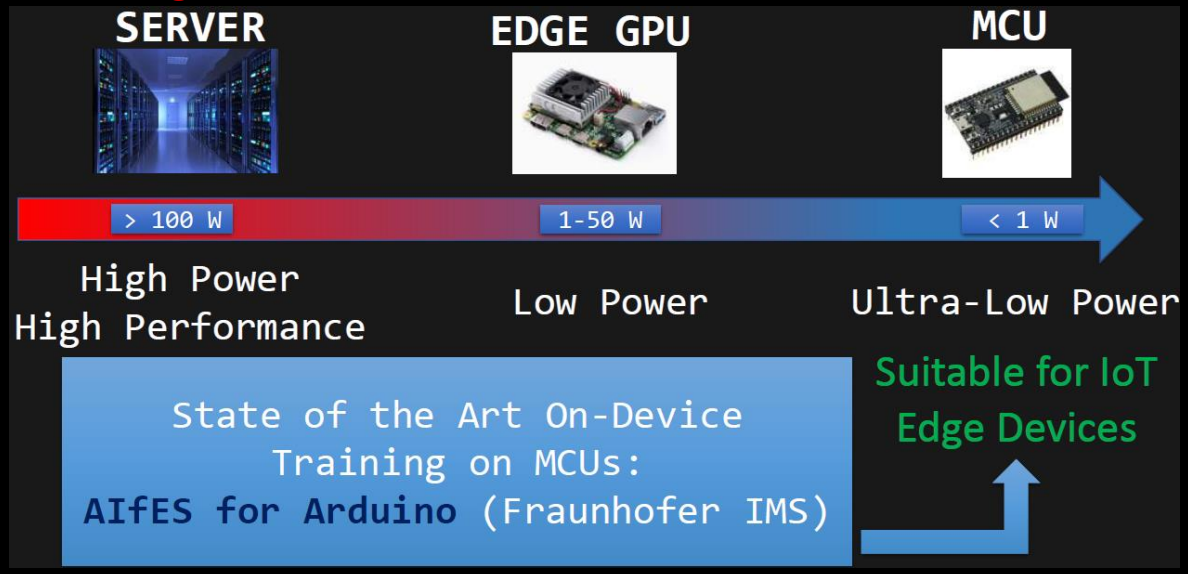
Source: https://pulp-platform.org/docs/samos2022/samos22_PULP-TrainLib.pdf

On-device training



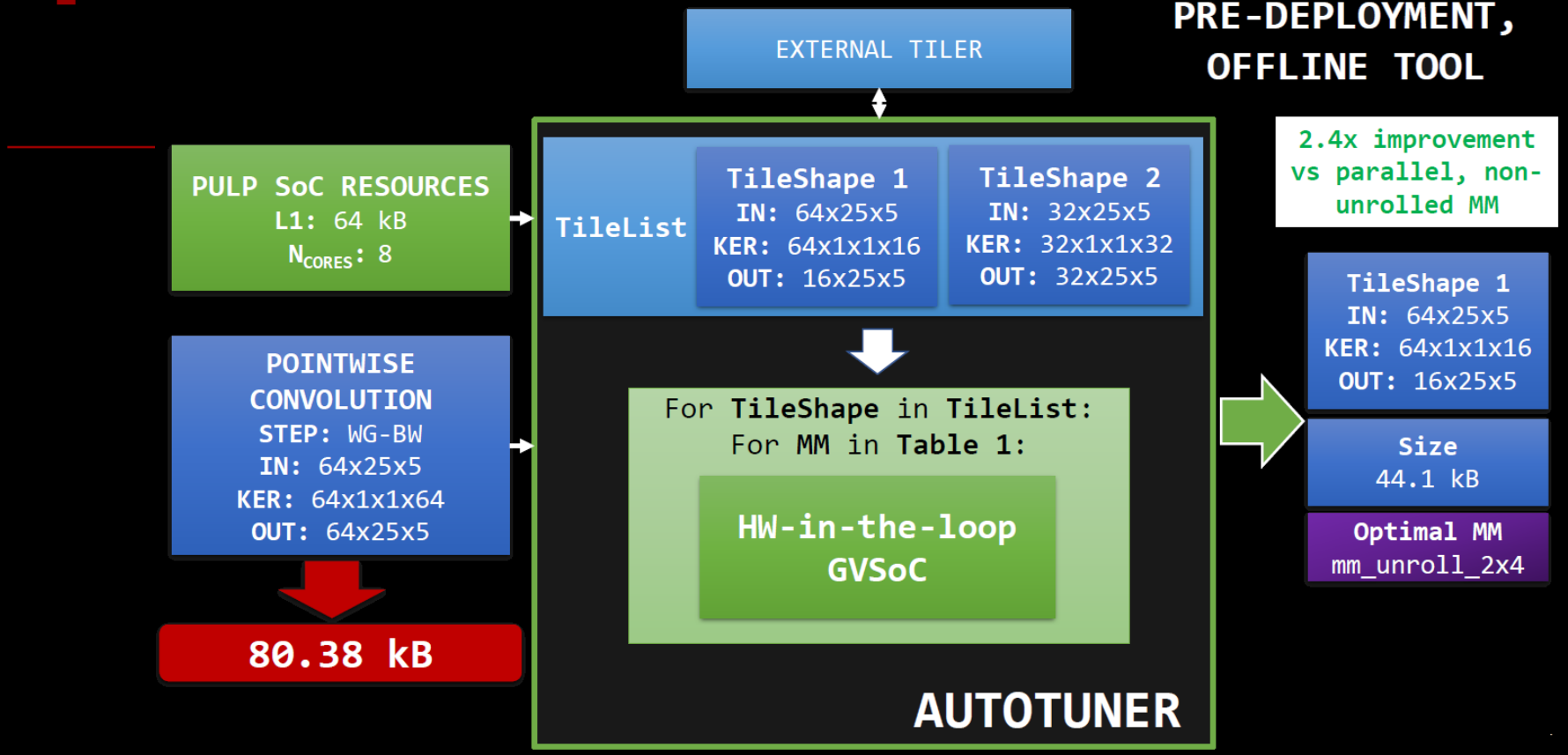
Source: https://pulp-platform.org/docs/samos2022/samos22_PULP-TrainLib.pdf

Training on MCUs



Source: https://pulp-platform.org/docs/samos2022/samos22_PULP-TrainLib.pdf

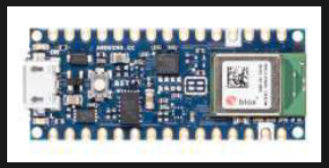
AutoTunner



Source: https://pulp-platform.org/docs/samos2022/samos22_PULP-TrainLib.pdf

Comparison with AIfES

AIfES: DNN on-device training/inference library for Arduino (with ARM MCUs)



Our target:
ARM CORTEX M4
vs
GVSoC

AIfES core:
ARM-CMSIS MM

PULP ISA
(2x wrt ARM)
+ AutoTuner

PARALLELISM
+ AutoTuner
(6.82x speedup wrt 1 RISC-V Core)

Source: https://pulp-platform.org/docs/samos2022/samos22_PULP-TrainLib.pdf

Summary

PULP-TrainLib

+

AutoTuner

The first **Open-Source SW Library for On-Device Training** on RISC-V Multicore MCUs

Up to **4.39 MAC/clock on 8 RISC-V cores** to execute FP32 DNN training primitives (with AutoTuner)

Almost real time training of TinyML models: 0.9 ms (Deep Autoencoder), 3 ms (DS-CNN) @450 MHz (PULP-based Vega SoC)

30.7x faster performance **wrt AIfES** on Cortex M4

Source: https://pulp-platform.org/docs/samos2022/samos22_PULP-TrainLib.pdf

2) AIRISC
2.1 Overview

- https://www.airisc.de
-

Trustworthy RISC-V Cores

The AIRISC family of RISC-V cores from Fraunhofer IMS enables efficient machine learning and AI in sensors, IoT devices and other embedded applications.
AIRISC variants with extended safety mechanisms such as dual core lock-stepping (DCLS) or error correcting codes (ECC) are available for use in fail-safe systems according to ISO 26262 (up to ASIL-D) and DIN 61508. Since intellectual property protection is highly relevant in these applications, the AIRISC family offers functions for cryptographic security, such as firmware protection and secure boot.

Designed for Embedded AI

In interaction with the AIfES software library invented by Fraunhofer IMS, the AIRISC processor family supports the inference and training of neural networks directly on the embedded device. Integrated accelerators enable distributed training in networked devices (federated learning) and also the calibration of sensors using AI. Image processing directly in the sensor, without the need for permanent communication with the cloud, makes the AIRISC processor interesting for energy-autonomous and wearable sensor systems.

Source: https://www.ims.fraunhofer.de/content/dam/ims/de/documents/Downloads/AIRISC%20-%20RISC-V%20CORESAIRISC%20CORES.pdf

- https://riscv.org/blog/2022/01/edge-ai-on-low-footprint-risc-v-alexander-stanitzki-fraunhofer-ims/

Features

Features

- 32-bit, 5-stage pipeline
- Highly modular: tune area, power and performance by enabling features in a single central configuration file
- E - reduced register set
- M - single cycle multiplier/divider
- C - compressed instructions
- F - single precision floating point unit (FPU)
- Branch prediction and Return Address Stack (RAS)
- Configurable hardware breakpoints/watchpoints
- Hardware support for AIfES embedded AI library

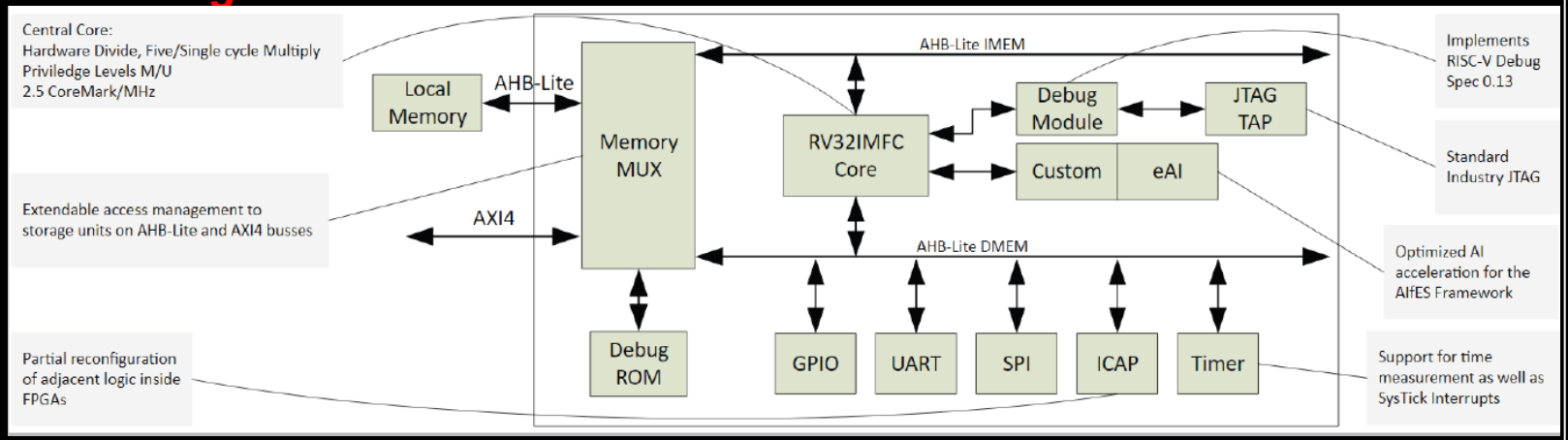
Modules/Peripherals

- AHB-Lite system interface
- GPIO
- UART
- SPI Master/Slave
- Timer
- JTAG
- Dynamic reconfiguration module (on Xilinx 7-series)

Deliverables

- RTL verilog sources
- Verification suite
- Synthesis and P&R scripts for Cadence Genus/Innovus
- Register and module description
- Integration guide
- Example projects for Diligent NexysVideo, CMOD A7 and Arty A7 boards
- Including Coremark, FreeRTOS-Demo, Peripheral Demo and embedded AI Demo

Block diagram

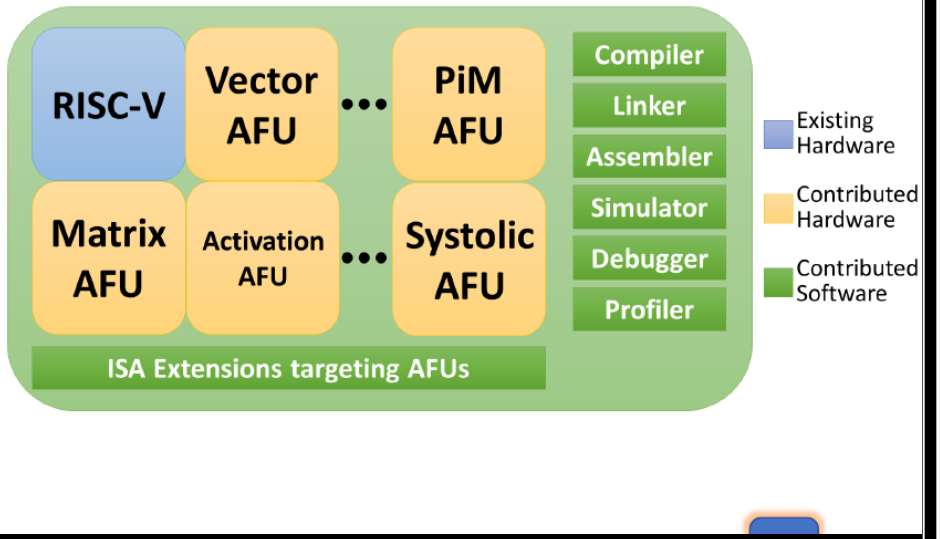


Source: https://www.ims.fraunhofer.de/content/dam/ims/de/documents/Downloads/AIRISC%20-%20RISC-V%20CORESAIRISC%20CORES.pdf

AFU

Tightly integrated AI Functional Units

- Custom RISC-V processor with ISA extensions targeting AI applications
- Tightly integrated AI accelerators for fine-grained offloading of AI tasks
- End-to-end hardware/software co-design solution
- Support for AI and non-AI applications on the same processor



Source: https://www.tvmcon.org/events/leveraging-tvm-as-a-front-end-compiler-for-risc-v-based-custom-tinyml-processor/

Hardware/Software Co-design for AI-RISC

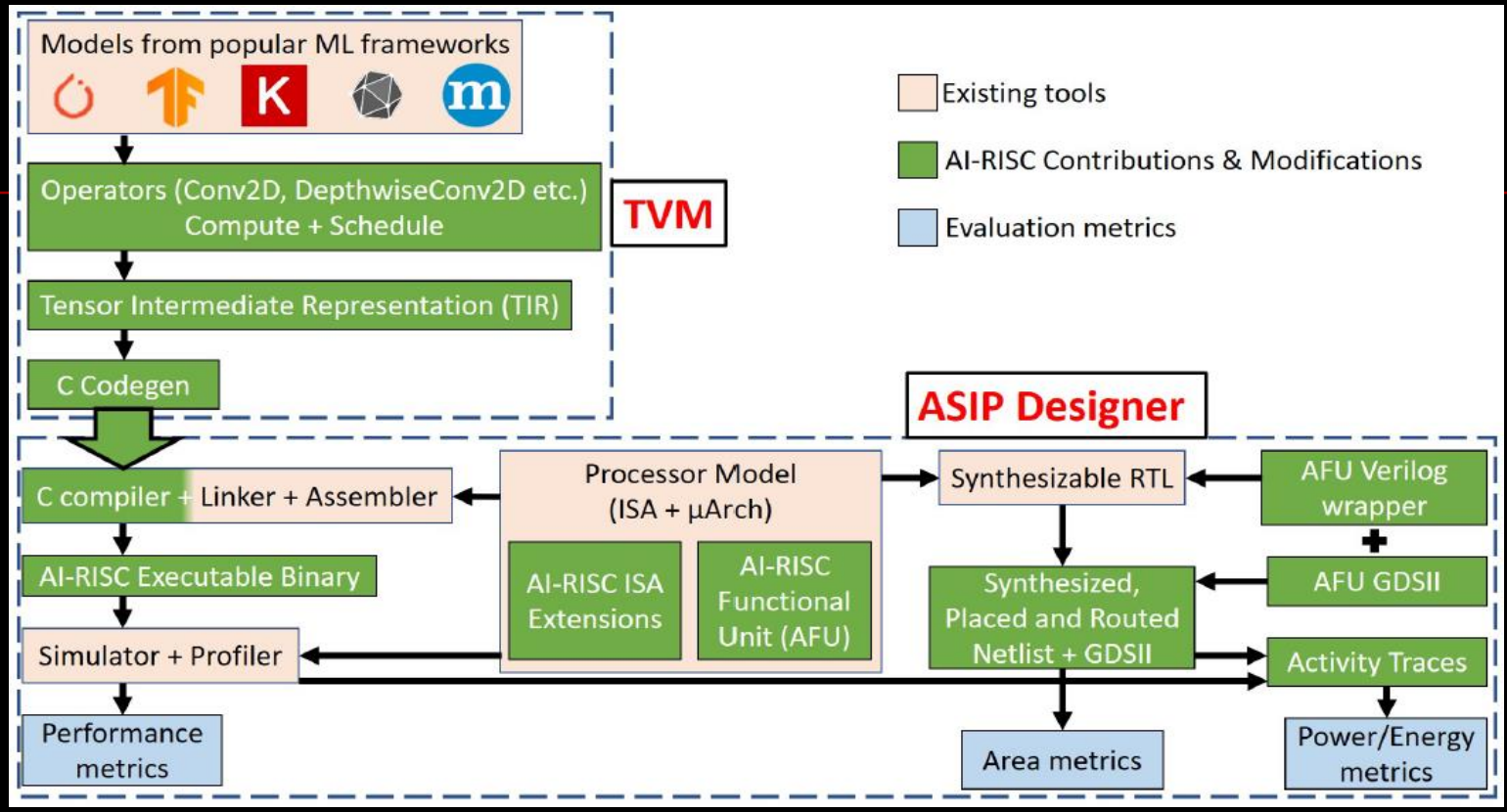


Figure 5.1: AI-RISC hardware, ISA and software co-design methodology consisting of TVM deep learning compiler [3] as the front-end compiler to map models from popular machine learning frameworks to C and Synopsys ASIP Designer [4] as the back-end processor design tool to generate complete SDK and synthesizable RTL. AI-RISC contributions and modifications to existing tools to support AI functional units across the hardware and software stacks have been highlighted in green.

Source: "AI-RISC: Scalable RISC-V Processor for IoT Edge AI Applications", Vaibhav Verma, PhD thesis 2022.

TVM generated C source code with and without custom instruction compiler intrinsics

```
    for (int32_t k_outer = 0; k_outer < 16; ++k_outer) {
      for (int32_t y_inner = 0; y_inner < 8; ++y_inner) {
        for (int32_t k_inner = 0; k_inner < 8; ++k_inner) {
          ((int32_t*)compute1)[(((z_outer_y_outer_fused * 8) + y_inner))] = (((int32_t*)compute1)[(((z_outer_y_outer_fused * 8) + y_inner)
)] + (((int32_t)((int8_t*)placeholder)[(((k_outer * 8) + k_inner))]) * ((int32_t)((int8_t*)placeholder1)[((((z_outer_y_outer_fused * 1024
) + (y_inner * 128)) + (k_outer * 8)) + k_inner))]));
        }
      }
    }
  }
```

Without Custom instructions

```
    for (int32_t k_outer = 0; k_outer < 16; ++k_outer) {
      (void)gemm_1x8x8_update_ISQWNLHM(((int8_t *)placeholder + ((k_outer * 8))), ((int8_t *)placeholder1 + (((z_outer_y_outer_fused * 102
4) + (k_outer * 8)))), ((int32_t *)compute1 + ((z_outer_y_outer_fused * 8))), 8, 128, 8);
    }
  }
```

```
void gemm_1x8x8_update_ISQWNLHM(
    int8_t *aa, int8_t *bb, int32_t *cc,
    int A_stride, int B_stride, int C_stride) {

  for (int i = 0; i < 8; i++) {
    PIM_mem_store((char*)i,*((long*)(bb+i*B_stride)));
  }
  chess_memory_fence();

  long out0 = PIM_mac_0(*((long*) (aa)));
  long chess_storage(x13) out1 = PIM_mac_1(*((long*) (aa)));
  long chess_storage(x14) out2 = PIM_mac_2(*((long*) (aa)));
  long out3 = PIM_mac_3(*((long*) (aa)));
  long out[4] = { out0, out1, out2, out3};

  for (int i = 0; i < 8; i++) {
    cc[i] += *((int32_t*)(&out) + i);
  }
}
```

With Custom instructions

Fig. 5.3 shows an example of the generated C code from TVM for consumption by the back-end compiler. It shows that starting from the same source code high-level description in TensorFlow Lite, AI-RISC software can generate code for either a standard RISC-V processor without the compiler intrinsics or for AI-RISC processor with the supported compiler intrinsics. This provides great flexibility in porting existing AI applications and their source code to AI-RISC.

Source: "AI-RISC: Scalable RISC-V Processor for IoT Edge AI Applications", Vaibhav Verma, PhD thesis 2022.

Performance

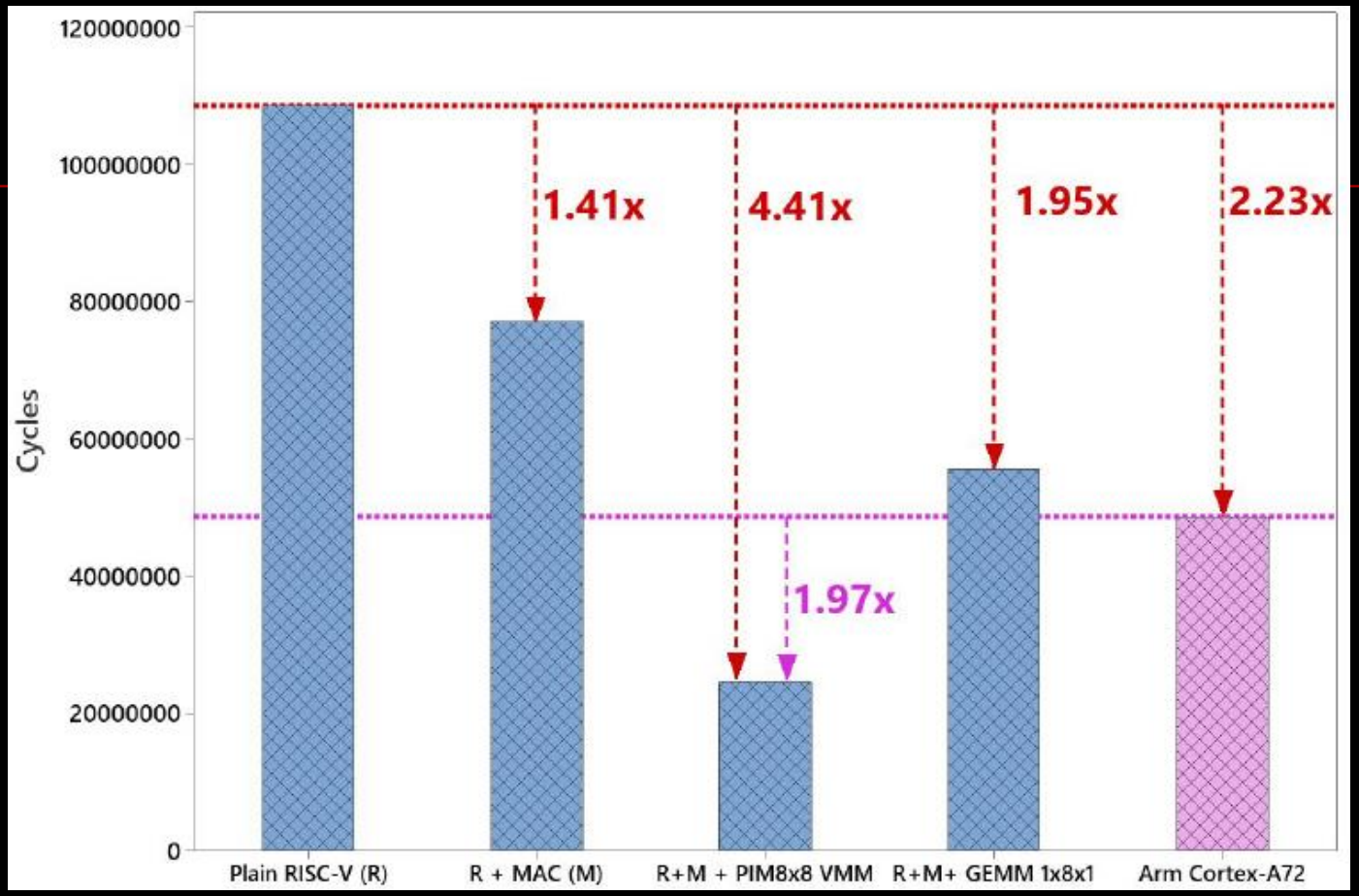


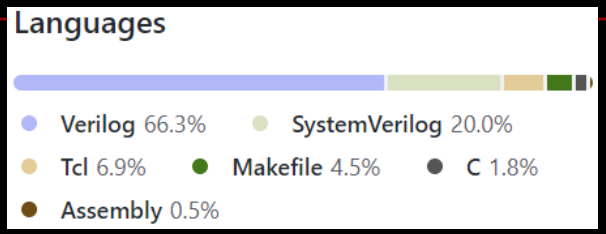
Figure 6.4: Performance improvement on ResNet-8 neural network model from MLPerf Tiny benchmark using different AI-RISC AFUs and custom instructions compared to RV64IMC RISC-V and Arm Cortex-A72 baseline.

Source: "AI-RISC: Scalable RISC-V Processor for IoT Edge AI Applications", Vaibhav Verma, PhD thesis 2022.

Open Source Version

- https://github.com/Fraunhofer-IMS/airisc_core_complex

  Fraunhofer IMS processor core. RISC-V ISA (RV32IM) with additional peripherals. FPGA implementation for Digilent Nexys Video.

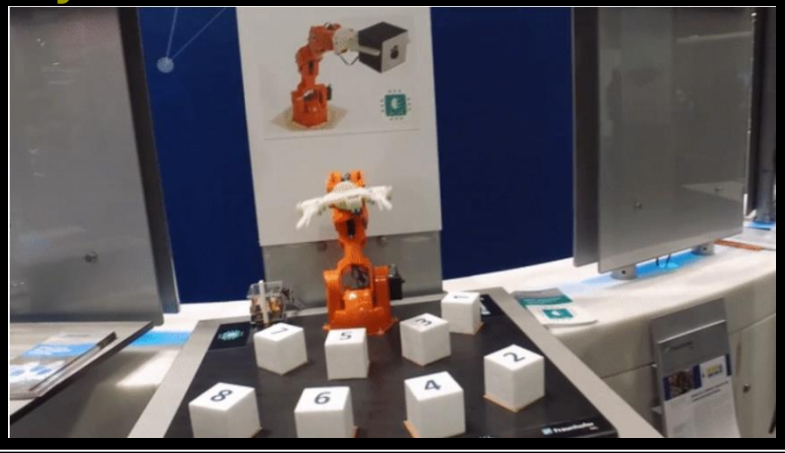


2.2 AIfES

- http://www.aifes.ai/
- https://github.com/Fraunhofer-IMS/AIfES_for_Arduino

AIfES (Artificial Intelligence for Embedded Systems) is a platform-independent and standalone AI software framework optimized for embedded systems. The Feedforward Neural Networks (FNN) implemented in AIfES can be freely parameterized, trained, modified or reloaded at runtime. In this version, it is optimized for the Arduino IDE and compatible to almost any Arduino board. AIfES is developed in the C programming language and uses only standard libraries based on the GNU Compiler Collection (GCC). AIfES thus runs on almost any hardware from 8-bit microcontrollers over Raspberry PI to smartphones or PCs. Not only inference of FNN is possible, but also training directly in the device. Furthermore, compatibility to other AI software frameworks such as Keras or TensorFlow is also given.

- https://www.ims.fraunhofer.de/en/Newsroom/Press-releases/2021/The_Release_of_AIFES.html
- https://www.seeedstudio.com/blog/2021/07/08/introducing-aifes-a-tinyml-framework-for-machine-learning-on-arduino-microcontrollers/



- **https://www.ims.fraunhofer.de/en/Business-Unit/Industry/Industrial-AI/Artificial-Intelligence-for-Embedded-Systems-AIfES/AIFES_Technical_Details.html**

Resource-saving programming

AIfES functions explicitly work with pointer arithmetic and only declare the most necessary variables in a function. This means, the storage areas for the training data and the weights are provided by the main program. AIfES functions access these memory areas by passing a pointer without requiring large resources themselves.

Platform independent and compatible

Due to the compatible programming with the GCC a porting to almost all platforms is possible. This enables the completely self-sufficient integration including the learning algorithm on an embedded system.

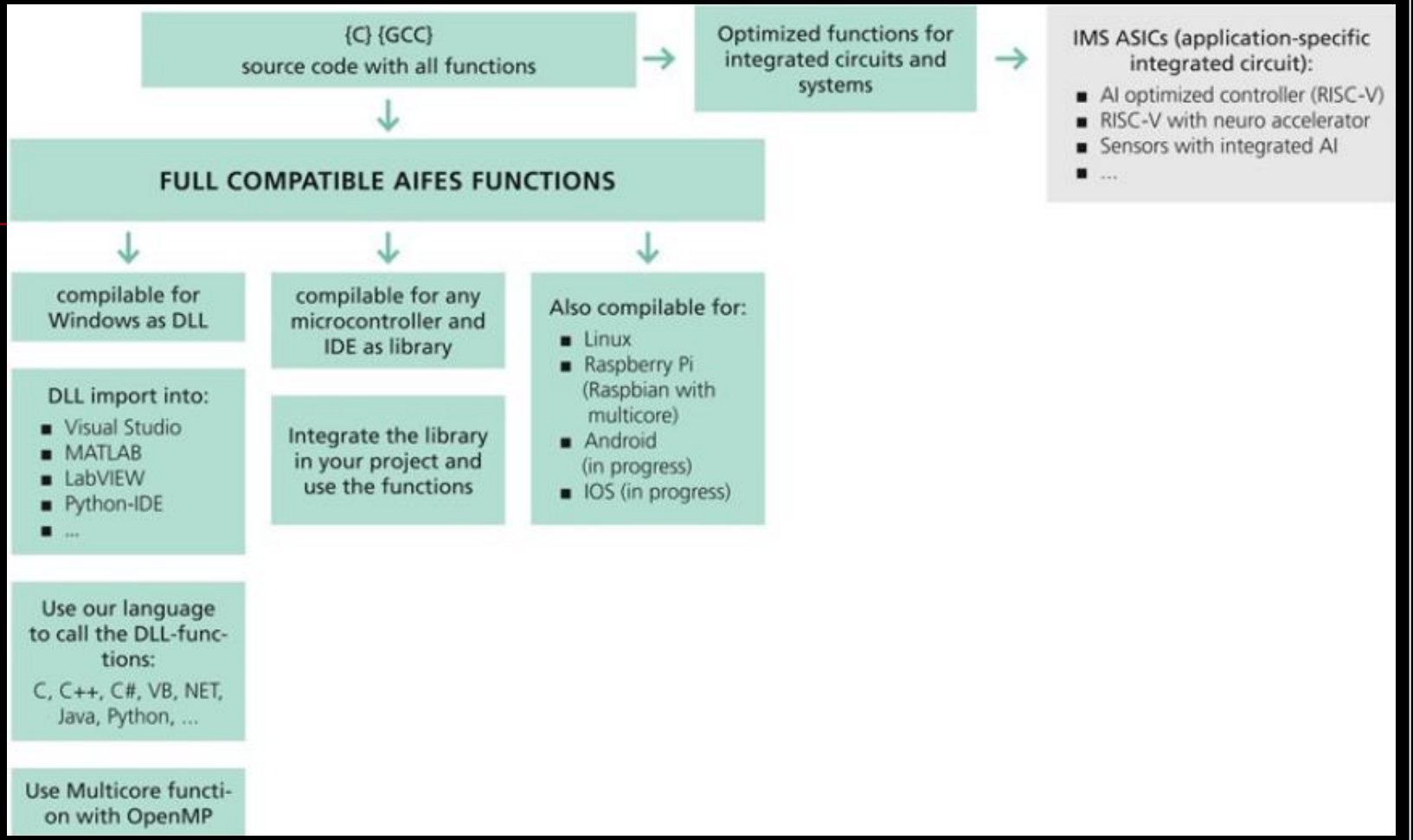
For use under Windows, for example, the source code can be compiled as a »Dynamic Link Library« (DLL) so that it can be integrated into software tools such as LabVIEW or MATLAB. Especially the direct connection to MATLAB is helpful to test e.g. different data preprocessings.

The integration in different software development environments like Visual Studio or a Python-IDE is also possible. The main program, which binds the DLL can therefore also be in a different programming language such as C++, C#, Python, VB.NET, Java.

For the first development of the individual FNN the Computer as a platform is a suitable choice to perform fast calculations. After the right configuration is completed the porting to the embedded system can be conducted.

A small selection of platforms and microcontrollers AIfES was already tested with:

- Windows (DLL)
- Raspberry Pi with Raspbian
- Arduino UNO
- Arduino Nano 33 BLE Sense
- Arduino Portenta H7
- ATMega32U4
- STM32 F4 Series (ARM Cortex-M4)


{C} {GCC}
source code with all functions
Optimized functions for integrated circuits and systems
IMS ASICs (application-specific integrated circuit):
AI optimized controller (RISC-V)
RISC-V with neuro accelerator
Sensors with integrated AI
...
FULL COMPATIBLE AIFES FUNCTIONS
compilable for Windows as DLL
compilable for any microcontroller and IDE as library
Also compilable for:
Linux
Raspberry Pi (Raspbian with multicore)
Android (in progress)
IOS (in progress)
DLL import into:
Visual Studio
MATLAB
LabVIEW
Python-IDE
...
Integrate the library in your project and use the functions
Use our language to call the DLL-functions:
C, C++, C#, VB, NET, Java, Python, ...
Use Multicore function with OpenMP

Are hardware accelerators supported?

Arm®

The Arm® CMSIS DSP library can be included in AIfES

- Hardware acceleration is possible for all Arm® controllers that support CMSIS DSP
- AIfES is a partner in Arm's AI Ecosystem

AIRISC by Fraunhofer IMS (link)

AIRISC: RV32IMEFC implementation – about 2,7 Coremark/MHz

Extensions for AI (specialized AIfES support)

Functional safety (Lockstep, ECC etc.) incl. ISO 26262 ASIL-D ready certification

Crypto functions for information security

Source: https://cms.tinyml.org/wp-content/uploads/talks2022/tinyML_Talks_Pierre_Gembaczka_211201.pdf

Function overview

Feedforward neural network inference

Float

Freely configurable (inputs, hidden layer, outputs)

Many activation functions

- Sigmoid, softsign, linear, RELU, Leaky RELU, softmax, tanh, ELU

Feedforward neural network Training

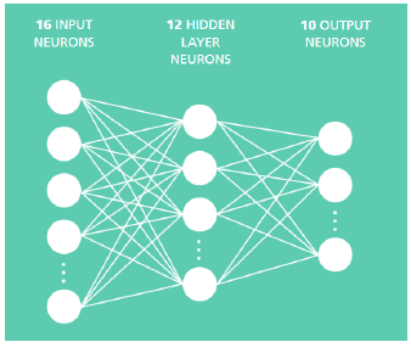
Full SGD and ADAM algorithm

Training types

- Online, Batch, Minibatch

Various loss functions

- mean squared error (MSE), cross-entropy



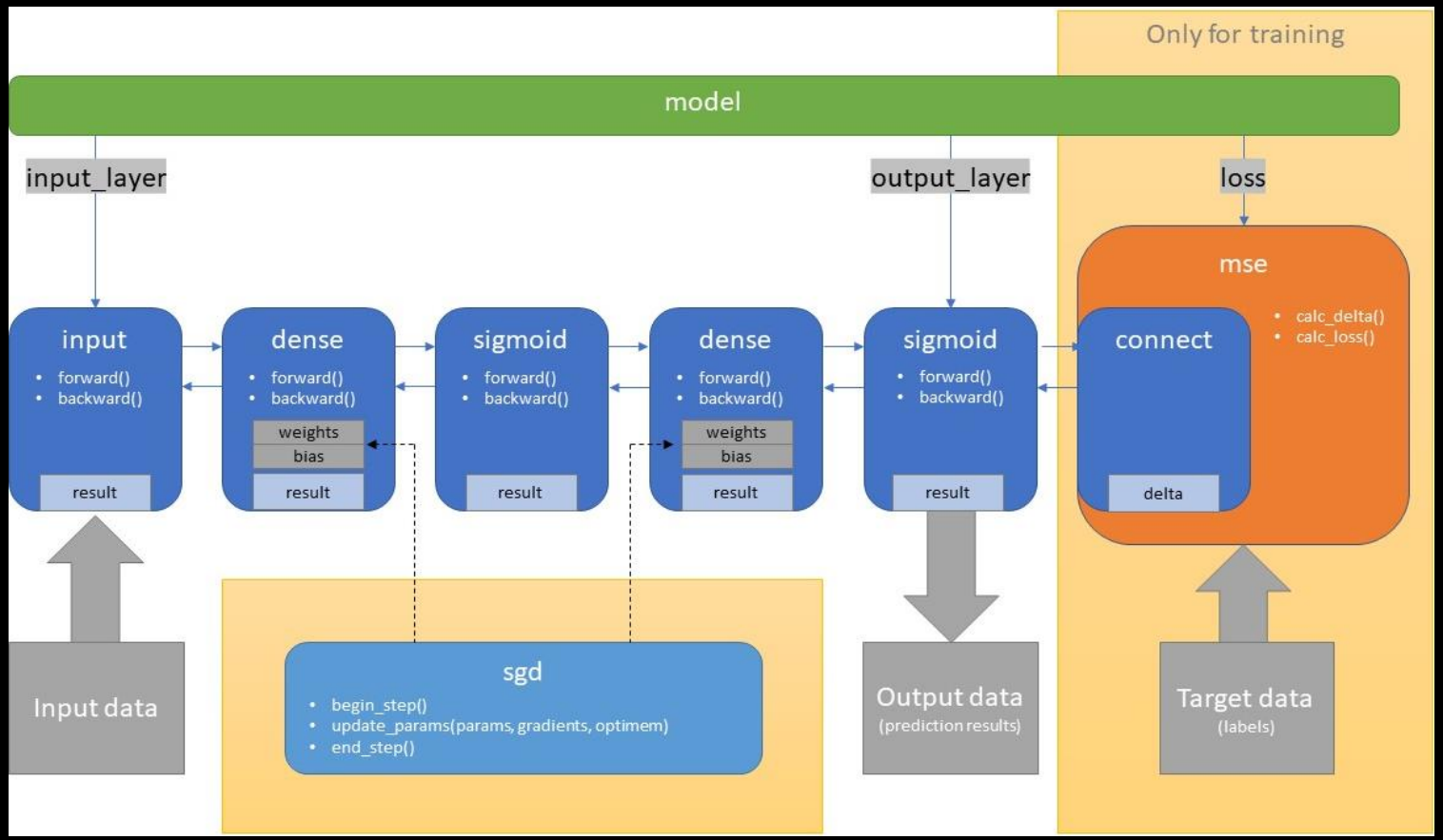
Source: https://cms.tinyml.org/wp-content/uploads/talks2022/tinyML_Talks_Pierre_Gembaczka_211201.pdf

Structure and design concepts

- https://fraunhofer-ims.github.io/AIfES_for_Arduino/index.html#autotoc_md1

AIfES was designed as a flexible and extendable toolbox for running and training or artificial neural networks on microcontrollers. All layers, losses and optimizers are modular and can be optimized for different data types and hardware platforms.

Example structure of a small FNN with one hidden layer in AIfES 2:



What's next

- **A little out-of-date:**

AIfES update in the next weeks

AIfES-Express API

- Simplified API that is directly integrated
- Inference and training with one function call

Fixpoint calculation with quantization of weights

- Automated Q7 quantization

Storage of weights in flash memory

Of course there are also new examples

Currently in development

CNN / ConvNets

Reinforcement learning

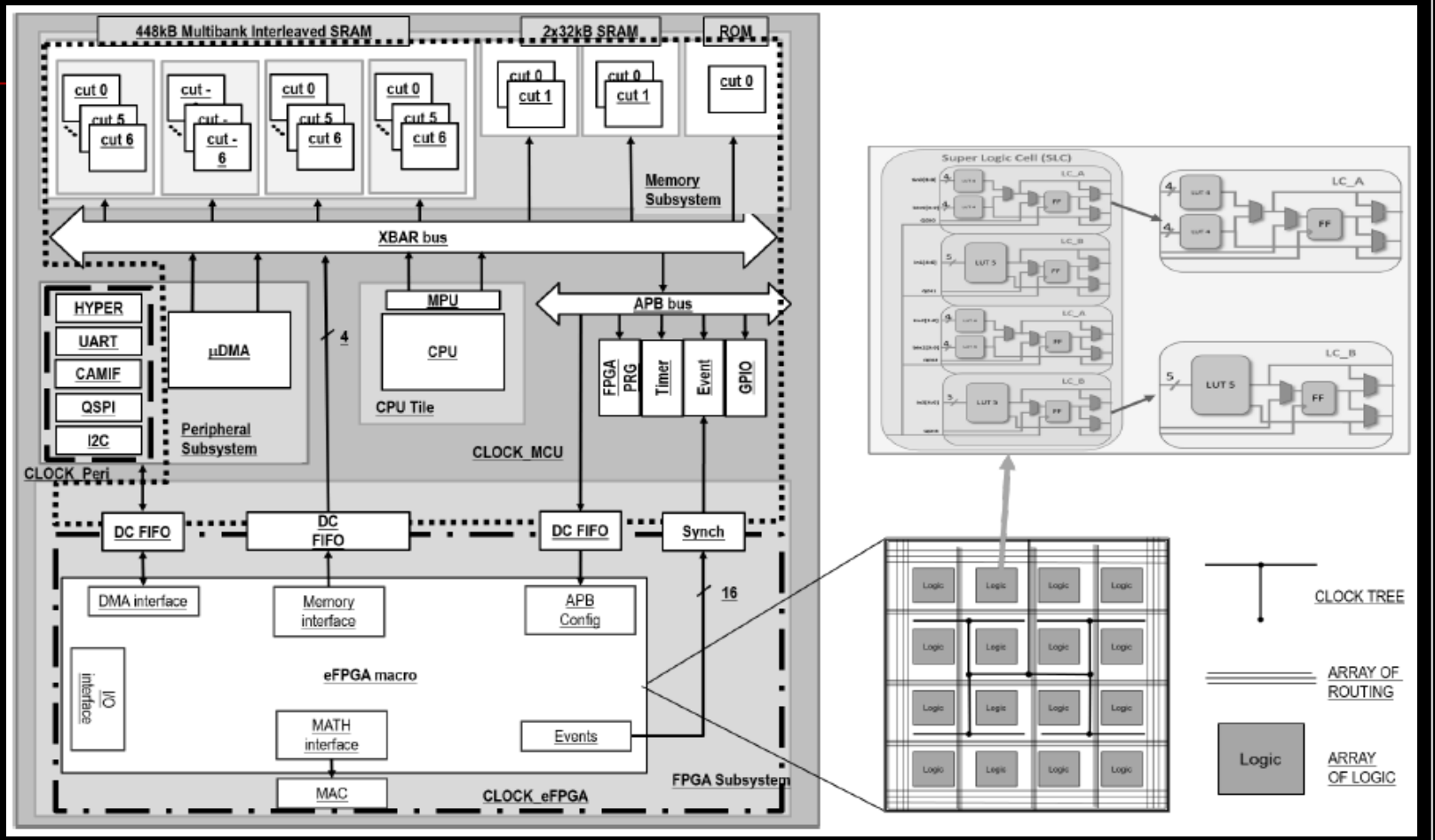
Source: https://cms.tinyml.org/wp-content/uploads/talks2022/tinyML_Talks_Pierre_Gembaczka_211201.pdf

3) Miscs

- https://www.edn.com/using-efpga-core-for-cpu-isa-extension-reconfigurability/
- …

3.1 Arnold

- An eFPGA-Augmented RISC-V SoC for Flexible and Low-Power IoT End-Nodes



Source: https://arxiv.org/pdf/2006.14256.pdf

4) eFPGA Framework
4.1 FABulous

- https://github.com/FPGA-Research-Manchester/FABulous

FABulous is designed to fulfill the objectives of ease of use, maximum portability to different process nodes, good control for customization, and delivering good area, power, and performance characteristics of the generated FPGA fabrics. The framework provides templates for logic, arithmetic, memory, and I/O blocks that can be easily stitched together, whilst enabling users to add their own fully customized blocks and primitives.

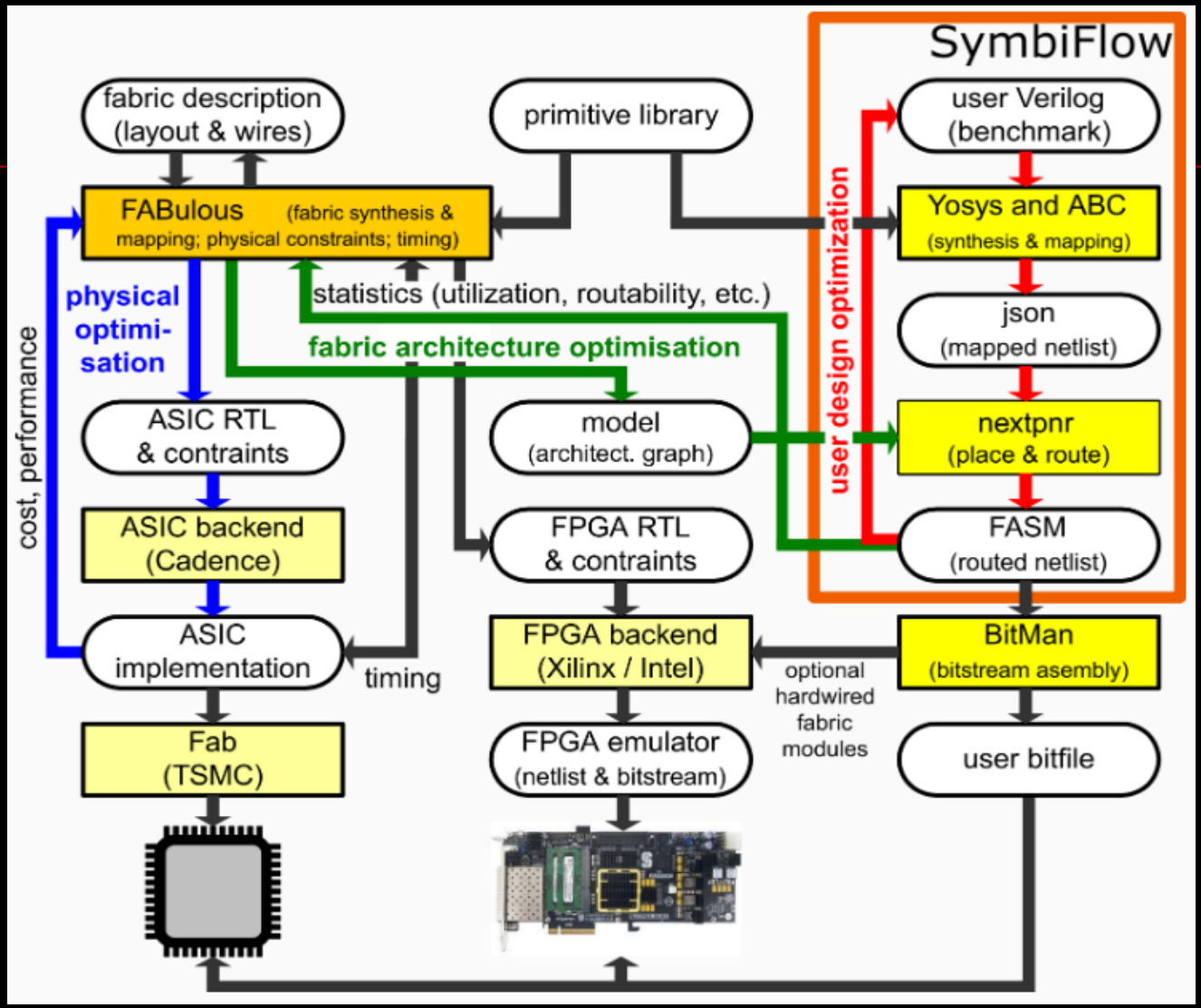
The FABulous ecosystem generates the embedded FPGA fabric for chip fabrication, integrates SymbiFlow toolchain release packages, deals with the bitstream generation and provides after-fabrication tests. Additionally, we plan to provide an emulation path for system development.

- **Wrap-up**

- Heterogeneous (FPGA) fabric (DSBs, BRAMs, CPUs, custom blocks)
  - Multiple tiles can be combined for integrating more complex blocks
  - Custom blocks can be instantiated directly in Verilog and are integrated in Yosys, VPR/nextpnr CAD tools (Synthesis, Place&Route) (as primitive blocks)
- Support for dynamic partial reconfiguration (some elements of XC6200, like wildcard configuration)
- Configuration through shift registers or latches (or custom cells)
- Support for custom cell primitives (passgate multiplexers)
- Good performance / area / power figures (about 1.3x worse than Xilinx) (could be narrowed down through customization)
- Usable by FPGA users (you don't have to be an FPGA architect)
  → there are FPGA classics that we have/will clone
- ToDo: multiple clock domains, mixed-grained granularity, more ReRAM, processes, ...

Source: https://invasic.informatik.uni-erlangen.de/publications/DirkKoch2022.pdf

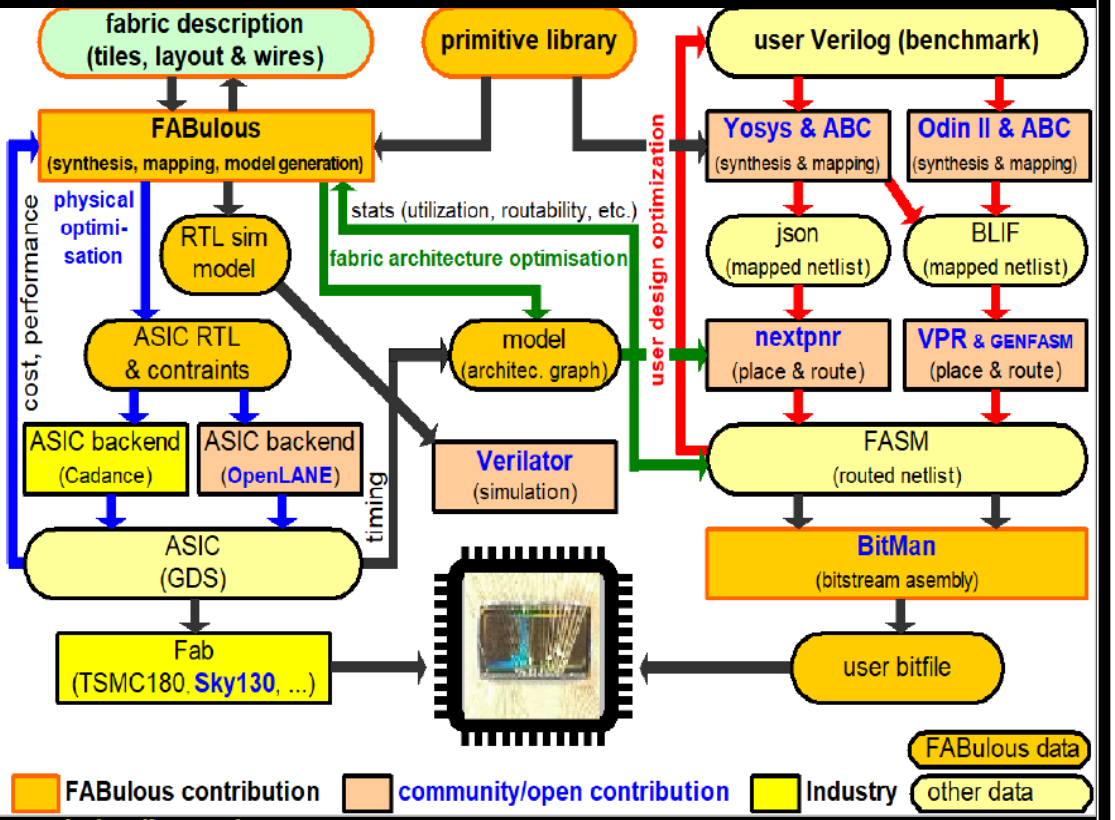
Workflows and dependencies



Source: https://fabulous.readthedocs.io/en/latest/

- FABulous eFPGA generator
  - ASIC RTL and constraints generation
  - Generating models for nextpnpr/VPR flows
  - FPGA emulation
- Virtex-II, Lattice clones (patent-free!)

- See our FPGA 2021 paper „FABulous: An Embedded FPGA Framework"



Source: https://fabulous.readthedocs.io/en/latest/

III. Project CFU-Playground

1) Overview

- https://cfu-playground.readthedocs.io/en/latest/

Accelerate ML models on FPGAs.

"CFU" stands for Custom Function Unit accelerator hardware that is tightly coupled into the pipeline of a CPU core, to add new custom function instructions that complement the CPU's standard functions (such as arithmetic/logic operations).

The CFU Playground is a collection of software, gateware and hardware configured to make it easy to:

- Run ML models
- Benchmark and profile performance
- Make incremental improvements
  - In software by modifying source code
  - In gateware with a CFU
- Measure the results of changes

ML acceleration on microcontroller-class hardware is a new area, and one that, due to the expense of building ASICs, is currently dominated by hardware engineers. In order to encourage software engineers to join in the innovation, the CFU-Playground aims to make experimentation as simple, fast and fun as possible.

As well as being a useful tool for accelerating ML inferencing, the CFU Playground is a relatively gentle introduction to using FPGAs for computation.

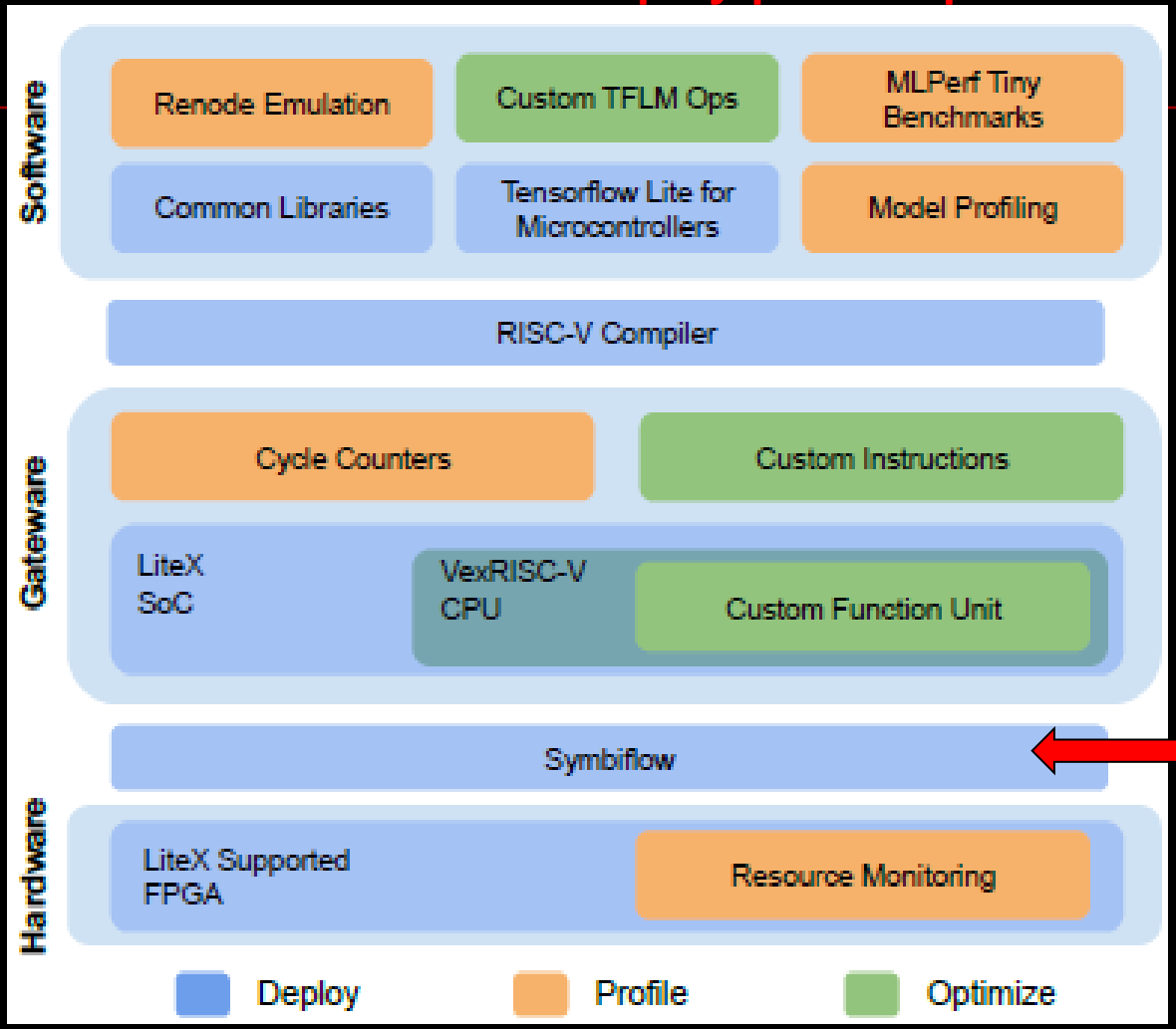
If you find that you need help or that anything is not working as you expect, please raise an issue and we'll do our best to point you in the right direction.

Disclaimer: This is not an officially supported Google project. Support and/or new releases may be limited.

- A new kind of ASIP/DSA?

Architecture & Design

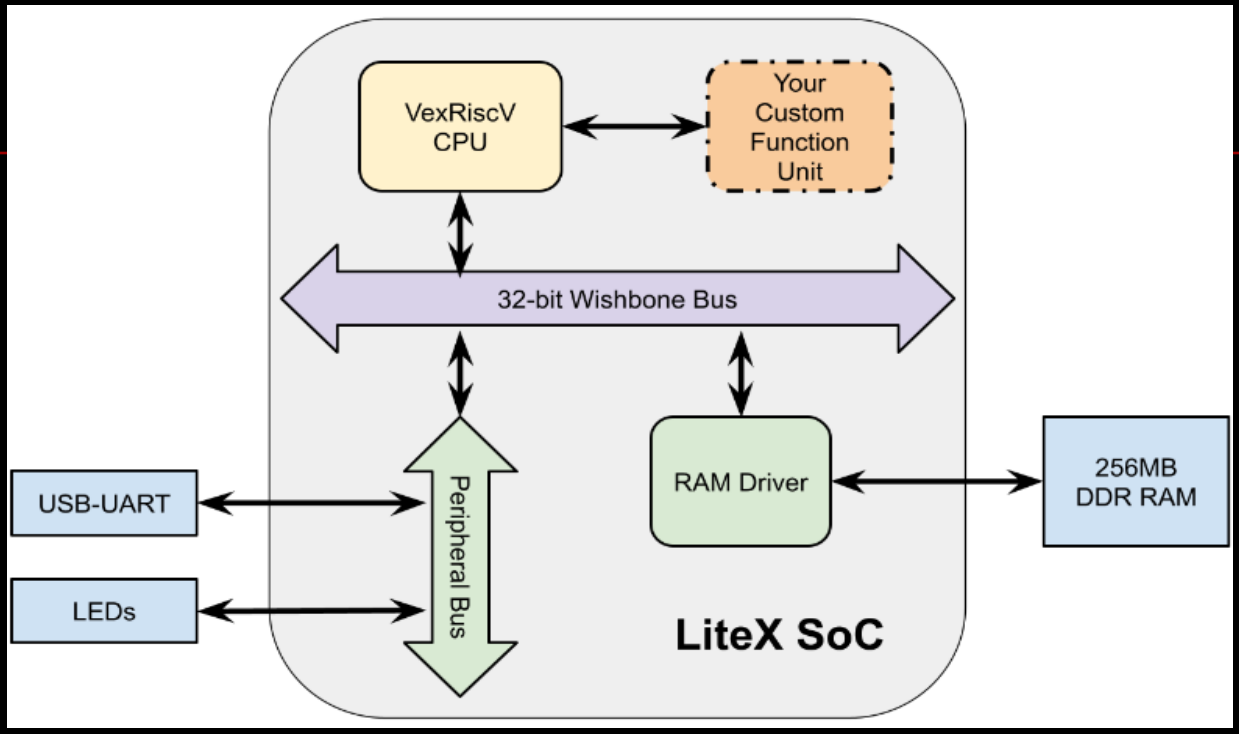
- Allows users to design and evaluate model-specific ML enhancements to a "soft" CPU core with deploy-profile-optimization feedback loop



Source: https://www.researchgate.net/publication/357647725_CFU_Playground_Full-Stack_Open-Source_Framework_for_Tiny_Machine_Learning_tinyML_Acceleration_on_FPGAs

- **Gateware**

Use LiteX to build a standard SoC that runs on the FPGA.



The CPU has an op code allocated to the CFU. When the CPU executes that op code, it passes the contents of two registers to the CFU, waits for a response and puts the result back into a third register. A notable feature of this architecture is that the CFU does not have direct access to memory. It relies on the CPU to move data back and forth.

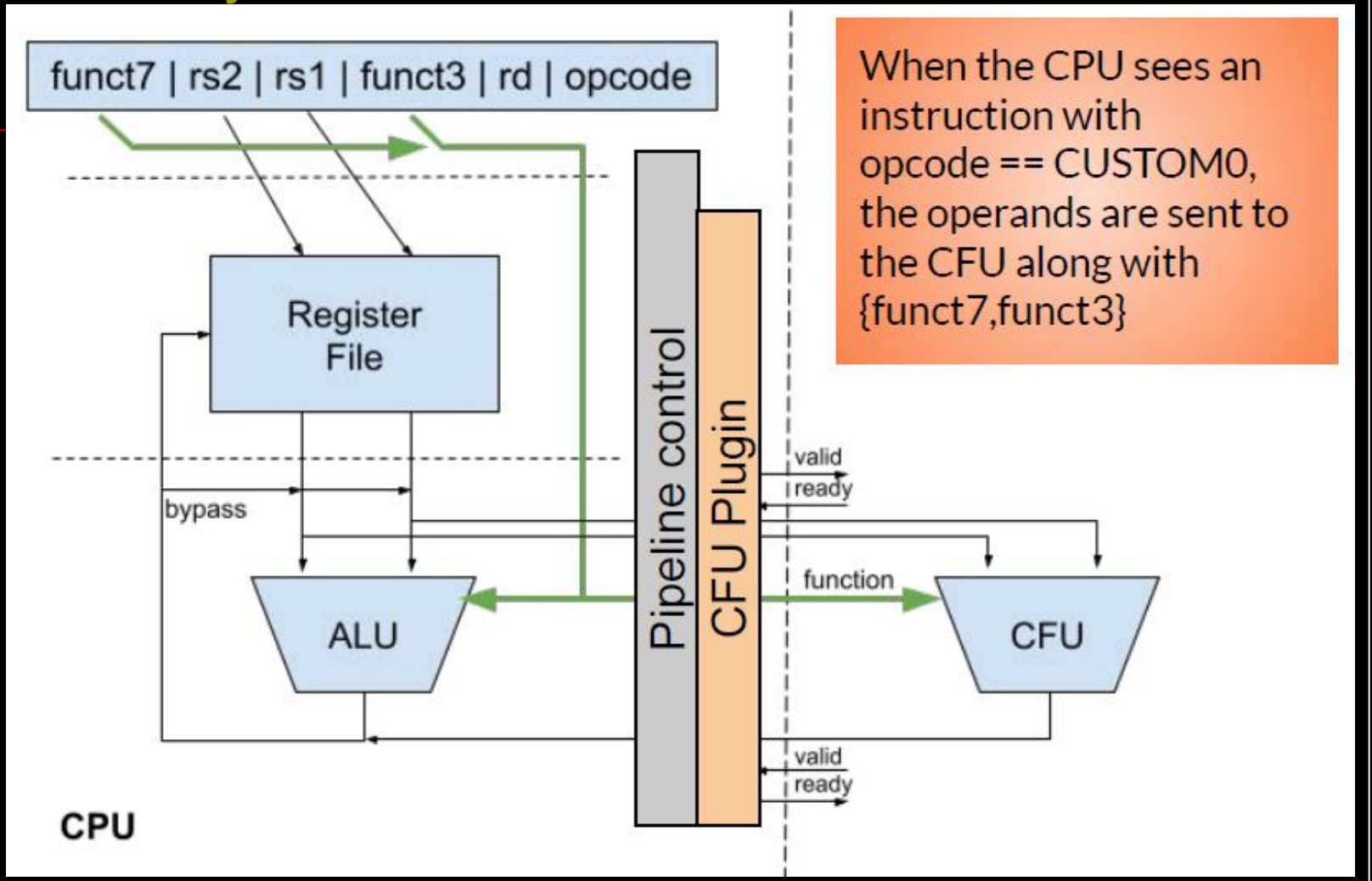
CFUs may be written in Verilog or with any tool that outputs Verilog. We prefer using Amaranth, because it has good support for composition, reuse and unit testing. The CFU Playground includes Amaranth library components to support CFU development.

We are currently using Vivado to synthesize the FPGA bitstream, and intend to move to Symbiflow in the near future.

Source: https://cfu-playground.readthedocs.io/en/latest/overview.html

How it works

- What exactly is a “CFU” here?



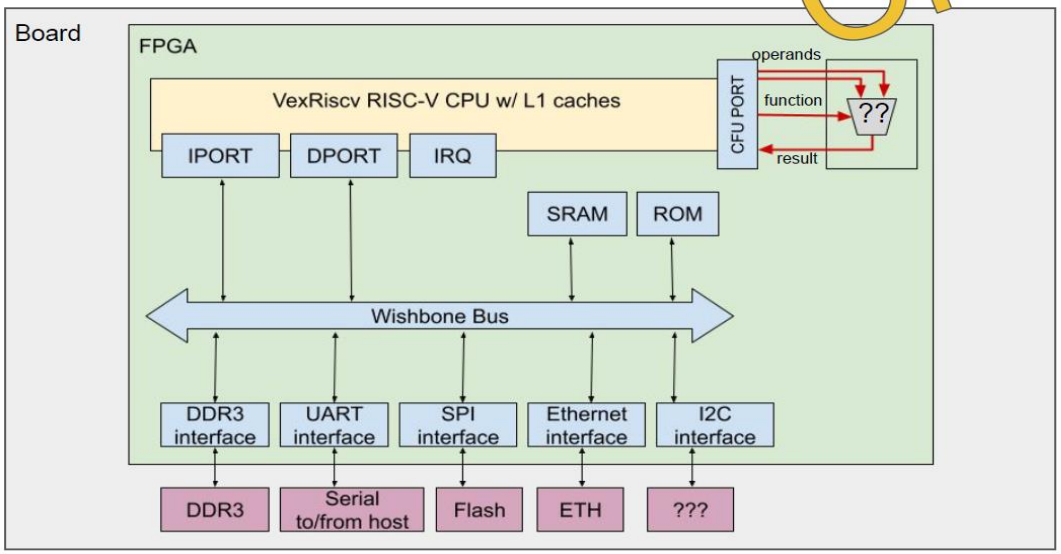
Source: https://static.sched.com/hosted_files/riscvforumdttc2021/fb/Share%20RISC-V%20Forum%20CFU%20Playground.pdf

- **VexRiscv**

  $SRC_VEXRISCV/src/main/scala/vexriscv/plugin/CfuPlugin.scala

  $SRC_VEXRISCV/src/main/scala/vexriscv/demo/GenSmallAndProductiveCfu.scala

FPGA System on a Chip (SoC): LiteX

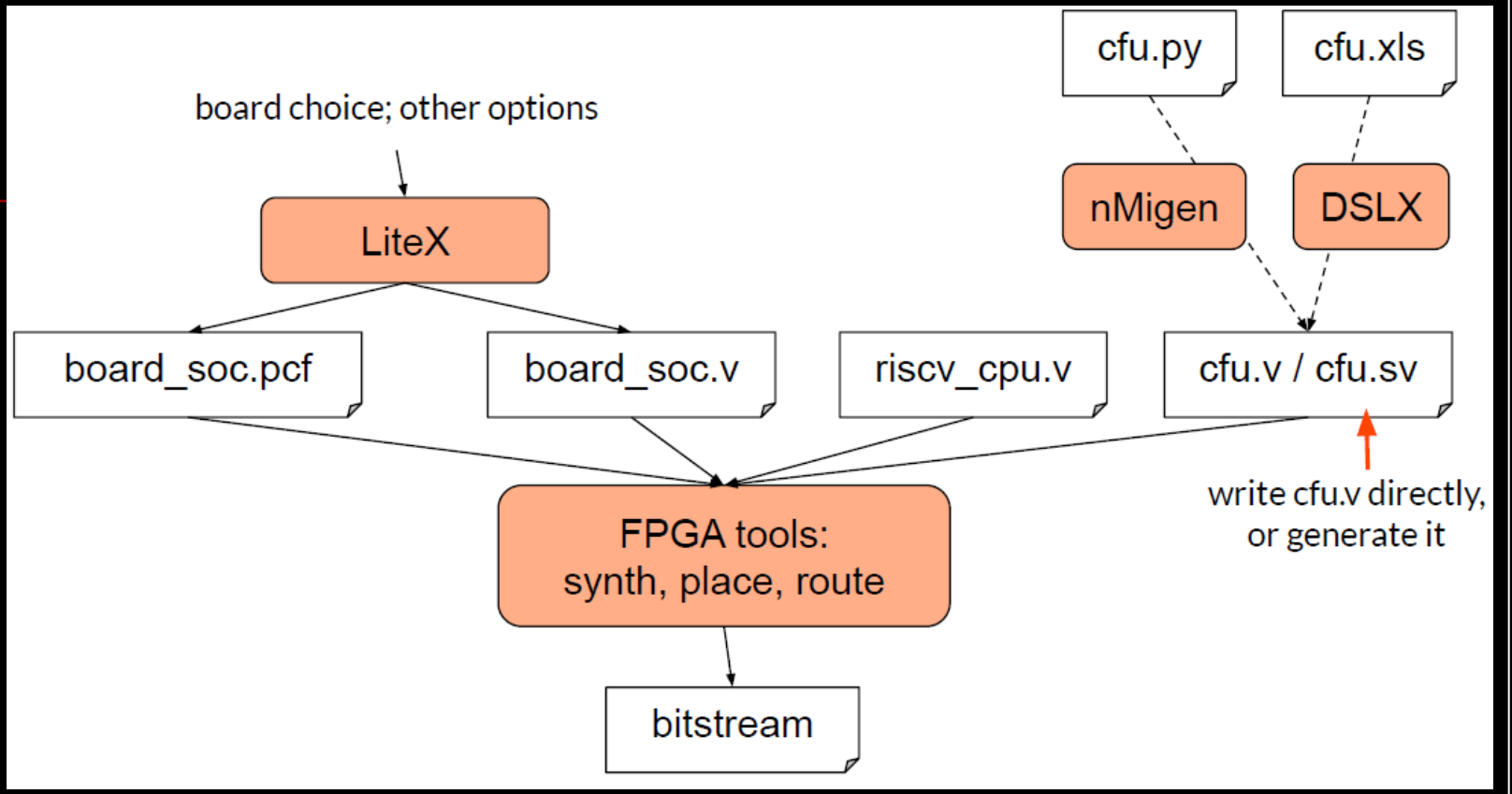


Source: "CFU Playground: Build your own ML Processor using Open Source", Tim Callahan etc, WOSET 2021.

- Choose a TensorFlow Lite model; a quantized person detection model is provided, or bring your own.
- Execute the inference on the Arty FPGA board to get cycle counts per layer.
- Choose an TFLite operator to accelerate, and dig into that code.
- Design new instruction(s) that can replace multiple basic operations.
- Build a custom function unit (a small amount of hardware) that performs the new instruction(s).
- Modify the TFLite/Micro library kernel to use the new instruction(s), which are available as intrinsics with function call syntax.
- Rebuild the FPGA Soc, recompile the TFLM library, and rerun to measure improvement.

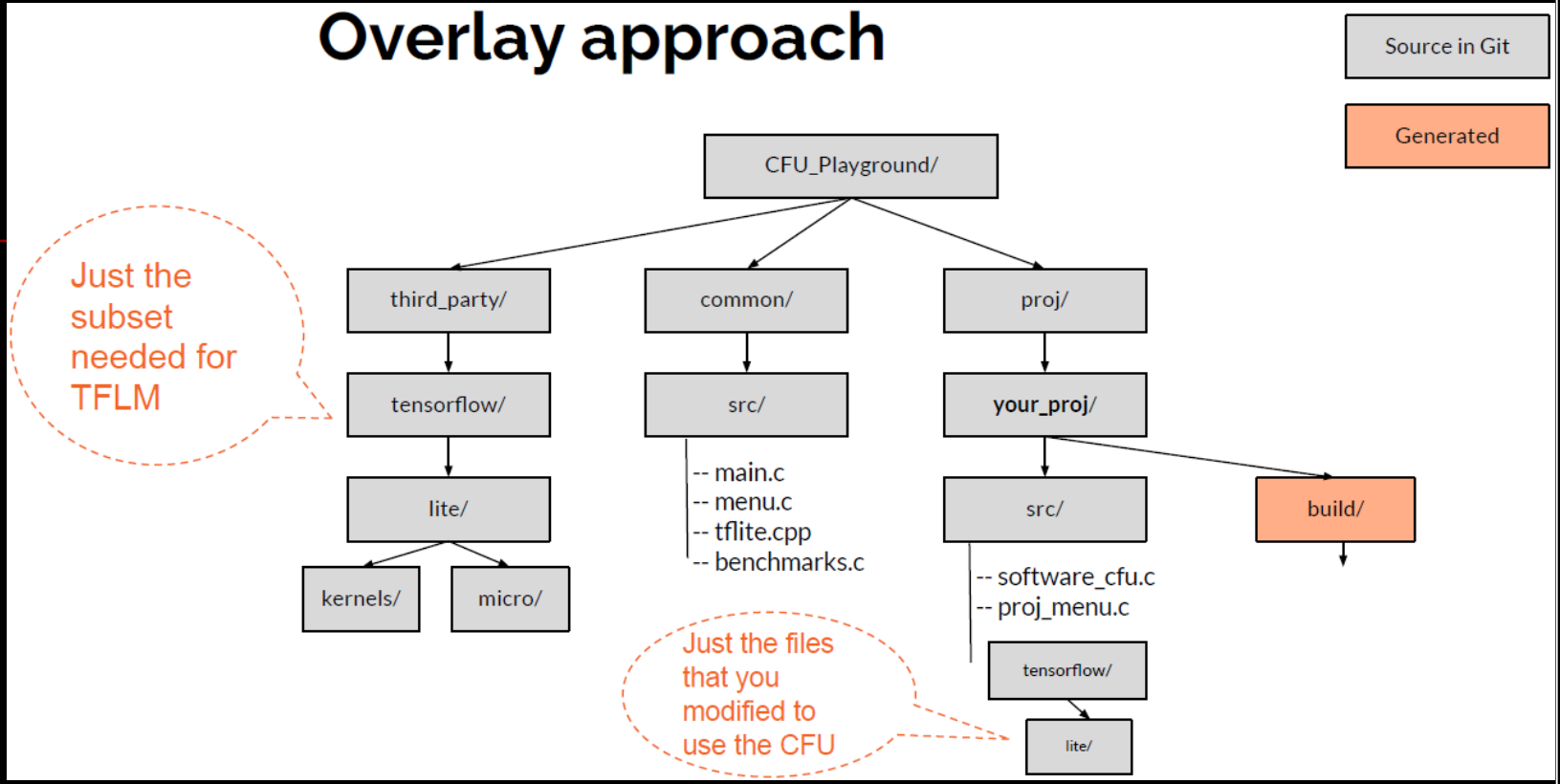
Source: https://github.com/google/CFU-Playground

Hardware Flow



Source: "CFU Playground: Build your own ML Processor using Open Source", Tim Callahan etc, WOSET 2021.

Overlay approach



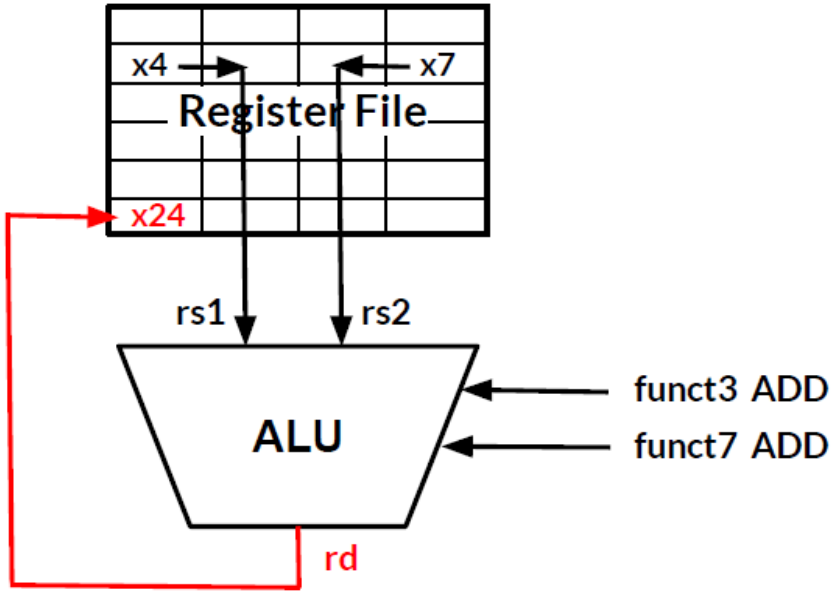
Source: https://static.sched.com/hosted_files/riscvforumdttc2021/fb/Share%20RISC-V%20Forum%20CFU%20Playground.pdf

Example

- The RISC-V Add instruction

00390C33	add x24,x4,x7	# x24 = x4 + x7

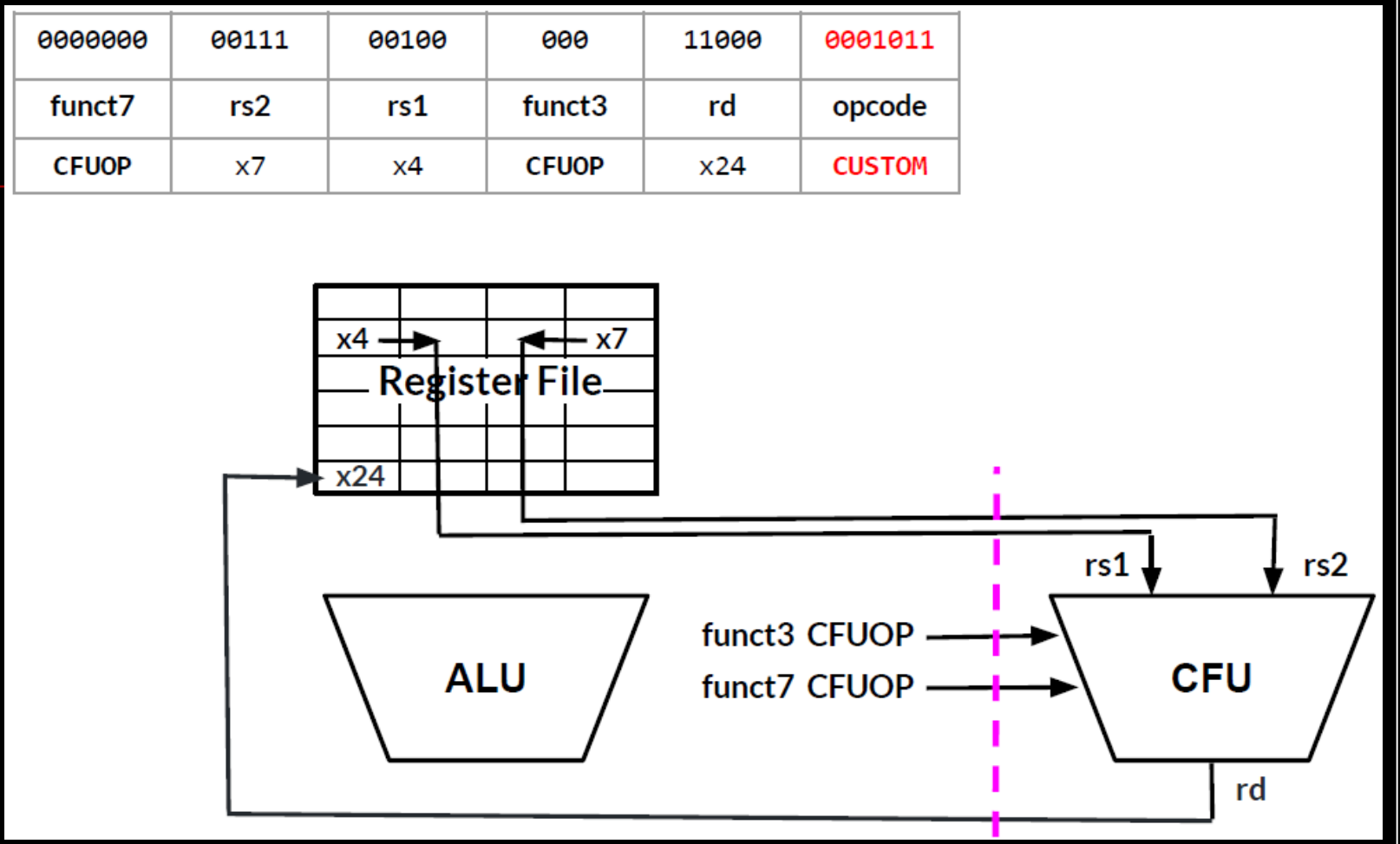
0000000	00111	00100	000	11000	0110011
funct7	rs2	rs1	funct3	rd	opcode
ADD	x7	x4	ADD	x24	ALU



Source: https://cms.tinyml.org/wp-content/uploads/talks2022/tinyML_Talks_Tim_Callahan_220111.pdf

Custom Instruction

0000000	00111	00100	000	11000	0001011
funct7	rs2	rs1	funct3	rd	opcode
CFUOP	x7	x4	CFUOP	x24	CUSTOM



Source: https://cms.tinyml.org/wp-content/uploads/talks2022/tinyML_Talks_Tim_Callahan_220111.pdf

Using CFU ops

- cfu_op macro expands to GCC inline asm

```
rslt = cfu_op(funct3, funct7, op1, op2);
```

Compile-time constants

C / C++ variables / expressions

$SRC_CFU-Playground/ common/src/cfu.h

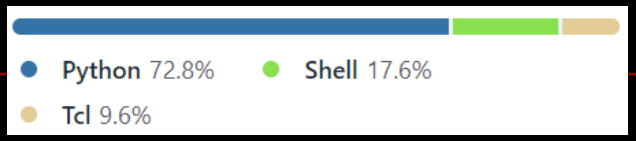
```
t1 = *x;
t2 = cfu_op(0, 0, t1, b);
t3 = cfu_op(1, 0, t2, b);
*x = t3;
```

```
400001a0:   00812783    lw        a5,8(sp)
400001a4:   00d7878b    cfu[0,0]  a5, a5, a3
400001a8:   00d7978b    cfu[0,1]  a5, a5, a3
400001ac:   00f12423    sw        a5,8(sp)
```

Source: "CFU Playground: Build your own ML Processor using Open Source", Tim Callahan etc, WOSET 2021.

2) F4PGA (Formerly known as SymbiFlow)

- https://github.com/chipsalliance/f4pga
  FOSS Flow For FPGA.



- https://f4pga.readthedocs.io/en/latest/index.html

F4PGA ↗, which is a Workgroup under the CHIPS Alliance ↗, is an Open Source solution for Hardware Description Language (HDL) to Bitstream FPGA synthesis, currently targeting Xilinx's 7-Series, QuickLogic's EOS-S3, and Lattice' iCE40 and ECP5 devices. Think of it as the GCC of FPGAs. The project aims to design tools that are highly extendable and multiplatform.

- Key Features

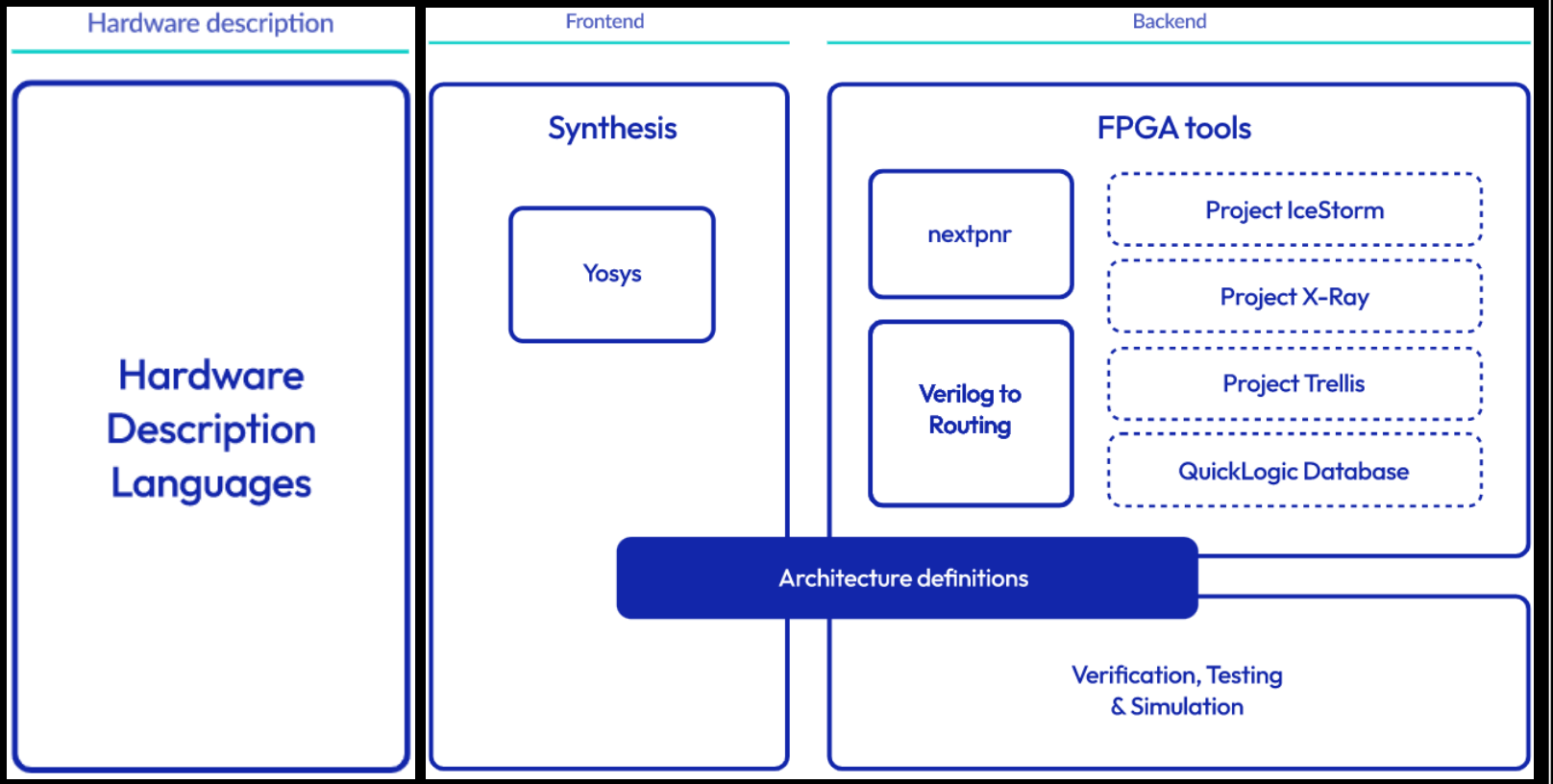
- https://www.design-reuse.com/news/51460/chips-alliance-f4pga-workgroup-open-source-fpga-tools.html

Architecture & Design

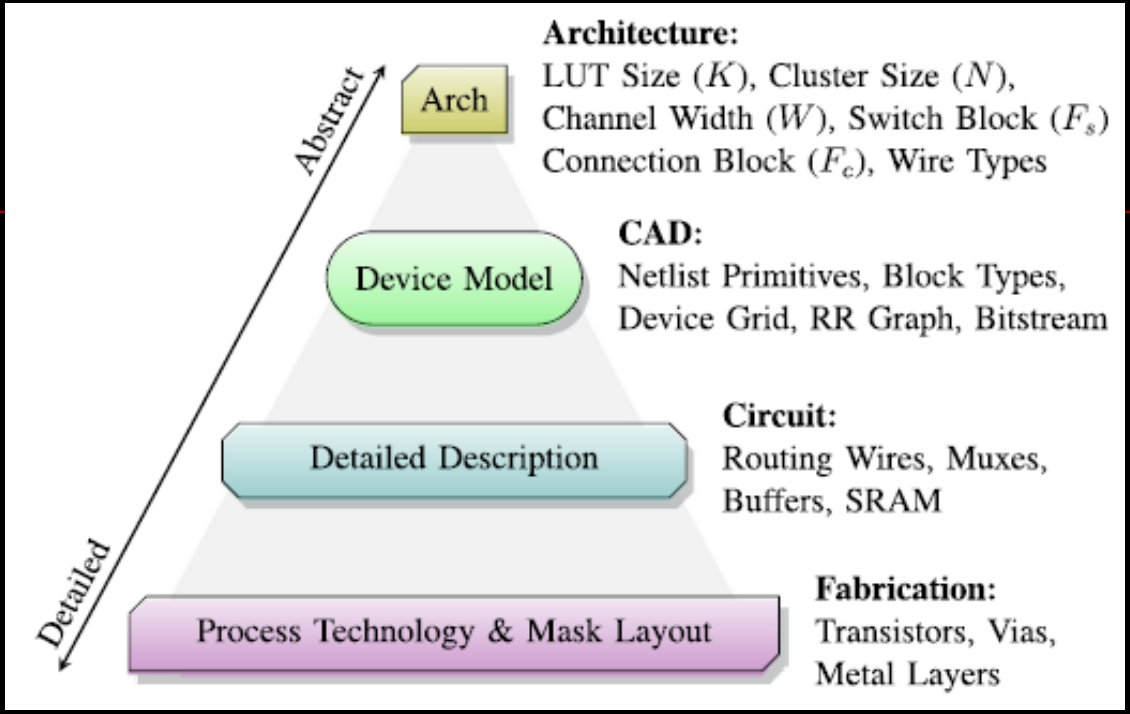
- https://f4pga.org/

To become a complete FOSS FPGA toolchain, F4PGA needs a number of tools and projects to be in place to provide an end-to-end flow. Thus, F4PGA serves as an umbrella framework for several activities, the central of which focuses on the creation of FPGA F4PGA Architecture Definitions, i.e. documentation of how specific FPGAs work internally.

Those definitions and serve as input to backend tools like nextpnr and Verilog to Routing, and frontend tools like Yosys. They are created within separate collaborating projects targeting different FPGAs - **Project X-Ray** for Xilinx 7-Series, **Project IceStorm** for Lattice iCE40 and **Project Trellis** for Lattice ECP5 FPGAs.



FPGA architecture representation



Source: https://ieeexplore.ieee.org/document/9103284

https://docs.verilogtorouting.org/en/latest/arch/

…

In F4PGA:

https://f4pga.readthedocs.io/projects/arch-defs/en/latest/index.html

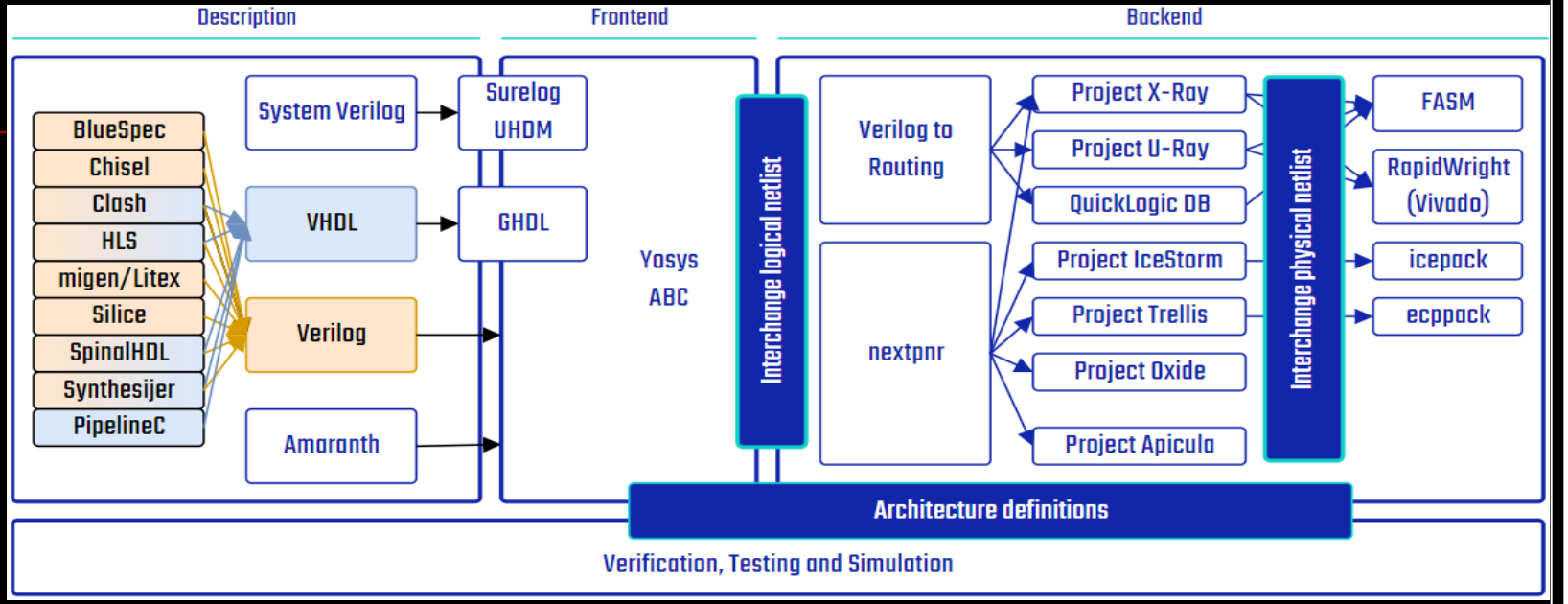
https://github.com/SymbiFlow/f4pga-arch-defs/

https://docs.verilogtorouting.org/en/latest/arch/#fpga-architecture-description

…

How it works

- https://f4pga.readthedocs.io/en/latest/how.html



- **Amaranth HDL (previously nMigen)**
  https://github.com/amaranth-lang/

The Amaranth project provides an open-source toolchain for developing hardware based on synchronous digital logic using the Python programming language, as well as evaluation board definitions, a System on Chip toolkit, and more. It aims to be easy to learn and use, reduce or eliminate common coding mistakes, and simplify the design of complex hardware with reusable components.

The Amaranth toolchain consists of the Amaranth hardware definition language, the standard library, the simulator, and the build system, covering all steps of a typical FPGA development workflow. At the same time, it does not restrict the designer's choice of tools: existing industry-standard (System)Verilog or VHDL code can be integrated into an Amaranth-based design flow, or, conversely, Amaranth code can be integrated into an existing Verilog-based design flow.

The development of Amaranth has been supported by SymbioticEDA, LambdaConcept, and ChipEleven.

Migration from Migen

If you have existing Migen code, you can use a comprehensive Migen compatibility layer provided in Amaranth. An existing Migen design can be synthesized and simulated with Amaranth in three steps:

1. Replace all `from migen import <...>` statements with `from amaranth.compat import <...>`.
2. Replace every explicit mention of the default `sys` clock domain with the new default `sync` clock domain. E.g. `ClockSignal("sys")` is changed to `ClockSignal("sync")`.
3. Migrate from Migen build/platform system to Amaranth build/platform system. Amaranth does not provide a build/platform compatibility layer because both the board definition files and the platform abstraction differ too much.

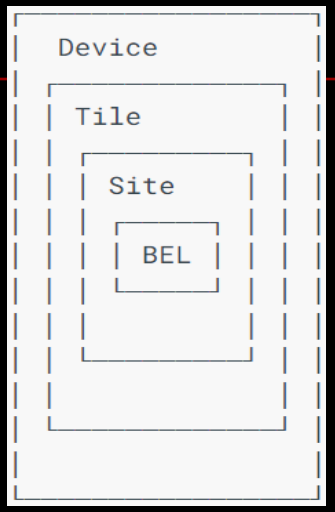
Note that Amaranth will **not** produce the exact same RTL as Migen did. Amaranth has been built to allow you to take advantage of the new and improved functionality it has (such as producing hierarchical RTL) while making migration as painless as possible.

Once your design passes verification with Amaranth, you can migrate it to the Amaranth syntax one module at a time. Migen modules can be added to Amaranth modules and vice versa, so there is no restriction on the order of migration, either.

2.1 FPGA interchange documentation

- https://fpga-interchange-schema.readthedocs.io/

Device Resources



- Device - A set of tiles and package pins.
- Tiles - An instance of a tile type which contains wires and sites
- Package pin - A boundry between the “interior” of the device and what is “outside” the package. Generally corresponds to a pin on a package, e.g. pin 1 on SOP-8 or A1 on CSG324.
- Wire - Also known as a “tile wire” . A wire is a piece of conductive material totally contained within a tile. Wires can be part of nodes. Wires can connect to PIPs or site pins.
- Node - A node is a set of 1 or more wires that are connected. Nodes can span multiple tiles. Nodes connect to PIPs or site pins via the wires that are part of the node.
- PIP - PIP is an abbreviation for programable interconnect point. A PIP provides a connection between two wires. PIPs can be unidirectional or bidirectional. Unidirectional PIPs always connect wire0 to wire1. Bidirectional PIPs can connect wire0 to wire1 or wire1 to wire0.
- Site - A collection of site pins, site wires and BELs.
- Site pin - The connection between a site and a wire. Site pins may connect to 0 or more site port BELs.
- Site wire - A piece of conductive material that connects to at most 1 output BEL pin and 0 or more input or inout BEL pins.
- BEL - BEL is an abbreviation of basic logic element. A BEL can be one of 3 types, site port, logic, routing. A BEL contains 1 or more BEL pins.
- BEL pin - A connection between a BEL and a site wire.
- Logic BEL - A placable logic element. May be subject to 0 or more placement constraints.
- Site port BEL - A site port BEL represents a connection to a site pin contained within the parent tile of the site. See Site port BEL.

- Routing BEL - A routing BEL connects at most 1 input BEL pin to the output BEL pin. See Routing BEL.
- Site PIP - A pair of input and output BEL pins belonging to a BEL that represents a logically connection.
- Cell - A logical element of a design that contains some number of cell ports and some number of cell instances, and some number of nets.
- Cell port - The boundary between the interior of a cell and the containing cell (if any).
- Cell instance - A instance of a cell. The cell ports of may be connected to nets within the parent cell.
- Net - A set of logically connected cell ports.

…

- https://antmicro.com/blog/2021/09/fpga-interchange-format/
- https://opensource.googleblog.com/2022/02/FPGA%20Interchange%20format%20to%20enable%20interoperable%20FPGA%20tooling.html

Components

The Interchange format provides three key descriptions to describe an FPGA and to interact with the various tools involved:

- Device resources: defines the FPGA internal structure and the technological cell libraries describing FPGA logic blocks (basic blocks like flip-flops and complex like DSP cells).
- Logical netlist: post-synthesized netlist compatible with the Interchange.
- Physical netlist: collection of all placement locations and physical routing of the nets and resources produced by the place and route tool.

Repositories

- **https://github.com/chipsalliance/fpga-interchange-schema**
  Contains the capnp schema for the FPGA interchange.
- **Tools around the FPGA interchange**

RapidWright:

- Provides support for 7-series, UltraScale and UltraScale+ parts
- Generate device database for parts
- Can convert DCPs to logical and physical FPGA interchange files.
- Can convert logical and physical FPGA interchange files to DCPs

python-fpga-interchange:

- Implements textual format conversions for FPGA interchange files.
- Provides (partial) generator for nextpnr to generate a nextpnr architecture for a FPGA interchange device database.

- **…**

2.2 FASM

- FPGA Assembly
- https://fasm.readthedocs.io/en/latest/

The FASM is a file format designed to specify the bits in an FPGA bitstream that need to be set (e.g. binary 1) or cleared (e.g. binary 0).

A FASM file declares that specific “Features” within the bitstream should be enabled. Enabling a feature will cause bits within the bitstream to be set or cleared.

A FASM file is illegal if a bit in the final bitstream must be set and cleared to respect the set of features specified in the FASM file.

An empty FASM file will generate a platform specific “default” bitstream. The FASM file will specify zero or more features that mutate the “default” bitstream into the target bitstream.

- https://github.com/chipsalliance/fasm

This repository documents the FASM file format and provides parsing libraries and simple tooling for working with FASM files.

It provides both a pure Python parser based on `textx` and a significantly faster C parser based on `ANTLR`. The library will try and use the ANTLR parser first and fall back to the `textx` parser if the compiled module is not found.

Which parsers are supported by your currently install can be found via `python3 -c "import fasm.parser as p; print(p.available)`. The currently in use parser can be found via `fasm.parser.implementation`.

It is highly recommended to use the ANTLR parser as it is about 15 times faster.

functions for parsing and generating FASM files.

Support in VTR

- **https://docs.verilogtorouting.org/en/latest/utils/fasm/**

After VPR has generated a placed and routed design, the `genfasm` utility can emit a FASM file to represent the design at a level detailed enough to allow generation of a bitstream to program a device. This FASM output file is enabled by FASM metadata encoded in the VPR architecture definition and routing graph. The output FASM file can be converted into a bitstream format suitable to program the target architecture via architecture specific tooling. Current devices that can be programmed using the vpr + fasm flow include Lattice iCE40 and Xilinx Artix-7 devices, with work on more devices underway. More information on supported devices is available from the Symbiflow website and an overview of the flow for Artix-7 devices is described in IEEE Micro [MurrayAnsellRothman+20].

2.3 UHDM

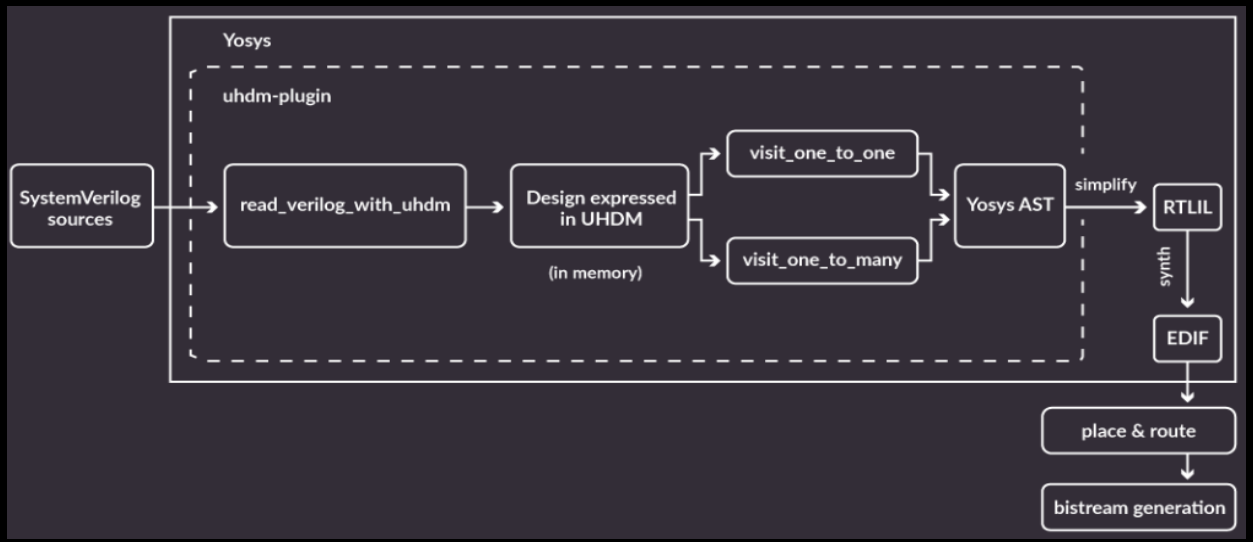
- https://github.com/chipsalliance/UHDM

Universal Hardware Data Model

Universal Hardware Data Model. A complete modeling of the IEEE SystemVerilog Object Model with VPI Interface, Elaborator, Serialization, Visitor and Listener. Used as a compiled interchange format in between SystemVerilog tools. Compiles on Linux gcc, Windows msys2-gcc & msvc, OsX

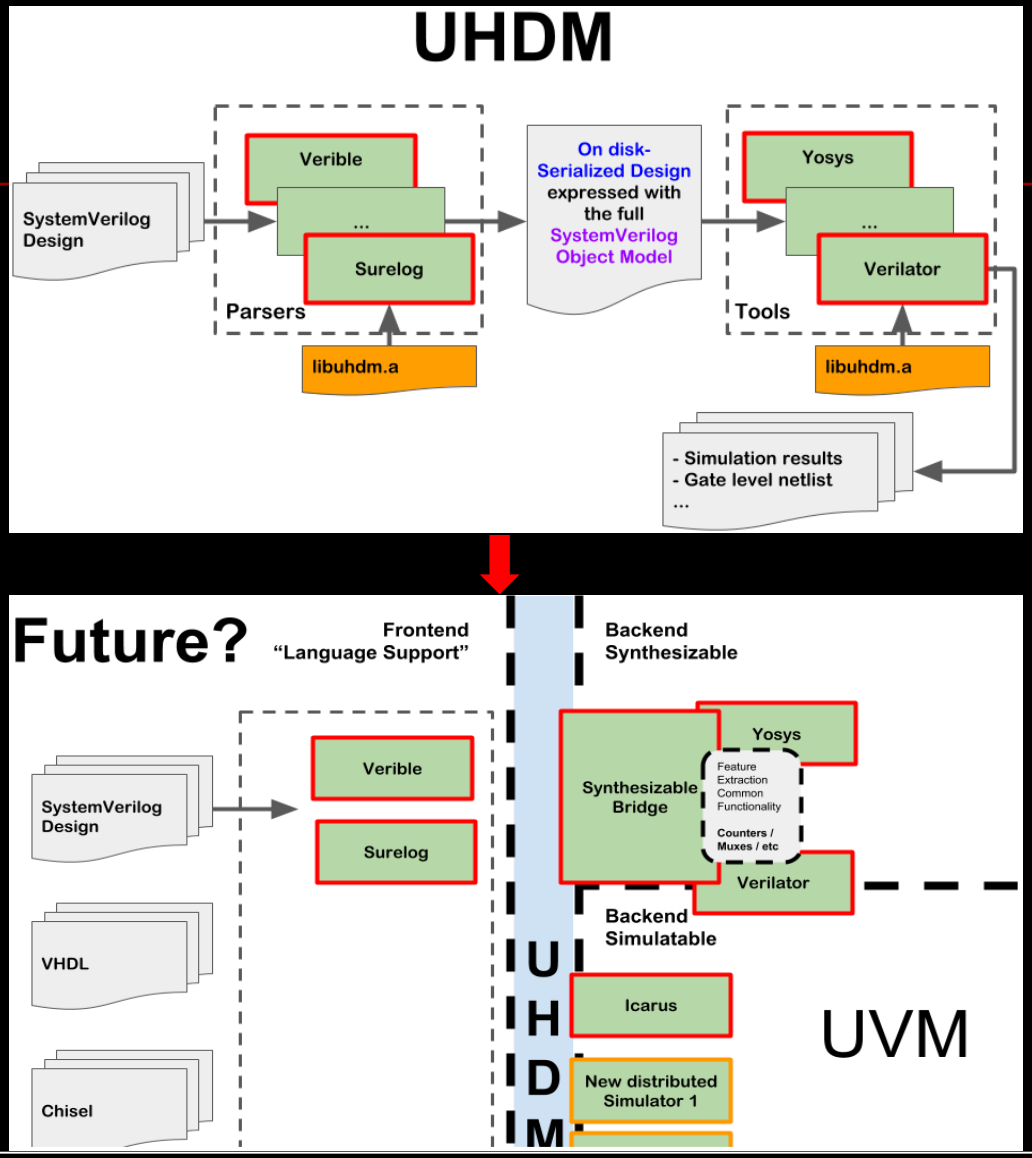
https://standards.ieee.org/ieee/1800/7743/

- https://antmicro.com/blog/2022/02/simplifying-open-source-sv-synthesis-with-the-yosys-uhdm-plugin/



- …

Workflow


UHDM
SystemVerilog Design
Verible
...
Surelog
Parsers
libuhdm.a
On disk-Serialized Design expressed with the full SystemVerilog Object Model
Yosys
...
Verilator
Tools
libuhdm.a
- Simulation results
- Gate level netlist
...
Future?
Frontend "Language Support"
Backend Synthesizable
SystemVerilog Design
Verible
Surelog
VHDL
Chisel
Synthesizable Bridge
Yosys
Feature Extraction Common Functionality
Counters / Muxes / etc
Verilator
Backend Simulatable
U
H
D
M
Icarus
New distributed Simulator 1
UVM

2.4 Design Flow
2.4.1 VTR

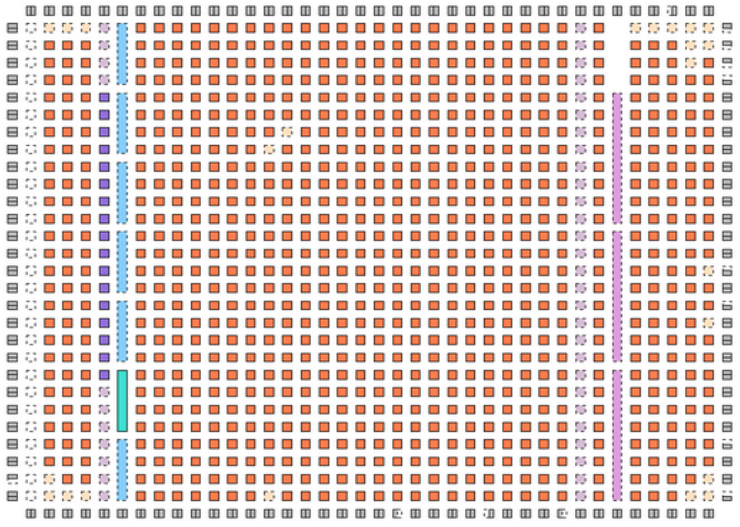
- https://verilogtorouting.org/

The Verilog to Routing (VTR) project provides open-source CAD tools for FPGA architecture and CAD research.

Open source CAD tools enable the investigation of new FPGA architectures and CAD algorithms, which are not possible with closed-source tools.

The VTR design flow takes as input a Verilog description of a digital circuit, and a description of the target FPGA architecture. It then perfoms:

- Elaboration & Synthesis (ODIN II)
- Logic Optimization & Technology Mapping (ABC)
- Packing, Placement, Routing & Timing Analysis (VPR)



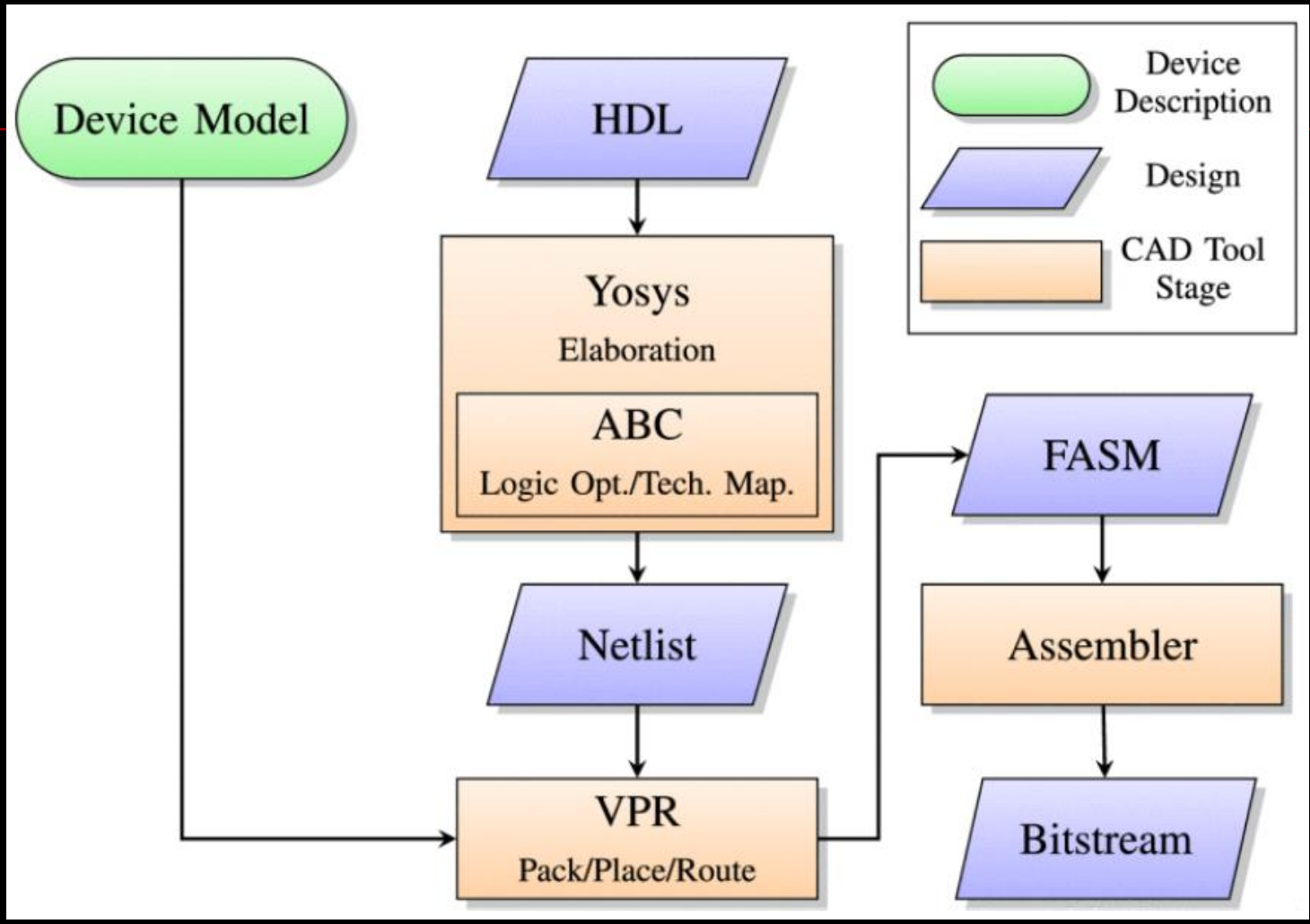
Device Floorplan & Placement

to produce FPGA speed and area results. VTR can also produce the information required for bitstream generation to target real FPGA devices.

- https://github.com/verilog-to-routing

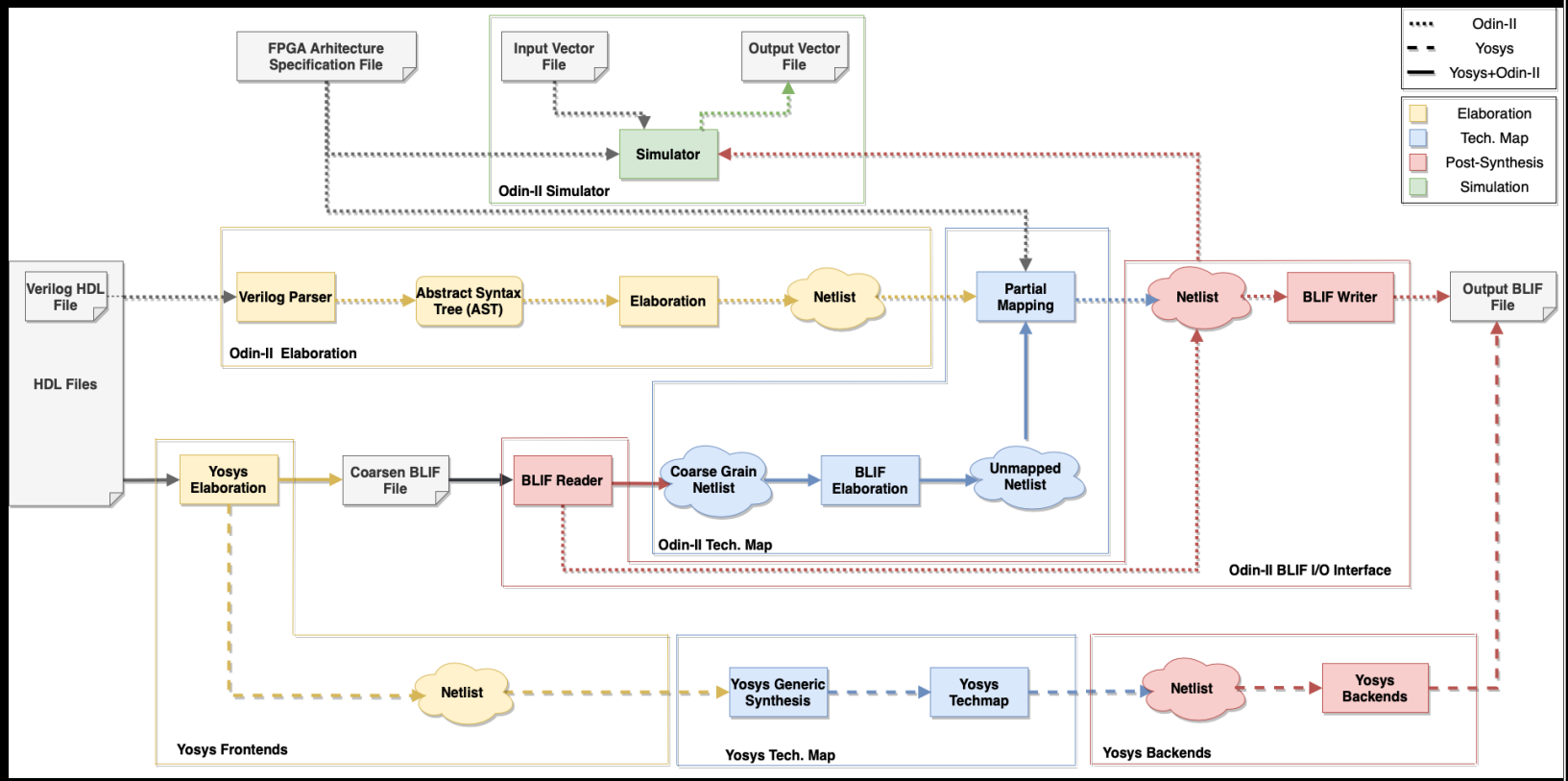
Overall

- https://ieeexplore.ieee.org/document/9103284



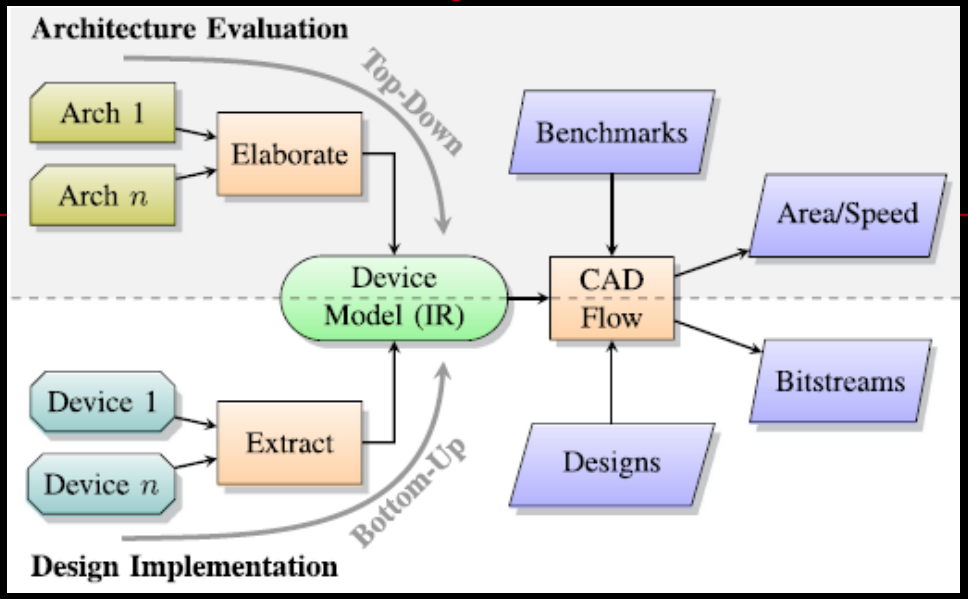
Yosys + Odin-II

Yosys+Odin-II is the combination of Yosys as the elaborator and the Odin-II partial mapper and logic optimizer to perform logic synthesis for a subset of the Verilog Hardware Description Language (HDL) into a BLIF netlist. Odin-II lacks support for the Verilog-2005 standard and SystemVerilog, but it provides complex partial mapping for balancing soft logic and hard blocks such as embedded multipliers, adders, and memories in FPGA architectures. Yosys, an open framework for RTL synthesis, provides extensive support for HDLs. However, the Yosys flow forces the user to decide the discrete circuit implementation manually.

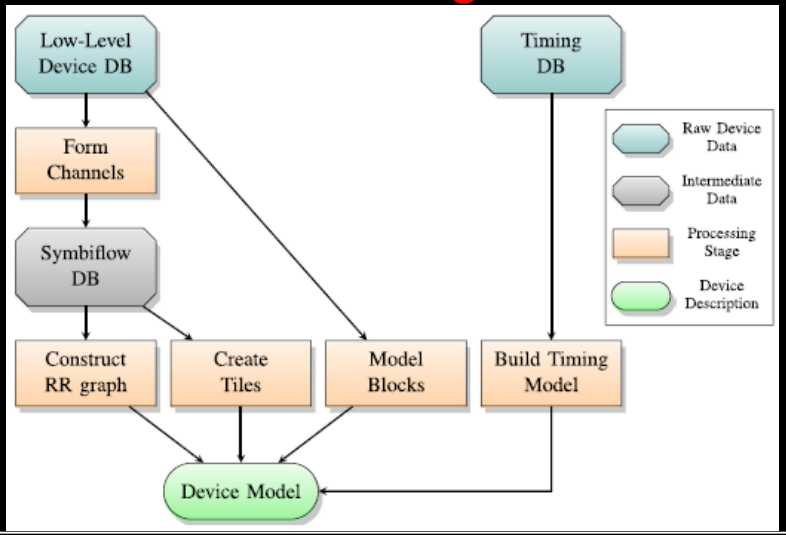


Source: https://docs.verilogtorouting.org/en/latest/yosys+odin/

- **Architecture and implementation flows**

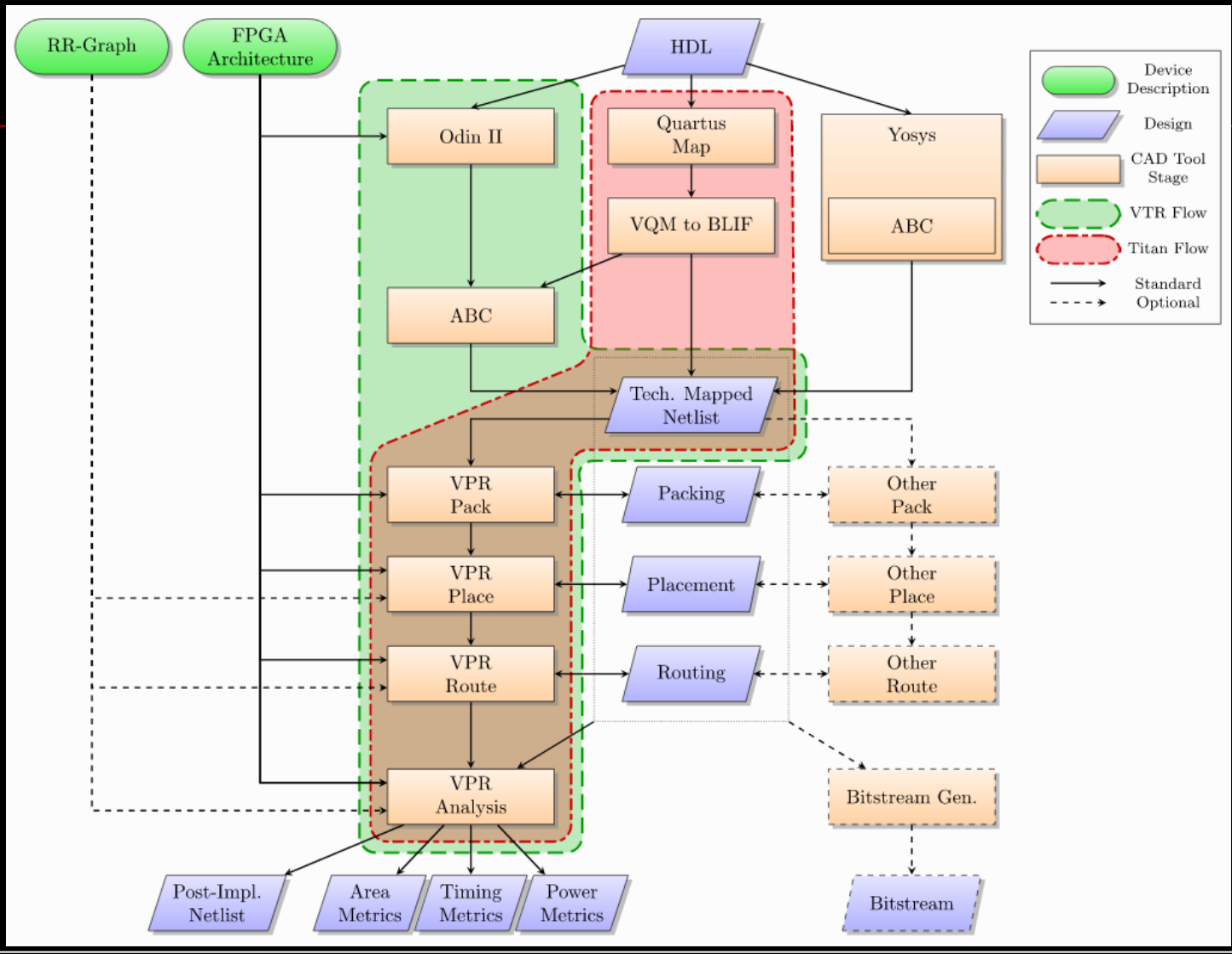


- **Architecture model generation**



VTR CAD Flow

- https://docs.verilogtorouting.org/en/latest/vtr/cad_flow/#vtr-cad-flow

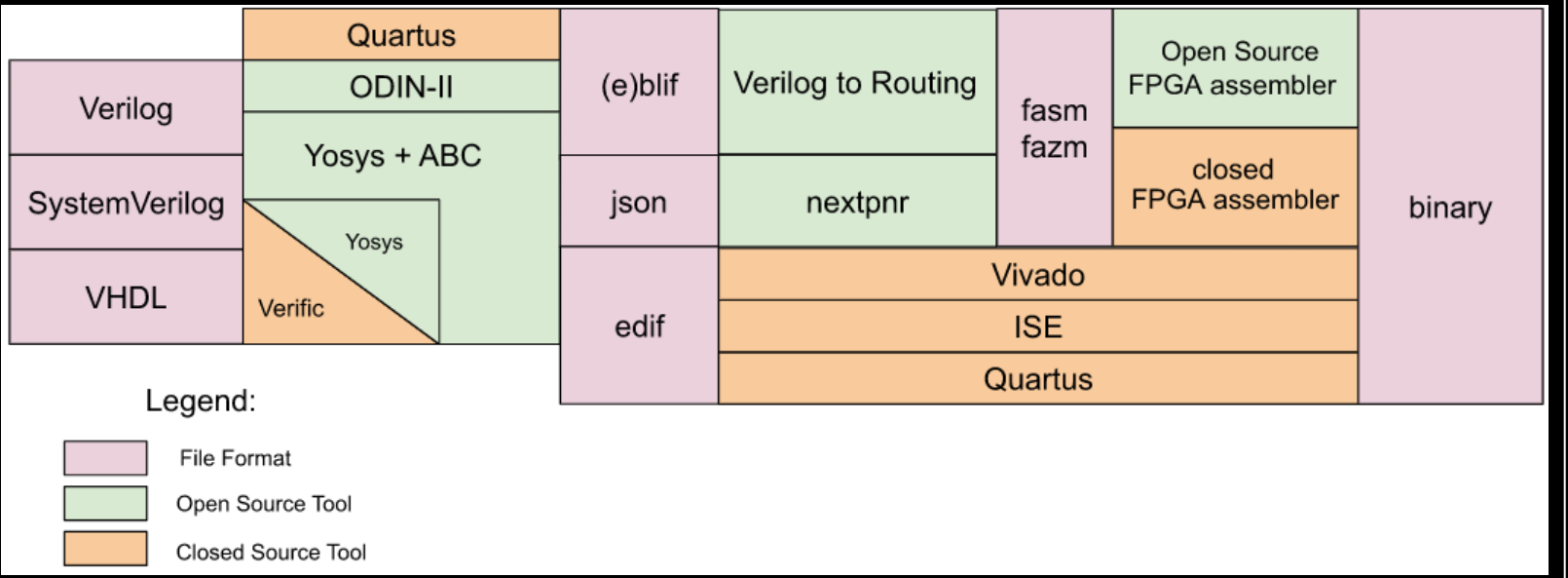


2.4.2 F4PGA

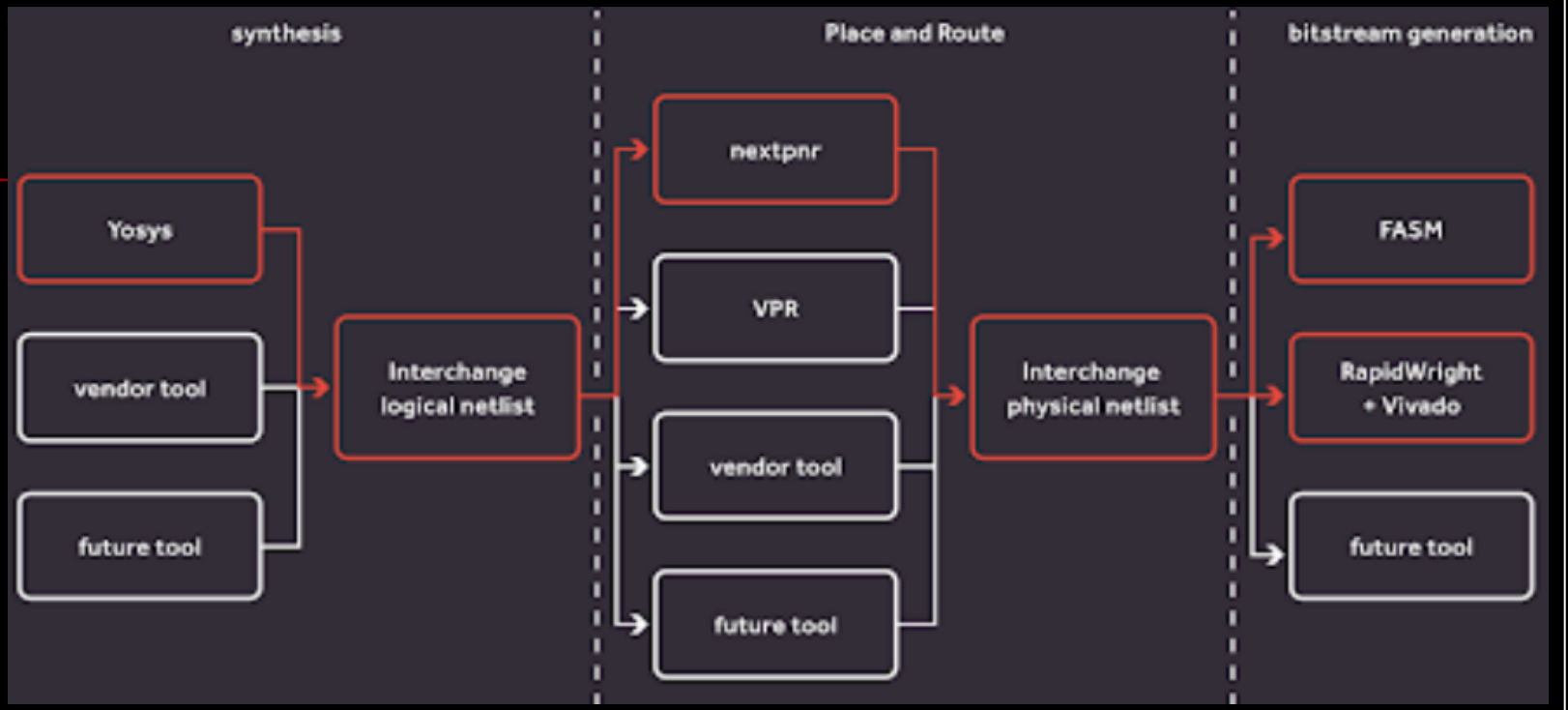
- https://f4pga.readthedocs.io/en/latest/flows/index.html

F4PGA is an end-to-end FPGA synthesis toolchain, because of that it provides all the necessary tools to convert input Hardware Description Language (HDL) sources into a final bitstream. It is simple to use however, the whole synthesis and implementation process is not trivial.

The final bitstream format depends on the used platform. What's more, every platform has different resources and even if some of them provide similar functionality, they can be implemented in a different way. In order to be able to match all that variety of possible situations, the creation of the final bitstream is divided into few steps. F4PGA uses different programs to create the bitstream and is responsible for their proper integration. The procedure of converting HDL files into the bitstream is described in the next sections.



Example Flow

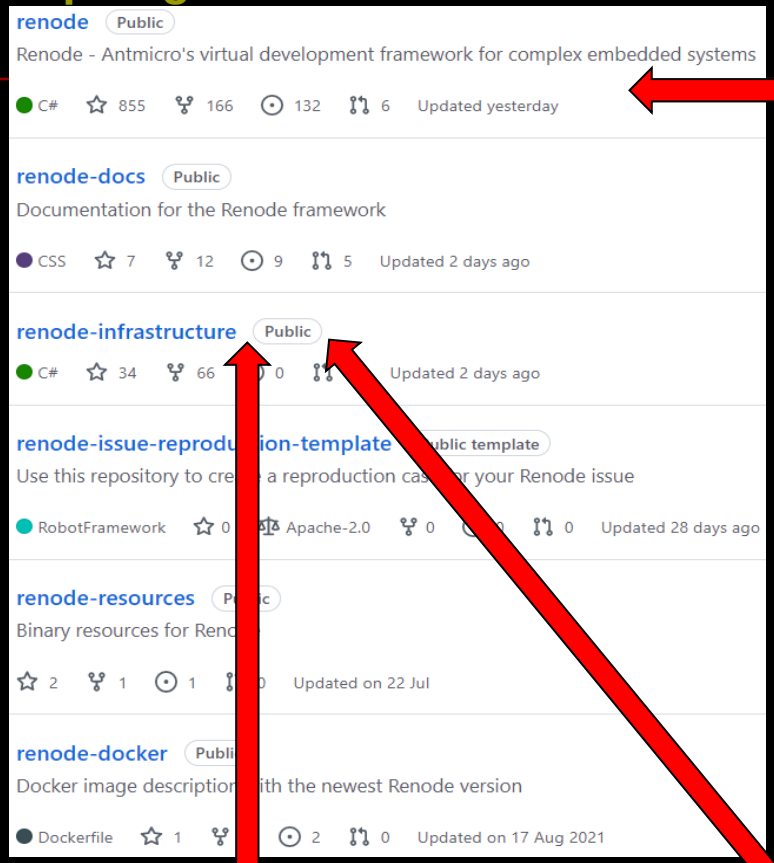


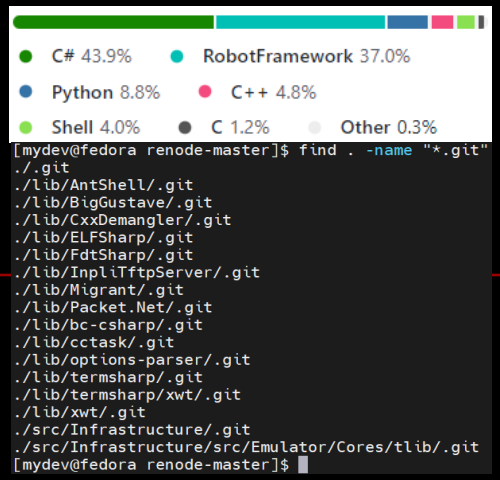
Source: https://opensource.googleblog.com/2022/02/FPGA%20Interchange%20format%20to%20enable%20interoperable%20FPGA%20tooling.html

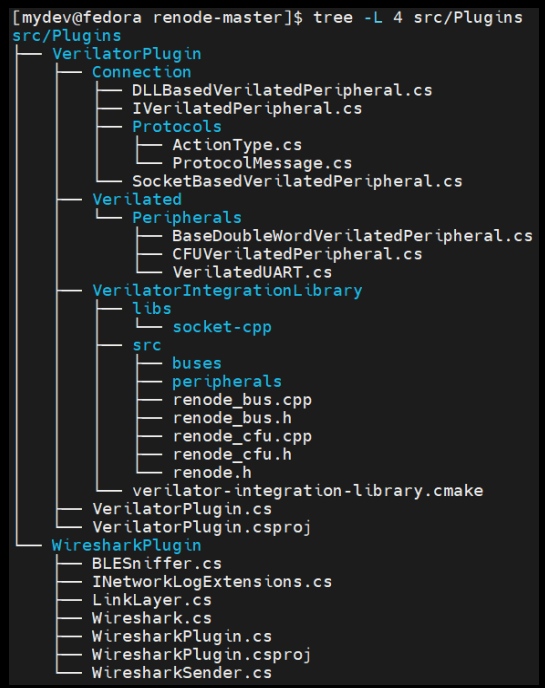
3) Developing with Renode

Renode

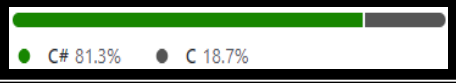
- https://github.com/renode







- https://github.com/antmicro/tlib



- …

Renode requires Mono >= 5.20 (Linux, macOS) or .NET >= 4.7 (Windows).

Linux	Install the `mono-complete` package as per the installation instructions for various Linux distributions which can be found on the Mono project website.
macOS	On macOS, the Mono package can be downloaded directly from the Mono project website.
Windows	On Windows 7, download and install .NET Framework 4.7. Windows 10 ships with .NET by default, so no action is required there.

Model files

- LiteX generates a platform description in csr.csv output file
- Automatic generation from LiteX configuration (CSV)
  - platform (REPL)
  - script (RESC)
  - Zephyr DTS overlay
- Perfect for CI setup

...

```
[mydev@fedora renode-master]$ tree -L 2 platforms/boards/
platforms/boards/
├── A2_CV32E40P.repl
├── arduino_101-shield.repl
├── arduino_nano_33_ble.repl
├── arty_litex_vexriscv.repl
├── arvsom.repl
├── beaglev_starlight.repl
├── colibri-vf61.repl
├── crosslink-nx-evn.repl
├── eos-s3-qomu.repl
├── eos-s3-quickfeather.repl
├── gr716-devboard.repl
├── ice40up5k-mdp-evn.repl
├── leon3-externals.repl
├── leon3.repl
├── mars_zx3-externals.repl
├── mars_zx3.repl
├── max32652-evkit.repl
├── mimxrt1064_evk.repl
├── miv-board-additional-uarts.repl
├── miv-board.repl
├── mpfs-icicle-kit.repl
├── quark_c1000-cc2520.repl
├── silabs
│   ├── brd4162a.repl
│   ├── sltb001a.repl
│   ├── slwstk6220a.repl
│   ├── stk3200.repl
│   ├── stk3600.repl
│   ├── stk3700.repl
│   └── stk3800.repl
├── sltb004a.repl
├── stm32f072b_discovery.repl
├── stm32f4_discovery-additional_gpios.repl
├── stm32f4_discovery-bb.repl
├── stm32f4_discovery-kit.repl
├── stm32f4_discovery.repl
├── stm32f7_discovery-bb.repl
├── tegra2.repl
├── tegra3.repl
├── tegra_externals.repl
├── versatile.repl
├── vexpress-externals.repl
├── vexpress.repl
├── zedboard-externals.repl
├── zedboard.repl
└── zolertia-firefly.repl
```

```
[mydev@fedora renode-master]$ tree -L 2 scripts/
scripts/
├── complex
│   ├── arduino_nano
│   ├── fomu
│   ├── litex_i2s
│   └── README.rst
├── monitor.py
├── multi-node
│   ├── cc2538
│   ├── nrf52840-ble-zephyr.resc
│   ├── quark-c1000-zephyr
│   └── sam_e70.resc
├── pydev
│   ├── counter.py
│   ├── flipflop.py
│   ├── repeater.py
│   ├── rolling-bit.py
│   └── ticker.py
└── single-node
    ├── ambiq-apollo4.resc
    ├── arty_litex_vexriscv.resc
    ├── arvsom.resc
    ├── beaglev_starlight.resc
    ├── cc2538.resc
    ├── efr32mg.resc
    ├── gr716_zephyr.resc
    ├── hifive_unleashed.resc
    ├── i386.resc
    ├── icicle-kit.resc
    ├── kendryte_k210.resc
    ├── leon3_zephyr.resc
    ├── litex_ibex.resc
    ├── litex_microwatt.resc
    ├── litex_minerva.resc
    ├── litex_nexys_video_vexriscv_linux.resc
    ├── litex_vexriscv_linux.resc
    ├── litex_vexriscv_micropython.resc
    ├── litex_vexriscv.resc
    ├── litex_vexriscv_sdcard.resc
    ├── litex_vexriscv_smp.resc
    ├── litex_vexriscv_tftp.resc
    ├── litex_vexriscv_tock.resc
    ├── litex_vexriscv_verilated_cfu.resc
    ├── litex_vexriscv_zephyr.resc
    ...
```

Official Guide

- https://cfu-playground.readthedocs.io/en/latest/step-by-step.html

develop with Amaranth

There is a fairly robust framework for building a CFU in Amaranth. Inside of `<CFU-Playground root>/python/amaranth_cfu` there are a set of helper files that you can `import` in your code. It's best to read through the doc comments in `<CFU-Playground root>/python/amaranth_cfu/cfu.py` and `<CFU-Playground root>/python/amaranth_cfu/util.py` before starting development, but you should be able to get a reasonable understanding of the framework through this example.

```
from amaranth import C, Module, Signal, signed
from amaranth_cfu import all_words, InstructionBase, InstructionTestBase, pack_vals, simple_cfu
import unittest


# Custom instruction inherits from the InstructionBase class.
class SimdMac(InstructionBase):
    def __init__(self, input_offset=128) -> None:
        super().__init__()

        self.input_offset = C(input_offset, signed(9))

    # `elab` method implements the logic of the instruction.
    def elab(self, m: Module) -> None:
        words = lambda s: all_words(s, 8)

        # SIMD multiply step:
        self.prods = [Signal(signed(16)) for _ in range(4)]
        for prod, w0, w1 in zip(self.prods, words(self.in0), words(self.in1)):
            m.d.comb += prod.eq(
                (w0.as_signed() + self.input_offset) * w1.as_signed())

        m.d.sync += self.done.eq(0)
        # self.start signal high for one cycle when instruction started.
        with m.If(self.start):
            with m.If(self.funct7):
                m.d.sync += self.output.eq(0)
            with m.Else():
                # Accumulate step:
                m.d.sync += self.output.eq(self.output + sum(self.prods))
            # self.done signal indicates instruction is completed.
            m.d.sync += self.done.eq(1)
```

...

```
# Expose make_cfu function for cfu_gen.py
def make_cfu():
    # Associate cfu_op0 with SimdMac.
    return simple_cfu({0: SimdMac()})
```

...

develop with Verilog

Developing CFUs in Verilog is lower-level and doesn't give you access to the nice testing features of Amaranth, but it does give you more control over the CFU. Firstly, delete the `cfu.py` and `cfu_gen.py` files from your project folder; we'll directly be creating and editing a file named `cfu.v`. To add the `cfu.v` file in `git`, you'll need to use the force option: `git add -f cfu.v`.

When doing CFU development with Amaranth, the CFU-CPU handshaking is implemented for you in the `Cfu` base class. In Verilog you will need to implement your own handshaking and for that it's important to know the CFU module specification.

The "CFU bus" provides communication between the CPU and CFU. The CFU Bus is composed of two independent streams:

- The CPU uses the command stream (cmd) to send operands and 10 bits of function code to the CFU, thus initiating the CFU computation.
- The CFU uses the response stream (rsp) to return the result to the CPU. Since the responses are not tagged, they must be delivered in-order.

Each stream has two-way handshaking and backpressure (`*_valid` and `*_ready` in the diagram below). An endpoint can indicate that it cannot accept another transfer by pulling its `ready` signal low. A transfer takes place only when both `valid` from the sender and `ready` from the receiver are high.

Note

The data values from the CPU (`cmd_function_id`, `cmd_inputs_0`, and `cmd_inputs_1`) are valid ONLY during the cycle that the handshake is active (when both `cmd_valid` and `cmd_ready` are asserted). If your CFU needs to use these values in subsequent cycles, it must store them in registers.

```
    >--- cmd_valid --------------->
    <--- cmd_ready ---------------<
    >--- cmd_function_id[9:0] ---->
    >--- cmd_inputs_0[31:0] ------>
    >--- cmd_inputs_1[31:0] ------>
CPU                                CFU
    <--- rsp_valid ---------------<
    >--- rsp_ready --------------->
    <--- rsp_outputs_0[31:0] -----<
```

With the previous specification in mind, here's an implementation of our SIMD multiply-and-accumulate instruction in `cfu.v`:

```
module Cfu (
  input               cmd_valid,
  output              cmd_ready,
  input      [9:0]    cmd_payload_function_id,
  input      [31:0]   cmd_payload_inputs_0,
  input      [31:0]   cmd_payload_inputs_1,
  output reg          rsp_valid,
  input               rsp_ready,
  output reg [31:0]   rsp_payload_outputs_0,
  input               reset,
  input               clk
);
  localparam InputOffset = $signed(9'd128);

  // SIMD multiply step:
  wire signed [15:0] prod_0, prod_1, prod_2, prod_3;
  assign prod_0 =  ($signed(cmd_payload_inputs_0[7 : 0]) + InputOffset)
                 * $signed(cmd_payload_inputs_1[7 : 0]);
  assign prod_1 =  ($signed(cmd_payload_inputs_0[15: 8]) + InputOffset)
                 * $signed(cmd_payload_inputs_1[15: 8]);
  assign prod_2 =  ($signed(cmd_payload_inputs_0[23:16]) + InputOffset)
                 * $signed(cmd_payload_inputs_1[23:16]);
  assign prod_3 =  ($signed(cmd_payload_inputs_0[31:24]) + InputOffset)
                 * $signed(cmd_payload_inputs_1[31:24]);

  wire signed [31:0] sum_prods;
  assign sum_prods = prod_0 + prod_1 + prod_2 + prod_3;

  // Only not ready for a command when we have a response.
  assign cmd_ready = ~rsp_valid;

  always @(posedge clk) begin
    if (reset) begin
      rsp_payload_outputs_0 <= 32'b0;
      rsp_valid <= 1'b0;
    end else if (rsp_valid) begin
      // Waiting to hand off response to CPU.
      rsp_valid <= ~rsp_ready;
    end else if (cmd_valid) begin
      rsp_valid <= 1'b1;
      // Accumulate step:
      rsp_payload_outputs_0 <= |cmd_payload_function_id[9:3]
          ? 32'b0
          : rsp_payload_outputs_0 + sum_prods;
    end
  end
endmodule
```

Summary

- For details, please refer to my previous talk "Renode: a Swiss Army Knife for RISC-V development" at .Net Conf China 2021(Online) and the upcoming follow-up "Revisiting Renode as a Swiss Army Knife for RISC-V development".
- By now we still fail to run Renode(unmodified) on RPi4 via various virtualization technologies, but keep on trying some new methods and wait for the upcoming development boards.
- RISC-V support in Linux distribution are more and more friendly, e.g.:

```
[mydev@fedora /]$ dnf search riscv
================================================ Name & Summary Matched: riscv ================
binutils-riscv64-linux-gnu.aarch64 : Cross-build binary utilities for riscv64-linux-gnu
gcc-c++-riscv64-linux-gnu.aarch64 : Cross-build binary utilities for riscv64-linux-gnu
gcc-riscv64-linux-gnu.aarch64 : Cross-build binary utilities for riscv64-linux-gnu
qemu-user-static-riscv.aarch64 : QEMU user mode emulation of riscv qemu targets static build
```

- ...

IV. Software stack for Edge AI

1) VEDLIoT

- https://vedliot.eu/

Very Efficient Deep Learning in IoT project.

- https://vedliot.eu/project-summary/
- https://antmicro.com/blog/2021/06/kenning-edge-ai-framework/

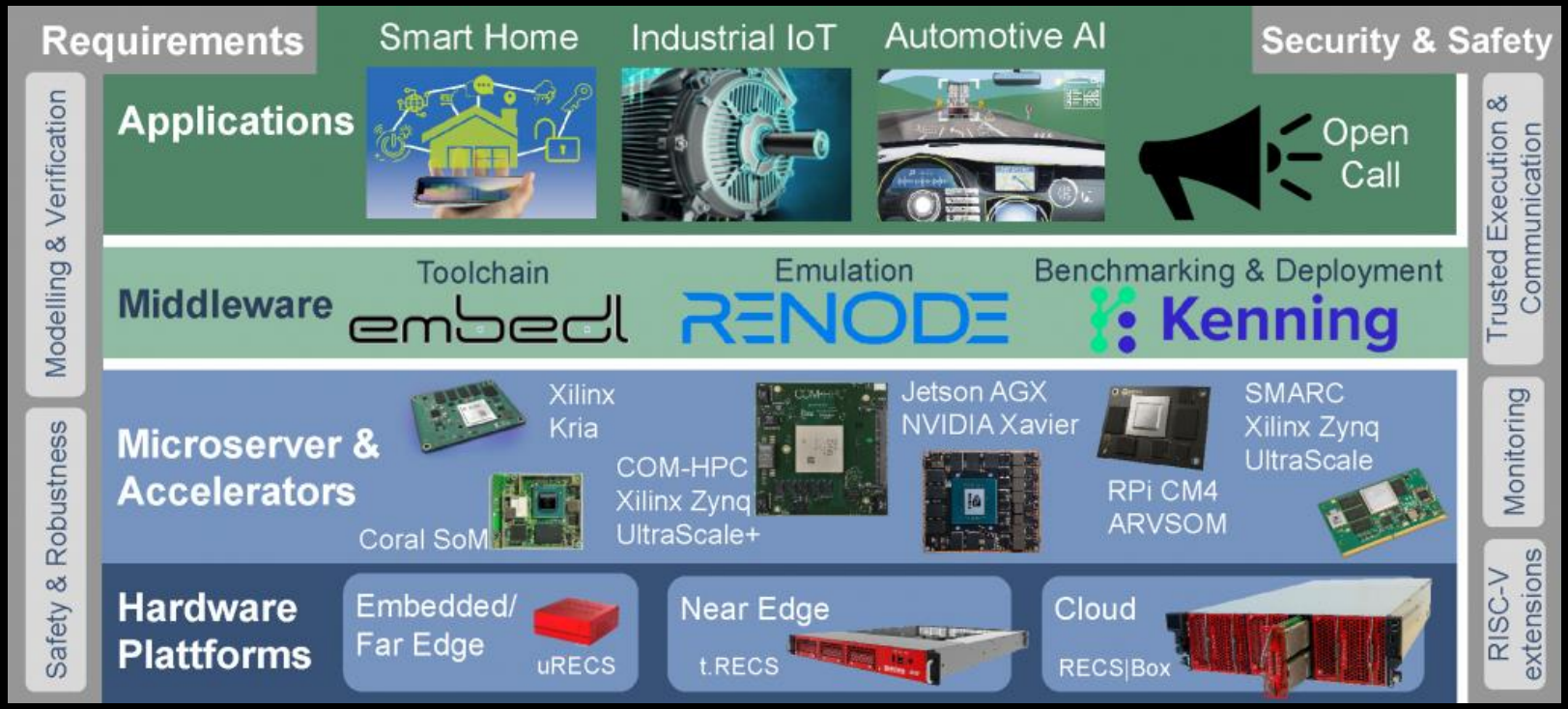
> Kenning's origin lies in a benchmarking framework implemented during our work on Very Efficient Deep Learning in IoT project. VEDLIoT is focused on building platform-independent AI for edge applications and exploring RISC-V (mentioned earlier in our blog in Very Efficient Deep Learning in IoT project with RISC-V and Renode.) That's another instance of our constant efforts on many fronts towards more open AI solutions in general, aligning with our work around open source FPGA tooling, TF Lite Micro, Renode, CFU extensions for RISC-V and with other global leaders at CHIPS Alliance.

- https://vedliot.eu/hardware/
- https://vedliot.eu/accelerators/
- …

Architecture & Design

- https://vedliot.eu/vedliot-platform/

VEDLIoT aims to provide a **modular, scalable hardware platform** for the next generation of AIoT devices (Artificial Intelligence of Things), covering the full spectrum from the cloud via near edge to far edge or embedded applications. Based on a modular approach, the platform provides a flexible and scalable architecture that supports the full spectrum of heterogeneous processing technology and supports regular CPU technology based on x86, ARM, RISC-V, and specialised accelerator ASICs.



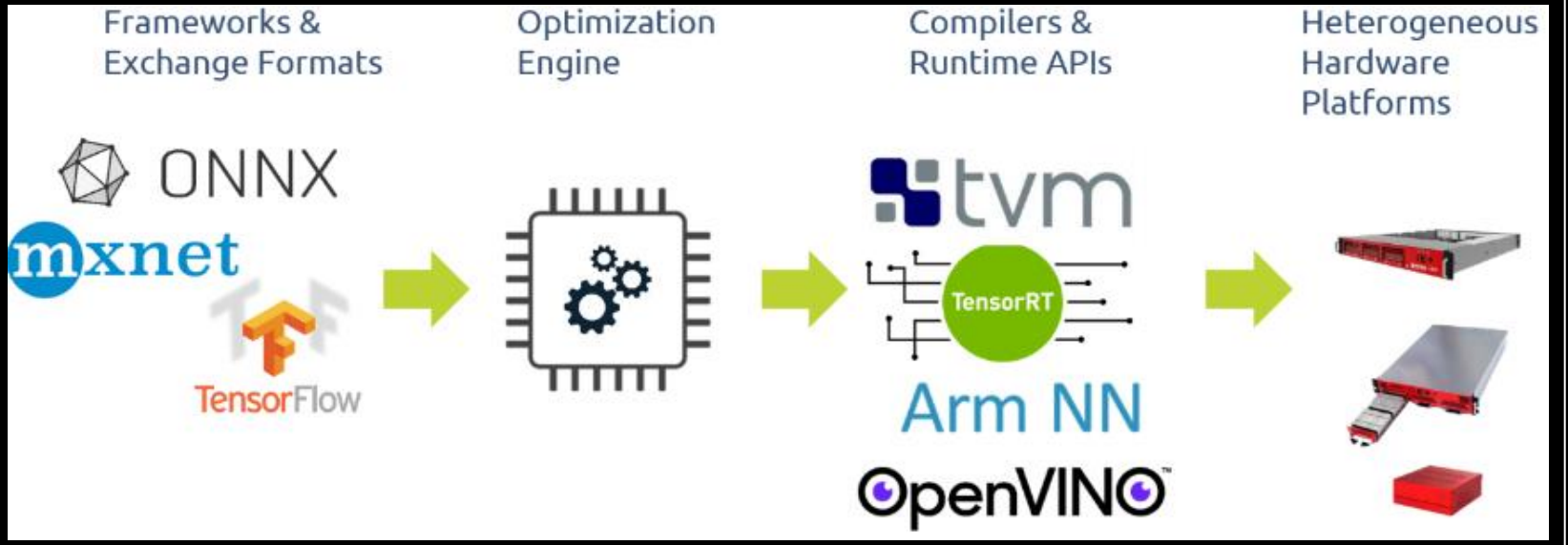
Toolchain

- **https://vedliot.eu/toolchain/**

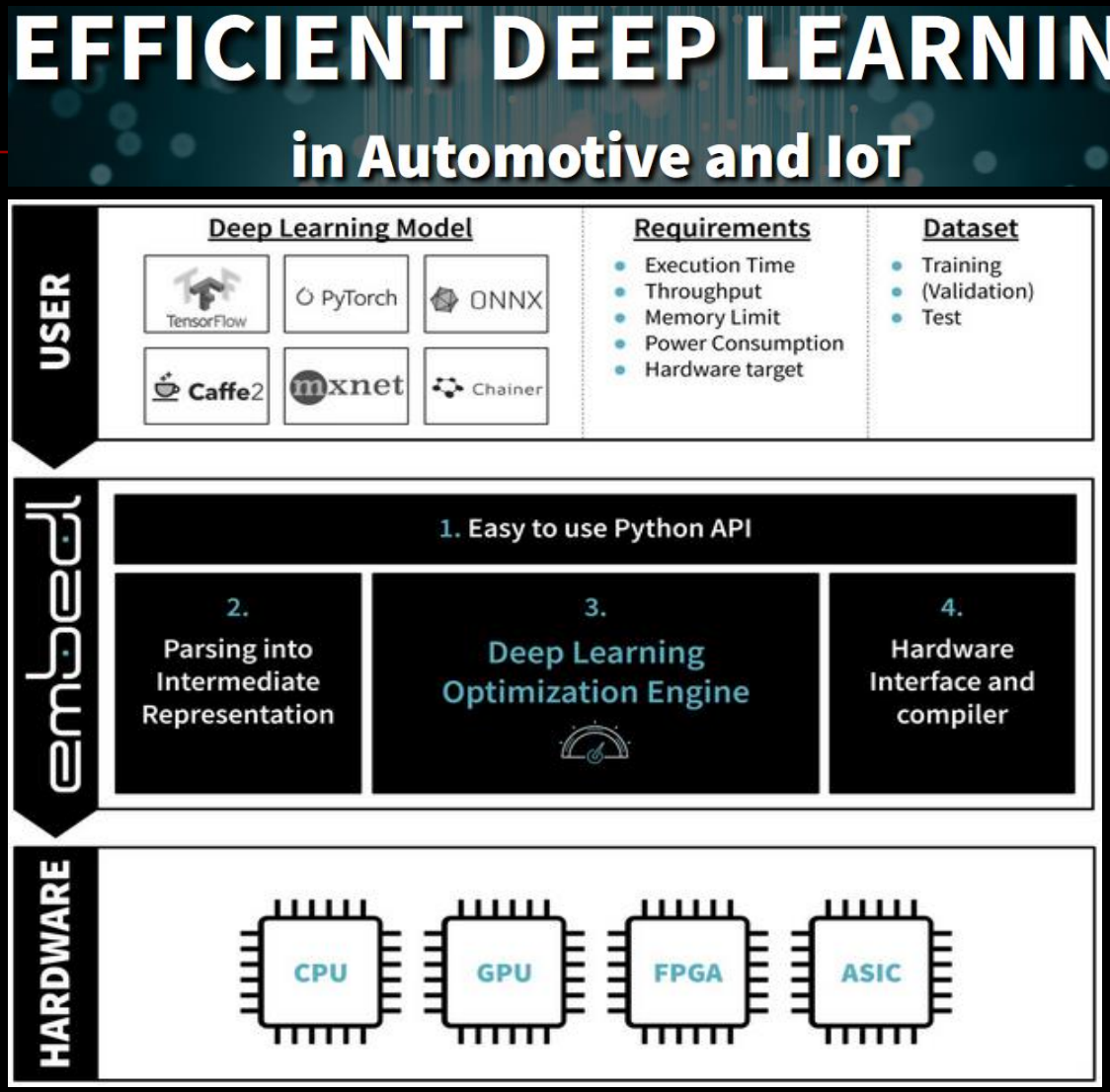
The primary goal of the VEDLIoT toolchain is the optimisation of existing Deep Neural Networks towards a specific target hardware using the EmbeDL optimiser technology.

The Kenning framework aims to create easy-to-use deployment flows and runtimes for DNN applications for various IoT hardware.

Another strong focus is on realizing hardened IoT platforms with built-in security features which can significantly enhance applications' security trustworthiness and safety.

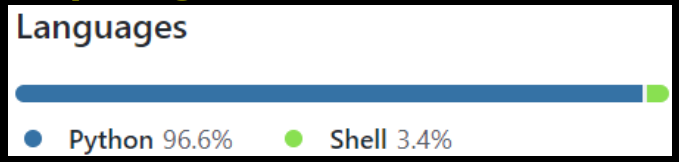


- EmbeDL
  https://www.embedl.com/

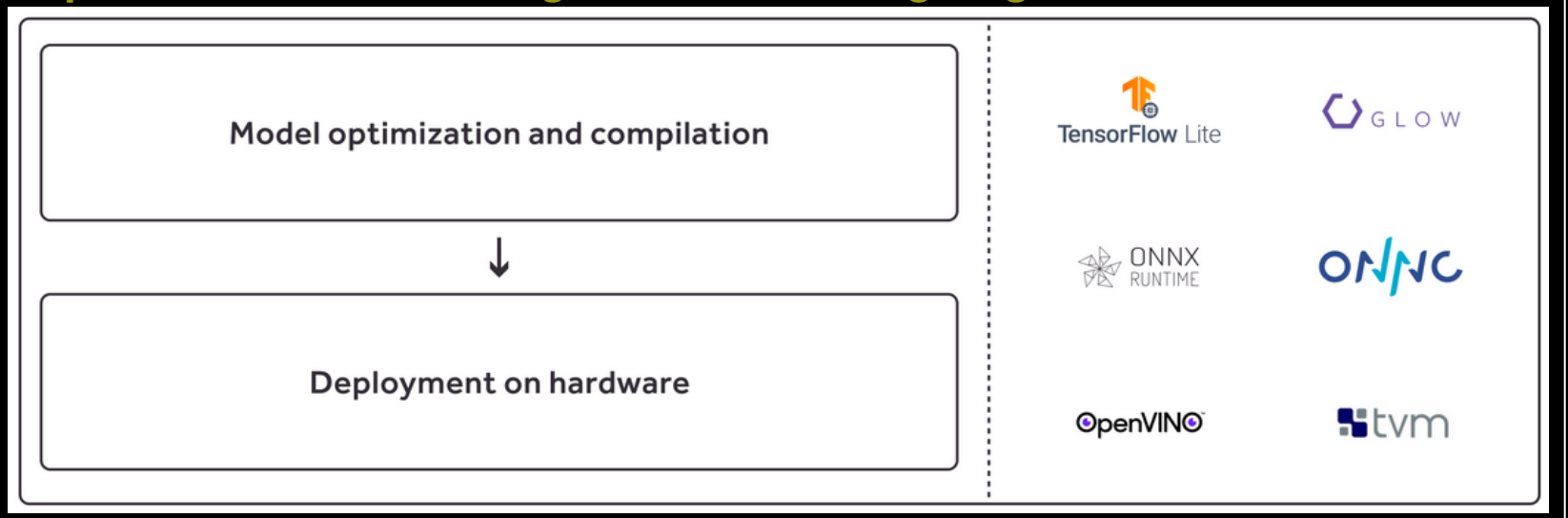


1.1 Kenning Framework

- https://antmicro.github.io/kenning/
  A framework for creating deployment flows and runtimes for Deep Neural Network applications on various target hardware. //New
  ~~Deploying deep learning models on the edge.~~ //Old
- https://github.com/antmicro/kenning/



- https://antmicro.com/blog/2021/06/kenning-edge-ai-framework/

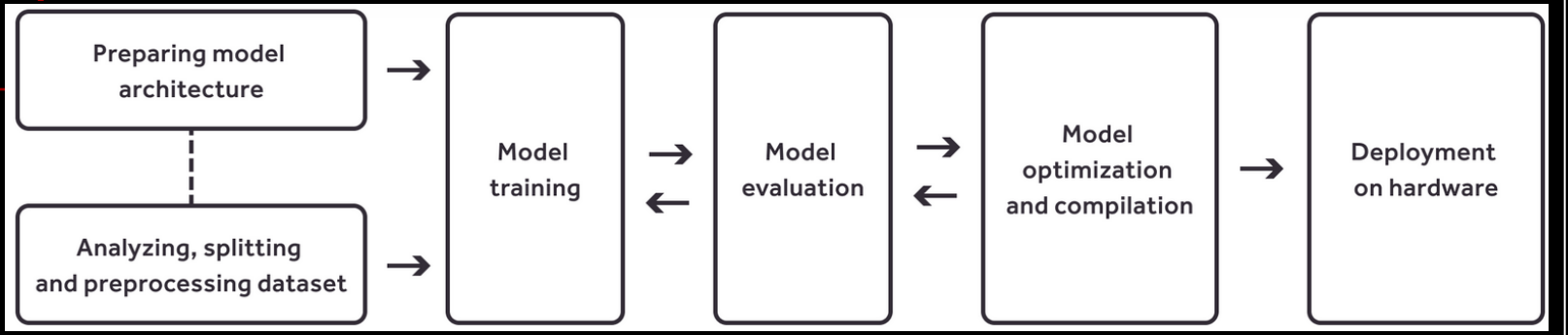


- https://antmicro.com/blog/2021/11/kenning-runtime/
- ...

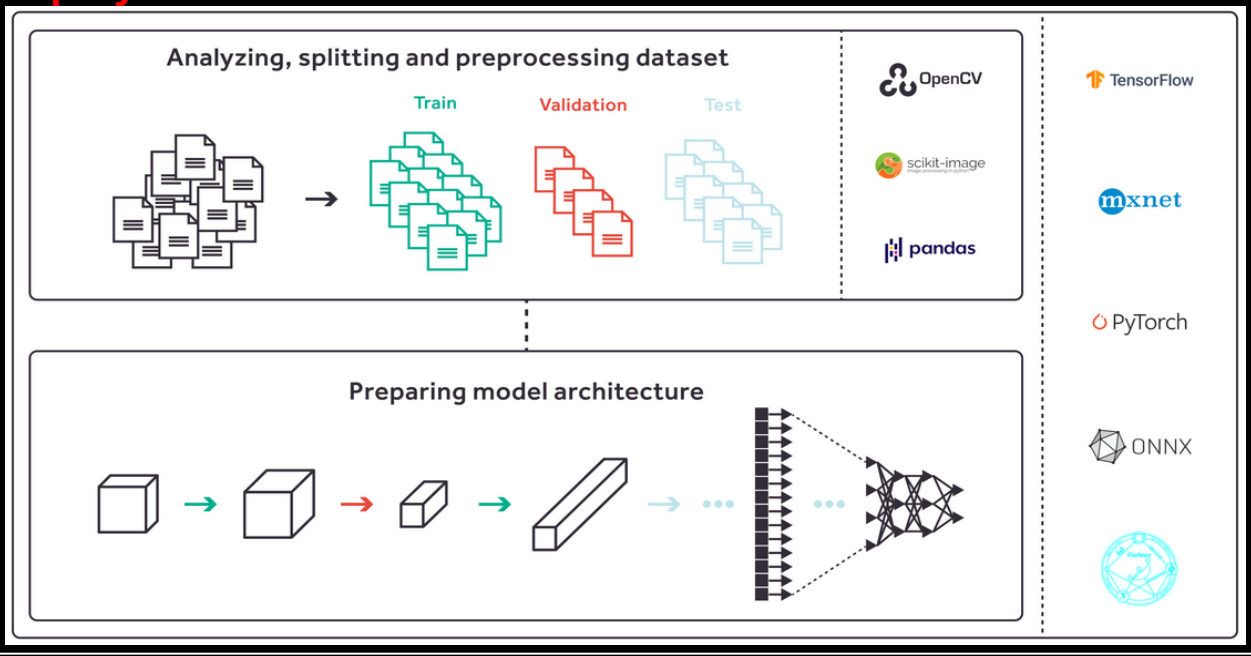
Kenning for Edge AI

- https://antmicro.com/blog/2021/06/kenning-edge-ai-framework/

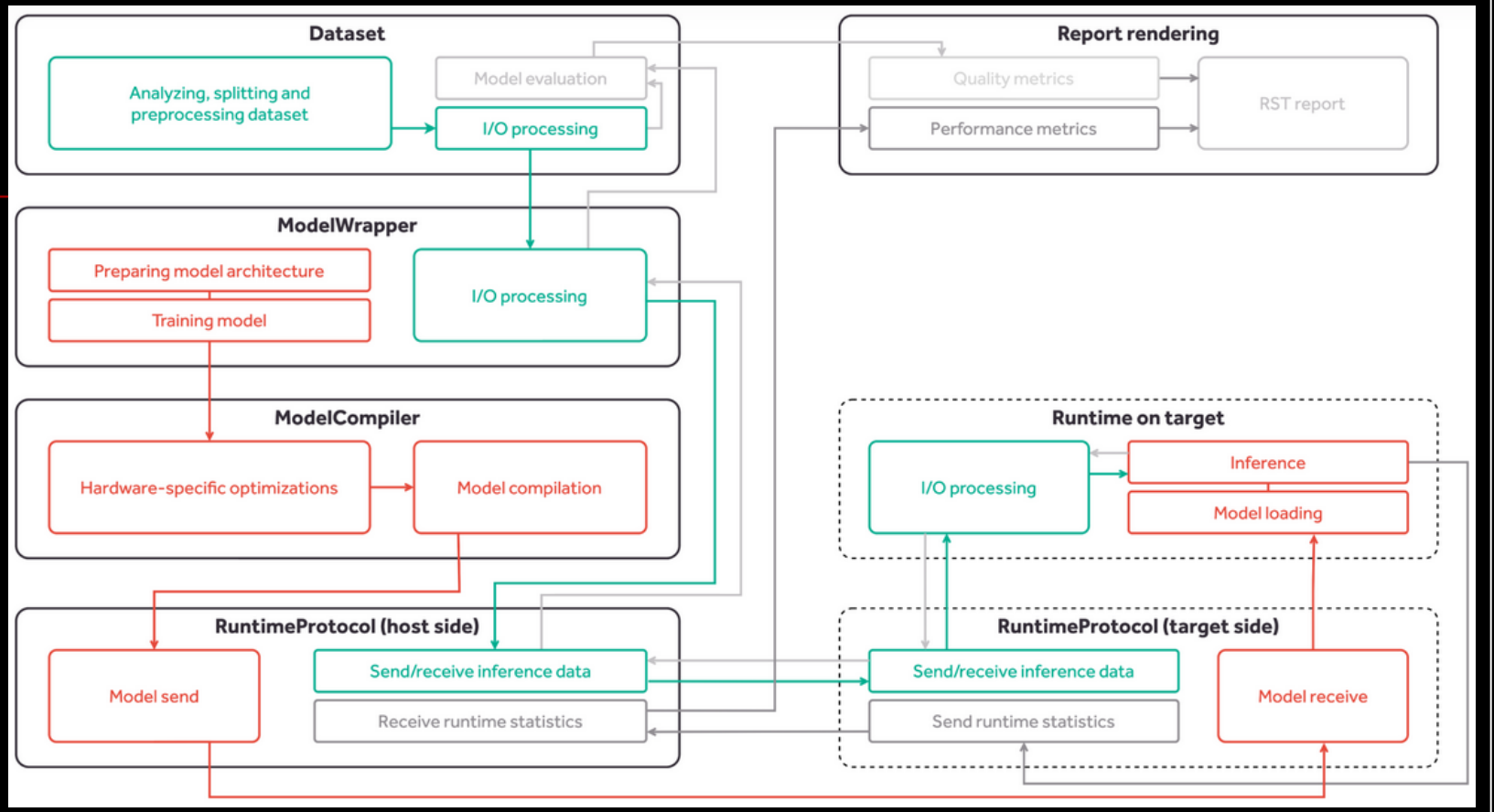
Pipeline



Deployment stack



Workflow and Classes



2) microTVM

- https://discuss.tvm.apache.org/t/coordination-of-risc-v-integration-in-tvm/13133

Src

- $SRC_TVM/python/tvm/target/target.py

```
def riscv_cpu(model="sifive-u54", options=None):
    """Returns a RISC-V CPU target.
    Default: sifive-u54 rv64gc

    Parameters
    ----------
    model: str
        CPU name.
    options : str or list of str
        Additional options
    """
    trans_table = {
        "sifive-e31": [
            "-model=sifive-e31",
            "-mtriple=riscv32-unknown-linux-gnu",
            "-mcpu=sifive-e31",
            "-mabi=ilp32",
            # cc: riscv64-unknown-linux-gnu-g++ -march=rv32imac -mabi=ilp32 -mcpu=sifive-e31
        ],
        "sifive-e76": [
            "-model=sifive-e76",
            "-mtriple=riscv32-unknown-linux-gnu",
            "-mcpu=sifive-e76",
            "-mabi=ilp32",
            # cc: riscv64-unknown-linux-gnu-g++ -march=rv32imafc -mabi=ilp32 -mcpu=sifive-e76
        ],
        "sifive-u54": [
            "-model=sifive-u54",
            "-mtriple=riscv64-unknown-linux-gnu",
            "-mcpu=sifive-u54",
            "-mabi=lp64d",
            # cc: riscv64-unknown-linux-gnu-g++ -march=rv64gc -mabi=lp64d -mcpu=sifive-u54
        ],
        "sifive-u74": [
            "-model=sifive-u74",
            "-mtriple=riscv64-unknown-linux-gnu",
            "-mcpu=sifive-u74",
            "-mabi=lp64d",
            # cc: riscv64-unknown-linux-gnu-g++ -march=rv64gc -mabi=lp64d -mcpu=sifive-u74
        ],
    }
```

$SRC_TVM/apps/microtvm/zephyr


```
[mydev@fedora tvm-main]$ tree -d apps/microtvm
apps/microtvm
├── arduino
│   └── template_project
│       ├── crt_config
│       ├── src
│       │   ├── example_project
│       │   └── host_driven
│       └── tests
├── cmsisnn
│   ├── include
│   └── src
├── ethosu
│   ├── include
│   └── src
├── reference-vm
│   ├── base-box
│   └── scripts
├── zephyr
│   └── template_project
│       ├── crt_config
│       ├── fvp-hack
│       ├── qemu-hack
│       └── src
│           ├── aot_standalone_demo
│           └── host_driven
│               └── fvp
└── zephyr_cmsisnn
    ├── include
    ├── model
    └── src
```

```
[mydev@fedora tvm-main]$ grep -ir "riscv" apps/microtvm/zephyr
apps/microtvm/zephyr/template_project/boards.json:    "qemu_riscv32": {
apps/microtvm/zephyr/template_project/boards.json:        "board": "qemu_riscv32",
apps/microtvm/zephyr/template_project/boards.json:    "qemu_riscv64": {
apps/microtvm/zephyr/template_project/boards.json:        "board": "qemu_riscv64",
apps/microtvm/zephyr/template_project/microtvm_api_server.py:            "qemu_riscv32",
apps/microtvm/zephyr/template_project/microtvm_api_server.py:            "qemu_riscv64",
[mydev@fedora tvm-main]$
```

3) Ray

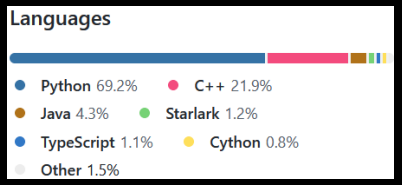
3.1 Overview

- **https://www.ray.io/**

  An open source project that makes it ridiculously simple to scale any compute-intensive Python workload—from deep learning to production model serving. With a rich set of libraries and integrations built on a flexible distributed execution framework, Ray makes distributed computing easy and accessible to every engineer.

- **https://github.com/ray-project/ray/**

  A unified framework for scaling AI and Python applications. Ray consists of a core distributed runtime and a toolkit of libraries (Ray AIR) for accelerating ML workloads.

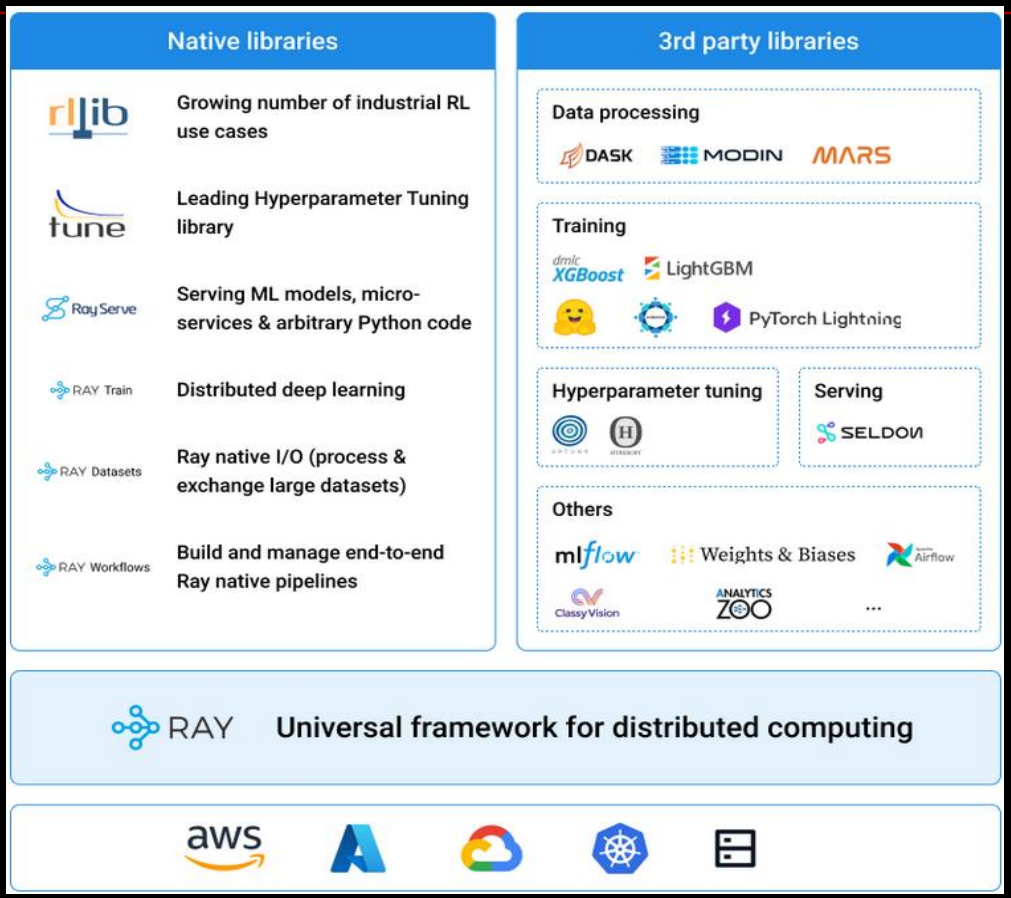


- **https://rise.cs.berkeley.edu/projects/ray/**

  Ray is a high-performance distributed execution framework targeted at large-scale machine learning and reinforcement learning applications. It achieves scalability and fault tolerance by abstracting the control state of the system in a global control store and keeping all other components stateless. It uses a shared-memory distributed object store to efficiently handle large data through shared memory, and it uses a bottom-up hierarchical scheduling architecture to achieve low-latency and high-throughput scheduling. It uses a lightweight API based on dynamic task graphs and actors to express a wide range of applications in a flexible manner.

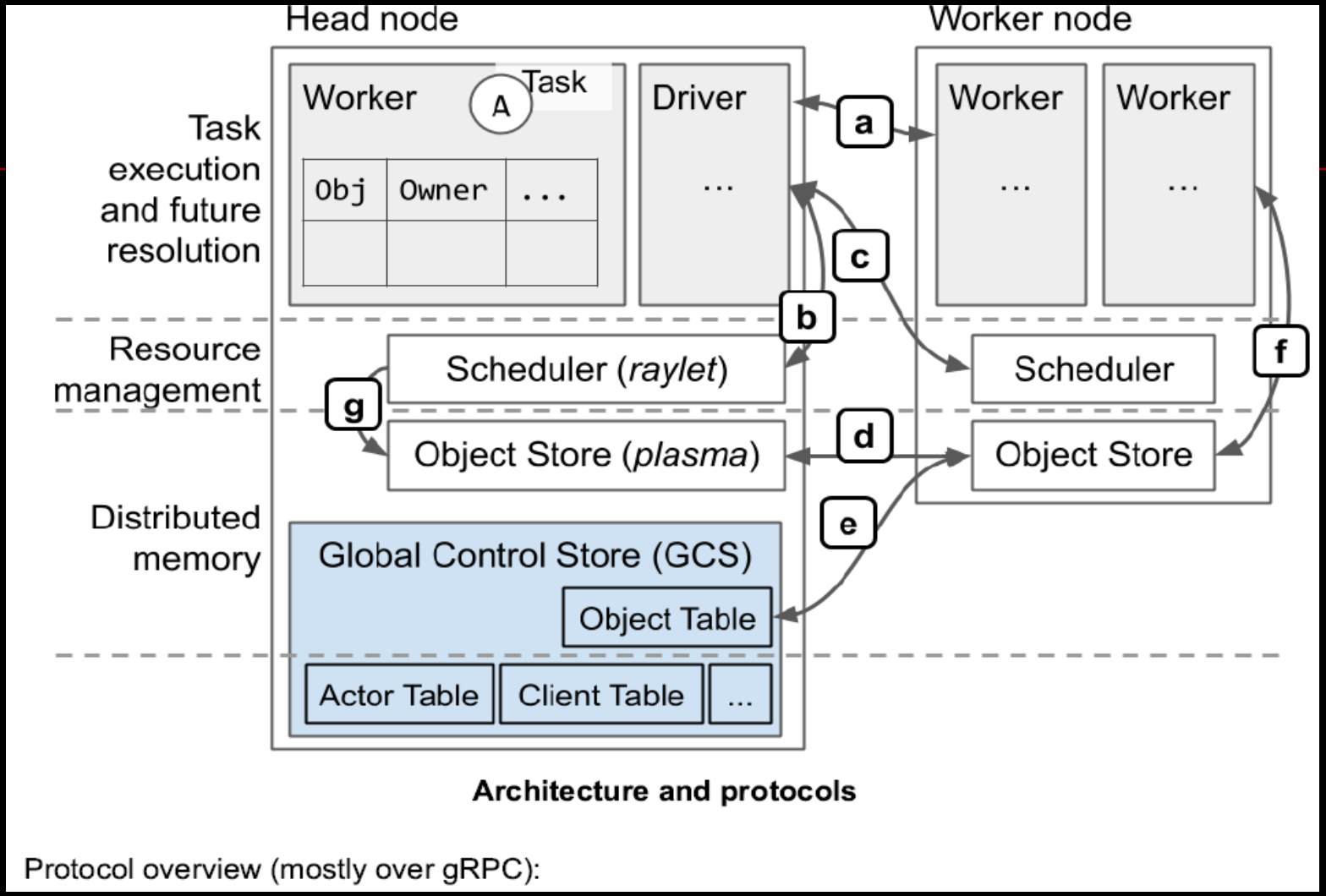
- **https://docs.ray.io/en/latest/index.html**

Ray is an open-source project developed at UC Berkeley RISE Lab. As a general-purpose and universal distributed compute framework, you can flexibly run any compute-intensive Python workload — from distributed training or hyperparameter tuning to deep reinforcement learning and production model serving.



- **https://www.anyscale.com/**
- ...

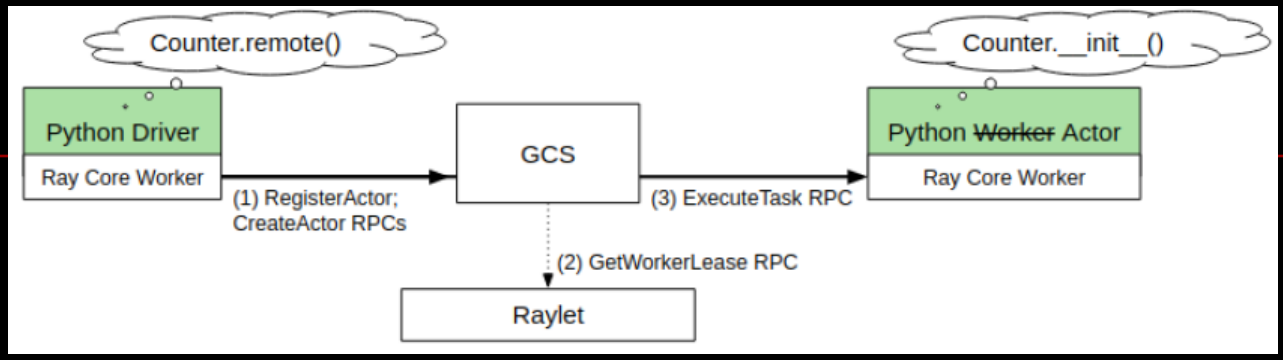
Architecture & Design



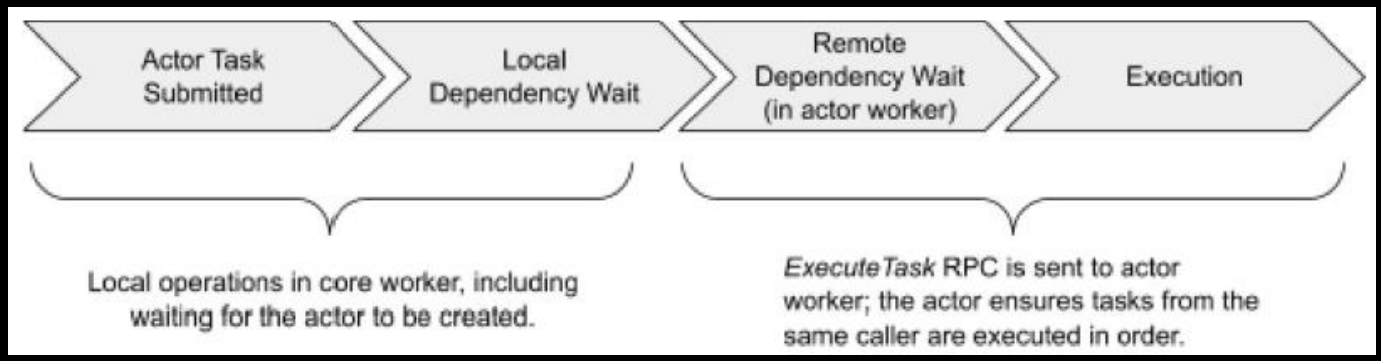
Source: Ray Architecture whitepaper 1.0

- **Actor Management**

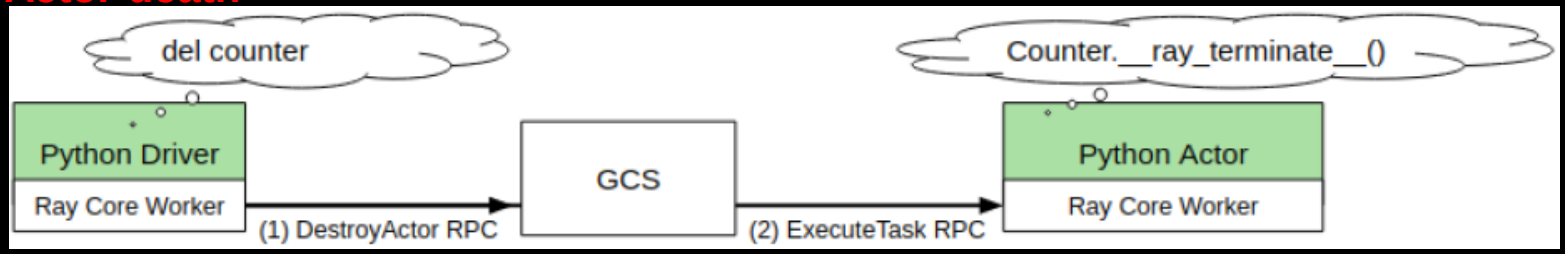
Actor creation



Actor task execution



Actor death



- **Ray Design Patterns 1.0**

Apache Arrow

- https://en.wikipedia.org/wiki/Apache_Arrow

Apache Arrow is a language-agnostic software framework for developing data analytics applications that process columnar data. It contains a standardized column-oriented memory format that is able to represent flat and hierarchical data for efficient analytic operations on modern CPU and GPU hardware.[3][4][5][6][7] This reduces or eliminates factors that limit the feasibility of working with large sets of data, such as the cost, volatility, or physical constraints of dynamic random-access memory.[8]

Interoperability [edit]

Arrow can be used with Apache Parquet, Apache Spark, NumPy, PySpark, pandas and other data processing libraries. The project includes native software libraries written in C, C++, C#, Go, Java, JavaScript, Julia, MATLAB, Python, R, Ruby, and Rust. Arrow allows for zero-copy reads and fast data access and interchange without serialization overhead between these languages and systems.[3]

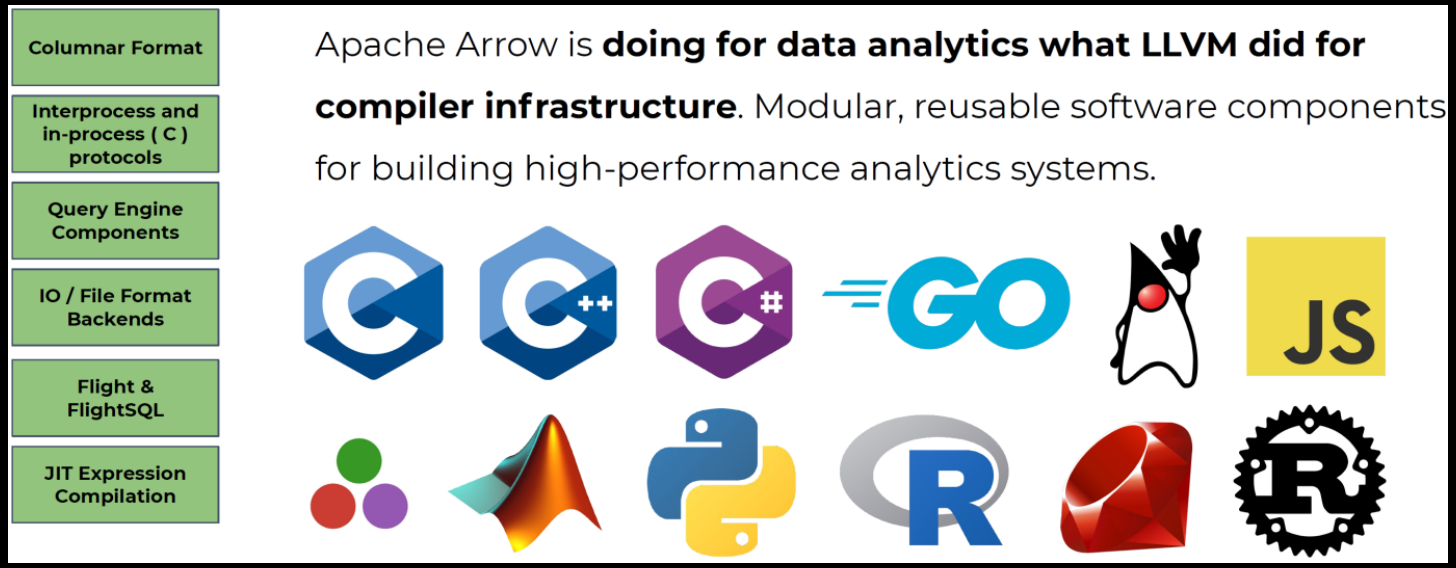
Applications [edit]

Arrow has been used in diverse domains, including analytics,[9] genomics,[10][8] and cloud computing.[11]

Comparison to Apache Parquet and ORC [edit]

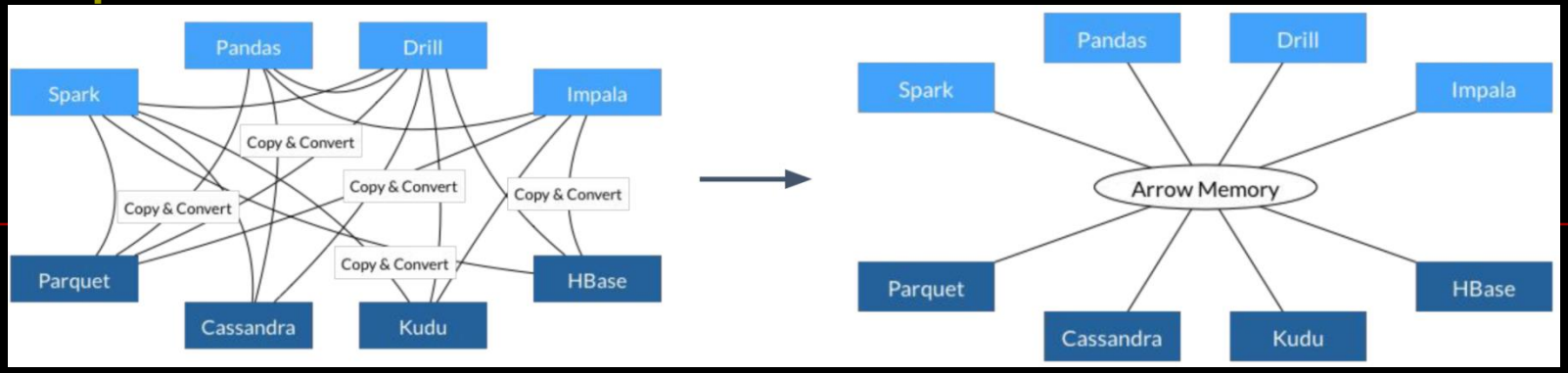
Apache Parquet and Apache ORC are popular examples of on-disk columnar data formats. Arrow is designed as a complement to these formats for processing data in-memory.[12] The hardware resource engineering trade-offs for in-memory processing vary from those associated with on-disk storage.[13] The Arrow and Parquet projects includes libraries that allow for reading and writing data between the two formats.[14]

- https://arrow.apache.org/
- Features



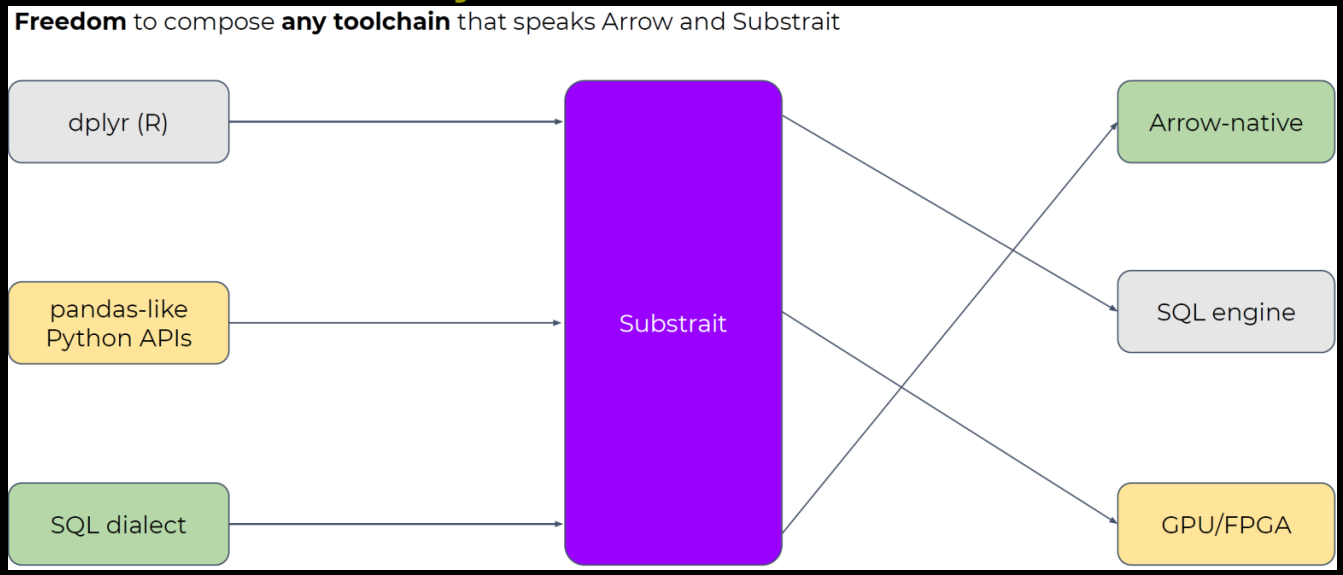
Source: https://www.slideshare.net/wesm/apache-arrow-high-performance-columnar-data-framework

- ## Simplified data access with the Arrow columnar format



Source: https://www.slideshare.net/wesm/apache-arrow-high-performance-columnar-data-framework

- ## LEGO for data ecosystem

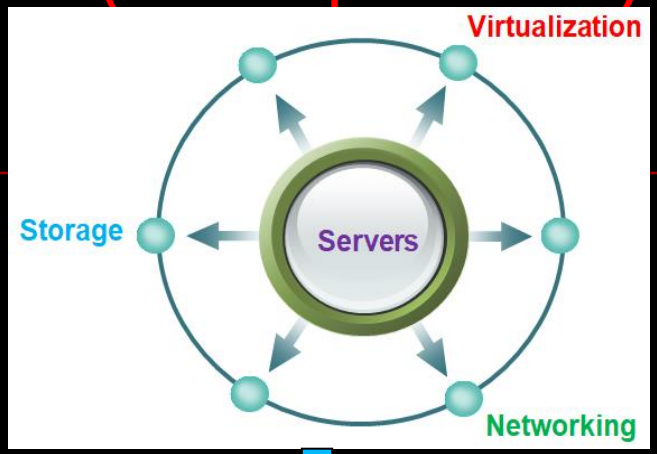


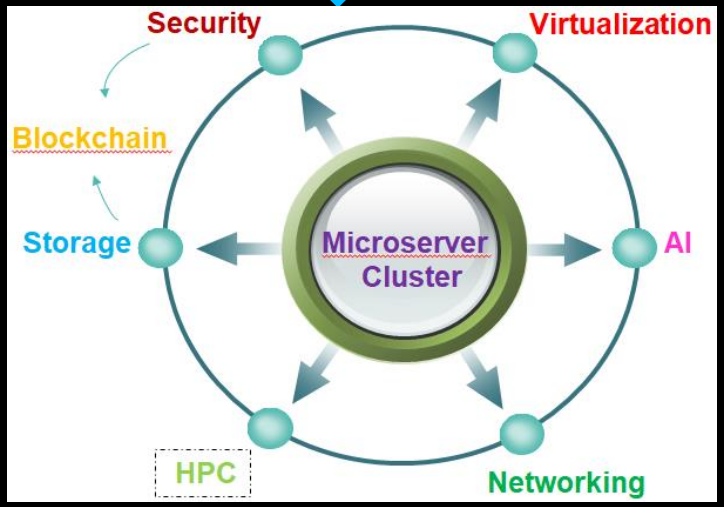
Source: https://www.slideshare.net/wesm/apache-arrow-high-performance-columnar-data-framework

- ## https://github.com/apache/arrow
- ...

Ray for Hyper-Converged Infrastructure

- Trend (from our point of view)





RAY

- Please look forward to our new talk "Ray as a universal infrastructure for distributed computing"…

V. Rethinking

1) PYNQ

- http://www.pynq.io/

PYNQ is an open-source project from Xilinx® that makes it easier to use Xilinx platforms.
Using the Python language and libraries, designers can exploit the benefits of programmable logic and microprocessors to build more capable and exciting electronic systems.
PYNQ can be used with *Zynq*, *Zynq UltraScale+*, *Zynq RFSoC*, *Alveo* accelerator boards and AWS-F1 to create high performance applications with:

- parallel hardware execution
- high frame-rate video processing
- hardware accelerated algorithms
- real-time signal processing
- high bandwidth IO
- low latency control

- Who is PYNQ for?

PYNQ is intended to be used by a wide range of designers and developers including:

- Software developers who want to take advantage of the capabilities of Xilinx platforms without having to use ASIC-style design tools to design hardware.
- System architects who want an easy software interface and framework for rapid prototyping and development of their Zynq, Alveo and AWS-F1 design.
- Hardware designers who want their designs to be used by the widest possible audience.

- http://www.pynq.io/board.html
- https://pynq.readthedocs.io/en/latest/

- **Key technologies**

Jupyter Notebook is a browser based interactive computing environment. Jupyter notebook documents can be created that include live code, interactive widgets, plots, explanatory text, equations, images and video.

A PYNQ enabled board can be easily programmed in Jupyter Notebook using Python. Using Python, developers can use hardware libraries and *overlays* on the programmable logic. *Hardware libraries*, or *overlays*, can speed up software running on a Zynq or Alveo board, and customize the hardware platform and interfaces.

PYNQ can be delivered in two ways; as a bootable Linux image for a Zynq board, which includes the *pynq* Python package, and other open-source packages, or as a Python package for an Alveo or AWS-F1 host computer.

python

jupyter

Linux

What software do I need?

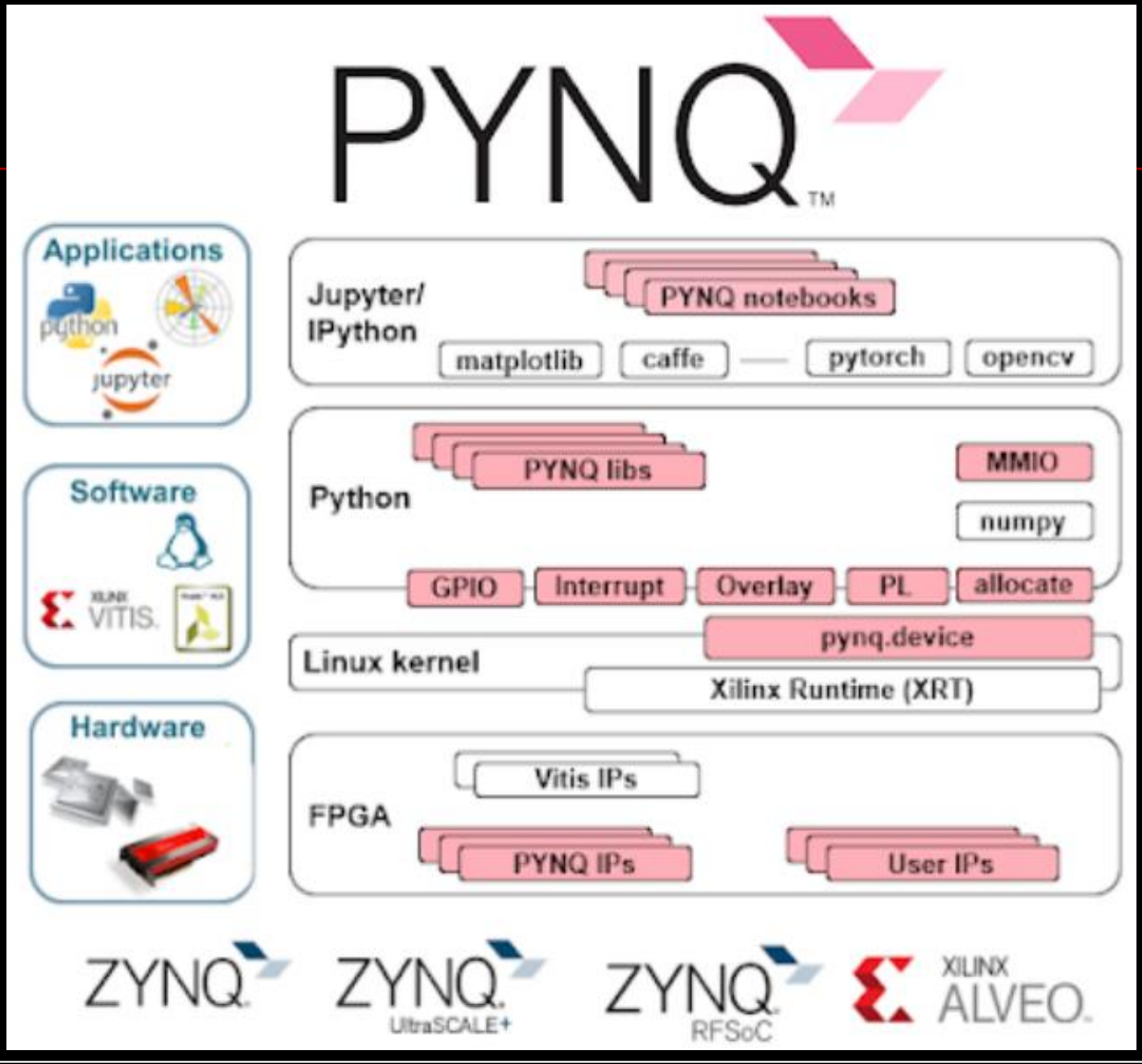
Jupyter notebook runs in a web browser. Only a **compatible web browser** is needed to start programming PYNQ with Python.

For higher performance, you can also use C/C++ with Python and PYNQ. The **Xilinx SDK software development environment** is available for free. You can also use third party software development tools.

New hardware libraries and overlays can be created using standard Xilinx and third party hardware design tools.
The **free WebPACK version of Xilinx Vivado** can be used with a wide range of Zynq boards **Vitis** and **Vitis open-source Accelerated Libraries** are free, and can be used for Alveo/AWS-F1.

- **https://github.com/Xilinx/PYNQ**
- **https://pynq.readthedocs.io/en/latest/pynq_overlays.html**
- **http://www.pynq.io/community.html**
- **…**

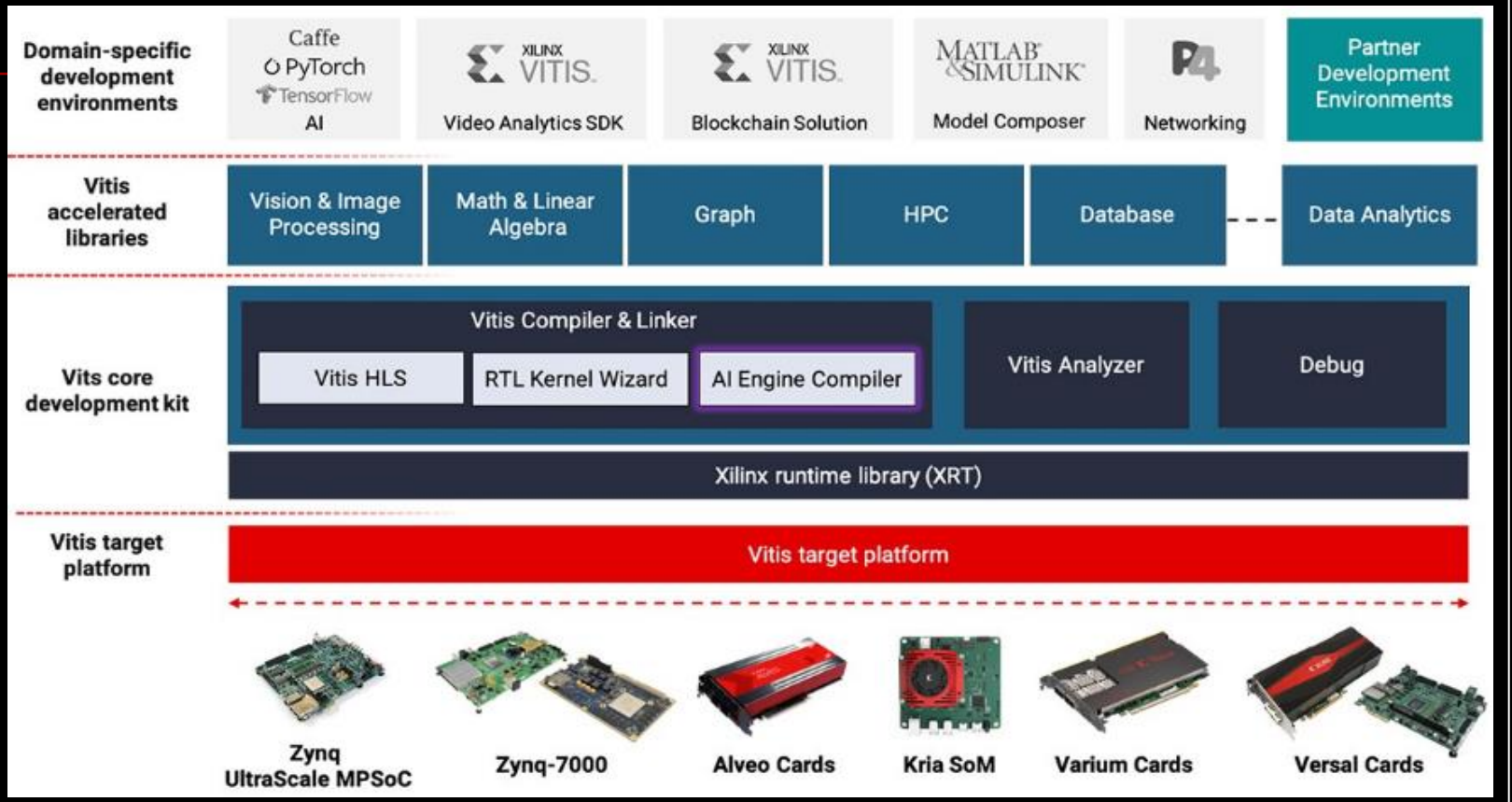
Architecture & Design



1.1 Vitis

- https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html

Unified Software Platform.



The Vitis unified software platform includes:

- Comprehensive core development kit to seamlessly build accelerated applications
- Rich set of hardware-accelerated open-source libraries optimized for Xilinx FPGA and Versal ACAP hardware platforms
- Plug-in domain-specific development environments enabling development directly in familiar, higher-level frameworks
- Growing ecosystem of hardware-accelerated partner libraries and pre-built applications
- Vitis Model Composer, a Model-Based Design tool that enables rapid design exploration and verification within the MathWorks MATLAB® and Simulink® environment and accelerates the path to production on Xilinx devices.
- Vitis Networking P4 allows for the creation of soft-defined networks. VitisNetP4 data plane builder generates systems that can be programmed for a wide range of packet processing functions from simple packet classification to complex packet editing.

https://xilinx.github.io/Vitis-Tutorials/2022-1/build/html/index.html

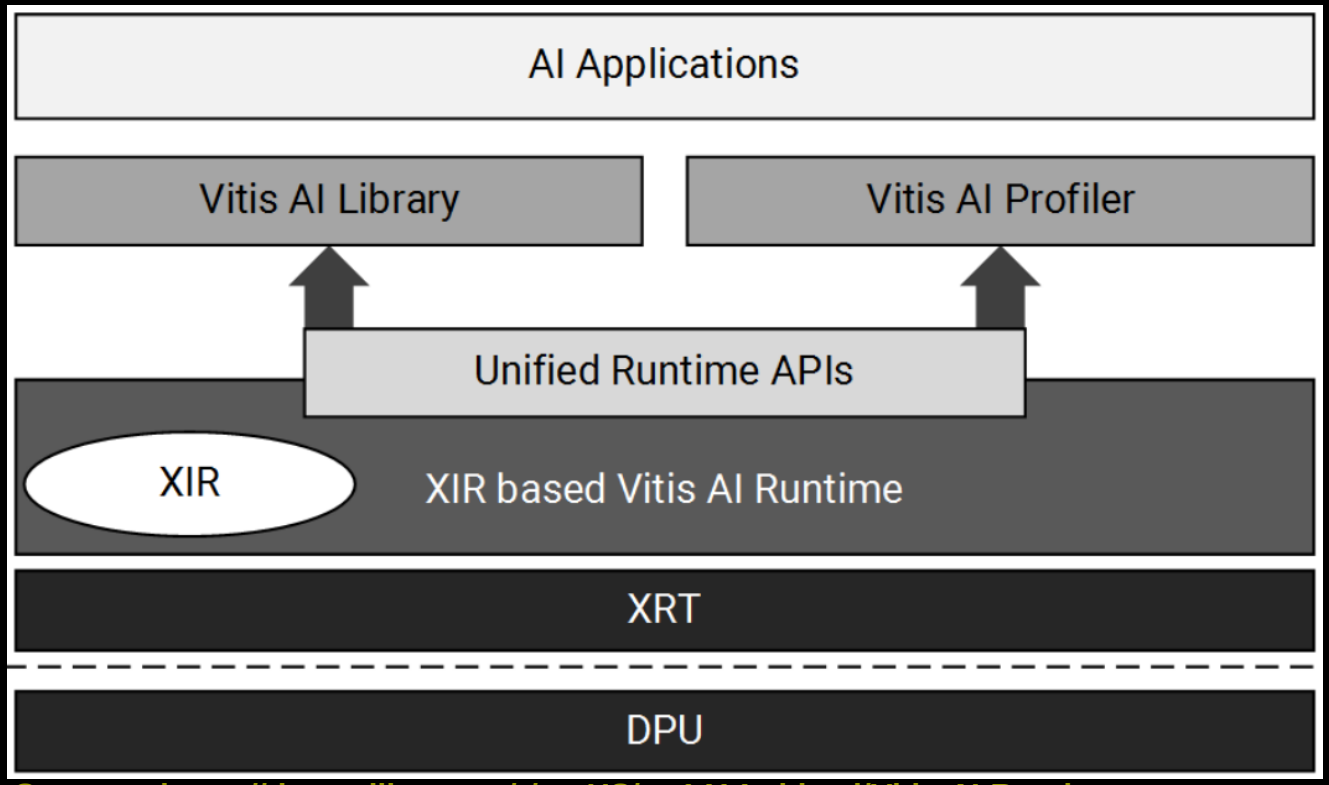
The Vitis unified software platform enables the development of embedded software and accelerated applications on heterogeneous Xilinx® platforms including FPGAs, SoCs, and Versal® ACAPs. It provides a unified programming model for accelerating Edge, Cloud, and Hybrid computing applications.

Leverage integration with high-level frameworks, develop in C, C++, or Python using accelerated libraries or use RTL-based accelerators & low-level runtime APIs for more fine-grained control over implementation — Choose the level of abstraction you need.

1.1.1 Vitis AI

- https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html
  Optimal Artificial Intelligence Inference from Edge to Cloud.

Vitis AI Runtime Stack



Source: https://docs.xilinx.com/r/en-US/ug1414-vitis-ai/Vitis-AI-Runtime

XIR

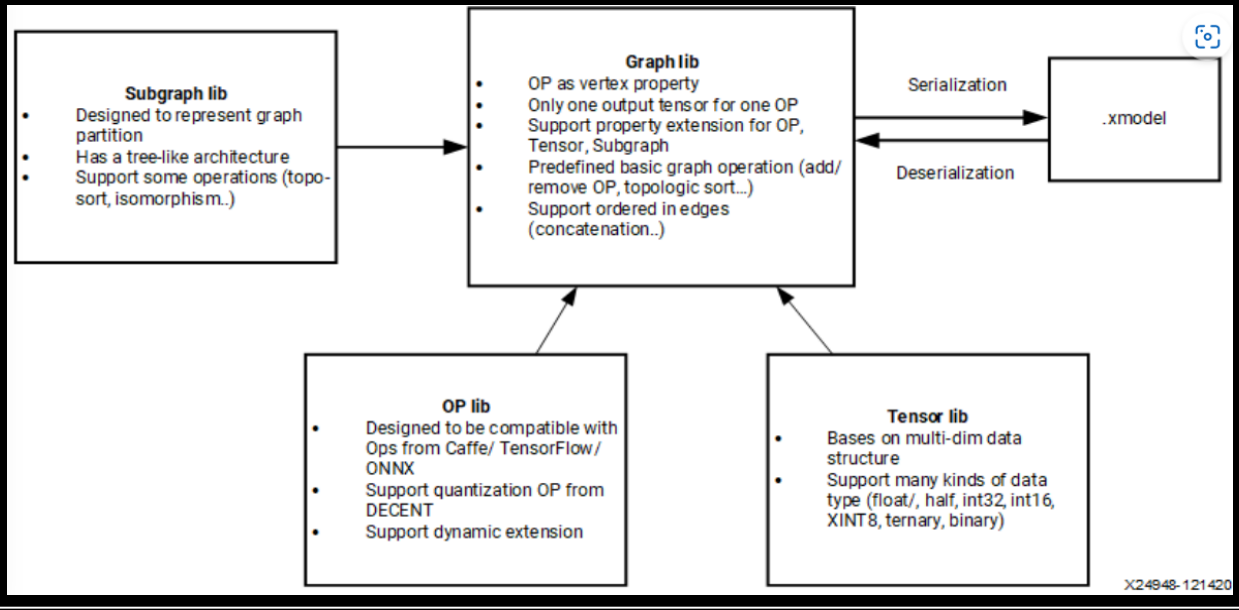
- https://docs.xilinx.com/r/en-US/ug1414-vitis-ai/XIR

XIR includes the Op, Tensor, Graph, and Subgraph libraries, which provide a clear and flexible representation of the computational graph. XIR has in-memory format and file format for different usage. The in-memory format XIR is a graph object and the file format is an xmodel. A graph object can be serialized to an XMODEL while an XMODEL can be deserialized to a graph object.

In the Op library, there is a well-defined set of operators to cover the popular deep learning frameworks, e.g., TensorFlow, PyTorch and Caffe[1], and all of the built-in DPU operators. This enhances the expression ability and achieves one of the core goals, which is eliminating the difference between these frameworks and providing a unified representation for users and developers.

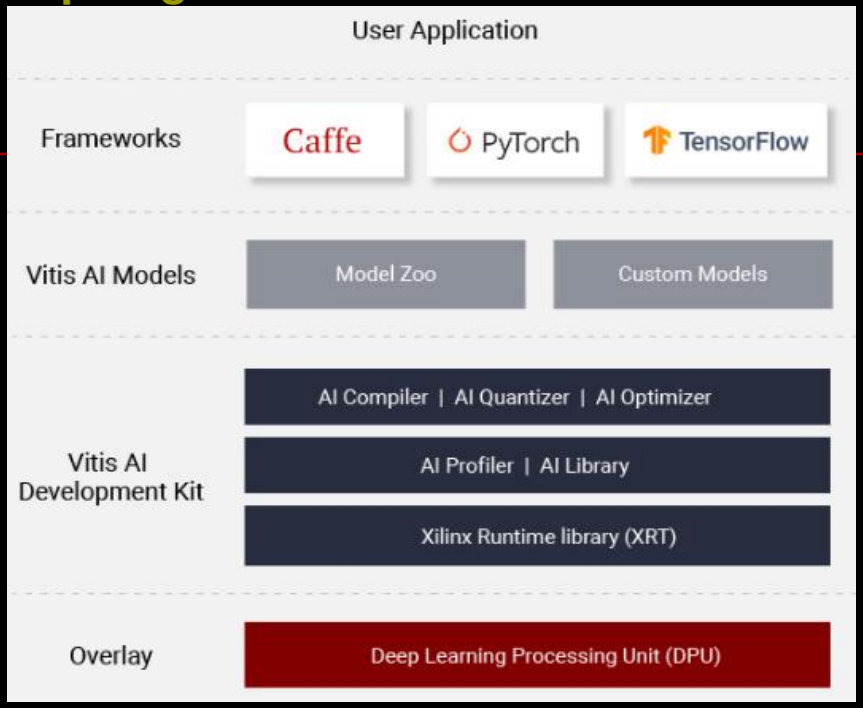
XIR also provides Python APIs named PyXIR, which enables Python users to fully access the XIR in a Python environment, e.g., co-develop and integrate users' Python projects with the current XIR-based tools without having to perform a huge amount of work to fix the gap between different languages.

XIR Based Flow



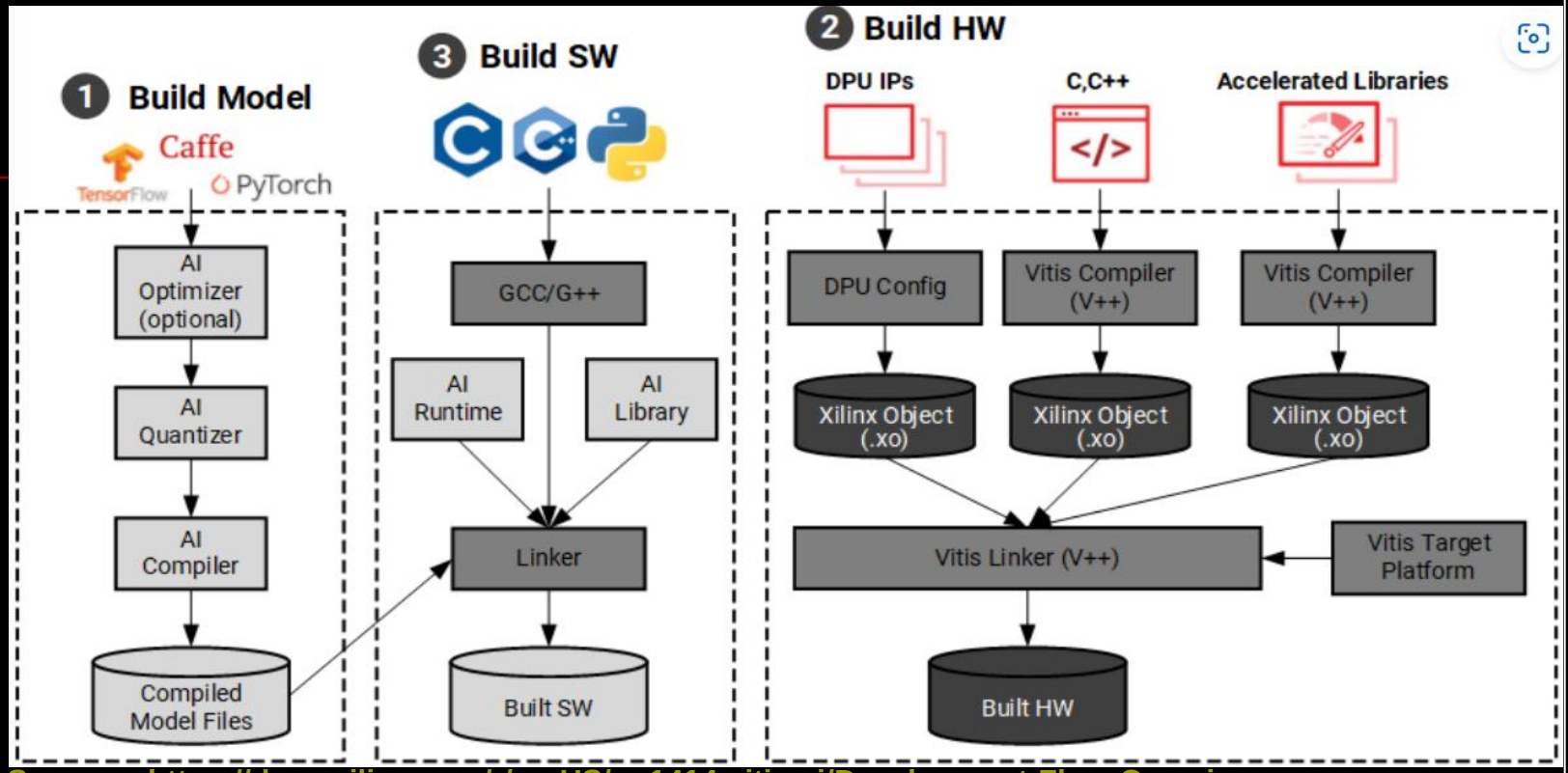
Architecture & Design

- https://github.com/Xilinx/Vitis-AI



- **AI Model Zoo** - A comprehensive set of pre-optimized models that are ready to deploy on Xilinx devices.
- **AI Optimizer** - An optional model optimizer that can prune a model by up to 90%. It is separately available with commercial licenses.
- **AI Quantizer** - A powerful quantizer that supports model quantization, calibration, and fine tuning.
- **AI Compiler** - Compiles the quantized model to a high-efficient instruction set and data flow.
- **AI Profiler** - Perform an in-depth analysis of the efficiency and utilization of AI inference implementation.
- **AI Library** - Offers high-level yet optimized C++ APIs for AI applications from edge to cloud.
- **DPU** - Efficient and scalable IP cores can be customized to meet the needs for many different applications.
  - For more details on the different DPUs available, refer to DPU Naming.

Development Flow



Source: https://docs.xilinx.com/r/en-US/ug1414-vitis-ai/Development-Flow-Overview

1.2 XRT

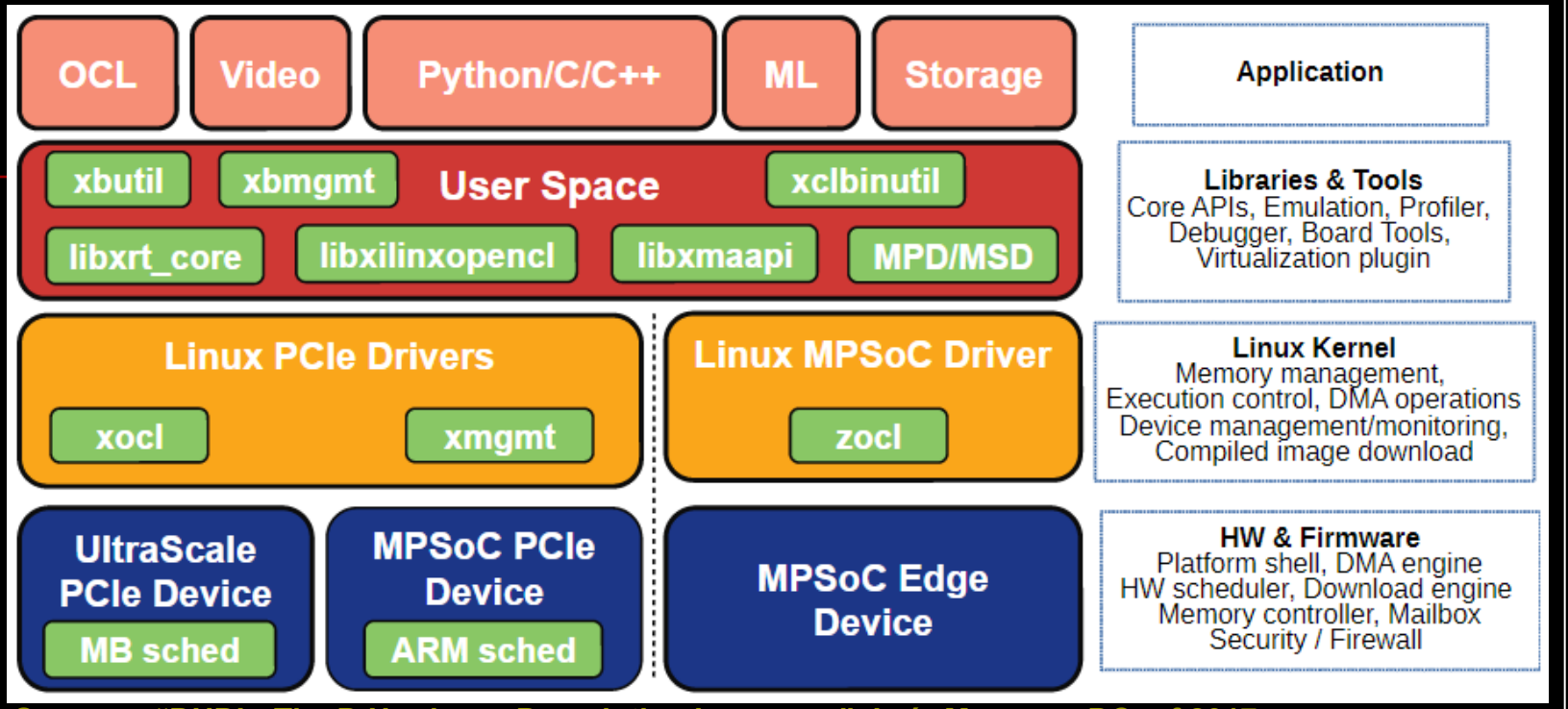
- https://xilinx.github.io/XRT/

  Xilinx Run Time for FPGA.

- https://xilinx.github.io/XRT/master/html/platforms.html

Xilinx Runtime library (XRT) is an open-source easy to use software stack that facilitates management and usage of FPGA/ACAP devices. Users use familiar programming languages like C/C++ or Python to write host code which uses XRT to interact with FPGA/ACAP device. XRT exports well defined set of software APIs that work across PCIe based datacenter platforms and ZYNQ UltraScale+ MPSoC/Versal ACAP based embedded platforms. XRT is key component of Vitis™ and Alveo™ solutions.

XRT SW Stack

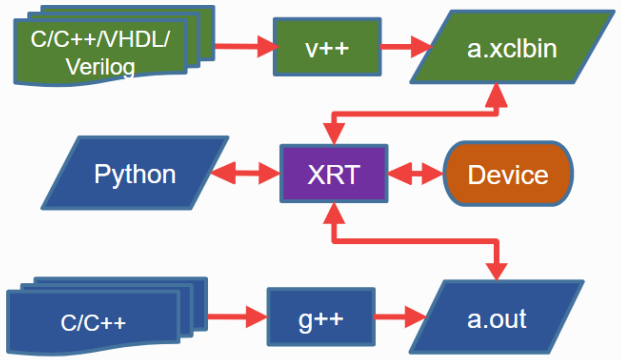


Source: "DHDL: The D Hardware Description Language", Luís Marques, DConf 2017.

XRT & Vitis

User Application Compilation

User application is made up of host code written in C/C++/OpenCL or Python. Device code may be written in C/C++/OpenCL or VHDL/Verilog hardware description language.

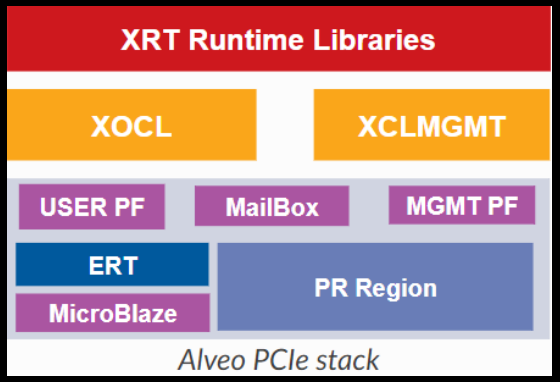


User application compilation and execution

Users use Vitis™ compiler, v++ to compile and link device code for the target platform. Host code written in C/C++/OpenCL may be compiled with gcc/g++. Host code may also be written in Python OpenCL (using PyOpenCL) or Python XRT (using built-in python binding).

Source: https://xilinx.github.io/XRT/master/html/platforms.html

PCIe Based Platforms



Alveo PCIe stack

Source: https://xilinx.github.io/XRT/master/html/platforms.html

2) Python-centric one-stop toolchain

Ideas

- For historical reasons, most of the popular FOSS EDA software that requires high performance are implemented by C++/C, e.g., Verilator, Yosys, KiCad…another "barrier" is that LLVM is also written in C++. But most of them provides Python binding or wrapper to various extents.
- Most of the main stream AI frameworks like Tensorflow, PyTorch and MXNet embrace Python/C++ for their software layers design.
  A similar phenomenon also takes place in the world of FOSS EDA…
- Today, there are a lot of new FOSS EDA tools like F4PGA, LiteX and Cocotb are choosing Python as their primary development language, some projects are even written in pure Python.
  And many Python-based HDLs such as Amaranth/FHDL and PyGears.
- We are also deeply impressed by the successful story of PYNQ – the productivity of Python and interactive development experience with Jupyter, which inspires us to try to build a fully open-sourced Python-centric one-stop toolchain for HW-SW co-designed systems that base on RISC-V and FPGA, especially for Edge AI.

2.1 Current plans

Design Goals

- A fully open-sourced alternative to PYNQ(Xilinx), while supports more FPGA/eFPGA hardware.
- Well-designed modular architecture as well as with flexible plug-in mechanism.

Intergration

- A customized Jupyter for FOSS EDA.
- F4PGA + LiteX + Cocotb + Amaranth + HWT + SpyDrNet + PlatformIO…
- Plugins: FABulous, CFU-Playground, Exo…

Edge AI

- A new Torch-MLIR centric framework.
- Wasm-based runtime for Edge device like Microcontroller.
- Add support for reinforcement learning and distributed AI by leveraging Ray.
- Make fully use of what from Antmicro etc.

Distributed Cooperation

- A customized version of Ray as a universal distributed engine.

Python Runtime

- Embrace new Python runtime for performance and HW-SW co-development purpose.

Virtual Prototyping

- Customize/re-implement Renode for our needs.

2.2 Existing and upcoming output

- Continuously updated "Revisiting the eBPF-centric new approach for Hyper-Converged Infrastructure & Edge Computing" at K+ Summit 2021(Shanghai), and the third-round discussion on this topic will be divided into two series "ARM + Python + Rust + Lua + GraalVM + …" and "RISC-V + Python + D + Lua +  + …" according to different technology roadmap(for details, please look forward to our upcoming technical report "The third-round discussion on eBPF-centric new approach for Hyper-Converged Infrastructure & Edge Computing" .
- "A survey of current Python implementations" at PyCon China 2021 (Online) and the upcoming follow-ups.
- "GraalVM-based unified runtime for eBPF & Wasm" at GOTC 2021(Shenzhen) & "Revisiting GraalVM-based unified runtime for eBPF & Wasm" at OpenInfra Days China 2021(Beijing), and the upcoming follow-ups.
- "AOT compilation based Wasm compiler and runtime for Serverless Edge computing" at OpenInfra Days China 2021(Beijing) and upcoming follow-ups.

- "GraalVM for Heterogeneous Parallel Computing" at OSDT China 2021 (Online) and the upcoming "The first exploration of Smart Runtime" together with corresponding follow-ups.
- "seL4 for secure Edge Cloud infrastructure" at CID 2021(Shanghai) and the upcoming follow-ups.
-  as a better system programming language series "Will D be a better system programming language" at OpenInfra Days China 2022(Online) and upcoming follow-ups like "First exploration of Dlang for HW-SW co-designed system" at 1st OSEDA Workshop 2022 (Online), "Revisiting D as a better system programming language" and "Rethinking D for HW-SW co-designed system".

VI. Wrap-up

- Edge computing is booming, and we have more freedom to innovate when compared with that in the Cloud.

- A New Golden Age for Computer Architecture!
  A New Golden Age for Compiler Design!
  A New Golden Age for Programming Language!
  A New Golden Age for HW-SW Co-design!

RISC-V Summit China 2022

Q & A

Reference

Slides/materials from many and varied sources:

- http://en.wikipedia.org/wiki/
- http://www.slideshare.net/
- https://viso.ai/edge-ai/artificial-intelligence-of-things-aiot/
- https://ptolemy.berkeley.edu/projects/embedded/Research/hsc/abstract.html
- https://evision-systems.com/electronic-design-automation/hardware-software-codesign/
- https://www.opencompute.org/projects/ai-hw-sw-codesign
- https://tec.ee.ethz.ch/education/lectures/hardware-software-codesign.html
- https://www.mathworks.com/help/supportpkg/usrpembeddedseriesradio/ug/hardware-software-co-design-overview.html
- https://www.embedded.com/top-5-predictions-for-efpga-in-2022/
- https://cdn.hackaday.io/files/1741677451560928/NEORV32.pdf
- https://stnolting.github.io/neorv32/#_zxcfu_custom_instructions_extension_cfu
- https://github.com/f4pga/
- https://en.wikipedia.org/wiki/Netlist
- https://qengineering.eu/deep-learning-with-raspberry-pi-and-alternatives.html

- https://www.hackster.io/aifes_team/aifes-express-tutorial-feedforward-neural-network-float32-e785f2
- https://www.hackster.io/aifes_team/how-to-use-aifes-on-a-pc-or-in-other-ides-ef20a0
- https://github.com/PlayOnLinux/

- https://blog.unity.com/technology/unity-and-net-whats-next
- https://zhuanlan.zhihu.com/p/507307850
- https://m-labs.hk/migen/manual/fhdl.html
- https://zhuanlan.zhihu.com/p/538432245
- https://zhuanlan.zhihu.com/p/149659607
- …