No-ISA Is The Best ISA

Shreeyash Pandey, Rishik Ram

Vicharak, India @ vicharak.in

28th September, 2024

About us

Contents

Chapter 1 Motivations for our work

Problems facing modern compute

Moores law is slowing down

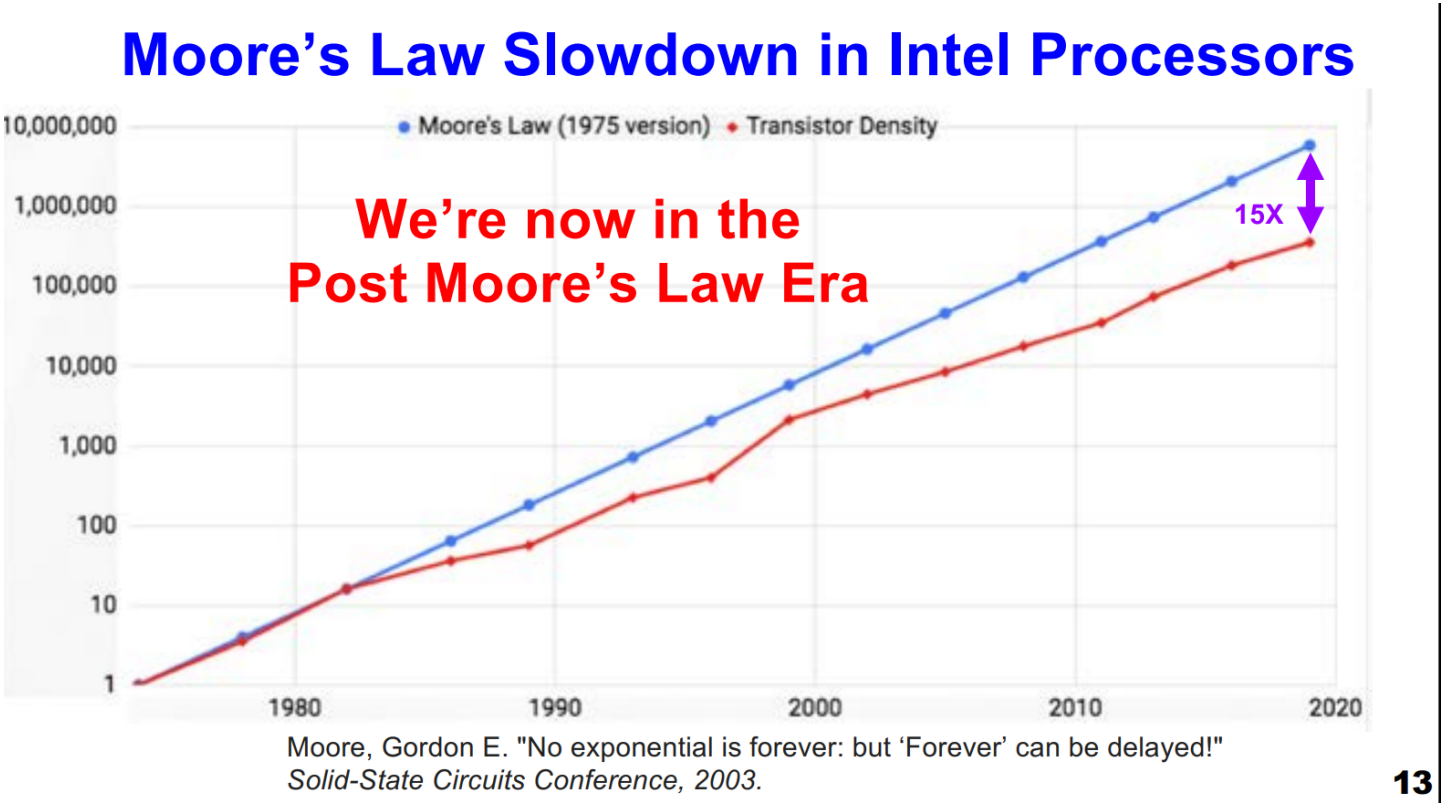


Figure: From "A Golden Age of Computers - David Patterson"

Problems facing modern compute

Dennard Scaling has stopped working

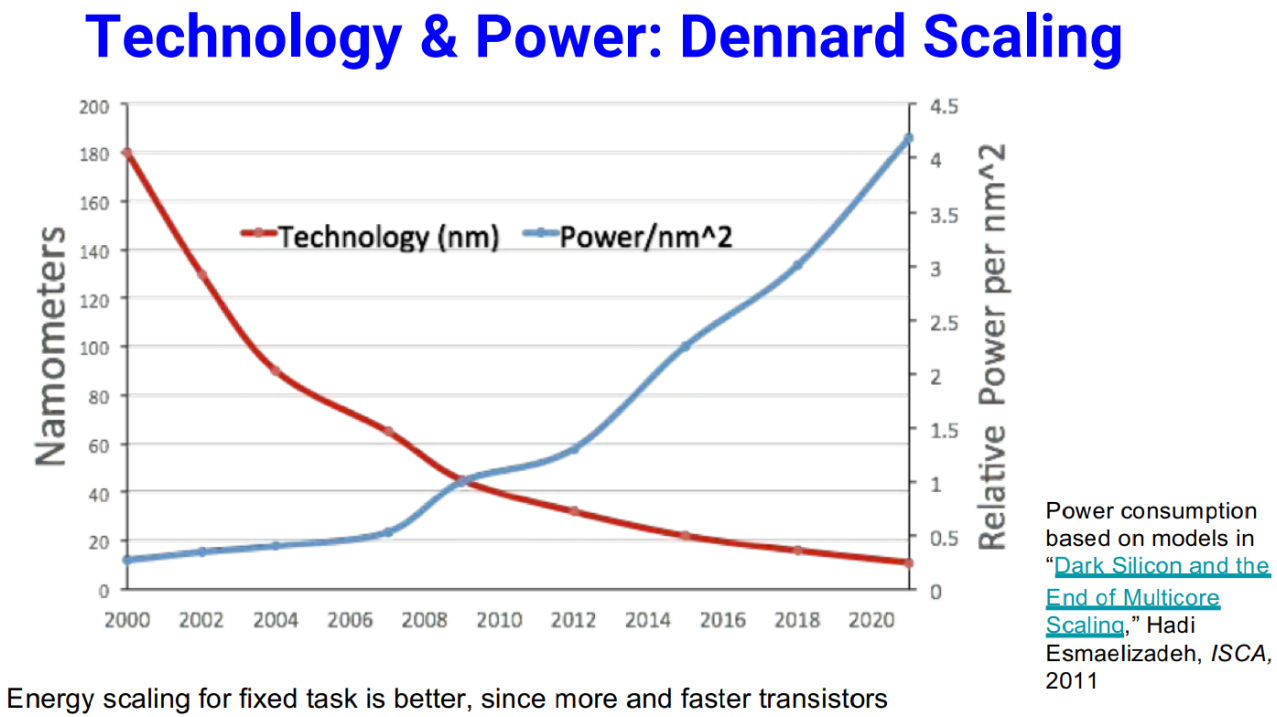


Figure: From "A Golden Age of Computers - David Patterson"

Problems facing modern compute

1. The free lunch afforded by hardware improvements over years is coming to an end
2. Hardwares are designed first and complying softwares to support them after it.
3. New and creative architectures need to be designed along with the softwares abstractions to use them.

Overview of Modern Compute

1. An average motherboard has a CPU and optionally a GPU
2. In specialized domains, one may find ASICs being used (for eg, ML acceleration)
3. ASICs are pretty cool (and fast) and solve domain specific problems that CPU/GPUs may not be able to solve, but are they for everyone?
4. For starters, they are expensive to engineer and require a team of expert hardware engineers to be designed and fabricated
5. Once that's done, expert systems software engineers are required to make the ASIC usable/compatible with the existing operating systems.
6. A lot of hardwork, definitely not for everyone. As a result, ASICs are far and few
7. Should modern compute be restricted to CPUs/GPUs and a handful of ASICs?
8. What about the problems where none of existing compute suffices?

Hard-to-solve Problems for Modern Compute

Example 1

Problems involving many peripherals as well as compute

For example,

An embedded application that uses object detection to find objects in a line of sight and responds to it by driving many motors in real time needs heavy compute (for OD) and flexible IO to be able to drive all the motors reliably.

Existing solution would involve usign a GPU for ML workload, an driving the motors from a CPU. A CPU may or may not have as many IOs as required, in which case an IO expander may have to be set up.

Hard-to-solve Problems for Modern Compute

Example 2

Unusual Representation of Numbers

Quantization is a technique of reducing precision of numbers at the loss of accuracy. Quantization is used extensively to speed up Neural Network inference. New techniques such as heterogenous quantization of layers (i.e. different bit-widths of numbers at layer granularity), odd-number quantization (such as 9-bit numbers), ternary computers etc. pose a significant challenge for existing fixed-bit-width computers.

Hard-to-solve Problems for Modern Compute

Example 3

New Architectures/Solutions for Old Problems

New solutions to old problems are those that are fundamentally different to all existing solutions. For example, Kolmogorov-Arnold Networks (KANs) propose an alternative to MLPs (which is at the core of machine learning today). KANs reeplace the static parameter of MLPs with a learnt spline function. Wrappers can be built around existing hardware to execute KANs too, but since its different on a fundamental level, dedicated hardwares would be beneficial.

Hard-to-solve Problems for Modern Compute

Example 4

Power-efficiency without sacrifices

Unlike general purpose chips, on FPGAs you only get what you need. As a result, the overall power efficiency of dedicated hardware tends to be higher than general purpose processors. Customizable architecture eliminates unused components and reconfigurability allows adapting to changing workloads

"Should I throw away my CPU?"

1. Strengths of existing compute are known. We would like to have these strengths in our systems and bring reconfigurable heterogenous compute to tackle the weaknesses.
2. We don't have to forego our CPUs, GPUs.
3. CPUs are good at running operating systems, they should continue doing it.
4. The goal is to complement existing compute not replace.

Chapter 2 Introduction To Reconfigurable And Heterogenous Computing

Setting the stage

Two key ideas:

1. Reconfiguration: The process through which a "reconfigurable processor" is re-programmed to implement a new circuit
2. Heterogeneity: A system must include processors of different capacities/abilites well integrated together.

Reconfigurability: An Introduction to FPGAs

1. FPGAs are a grid of cells that can be reprogrammed to implement any circuit.
2. Digital circuits consist of gates (that implement logic) and connections (that connect gates to each other).
3. FPGAs, at the core, consist of SRAM cells (that implement the functionality of gates by storing their truth-tables in SRAM cells) and programmable interconnect (implemented via switch boxes) that allow connections
4. Circuits for FPGAs are described using Hardware Descriptions Languages (HDLs) such as Verilog, VHDL.
5. High level description of a circuit is compiled into real hardware (i.e. a representation that only uses FPGA primitives) by a "compiler"

Key problems with reconfigurable-heterogenous computing

1. To implement a reconfigurable heterogenous computer with FPGAs, the problems are two-fold:
2. Problem 1: Using FPGAs with traditional softwares are in-convenient.
3. Problem 2: Writing new hardwares for FPGAs, implementing custom solutions is tedious with a very steep learning curve, often times requiring domain expertise.

Problem 1: Programming model for FPGAs

1. GPUs enjoy a concrete and abstract programming model
2. No true industry grade programming model exists for FPGAs
3. There's OpenCL support for FPGAs. But that involves treating FPGAs like an ASIC.
4. A true programming model for FPGAs would heavily exploit reconfigurability

Comparison of a reconfigurable-heterogenous programming model with a von-neumann computer

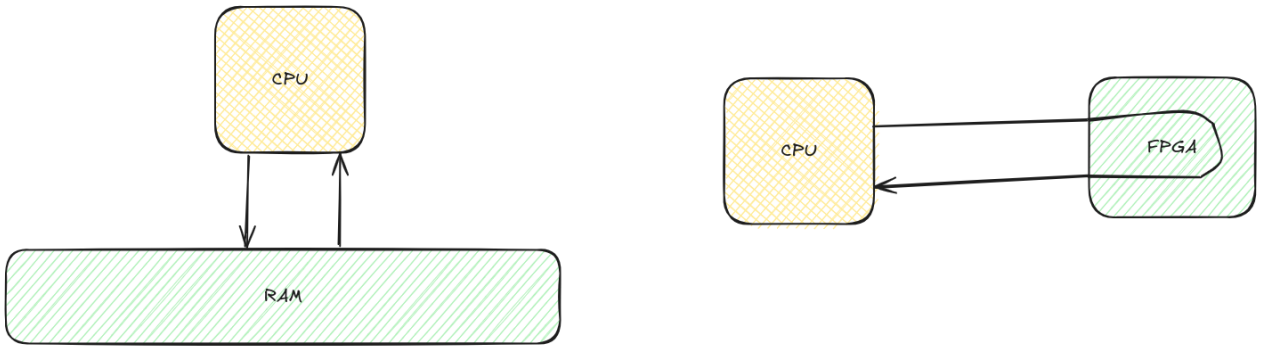


Figure: a) A von-neumann computer b) a flowing reconfigurable computer

Figure a) shows the difference b/w a von-neuman computer (which executes **instructions** on **data** over a **bus** resulting in back-and-forth of computation.
Figure b) is a flow computer where the hardware is configured to cause incoming data to be transformed in the way desired. There are no instructions as the hardware is configured to a desired operation. Data flows in and out of the chip transformed.

Comparison of a reconfigurable-heterogenous programming model with a von-neumann computer

It could be said from the previous slide that the reconfigurable style of architecture has no Instruction Set Architecture (ISA) (hence the title of this talk). "What to do with data" is a part of the hardware, instead of being attached with the data in the form of instructions. It's the only thing that it does. Following are a few examples of reconfigurable no-ISA architecture. They include a JPEG encoder and a CNN accelerator:

Flow architecture for JPEG encoding

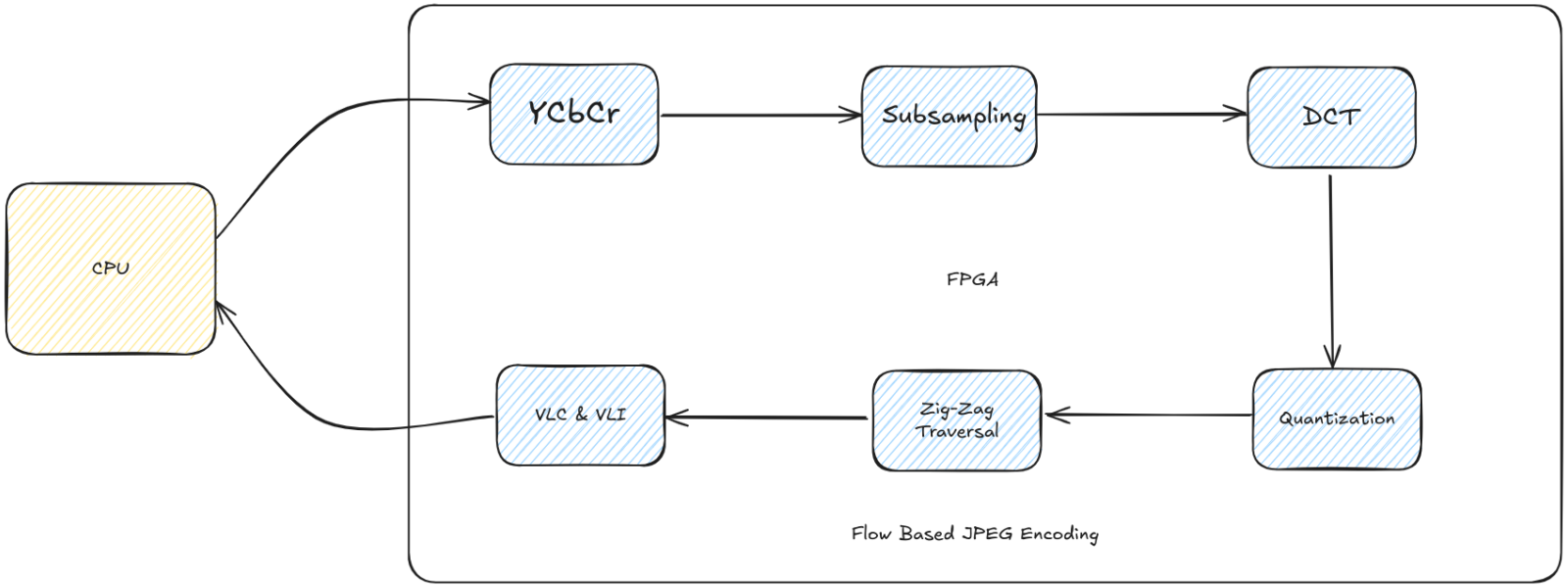


Figure: JPEG compression. Each operation has its own hardware

RAW images flow in, pass through the blocks, being encoded and the process and JPEG compressed images come out

Flow architecture for CNN inference

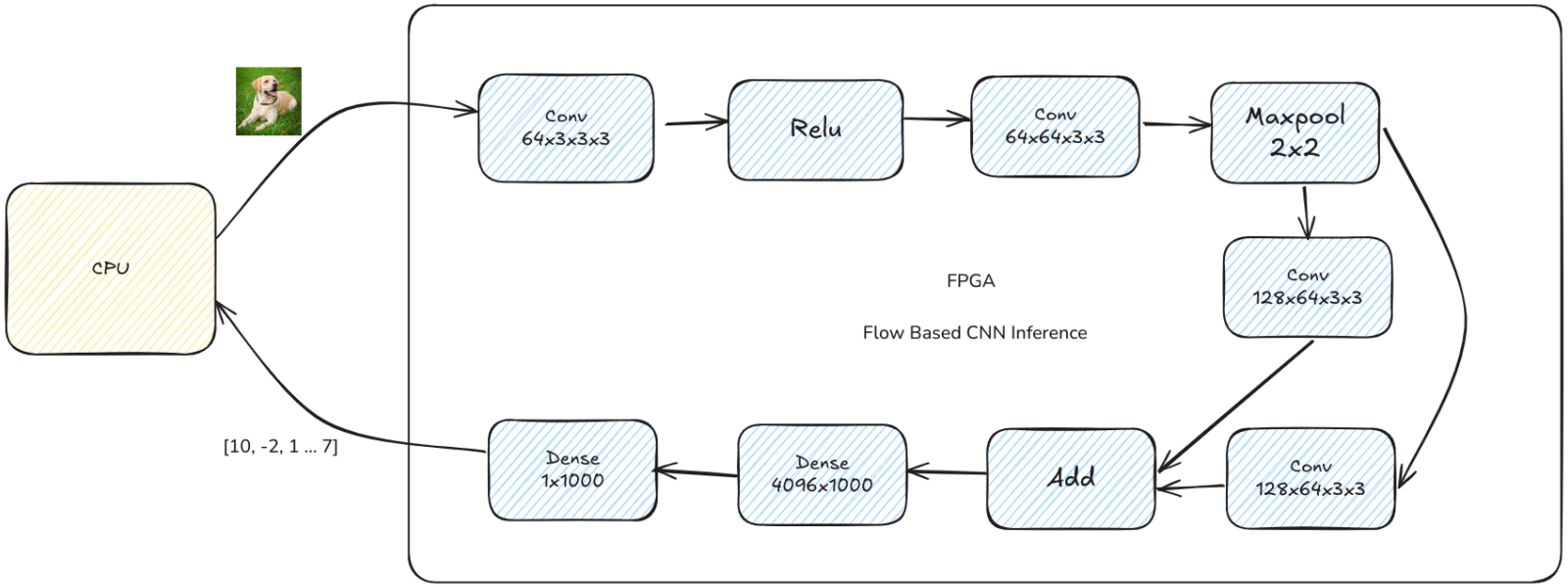


Figure: CNN inference. Each layer of a network has its own hardware

Images (according to the pre-process pipeline of a network) flow in, each layer manipulates and passes its computation to the hardware after it and end-results are returned by the last block.

Observation on flow-based computers

1. Hardwares for a problem are generated by a "Compiler" from a high-level specification that describes connection of coarse functions.
2. Any coarse hardware can be programatically be plucked and placed in a different setting thanks to the compilers ability to reason with hardware connections.
3. Flow based computer exibit a more functional approach towards computation
4. On a coarser scale, purity of computation is maintained as hardware blocks do not depend on a global state to execute

An exemplary DSL for reconfigurable compute architectures

Following is an example of a DSL that allows specification of coarse hardware. It provides an interface to define connections b/w hardware, control reconfiguration (through existing programming constructs (slide 3) and integrate it with existing codebases

An exemplary DSL for reconfigurable compute architectures

```
Base *input = new Tensor(std::vector<int>{1,2,3,4},
                  "t1");
Base *b = new MLEngineCore(input, "gc1");
*b = input;
Base *b1 = new PeripheralMultiplex(b, "AHB", "pp1");
*b1 = b;
for (int i = 0; i < 100; ++i) {
Base *b1 = new Sha256(b);
*b1 = b;
}
Model m1 = new Model(input, b1);
```

Describes an MIaccelerator connected to a peripheral multiplexor which is connected to 100 Sha256 blocks, all through c++.

An exemplary DSL for reconfigurable compute architectures (2)

```
Base *cam_in = new CameraCore("MIPI0", "cam1");
Base *proc_one = new GaussianBlurCore(input, "gc1");
*proc_one = cam_in;
Base *display_out = new
    PeripheralMultiplex(b, "LVDS", "pp1");
*display_out = proc_one;
Model m2 = new Model(cam_in, display_out);
```

Describes an application that takes input from camera, implements gaussian blur on it and returns it to an LVDS interface on the FPGA.

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. Demonstrates conditional reconfiguration where based on `m1->compute`'s result. If the result meets a user specified condition, m2's hardware is generated, flashed and computation begins for it.

Chapter 3 Need for modern EDA compilers

Problem 2: Writing hardwares is hard

1. Writing HDLs is a tedious task often requiring domain expertise
2. EDA tools are propreitary and hard-to-work-with
3. The general problem of compilation of hardwares is NP-Complete but there are special cases that can be exploited.

Opensource EDA Compilers

1. Groups such as f4pga, YosysHQ, openfpga are trying to create opensource alternatives for proprietary CAD tools by reverse engineering FPGAs but are limited by the resources
2. Creating Open Software infrastructure for Hardware (flexibility), which is community driven
3. Most of the opensource EDA compilers such as Yosys, CIRCT, Verilator, openvaf can't create real hardware. They are limited to logical synthesis and simulation.

Compilers in EDA

Yosys

1. Compiler that generates verilog to netlist format(support Technology mapping)
2. IR : RTLIL
3. SupportSimulation: CXXRTL (cycle driven simulator)(supports only 2 states
4. Largely community Driven

Verilator

1. Compiler that generates Cpp code from Verilog files
2. Used Extensively for cycle based simualation (supports only 2 states)
3. competes with proprietary simulators ,Community Driven

Compilers in EDA

CIRCT

1. Modular usage of libraries, designs similar to LLVM/MLIR in Hardware
2. {HLS,sv} to {sv,vcd etc ... }
3. Hardware MLIR Dialects
4. Arcilator used for simulation
5. cycle based simualation (supports only 2 states)
6. Supports only simulation

Openvaf

1. Verilog-A frontend
2. uses LLVM and generates a binary file for simulation

FPGA CAD Toolflow

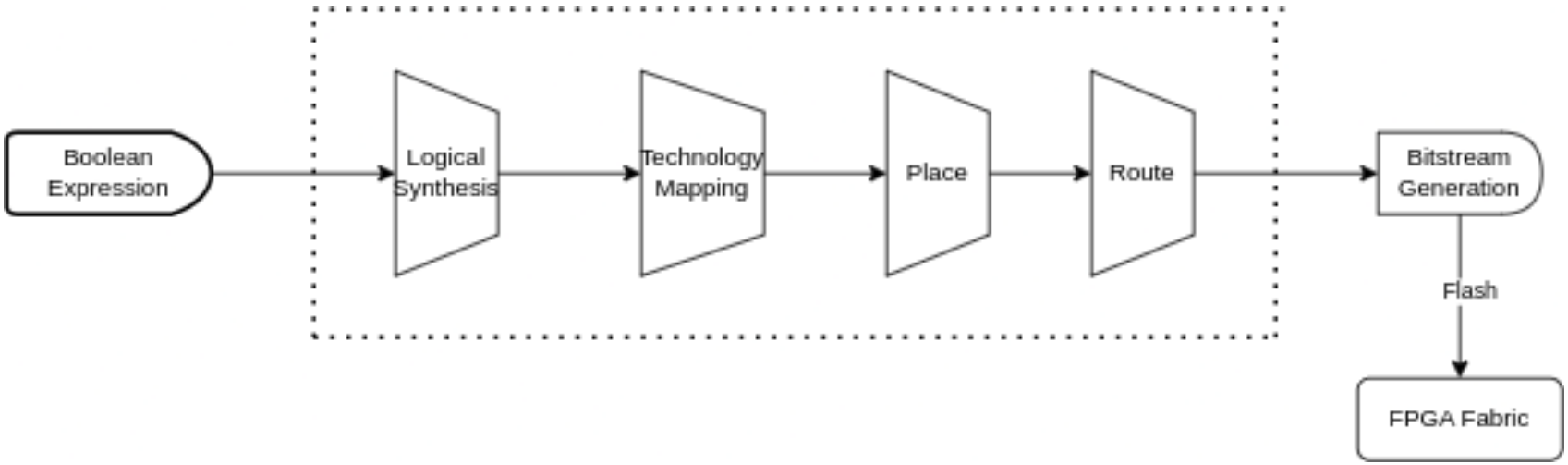


Figure: FPGA CAD Tool Flow

FPGA CAD Toolflow: The Frontend

Logical Synthesis,Technology Mapping

1. Logical synthesis is the process that parses HDL, performs technology-agnostic optimizations, and outputs a circuit (netlist) of generic primitives
2. Technology Mapping maps generic primitives generated by synthesis to FPGA-specific primitives.

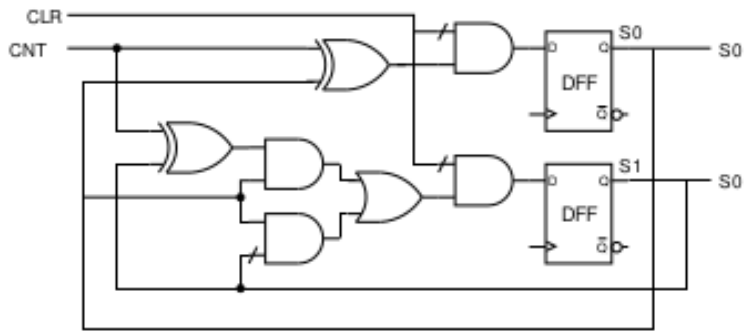


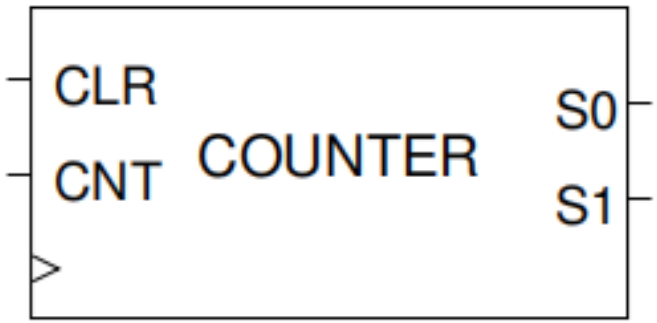
Figure: Gates Mapped for given Expression

FPGA CAD Toolflow: Synthesis/Mapping Via Example

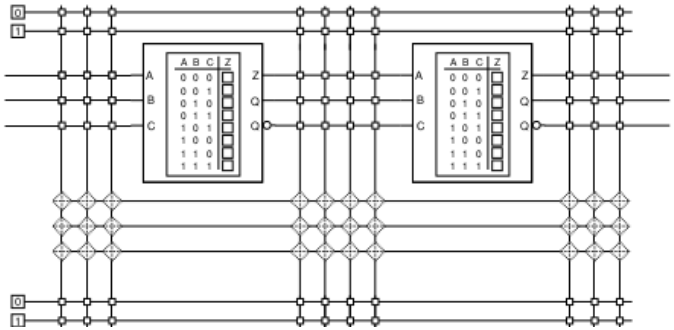
boolean expression:

$$S_0 = \overline{CNT} \cdot S_0 + CNT \cdot \overline{S_0}$$

$$S_1 = S_1(\overline{CNT} \cdot \overline{S_0}) + CNT \cdot S_0 \cdot \overline{S_1}$$



(a) Two bit counter block



FPGA CAD Toolflow: The Backend

Placement

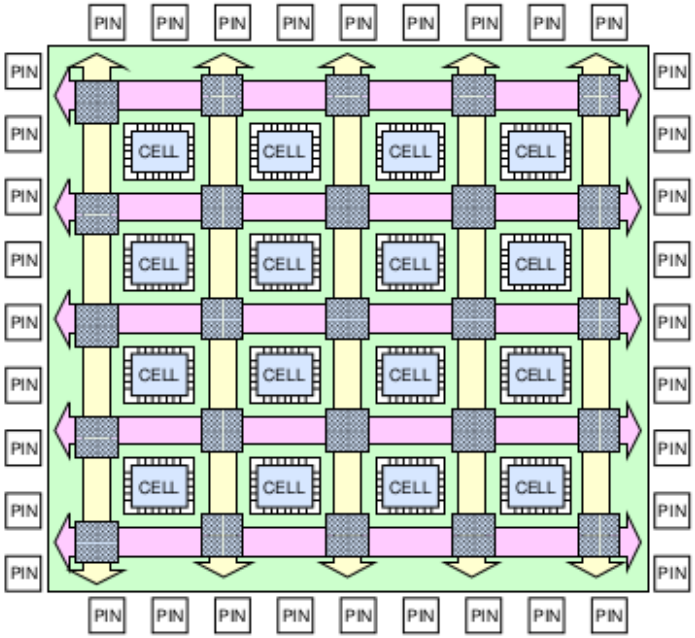


Figure: Example FPGA Fabric

Simulated Annealing (industry standard Algorithm) to place based on Minimum cost model

FPGA CAD Toolflow: The Backend

Routing

Routing : interconnect the Configurable Logic blocks with minimum timing cost

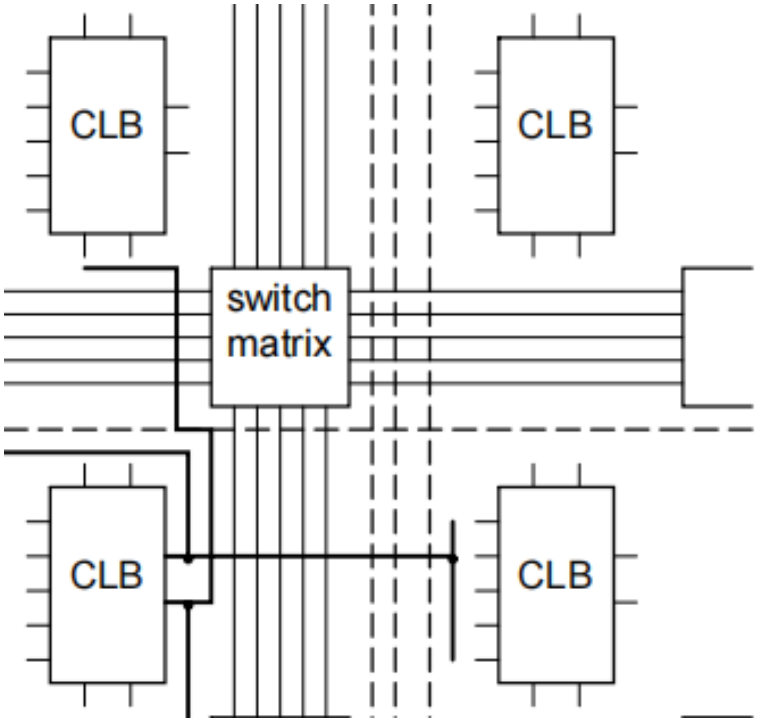


Figure: FPGA Interconnect

FPGA CAD Toolflow: Backend via Example

Placement, Routing

Placement and Routing for Two Bit counter would be

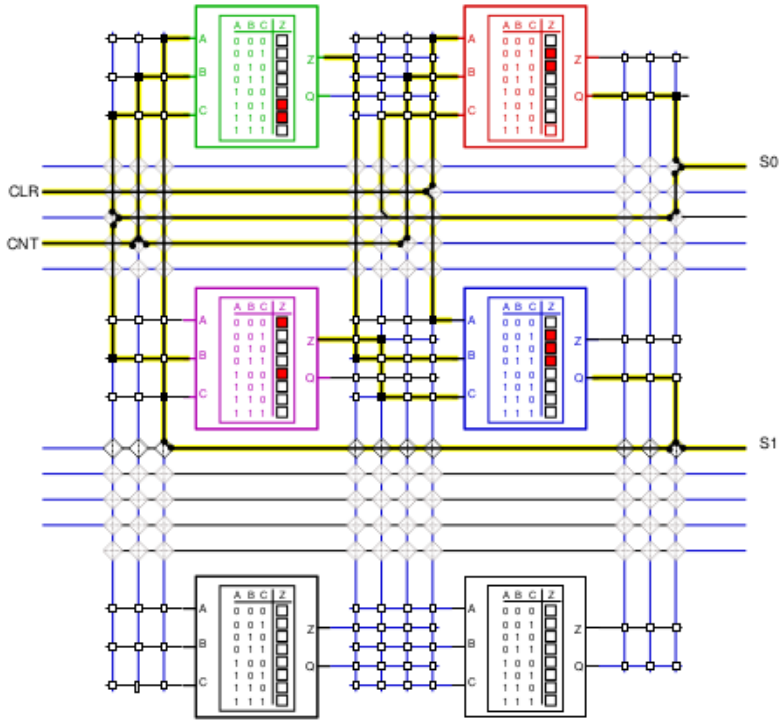


Figure: LUTs Connected with wire[1]

[1] Stephen D. Brown et al. *Field-programmable gate arrays*. USA: Kluwer Academic Publishers, 1992. ISBN: 0792392485.

Optimization oppurtunity for EDA compilers

1. Our DSL compiler connects hardwares together. The mapping phase of hardware generation can be completely by-passed if the compiler can be designed to operate on netlists directly instead of verilog.
2. 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 can be cached and further sped up by performing a look up in this cache instead of searching all over again.
3. Routing can be designed to make use of GPUs

Chapter 4 Work Done Towards Implementation

Realizing this goal

1. Realizing this goal requires designing and implementation from first principles
2. To achieve this, we designed our own hardware: Vaaman
3. To understand the nature of applications (in the sense of what bottlenecks exist and whether or not a certain application would benefit from reconfigurable-heterogenous architecture), projects have been implemented
4. These include: Gati (an ML accelerator) and Periplex (a peripheral multiplexor)
5. Discussion on this work follows:

The Hardware (Vaaman)

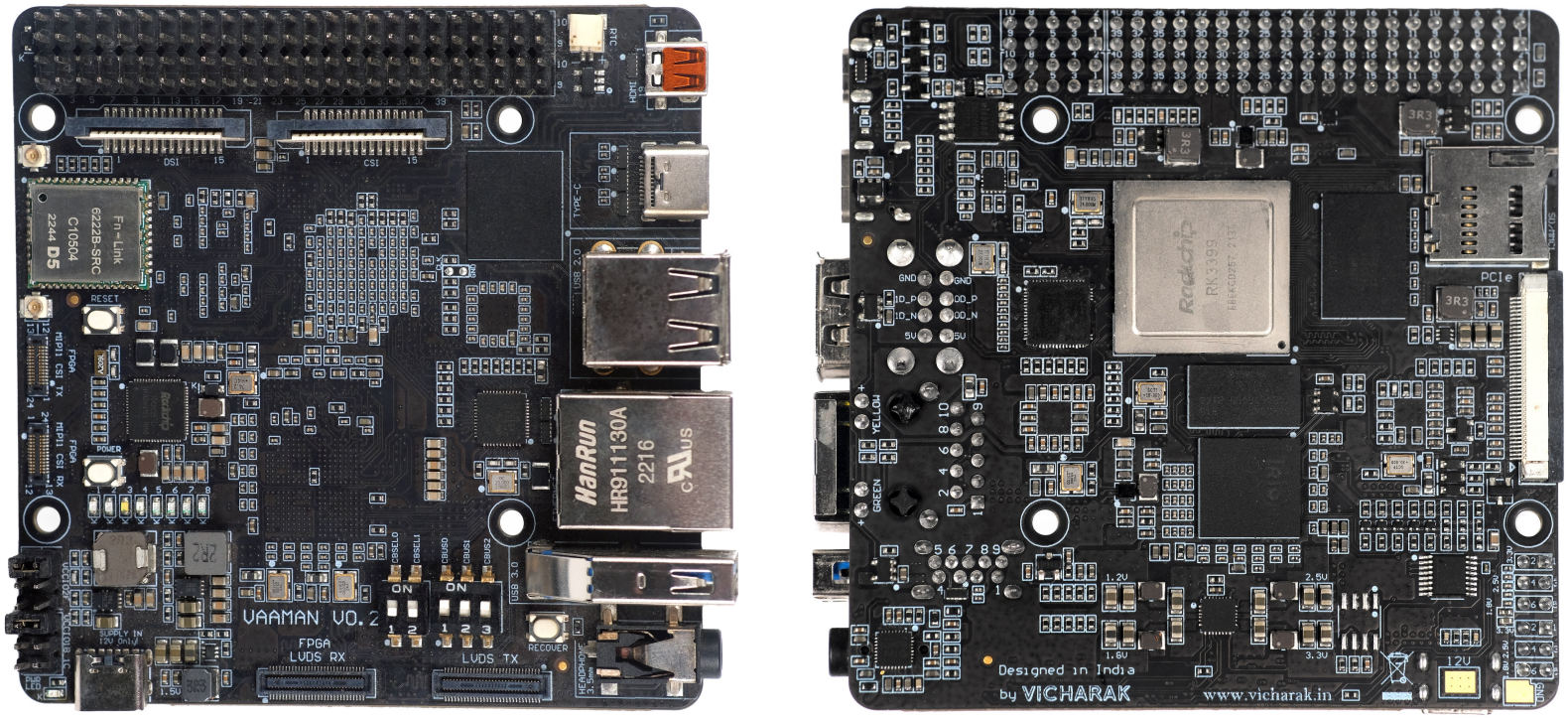


Figure: Vaaman: A heterogenous SBC

Implementation: ML Accelerator (Gati)

Gati is a set of hardware and software programs that perform CNN acceleration with FPGA as a co-processor.

1. At the core of gati is a systolic array pipeline based MAC engine
2. The Gati-ISA is macro (i.e. implements complex operations directly like Convolution) instead of breaking them down into primitives
3. The instructions have almost a one-to-one match with 'layers' from a neural network
4. Assisting this hardware is a Compiler/Runtime.
5. The compiler does two primary things:
    1. Parsing of input data and NN models (protobufs etc.), transpositions of kernels to allow contiguos memory access, and generation of a byte stream that can be fed to the FPGA
    2. Generating custom hardware for every nn model
6. The runtime partitions a network into execute-on-host and execute-on-device, re-orders inputs, and offloads computation

Gati has an ISA? But you said ISAs are bad?

Gati is a testbed for modelling complex problems found in real world. At the moment it does and doesn not do many things that we eventually want from it. For example:

1. Gati has a hardware generator. If this generator is generalized enouugh, we end up solving a part of problem 1.
2. It still uses an ISA. But its possible to partition an ML model so that it can entirely fit into the FPGA hardwired to do only a part of the model followed by reconfiguration to execute later parts.

Implementation: Peripheral Multiplexor (Periplex)

1. Periplex is a interface translator that allows communication between a set of inputs from a set of protocols to a set of outputs in another set of protocols
2. Consider, an application where 2 inputs each from an SPI bus and an I2C bus need to drive 2 motors whose controller speaks CAN. Periplex can be used to easily make hardware and enable this communication.
3. Periplex is, in a sense, the swiss army knife of embedded communication protocols.

Conclusion

1. Reconfigurable architectures can provide a way to solve many problems that existing compute struggle with
2. Reconfigurable architectures as demostrated previously can help alleviate von-neumann bottleneck.

References I

[1] Stephen D. Brown et al. *Field-programmable gate arrays*. USA: Kluwer Academic Publishers, 1992. ISBN: 0792392485.