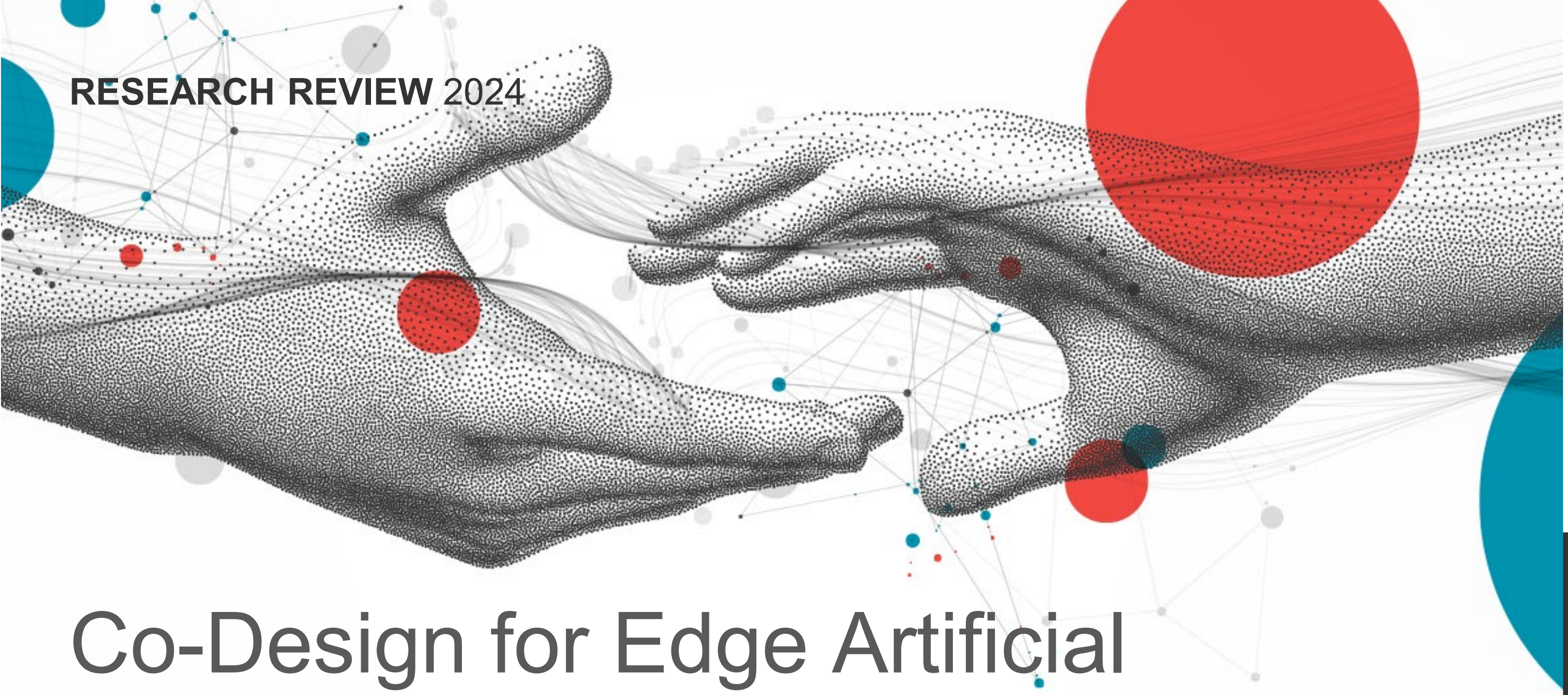


Co-Design for Edge Artificial Intelligence: Application-Specific System on Chip

NOVEMBER 13, 2024

Dr. John G. Wohlbier
Principal Research Scientist
Advanced Computing Lab Lead

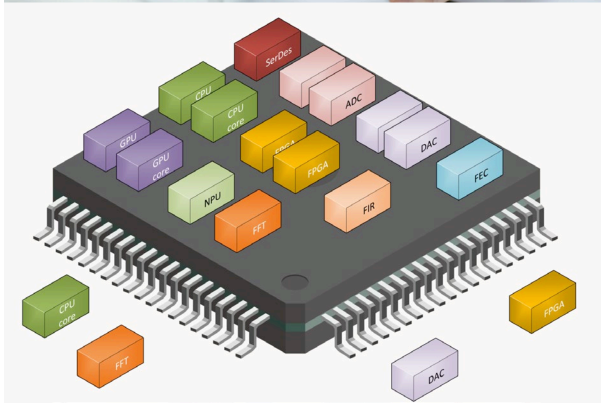
Document Markings

Agenda

- Application-specific system on chips (SoCs)
- Modulation recognition
- Accelerator implementation
- SoC integration
- Future work
- Resources

Carnegie Mellon University Software Engineering Institute

Application-Specific SoCs



- Edge devices, such as drones, are powered by microelectronics.
  - Accelerometer
  - Camera
  - Flight controller
  - GPS module
  - Speed controller
  - Radios (receive and transmit [Tx/Rx])
- What is an SoC?
  - A chip containing components that comprise a system
  - Components: CPU cores, GPU cores, memory cores, accelerators
  - Accelerators: fast Fourier transform (FFT), natural language processing (NLP), NVIDIA Deep Learning Accelerator (NVDLA), Tx/Rx

Application-Specific SoCs

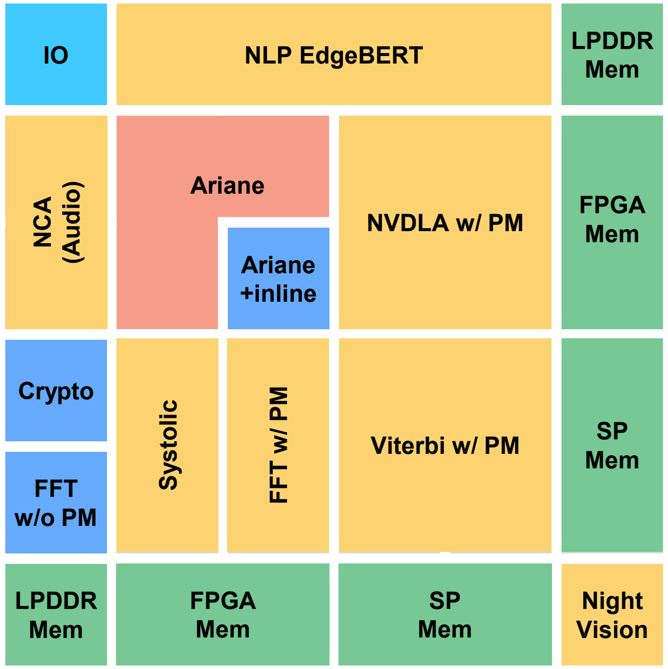


Image courtesy of IBM EPOCHS

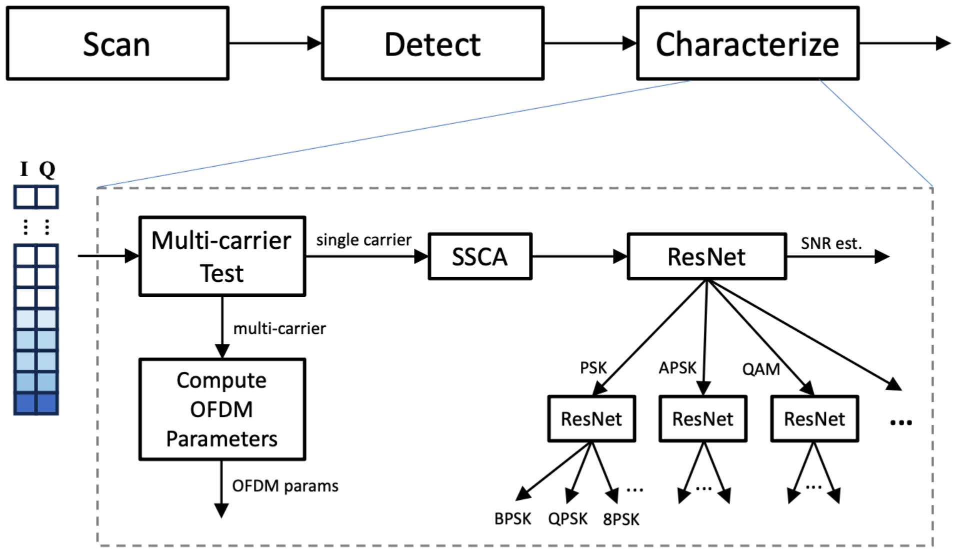
- Electronics' efficiency plays a major role in system performance.
  - Designing an SoC for a particular deployment can yield major benefit.
- What makes an SoC application specific?
  - The mix of system components is designed from the application.
  - Examples: object detection, object classification, signal characterization
- How do you design an application-specific SoC?
  - Hardware description languages (HDL): Chisel, Verilog
  - High-level synthesis (HLS): Bambu, oneAPI, AMD/Xilinx
  - Frameworks: Chipyard, Embedded Scalable Platforms (ESPs)

"In general, ***compute*** can improve mission time and lower energy consumption by as much as 5X."

Boroujerdian, Behzad; Genc, Hasan; Krishan, Srivatsan; Faust, Aleksandra; & Reddi, Vijay Janapa. Why Compute Matters for UAV Energy Efficiency? *International Symposium on Aerial Robotics*. 2018.

Carnegie Mellon University Software Engineering Institute

Modulation Recognition



- Edge environment senses radio frequency spectrum
- Workload designed to make sense of detected signals
- Signal processing for detection and data rate computation
- Machine learning (ML) for modulation classification

Accelerator Design Flow

- Identify hot spots of application(s) to accelerate.
  - Profiling, source code analysis, expert knowledge
- Accelerator design options
  - HDL: code the accelerator by hand (time consuming, but can achieve optimal performance)
  - HLS: tools lower code to HDL (much faster, but unlikely to achieve optimal performance)
- High-level synthesis
  - HLS tools supporting PyTorch and Tensorflow are immature
- Emerging approaches
  - Multi-Level Intermediate Representation (MLIR) compiler framework for model lowering
  - Dialect abstraction to enable different types of optimizations

Strip Spectral Correlation Algorithm (SSCA)

Definition: The cyclic auto-correlation function of a time-series x(t) is calculated as follows:

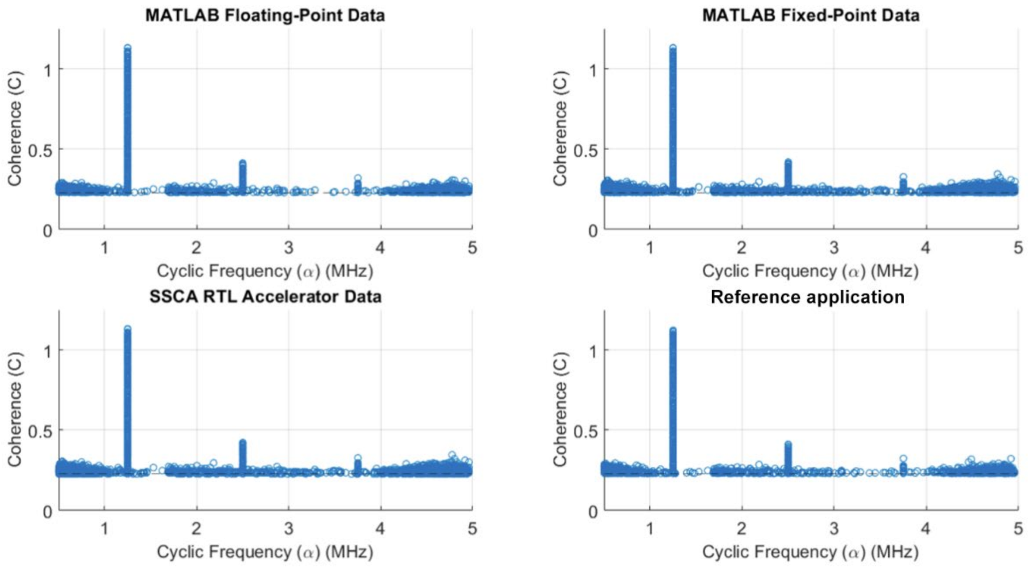
$$R_x^\alpha(\tau) = \int_{-\infty}^{\infty} x\left(t - \frac{\tau}{2}\right) x^*\left(t + \frac{\tau}{2}\right) e^{-i2\pi\alpha t} dt$$

where (*) denotes complex conjugation. By the Wiener-Khinchin theorem, the spectral correlation density is then:

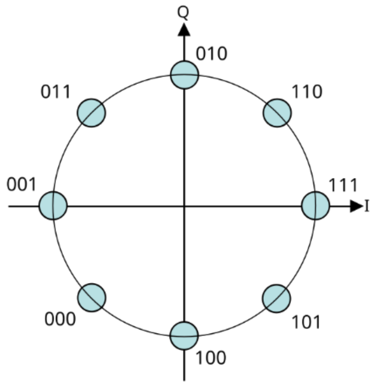
$$S_x^\alpha(f) = \int_{-\infty}^{\infty} R_x^\alpha(\tau) e^{-i2\pi f\tau} d\tau$$

- Cyclostationary processes
  - Signal having statistical properties that vary cyclically with time
  - Functions: cyclic autocorrelation, spectral correlation function (SCF)
  - α is the cyclical frequency (CF)
- SSCA estimates the SCF for all CFs
- For example, 8-phase-shift keying (8-PSK) signal at a specific modulation rate has the detection shown on the next slide.

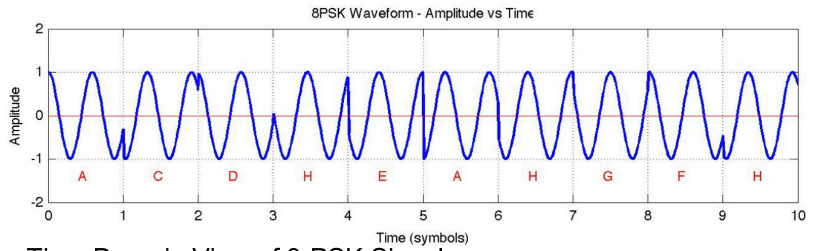
SSCA



Detect Coherence vs. Cyclic Frequency

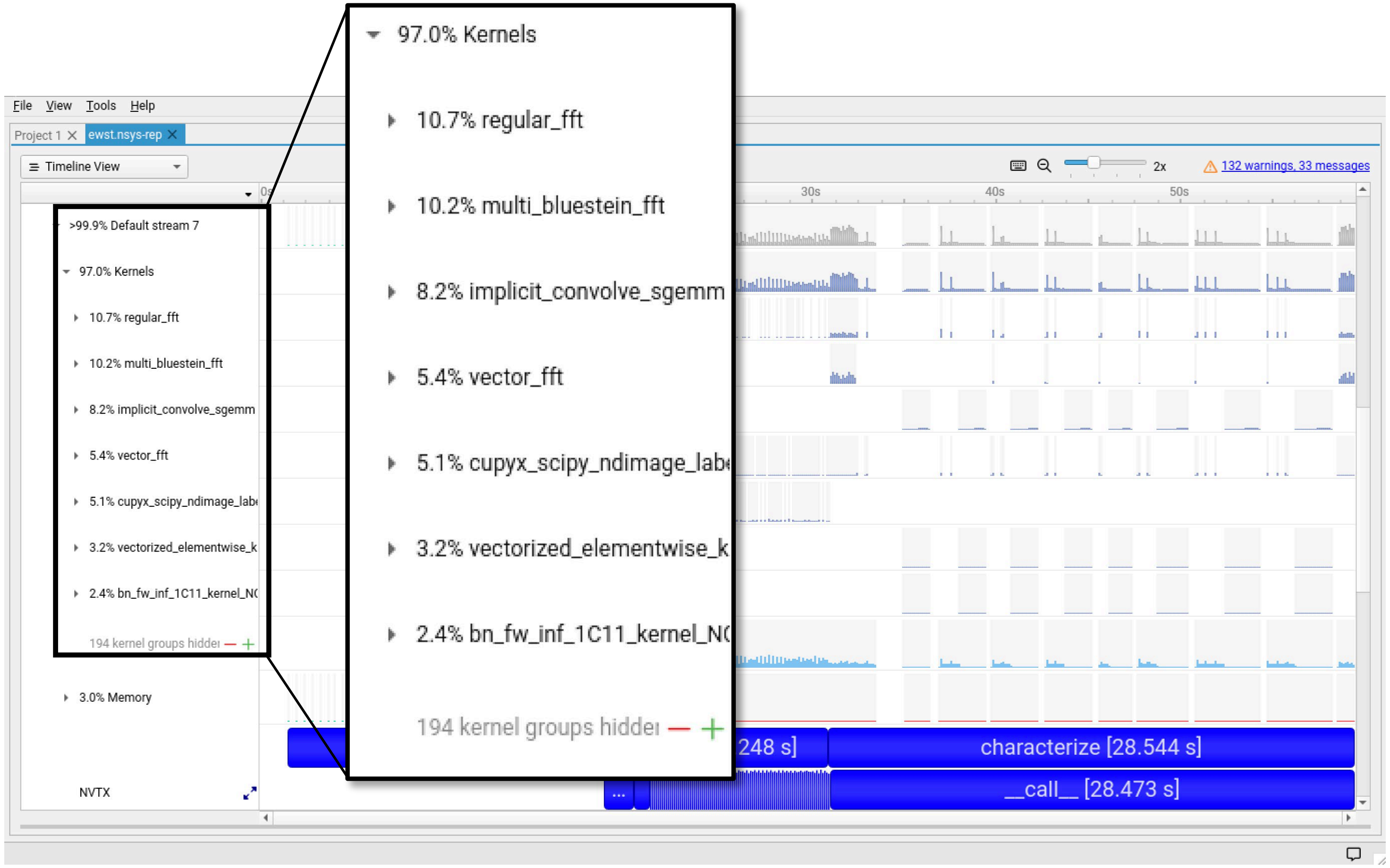


8-PSK Symbol Constellation



Time Domain View of 8-PSK Signal

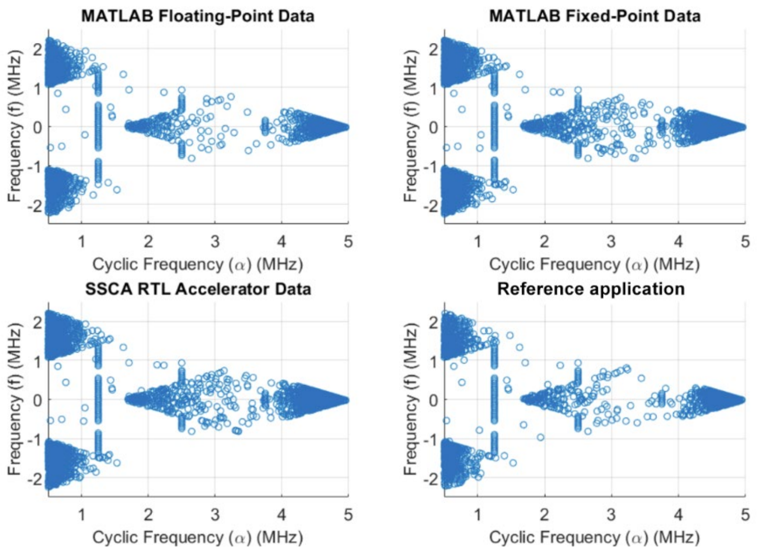
Modulation Recognition Profile



```
>99.9% Default stream 7
  97.0% Kernels
    10.7% regular_fft
    10.2% multi_bluestein_fft
    8.2% implicit_convolve_sgemm
    5.4% vector_fft
    5.1% cupyx_scipy_ndimage_labe
    3.2% vectorized_elementwise_k
    2.4% bn_fw_inf_1C11_kernel_N(
    194 kernel groups hidde
  3.0% Memory
```

- Scan
  - Radio sweeps
- Detect
  - Noise floor estimate
- Characterize
  - SSCA symbol rate
  - ML classify for modulation types

SSCA Accelerator



Detects in Bi-Frequency Plane

SSCA Algorithm Implementation	Total Duration (s)
Reference Application	334.12
SSCA RTL Accelerator - 1 Length N FFT	67.04
SSCA RTL Accelerator - 16 Length N FFTs	15.59

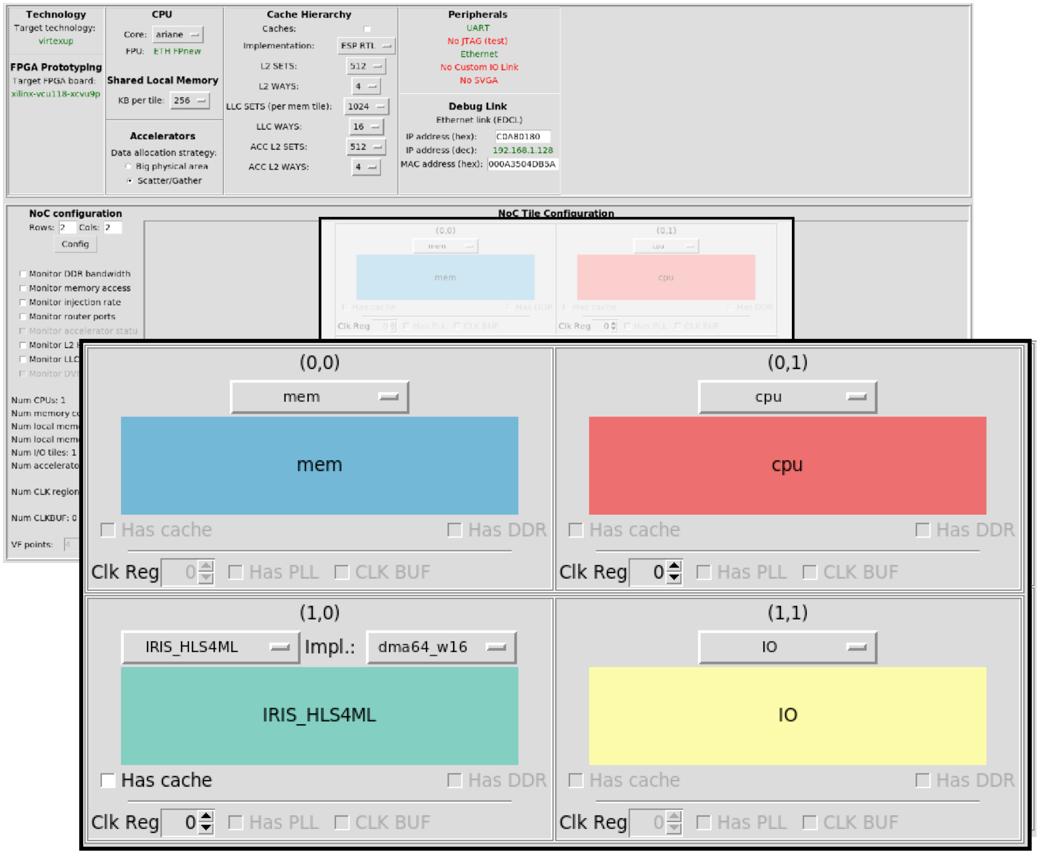
- SSCA: FFT heavy workload
- Implemented in VHDL with Xilinx FFT IP
- Synthesized into SoC with RISC-V CPU core and SSCA run on VCU118
- Defined benchmark to compare PyTorch implementation on Orin to field-programmable gate array (FPGA)

~ 5x – 20x acceleration

SSCA Accelerator

Device Type	LUTs (%)	Registers (%)	F7 Muxes (%)	F8 Muxes (%)	LUT as Logic (%)	LUT as Memory (%)	Block RAM Tile (%)	DSPs (%)	Bonded IOB (%)
profpga-xc7v2000t	35	24	1	0	28	24	18	91	12
profpga-xcvu440	17	12	1	0	13	18	9	68	10
xilinx-vc707-xc7vx485t	139	97	4	2	112	64	23	70	20
xilinx-vcu118-xcvu9p	36	25	1	1	29	14	11	29	17
xilinx-vcu128-xcvu37p	32	23	1	0	26	14	12	22	23
xilinx-zcu102-xczu9eg	154	107	5	2	124	59	26	78	44
xilinx-zcu106-xczu7ev	184	128	6	3	147	83	76	113	40
xilinx-z7020-xc7z020	795	553	24	11	637	485	170	889	72
xilinx-z7045-xc72045	193	135	6	3	155	120	44	217	40

SoC Integration



Technology
Target technology: virtexup

FPGA Prototyping
Target FPGA board: xilinx-vcu118-xcvu9p

CPU
Core: ariane
FPU: ETH FPnew

Shared Local Memory
KB per tile: 256

Accelerators
Data allocation strategy:
○ Big physical area
● Scatter/Gather

Cache Hierarchy

Caches:	☐
Implementation:	ESP RTL
L2 SETS:	512
L2 WAYS:	4
LLC SETS (per mem tile):	1024
LLC WAYS:	16
ACC L2 SETS:	512
ACC L2 WAYS:	4

Peripherals
UART
No JTAG (test)
Ethernet
No Custom IO Link
No SVGA

Debug Link
Ethernet link (EDCL)

IP address (hex):	C0A80180
IP address (dec):	192.168.1.128
MAC address (hex):	000A3504DB5A

NoC configuration
Rows: 2 Cols: 2
Config

☐ Monitor DDR bandwidth
☐ Monitor memory access
☐ Monitor injection rate
☐ Monitor router ports
☐ Monitor accelerator statu
☐ Monitor L2
☐ Monitor LLC
☐ Monitor DV

Num CPUs: 1
Num memory c
Num local mem
Num local mem
Num I/O tiles: 1
Num accelerato

Num CLK region

Num CLKBUF: 0

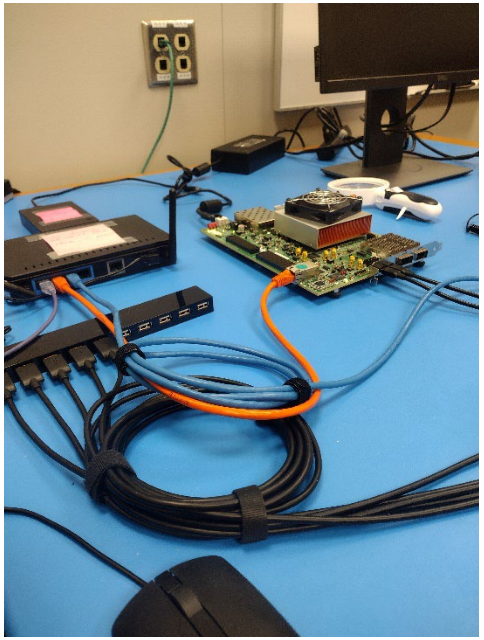
VF points: 4

NoC Tile Configuration

(0,0) mem — mem ☐ Has cache ☐ Has DDR — Clk Reg 0 ☐ Has PLL ☐ CLK BUF	(0,1) cpu — cpu ☐ Has cache ☐ Has DDR — Clk Reg 0 ☐ Has PLL ☐ CLK BUF
(1,0) IRIS_HLS4ML Impl.: dma64_w16 — IRIS_HLS4ML ☐ Has cache ☐ Has DDR — Clk Reg 0 ☐ Has PLL ☐ CLK BUF	(1,1) IO — IO ☐ Has cache ☐ Has DDR — Clk Reg 0 ☐ Has PLL ☐ CLK BUF

- Using ESP from Columbia University
  - Includes selectable CPUs, memory cores, input/output (I/O) cores, selection of accelerators
- Example: four-tile design with an RISC-V CPU core, memory tile, I/O tile, and PyTorch-Iris accelerator
- Implementation on FPGA with AMD/Xilinx tools
  - Design flow for application-specific integrated circuit (ASIC) also available

SoC Testing: FPGA Implementation



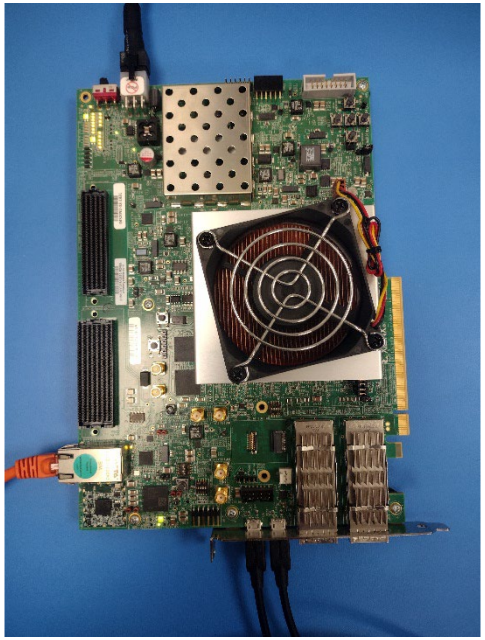
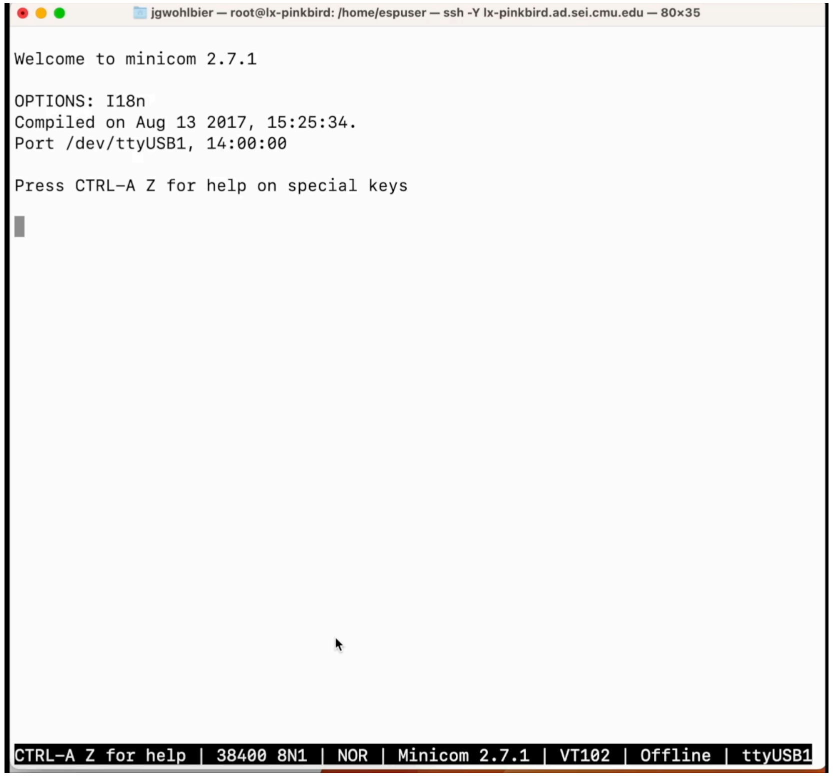


Image 1: FPGA Dev Board Xilinx VCU118 connected to host
Image 2: Booting Linux on RISC-V core and then running application, invoking the accelerator

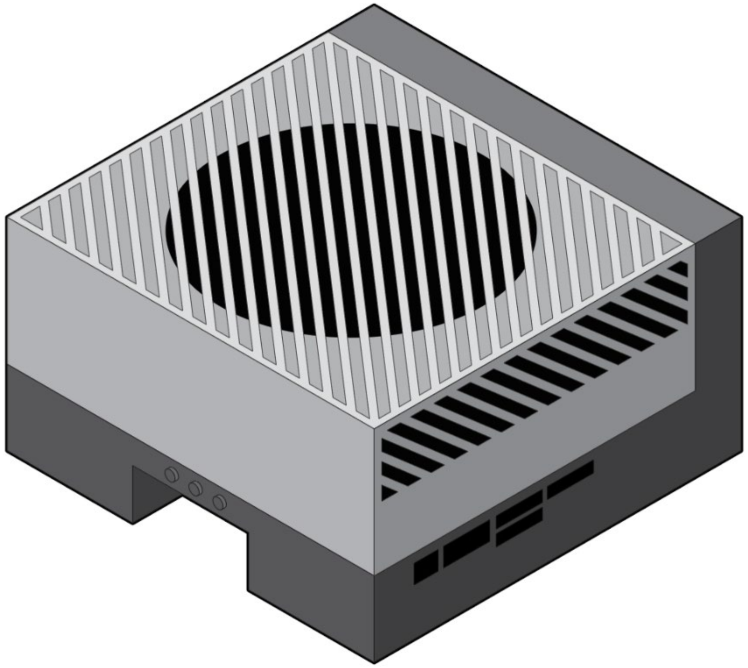


jgwohlbier — root@lx-pinkbird: /home/espuser — ssh -Y lx-pinkbird.ad.sei.cmu.edu — 80×35

```
Welcome to minicom 2.7.1

OPTIONS: I18n
Compiled on Aug 13 2017, 15:25:34.
Port /dev/ttyUSB1, 14:00:00

Press CTRL-A Z for help on special keys



CTRL-A Z for help | 38400 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyUSB1
```

Carnegie Mellon University Software Engineering Institute

Future Work



- Accelerator synthesis for Resnet neural network
- Integration of SoC containing neural network accelerator, SSCA accelerator, and RISC-V CPU core
- Performance quantification and comparison to NVIDIA Orin

Resources

Carnegie Mellon University Software Engineering Institute

- Reproduce these results using open source code
- CMU-SEI GitHub
  - PyTorch-Iris – hls4ml branch – simple neural network code
    - https://github.com/cmu-sei/pytorch-iris
  - Docker environment for using hls4ml
    - https://github.com/cmu-sei/hls4ml-docker
  - Docker environment for using ESP
    - https://github.com/cmu-sei/esp-docker
- ESP tutorials
  - https://www.esp.cs.columbia.edu/docs/

Presenter

Dr. John G. Wohlbier
Principal Research Scientist
Advanced Computing Lab Lead

Telephone: +1 412.268.5800
Email: info@sei.cmu.edu