No-ISA is the Best ISA

Shreeyash Pandey, Rishik Ram Jallarapu

Vicharak, India @ vicharak.in

28th September, 2024

About us

Vicharak was founded with the idea: to introduce software-level reconfigurability to the hardware world with real parallel machines to enhance computing.
Our **goal** is to create consumer and industrial computing hardware, as well as an entirely new kind of computing ecosystem that can sit on your desk, rest in your palm, or exist in the cloud.
For such an ambitious goal, Vicharak is ready to work on every vertical with a team of ~50 passionate engineers and thinkers deeply entrenched to make this vision a reality.
- Akshar Vastarpara (Founder and CEO, Vicharak)

Contents

Chapter 1 Motivations for our Work

Problems Facing Modern Compute

Moore's law

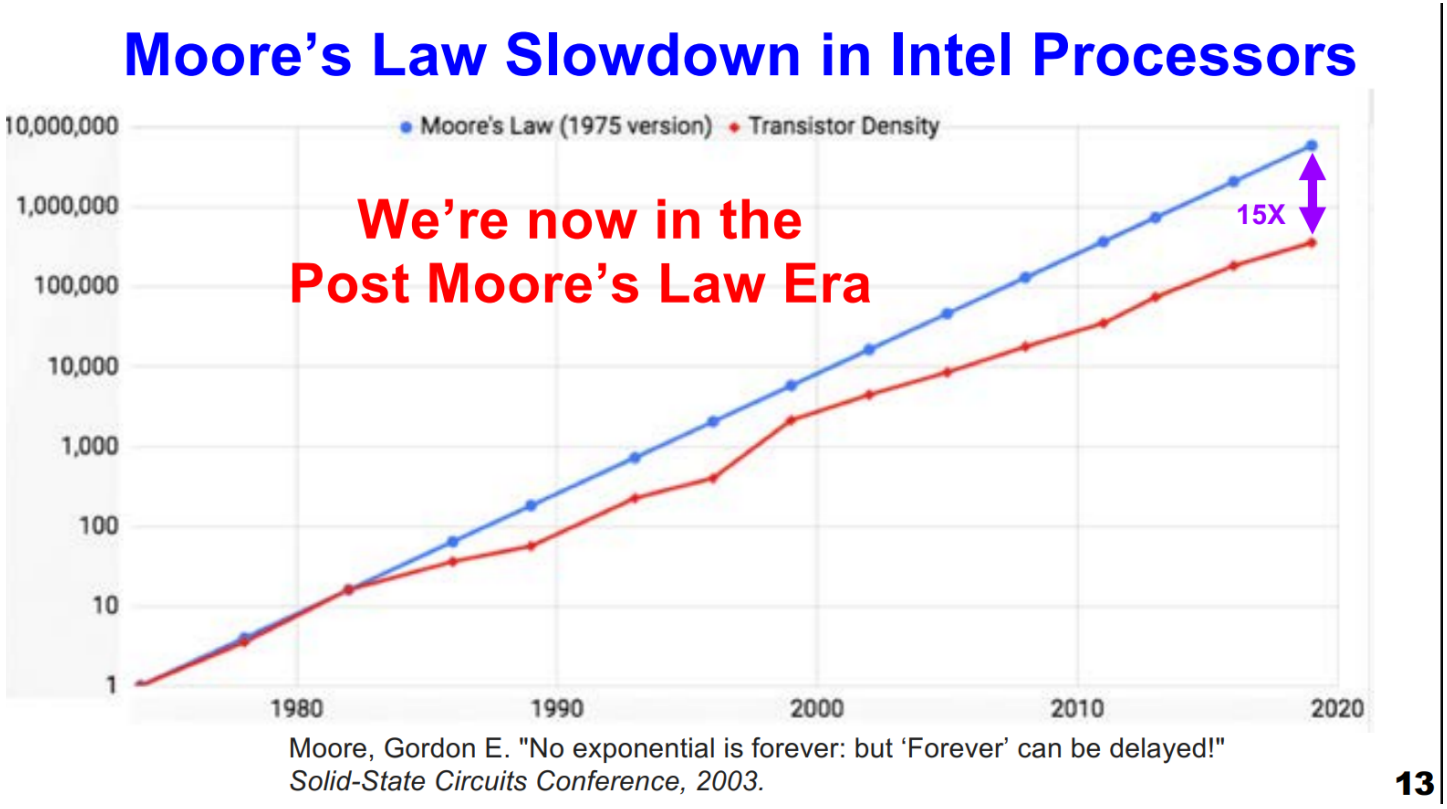


Figure: From "A Golden Age of Computers - David Patterson" [1]

Problems Facing Modern Compute

Dennard Scaling

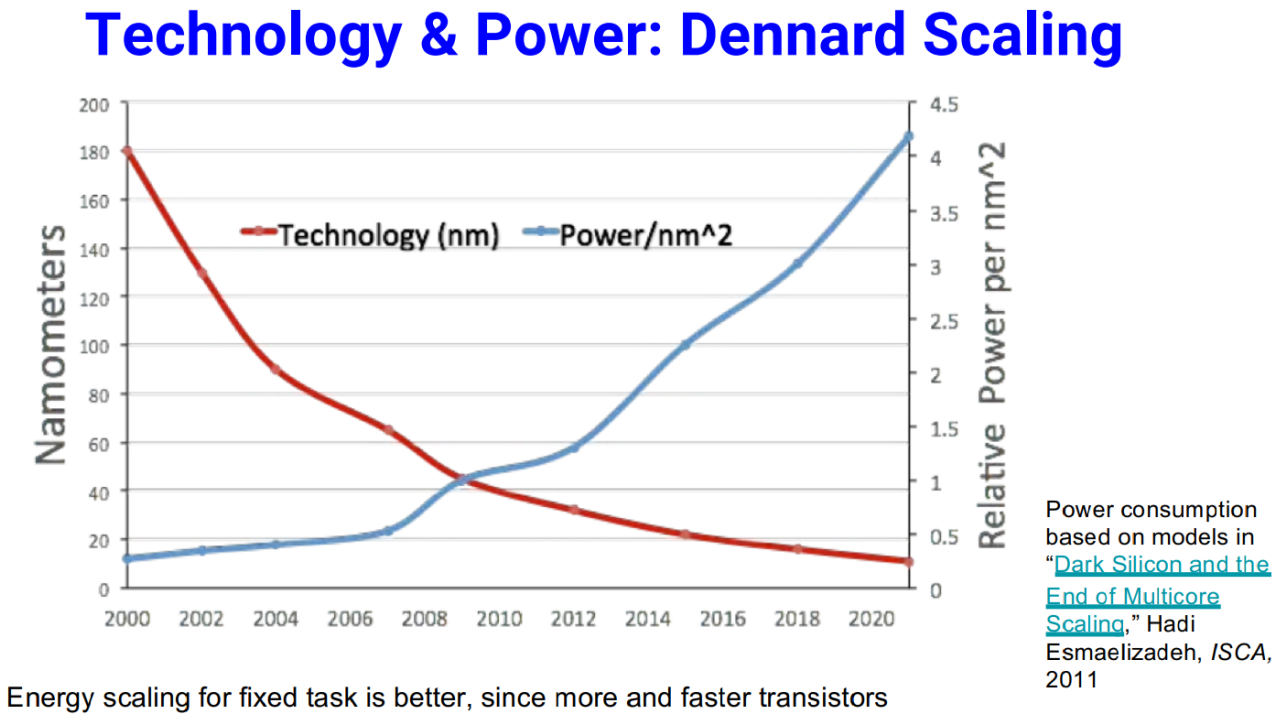
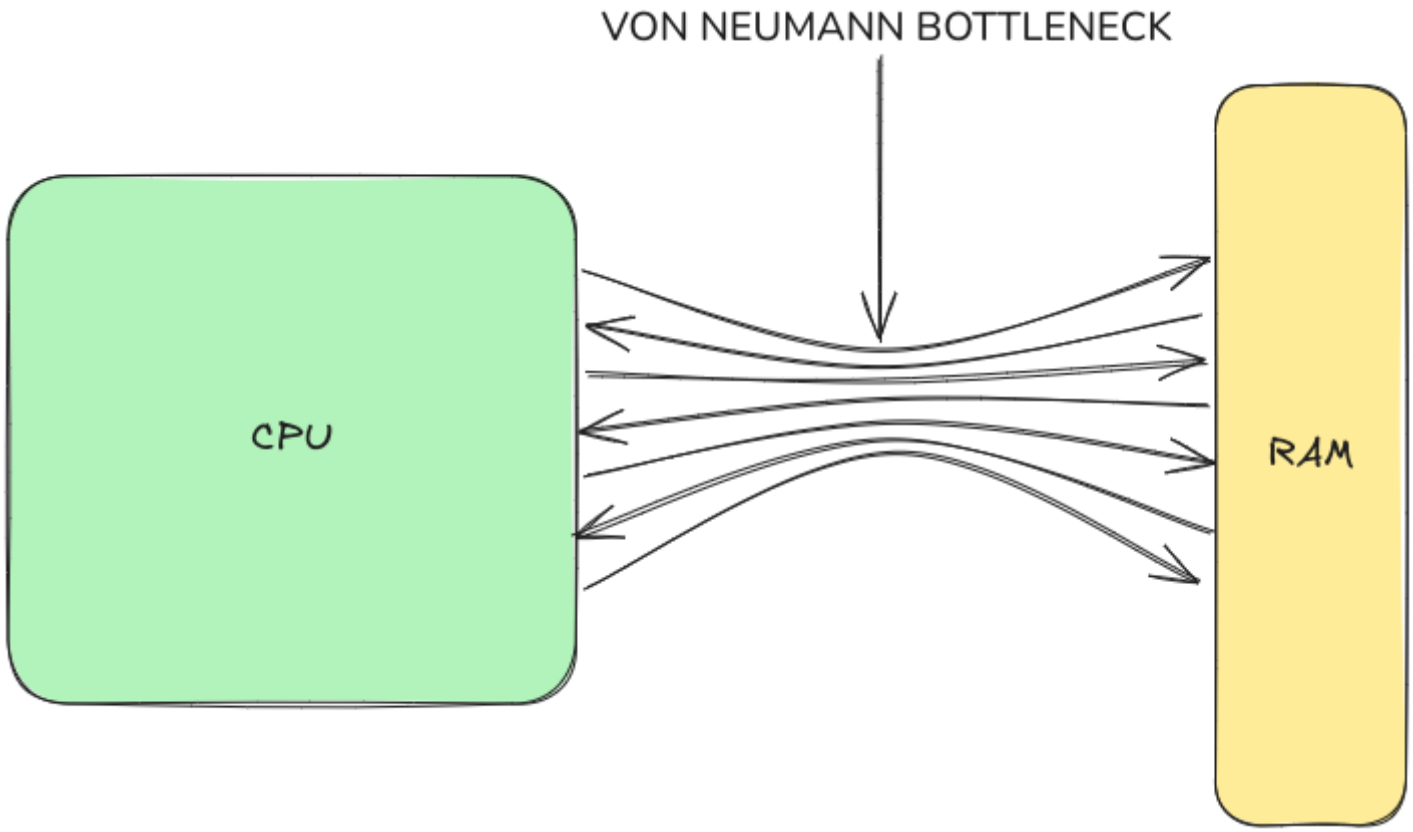


Figure: From "A Golden Age of Computers - David Patterson"

Problems Facing Modern Compute

Von Neumann Bottleneck



Overview of Modern Compute

- ASICs are expensive to engineer, CPUs are slow, GPUs are power hungry and inflexible
- Should modern compute be restricted to CPUs/GPUs and a handful of ASICs?
- Following are four instances where existing compute falls short:

Hard-to-Solve Problems for Modern Compute

- Problems involving many peripherals and real-time compute
- Unusual Representation of Numbers (In contrast to fixed width) (quantization for neural networks)
- New architectures/solutions for old problems (KANs)
- Power effiecient and flexible

Problems Facing Modern Compute

1. The free lunch afforded by hardware improvements over years is coming to an end.
2. New, creative, first-principles architectures and compute infrastrucutre need to be designed along with the software abstractions to use them.

Chapter 2 Introduction To Reconfigurable And Heterogeneous Computing

Setting the stage

Two key ideas:

1. Reconfiguration: The process through which a "reconfigurable processor" is re-programmed to implement a new circuit. Reconfiguration can be achieved by using circuits such as FPGAs.
2. Heterogeneity: The idea that a system must include processors (such as CPUs/GPUs/DSPs/Other ASICs) of different capacities/abilities well integrated together. Heterogeneity promotes complementing existing compute instead of replacing.

Key Problems With Reconfigurable-Heterogenous Computing

To implement a reconfigurable heterogeneous computer with FPGAs, the problems are two-fold:

1. Problem 1: Using FPGAs with traditional software is in-convenient.
2. Problem 2: Writing new hardwares for FPGAs, implementing custom solutions is tedious with a very steep learning curve, often times requiring domain expertise.

Reconfigurability: An Introduction to FPGAs

1. Digital circuits are made up of gates and connections between them.
2. FPGAs are a grid of programmable cells and switches that allows a user to create new gates and connections after the IC has been fabricated.
3. Circuits for FPGAs are described using Hardware Descriptions Languages (HDLs) such as Verilog, VHDL.
4. High level description of a circuit is compiled into real hardware (i.e. a representation that only uses FPGA primitives) by a "compiler".

Problem 1: Programming model for FPGAs

1. A true programming model for FPGAs would heavily exploit reconfigurability.
2. We introduce a non-Von Neumann flow-based reconfigurable architecture and a programming model for it.

Comparison

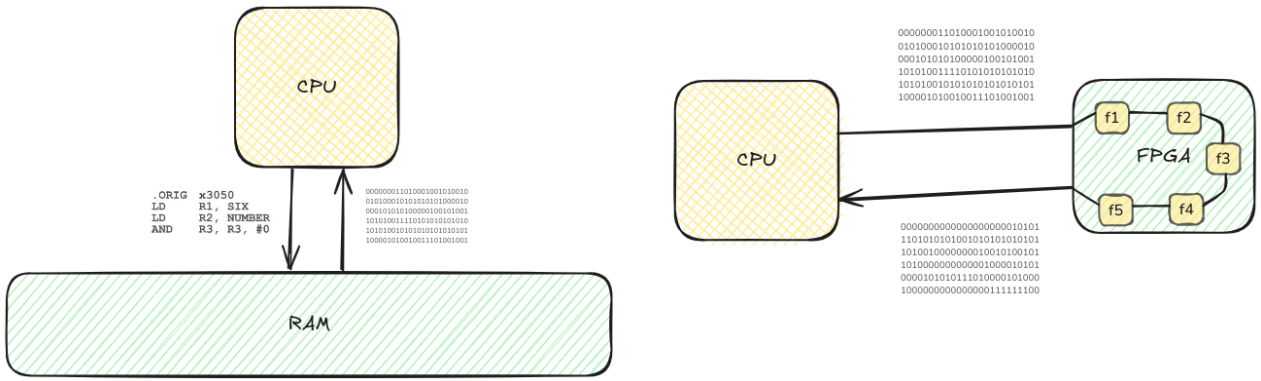


Figure: a) A Von-Neumann Computer b) A Flowing Reconfigurable Computer

Differences

- There are no instructions as the hardware is configured to a desired operation.
- Data flows in and out of the chip transformed.
- No instructions -¿ No ISA!
- "What to do with data" is a part of the hardware

An exemplary DSL for reconfigurable compute architectures

```
Base *cam_in = new CameraCore("MIPI0", "cam1");
Base *proc_one = new JPEGEncoderTillDct(input,
                                        "jpeg_encoder");
*proc_one = cam_in;
Base *proc_two = new MlEngineCore(input, "ml_core");
*proc_two = proc_one;
Base *display_out = new PeripheralGen(proc_one,
  1                                     "LVDS", "out1");
*display_out = proc_two;
Model m2 = new Model(cam_in, display_out);
```

An exemplary DSL for reconfigurable compute architectures (2)

Continue

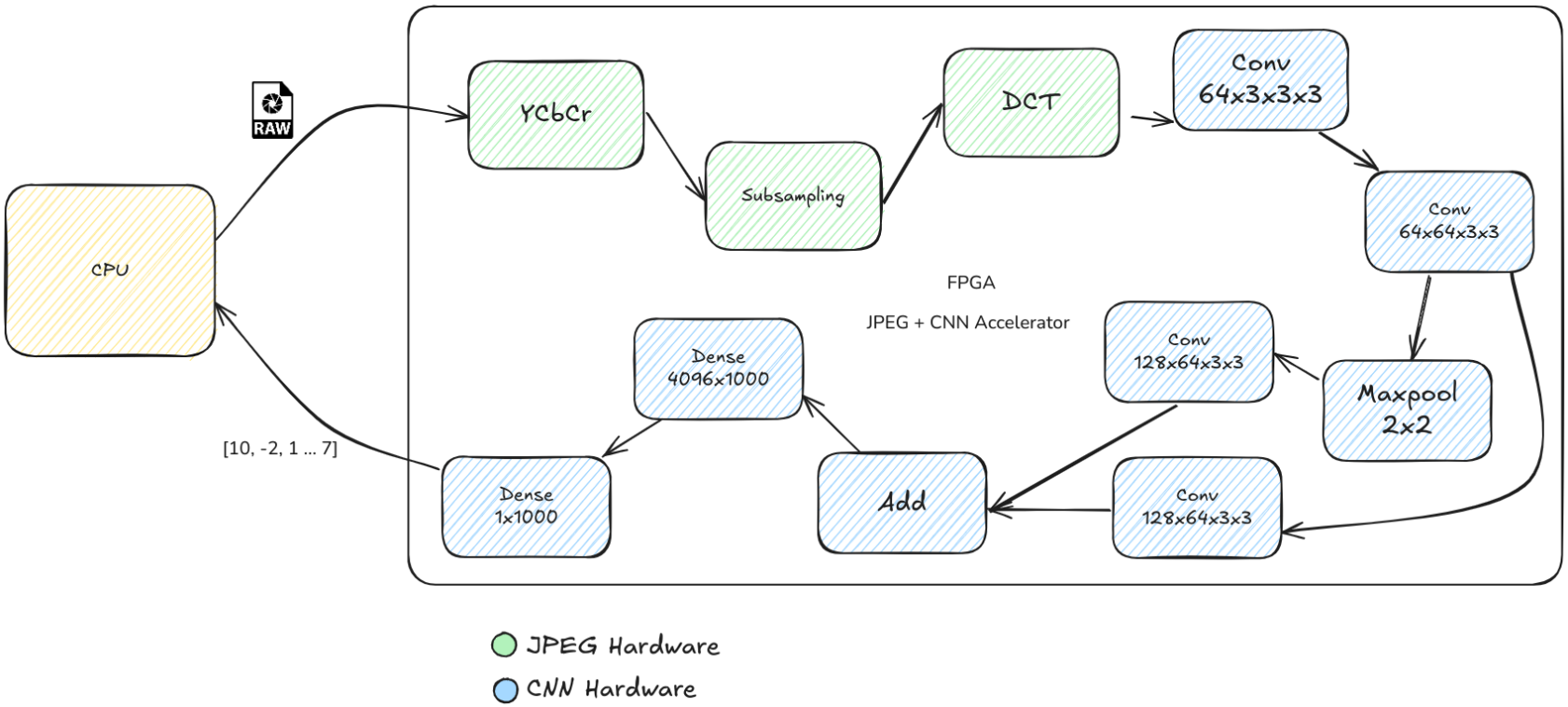


Figure: Data flow for JPEG (Partial) + CNN inference

An exemplary DSL for reconfigurable compute architectures (3)

```
m1->compute(input);
if (some_user_defined_condition(m1->output())) {
  m2->compute(m1->output());
} else {
  return m1->out();
}
```

`model->compute` is the function that triggers generation, flashing and computation on a hardware described by a Model.
The code above demonstrates *conditional reconfiguration*.

Chapter 3 Need for modern EDA Compilers

Problem 2: Writing Hardware Is Hard

1. Hardwares are written using HDLs like Verilog/VHDL
2. Problems can be categorized as those above HDL level and those below
3. Problems above HDL are solved by new tools and frameworks that such HDL wrappers (SpinalHDL, Chisel etc), High-Level Synthesis tools (name them), simulators/verifiers (verilator, CIRCT etc.)
4. We are particularly interested in solving problems below HDLs.
5. EDA tools are propreitary and hard-to-work-with.
6. The general problem of compilation of hardwares is NP-Complete but there are special cases that can be exploited.

FPGA CAD Toolflow

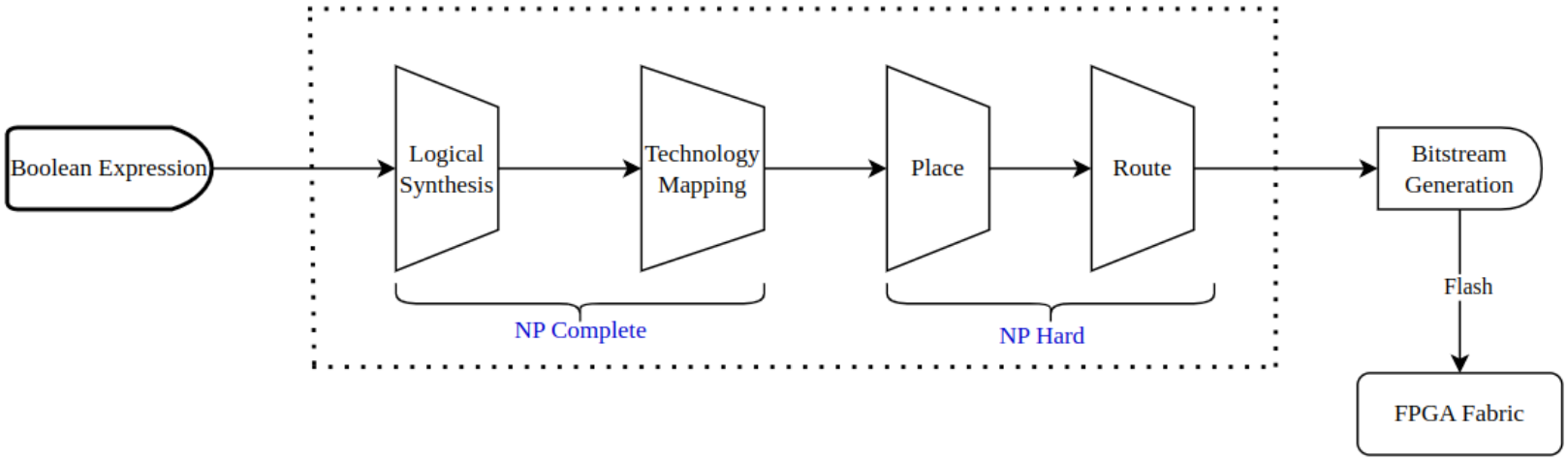


Figure: FPGA CAD Tool Flow

FPGA CAD Toolflow

- Logical Synthesis: HDL to primitive gates (AND/OR).
- Technology Mapping: Mapping generic AND/OR gates to k-LUTs.
- Placement: Placing k-LUTs in a constrained grid representing the FPGA based on a cost model
- Routing: Connection placed LUTs so that the latency is minimum
- Bitstream Generation: Generating a final bitstream that the FPGA can understand.

The Current State of EDA tools

1. Like the FPGA architectures, tools to work with them are propreitary.
2. Problems and sub-problems in the flow are ridden with NP-Hard and NP-Complete problems.
3. There is a gap between CAD research and production ready CAD tools.

Optimization Opportunities for EDA Compilers

1. Our DSL compiler connects hardware together. The mapping phase of hardware generation can be completely bypassed if The compiler can be designed to operate on netlists directly instead of verilog.
2. The mapping process involves, among many steps, a phase where it looks for a minimal boolean expression. In iterative write-compile-debug loops entire hardware may not change frequently so their resulting minimal boolean expressions history can be saved and revisited for next iterations.
3. Routing can be designed to make use of GPUs.

Chapter 4 Work Done Towards Implementation

Realizing this goal

1. Realizing this goal requires design and implementation from first principles.
2. To achieve this, we designed our own hardware: Vaaman.
3. Vaaman is a reconfigurable heterogenous computer.

The Hardware (Vaaman)

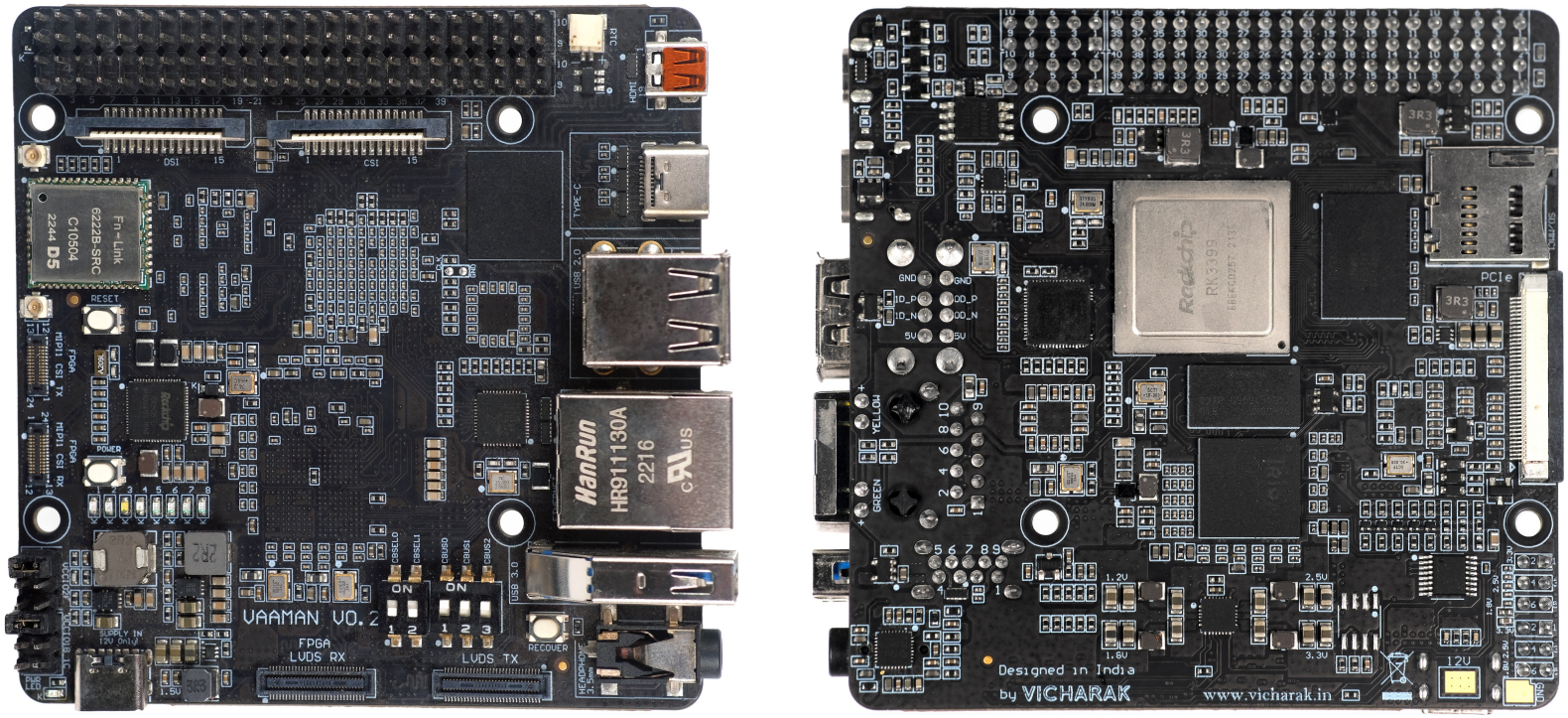


Figure: Vaaman: A heterogenous SBC
`https://shorturl.at/5y9QA`

ML Accelerator (Gati)

- Gati is a set of hardware and software programs that perform CNN acceleration with FPGA as a co-processor on Vaaman.
- Gati accelerates MAC operations with a Systolic-Array based MAC engine, and handles control flow with a macro-ISA.
- Assisting this hardware is a Compiler/Runtime.

ML Accelerator (Gati)

1. The compiler does two primary things:
   1.1 Parsing of input data and NN models (protobufs (ONNX) etc.), transpositions of kernels to allow contiguous memory access on the FPGA, and generation. of a byte stream that can be fed to the FPGA
   1.2 Generating custom hardware for every nn model.
2. The runtime partitions a network into execute-on-host and execute-on-device, re-orders inputs, and offloads computation to the FPGA.

Periplex - On-the-Fly Peripheral Generation

Problem

1. Hardware peripherals are limited by fabrication at the ASIC level or in MCUs and MPUs.
2. The world is moving towards more complex combinations of physical peripherals, instead of just (2 UARTs, 4 SPI, etc.).
3. Due to ASIC manufacturing costs and time, the hardware peripherals' average innovation/scale period from concept to fabricated chip and to consumers is around 2-3 years.
4. Emerging embedded industries like drones, autonomous vehicles,industrial gateways, robotics require more hardware peripheral accessibility and innovation due to real-time operations.

Periplex - On-the-Fly Peripheral Generation

Solution

1. Periplex allows software-defined generation of peripherals on reconfigurable hardware (FPGAs) which allows rapid prototyping and development hardware supporting newer peripherals in days not years.
2. Periplex operates on a JSON-driven configuration that specifies what peripherals are needed and how they should be connected, generates FPGA bitstream and uploads it the FPGA.
3. Periplex accomodates drivers for these peripherals directly into the linux kernel so that it can be accessed through linux APIs easily.

Conclusion

1. Reconfigurable architectures can provide a way to solve many problems that existing compute struggle with and help alleviate the von-neumann bottleneck.
2. Heterogenous approach of assisting instead of replacing integration of new hardwares in systems can allow existing infrastructure to be used.
3. Two of the biggest problems with achieving this are a) programming model that exploits reconfigurability b) fast and flexible hardware compilers (EDA tools).
4. Solutions to problem a) manifest themselves in the form of novel DSLs and compiler/runtime toolchains compatible with current toolchain/workflows used by CPUs.
5. Solutions to problem b) involve finding optimization oppurtunities to speed up EDA tools, making use of modern parallel hardwares such as GPU, and other accelerators.