CoDesign in Action: Dynamic Infrastructure Services Layer (DISL)

Ahmed Sanaullah
Senior Data Scientist
Red Hat Research

Jason Schlessman
Principal Software Engineer
Red Hat Research

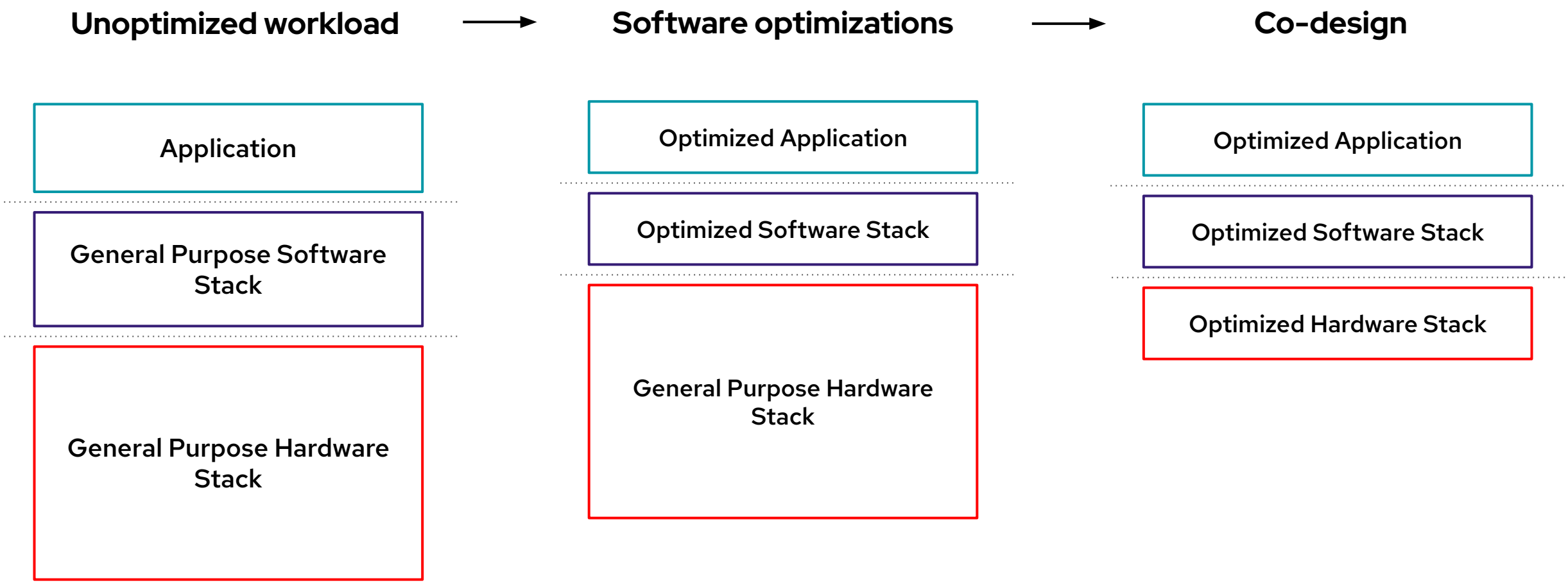
Ulrich Drepper
Distinguished Engineer
Red Hat Research

Red Hat Research

Overview

- The value of co-design
- The value of open source hardware tooling
- CoDes lab @ Red Hat - BU Collaboratory
- The value of FPGAs in particular
- Why are FPGAs so difficult to use?
- Dynamic Infrastructure Services Layer (DISL)
- Demo: Building a custom wireless security system using off-the-shelf components

The value of co-design



Red Hat Research

The value of open source hardware tooling

Lower cost and greater accessibility

Greater flexibility / customizability for IP blocks and tooling

Increased innovation and community collaboration

No lock-ins

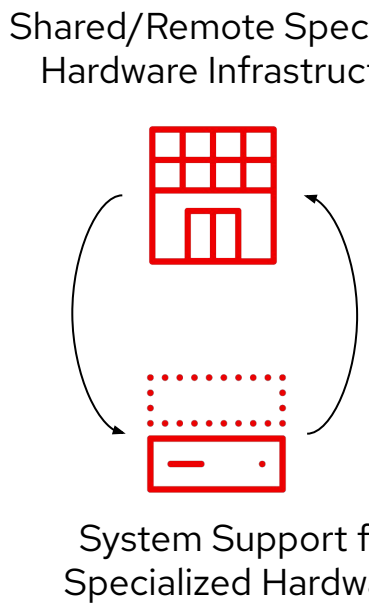
Better security and transparency

More applications and use cases

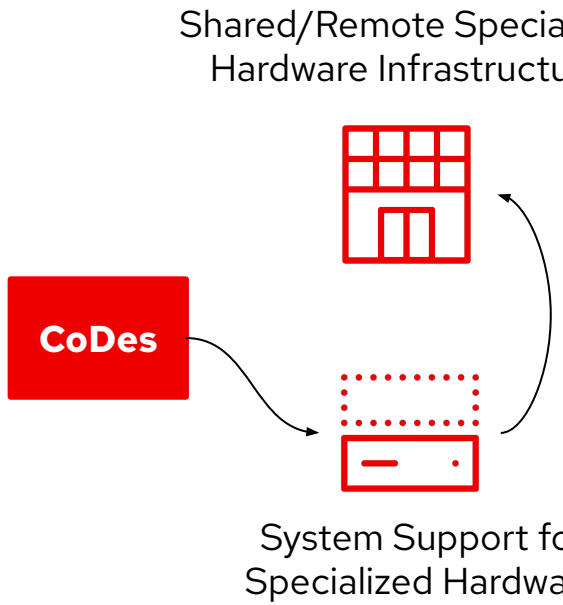
CoDes Lab @ Red Hat - BU Collaboratory

An on-premise research lab at Boston University as part of the Red Hat - BU collaboratory

Provides the infrastructure and engineering foundation needed to support co-design research



The shared infrastructure deadlock



CoDes as an on-premise incubation step

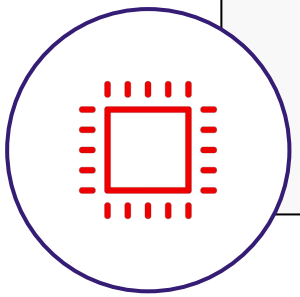
The value of Field Programmable Gate Arrays (FPGAs)

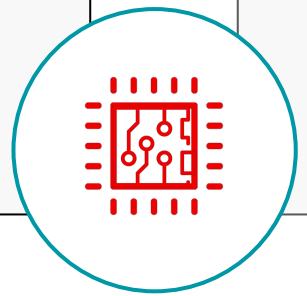
vs. Microcontrollers

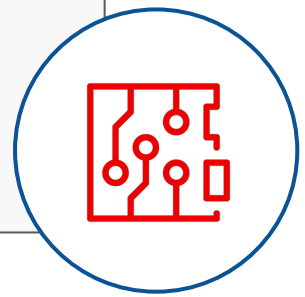
- Greater flexibility in how applications are deployed
- More connectivity options for external I/O
- Higher energy efficiency
- Better performance
- Prevents vendor lock-in of the software stack
- Do not have to compete with tools like FreeRTOS

vs. ASICs

- Faster time to market
- Lower risk of over-specialization
- Prevents vendor lock-in of the software stack
- Can emulate/test software for ASICs
- Cheaper for smaller numbers







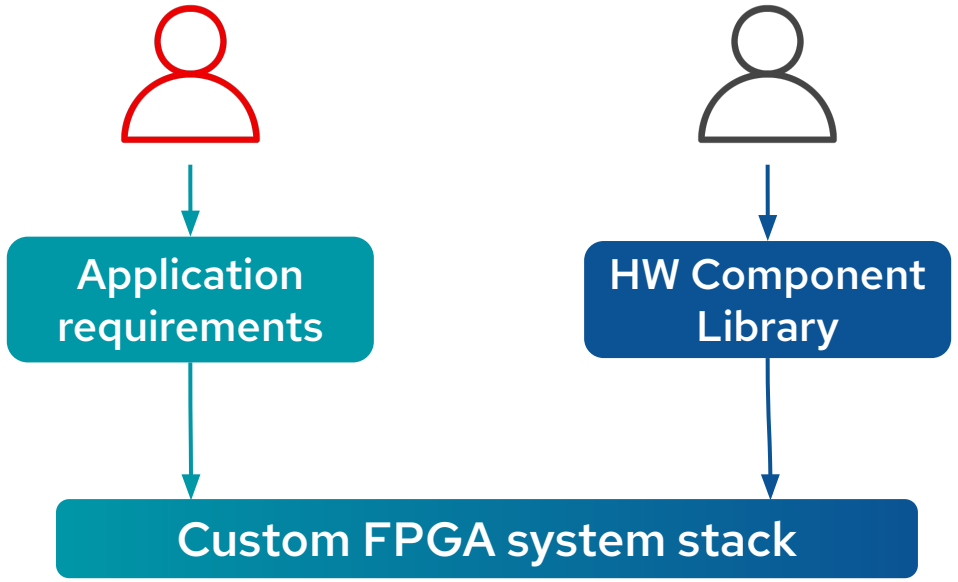
Microcontrollers

FPGAs

Application Specific Integrated Circuits

Programmability ⟷ **Performance**

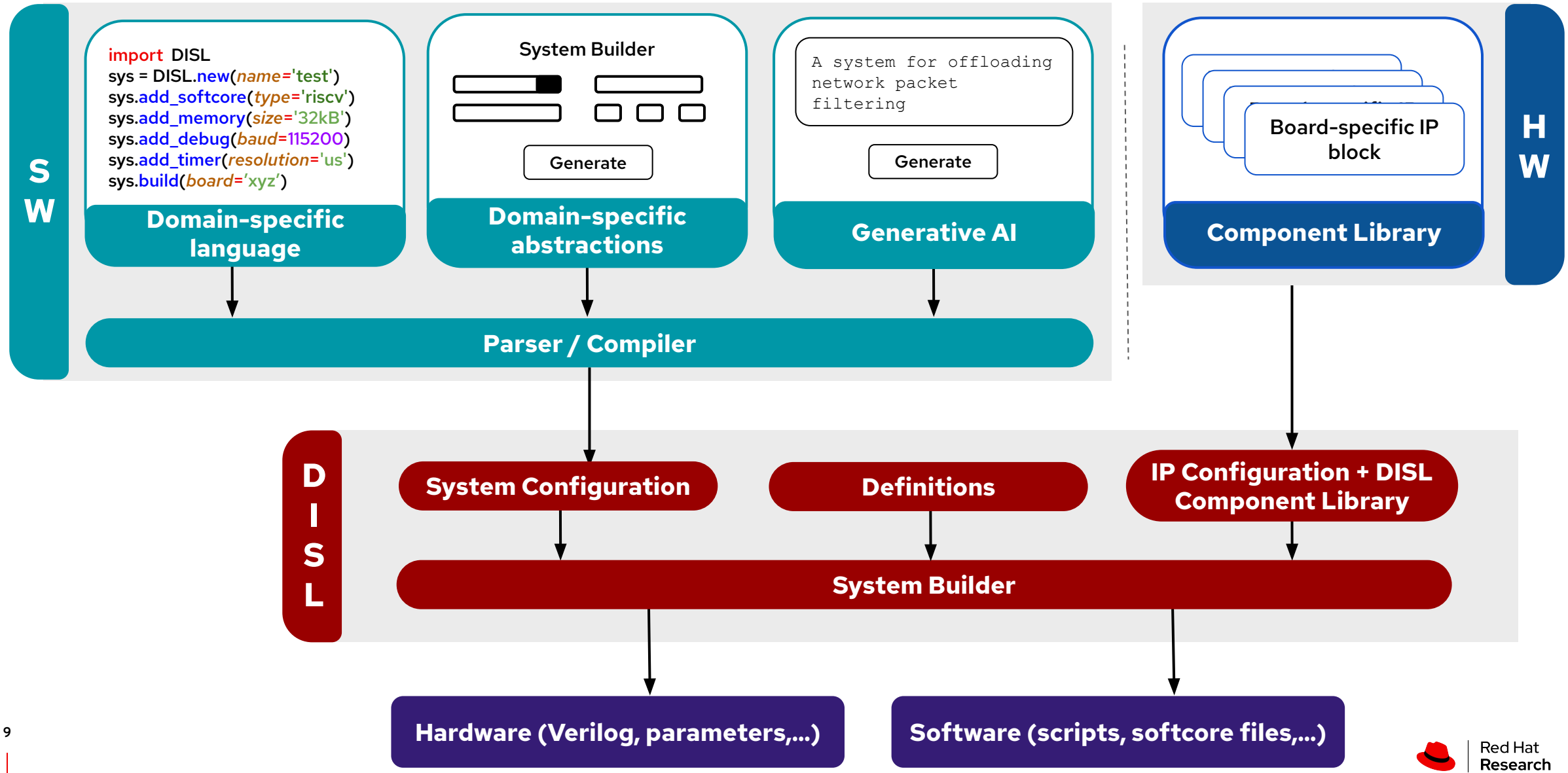
Why are FPGAs so difficult to use?



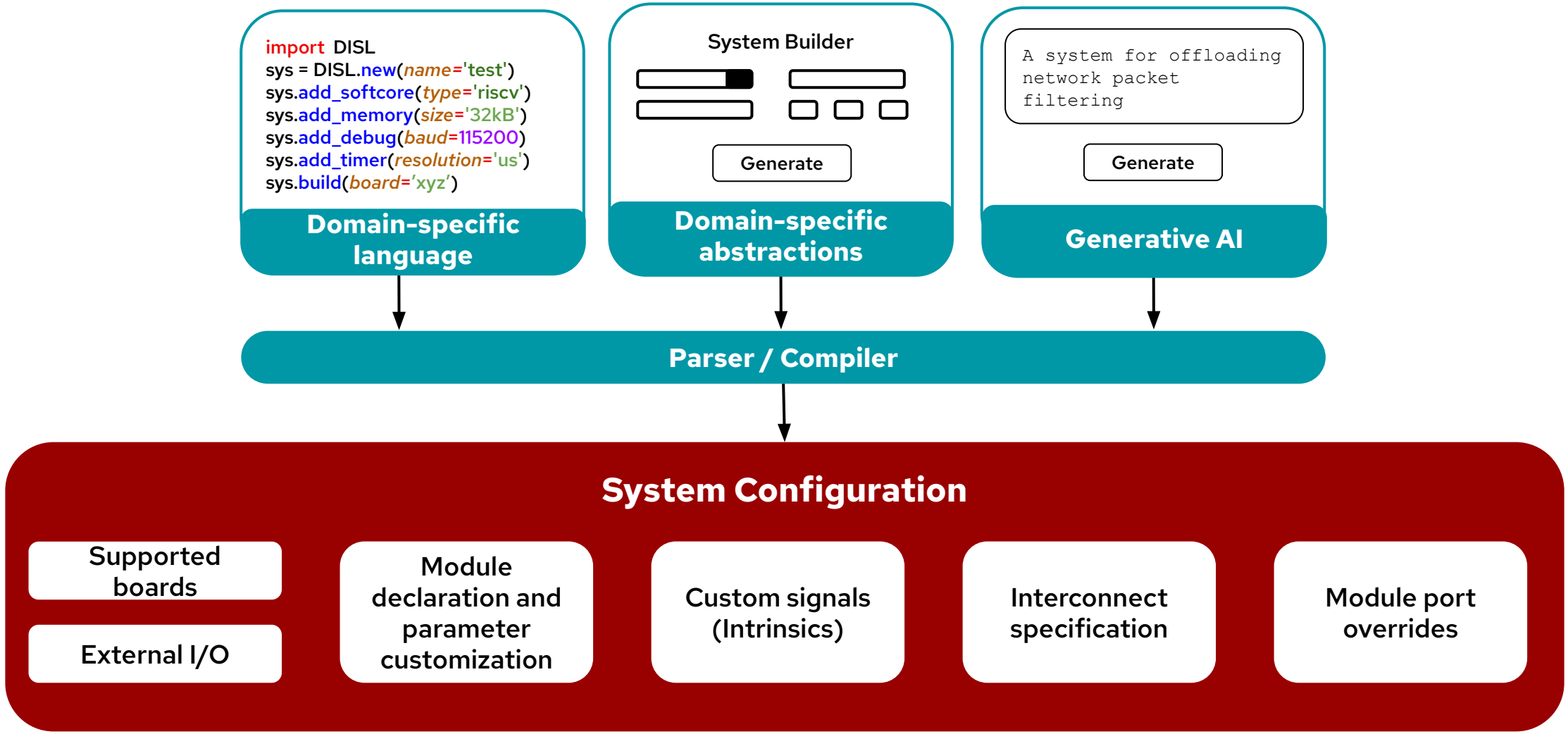
Inefficient development flow between software developer and hardware developer

Dynamic **I**nfrastructure **S**ervices **L**ayer (DISL)

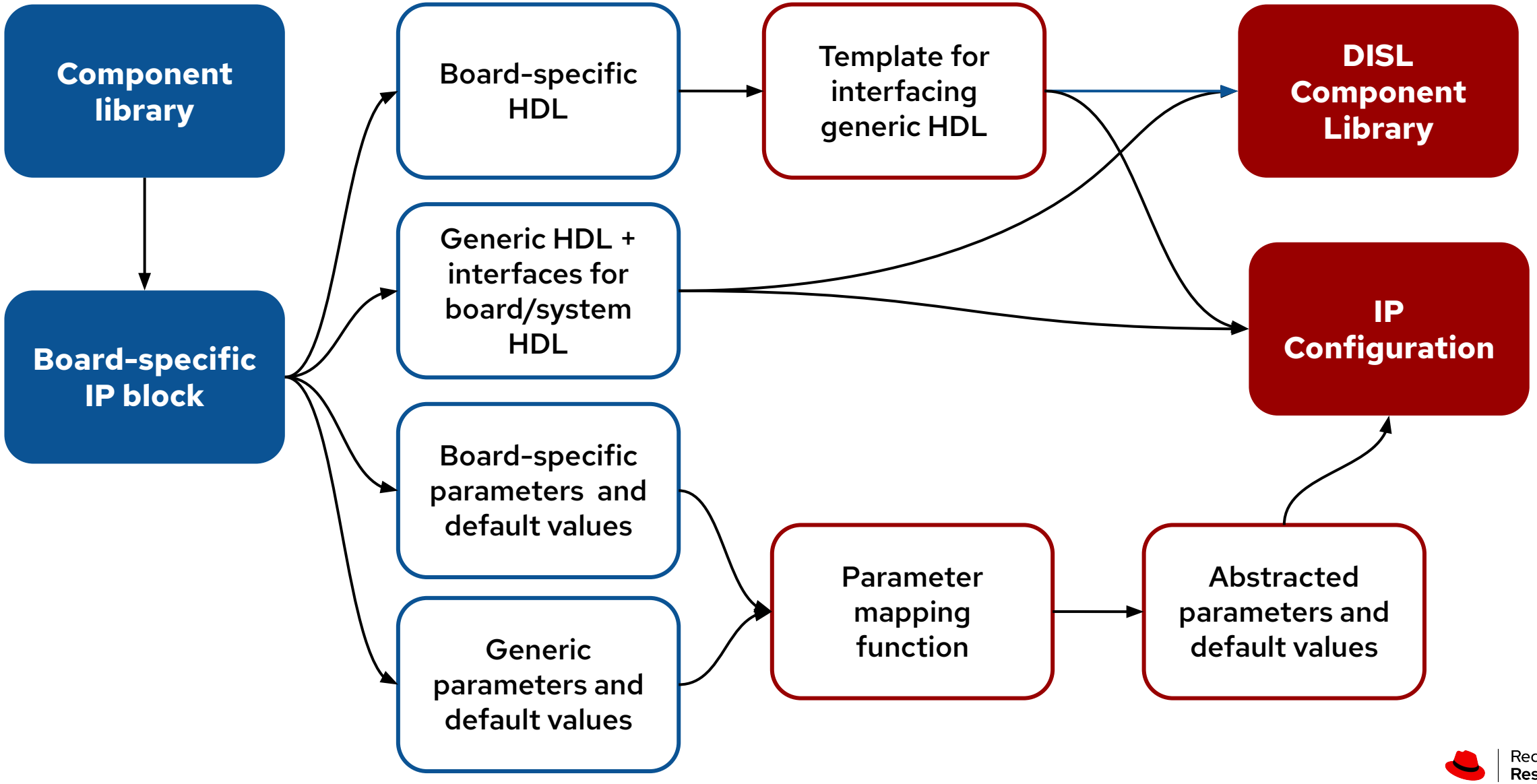
The DISL abstraction layer



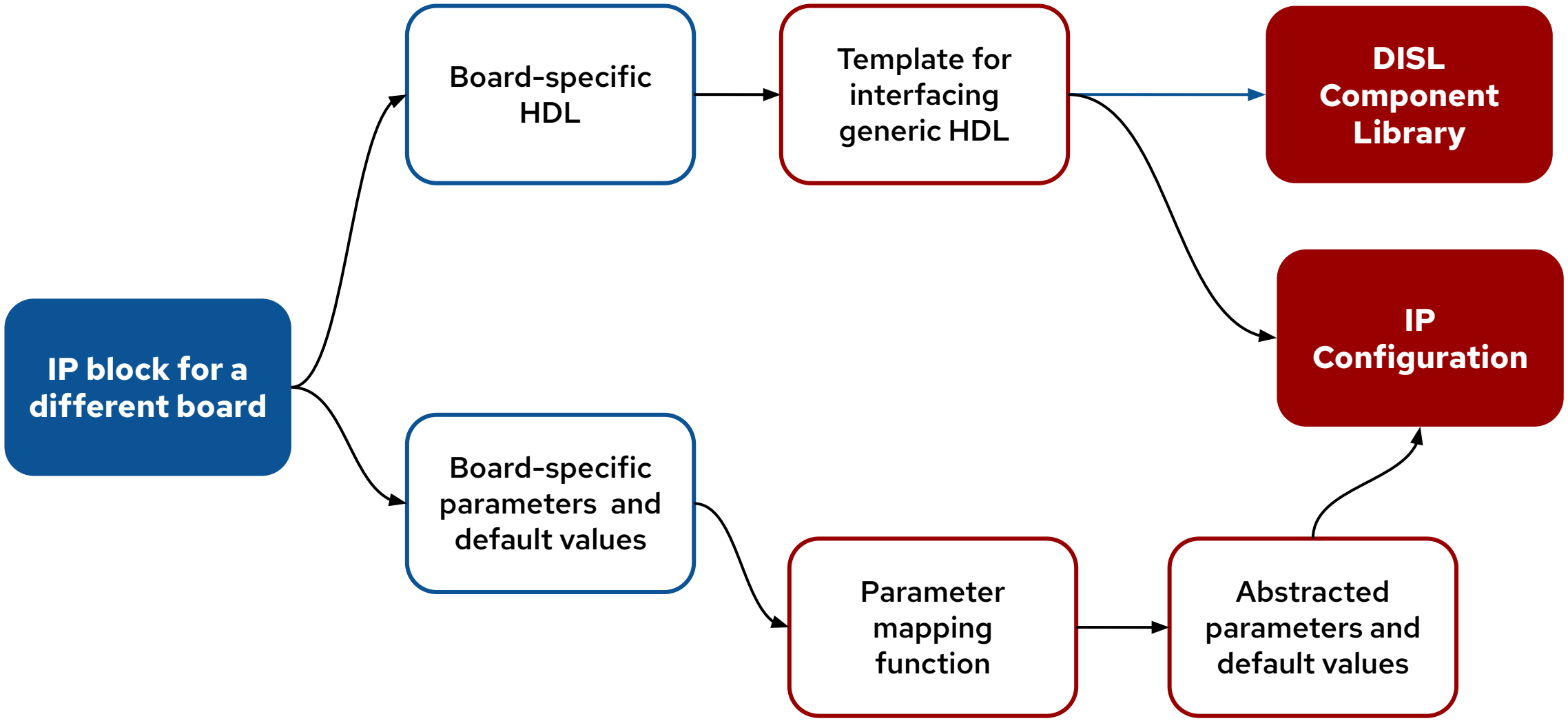
SW interface: System Configuration



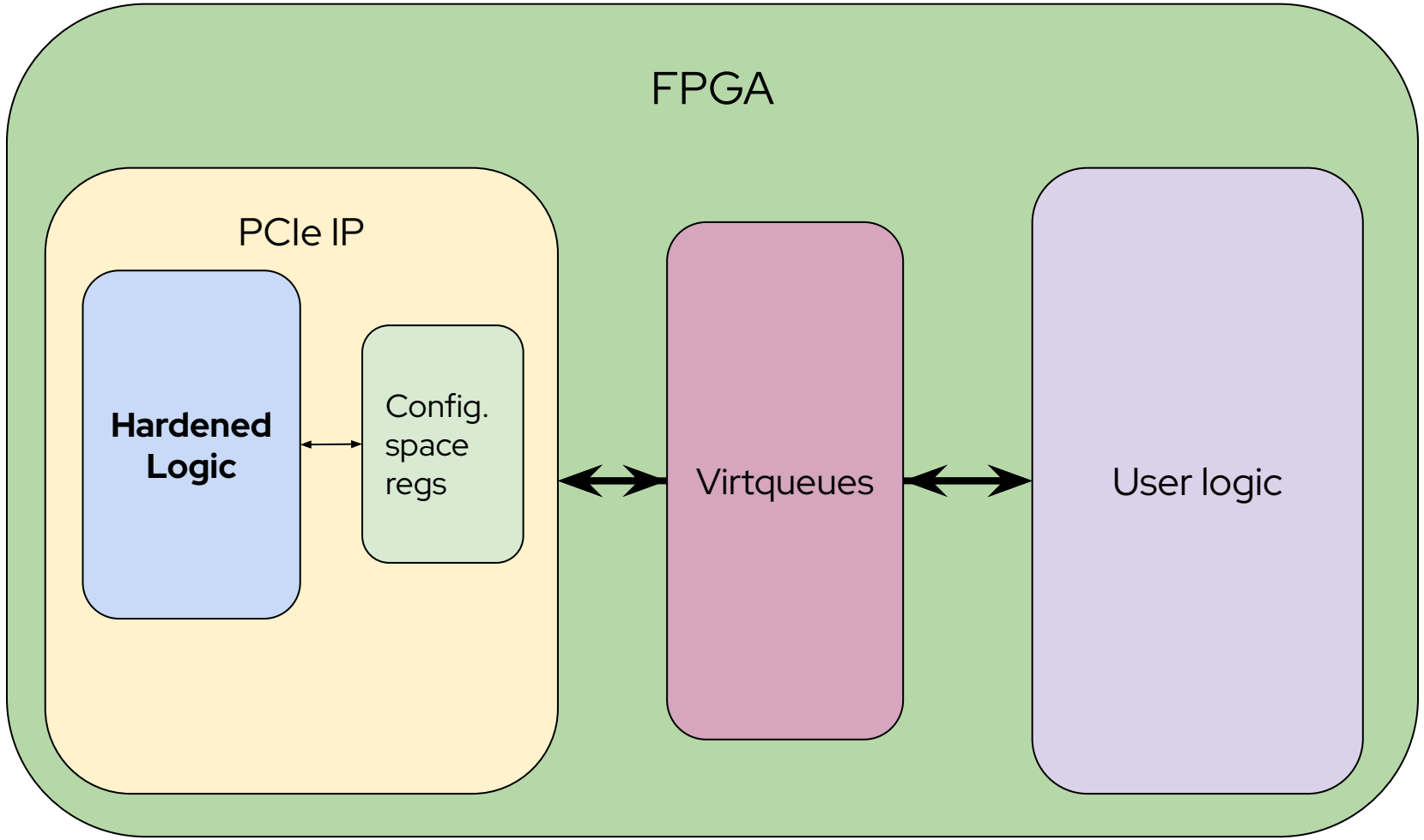
HW interface: IP Configuration and DISL Component Library



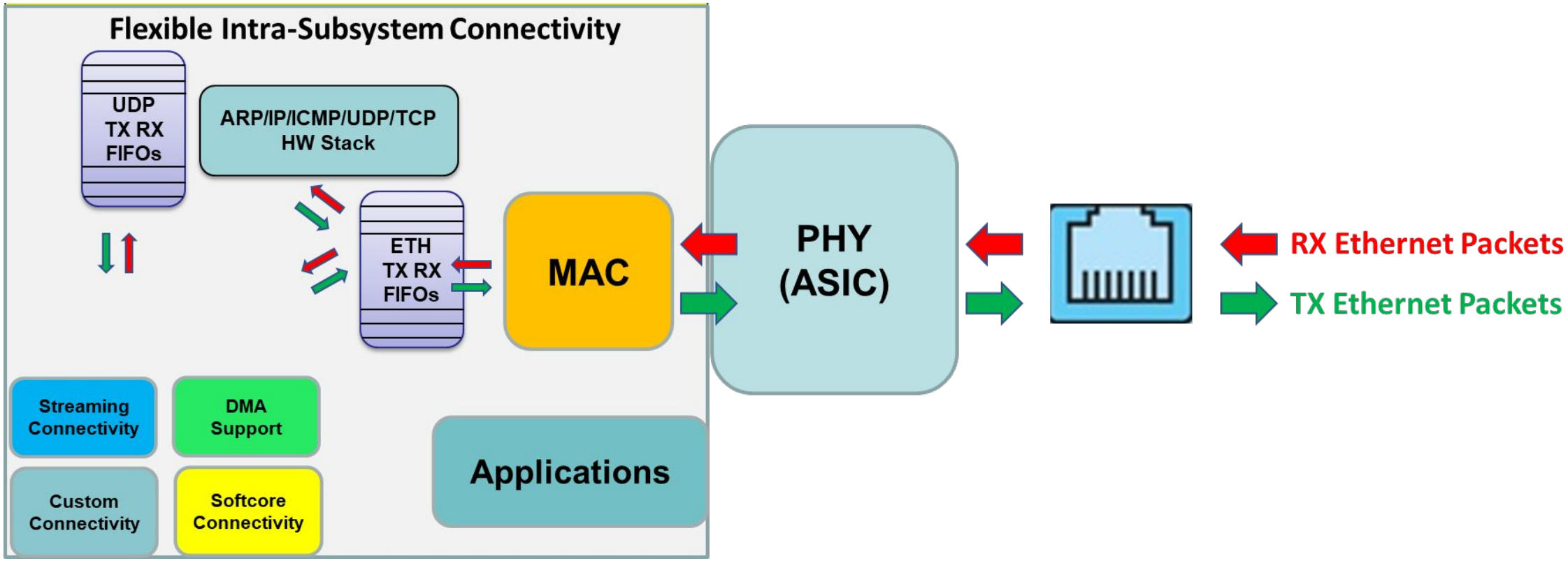
HW interface: IP Configuration and DISL Component Library



Building the DISL component library: PCIe subsystem

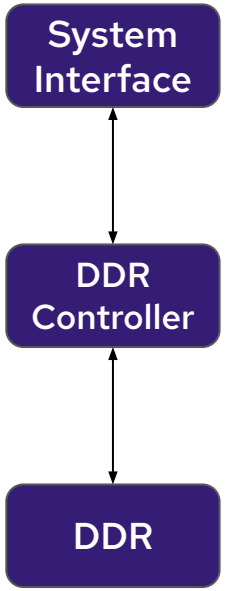


Building the DISL component library: Ethernet subsystem

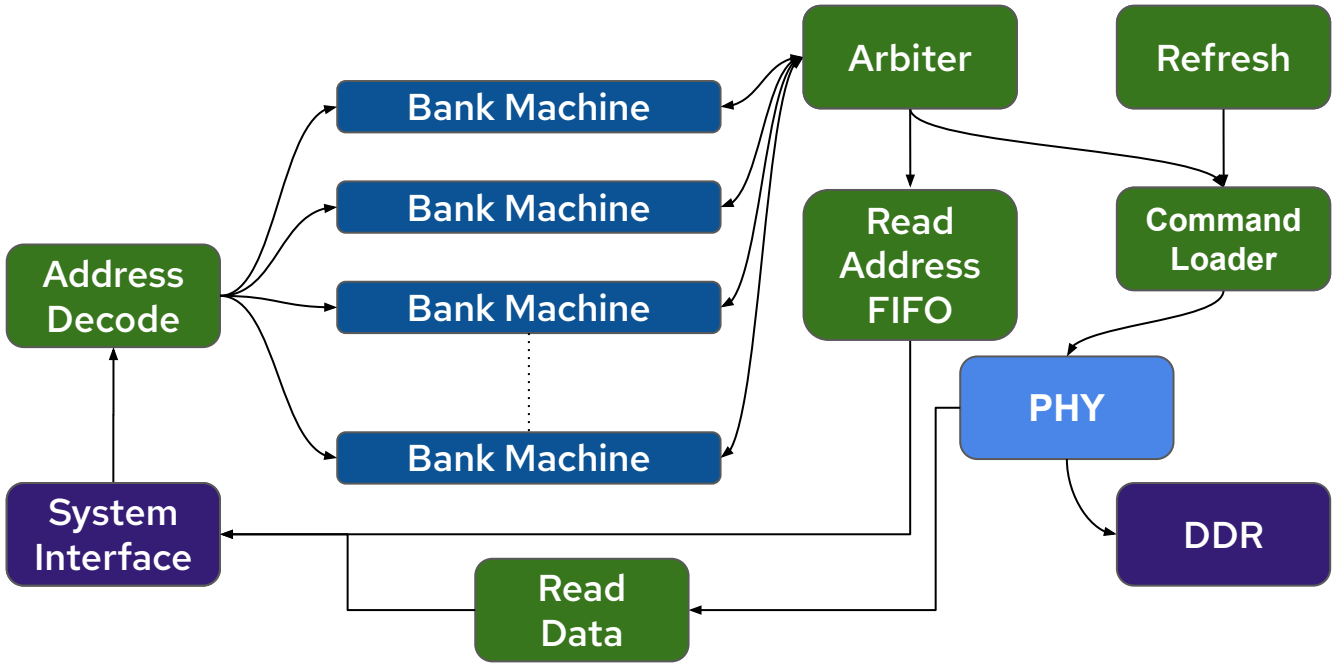


Building the DISL component library: DDR subsystem

Typical IP usage

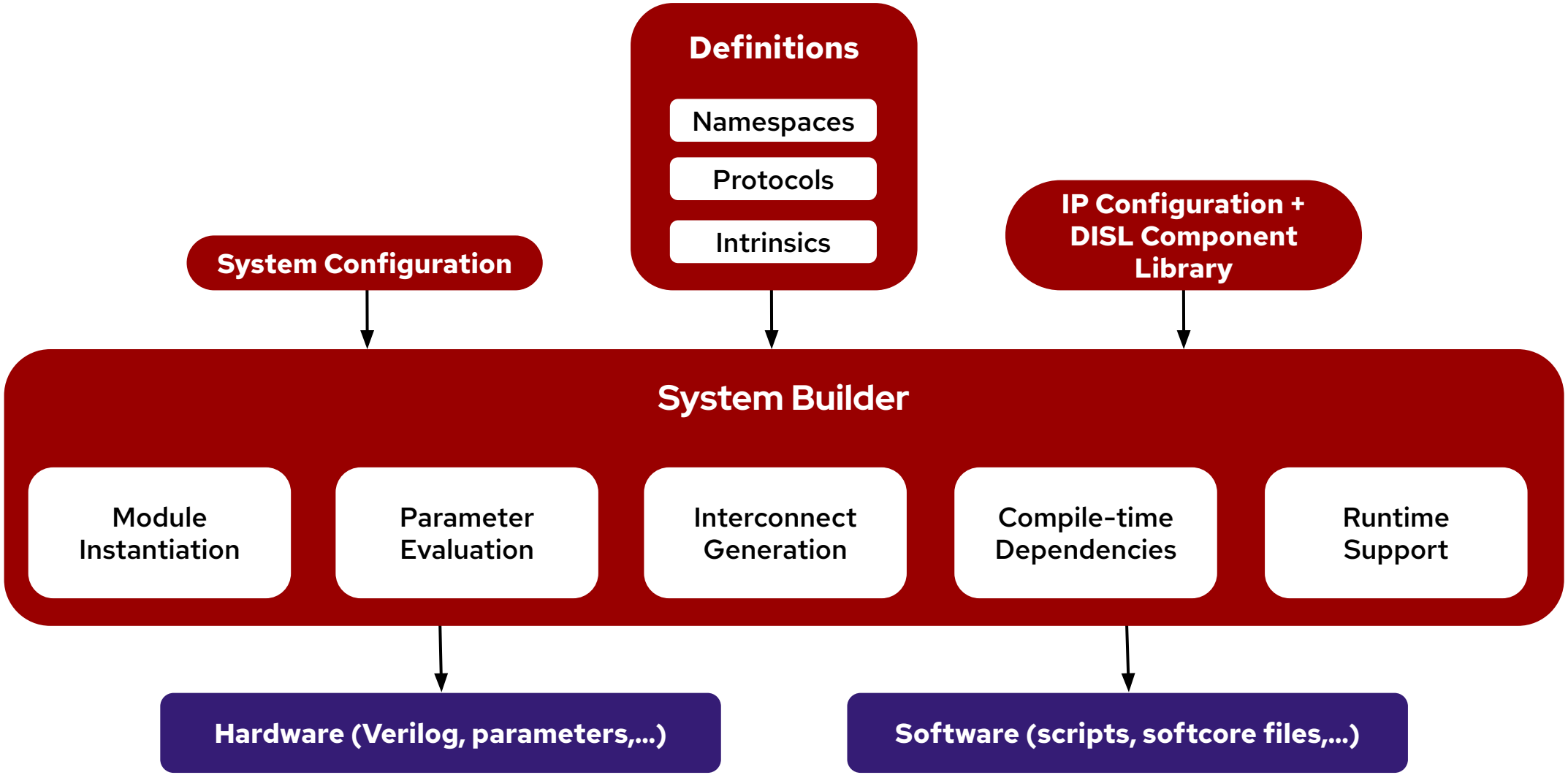


Potential for compile-time and run-time configurability



Red Hat Research

DISL System Builder



Demo:
Building a custom wireless security system using off-the-shelf components

Requirements

- No vendor cloud lock-in
- Low cost, off-the-shelf components
- Open source hardware IP and software tooling
- Highly flexible design that can be customized to meet performance/energy constraints
- Ability to wirelessly: i) reconfigure the FPGA, ii) reprogram any running softcores, and iii) communicate with the application.
- Provide a secure design for managing devices in the field

What you'll see in this demo: open tooling and IP blocks

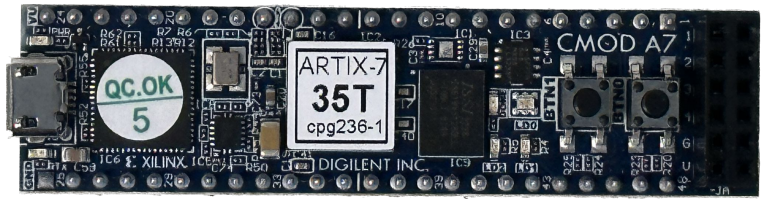
(to the greatest extent possible)

Major open source tooling and IP blocks	
RISC-V toolchain	https://github.com/riscv-collab/riscv-gnu-toolchain
OpenOCD (ported)	https://github.com/openocd-org/openocd
ArduCam Arduino library	https://github.com/ArduCAM/Arduino
Espressif ESP-IDF	https://github.com/espressif/esp-idf
Mosquitto	https://github.com/eclipse/mosquitto
Tensorflow	https://github.com/tensorflow/tensorflow
PicoRV32 RISC-V Softcore	https://github.com/YosysHQ/picorv32
JPEG decoder IP block	https://github.com/ultraembedded/core_jpeg
UART controller	https://nandland.com/uart-serial-port-module/

Major proprietary tooling and IP blocks	
Vivado	For synthesis + Place & Route - open source tooling currently does not support certain PHYs
FPGA PHYs	Vivado MIG PHY for DDR3, BSCANE2 PHY for JTAG support, PLLs for clock generation

What you'll see in this demo: multiple FPGA boards

Cmod A7-35T



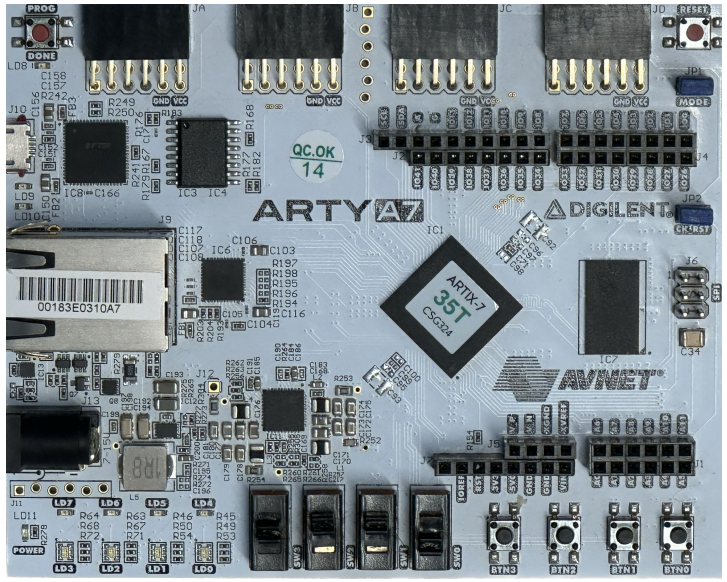
12MHz oscillator

Block RAM only

44 GPIOs

1x PMOD connector

Arty A7-35T



100MHz oscillator

Block RAM + 256MB DDR3 Memory

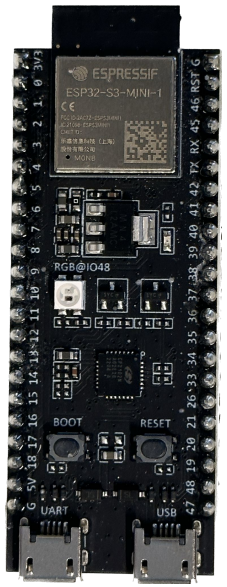
Arduino/chipKIT connectors

4x PMOD connectors

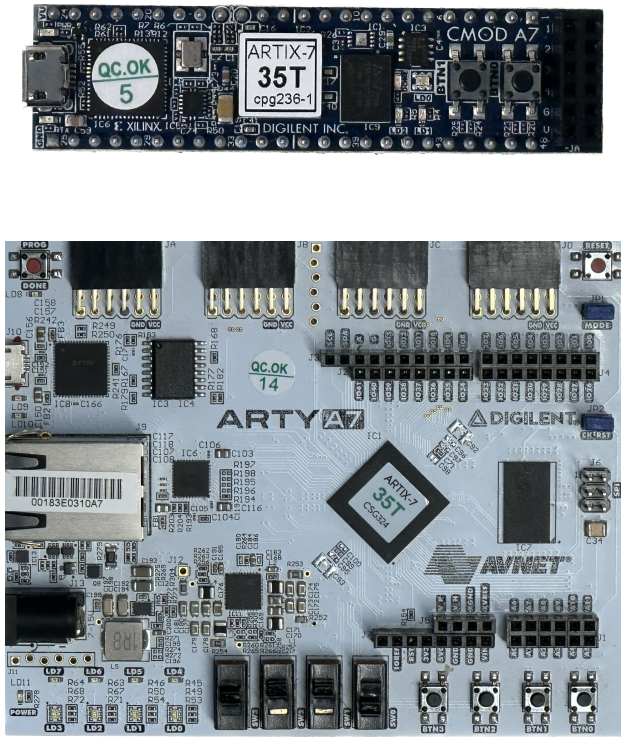
Ethernet PHY

Red Hat Research

What you'll see in this demo: multiple hardware types



Microcontroller
(ESP32-S3)



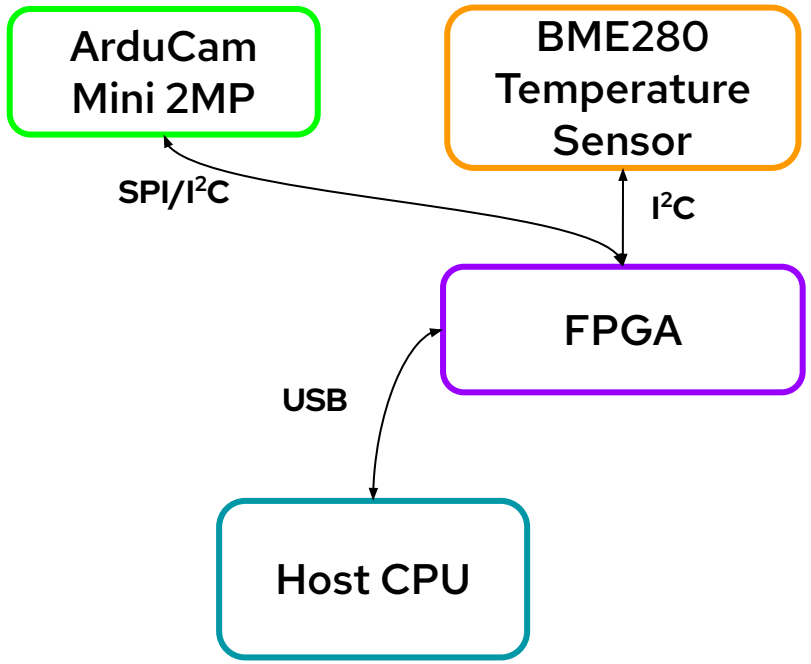
FPGA
(Cmod A7, Arty A7)

ASIC
(ArduCam)

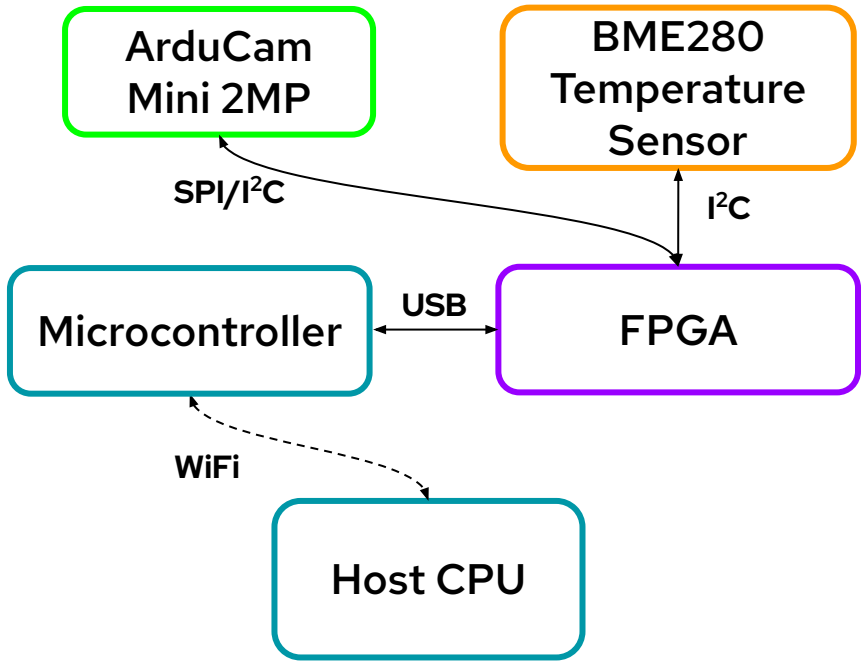
Red Hat Research

Demo overview

Part A: Application co-design

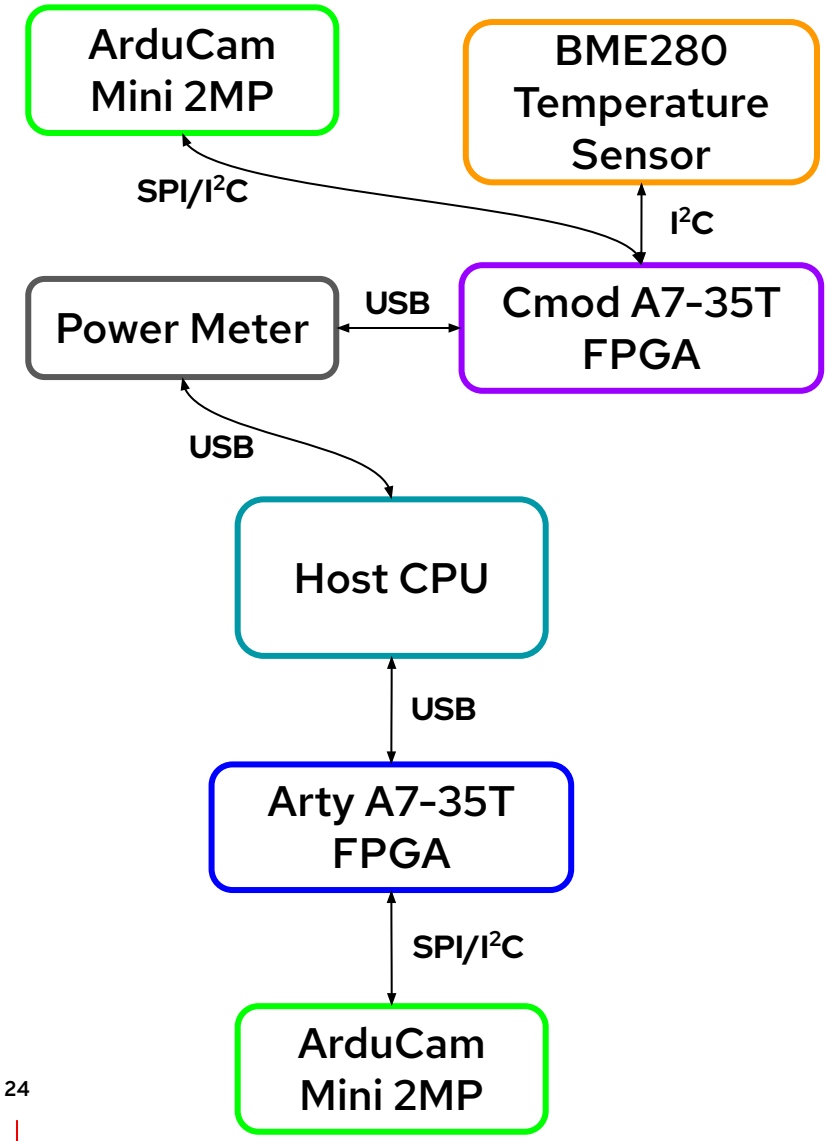


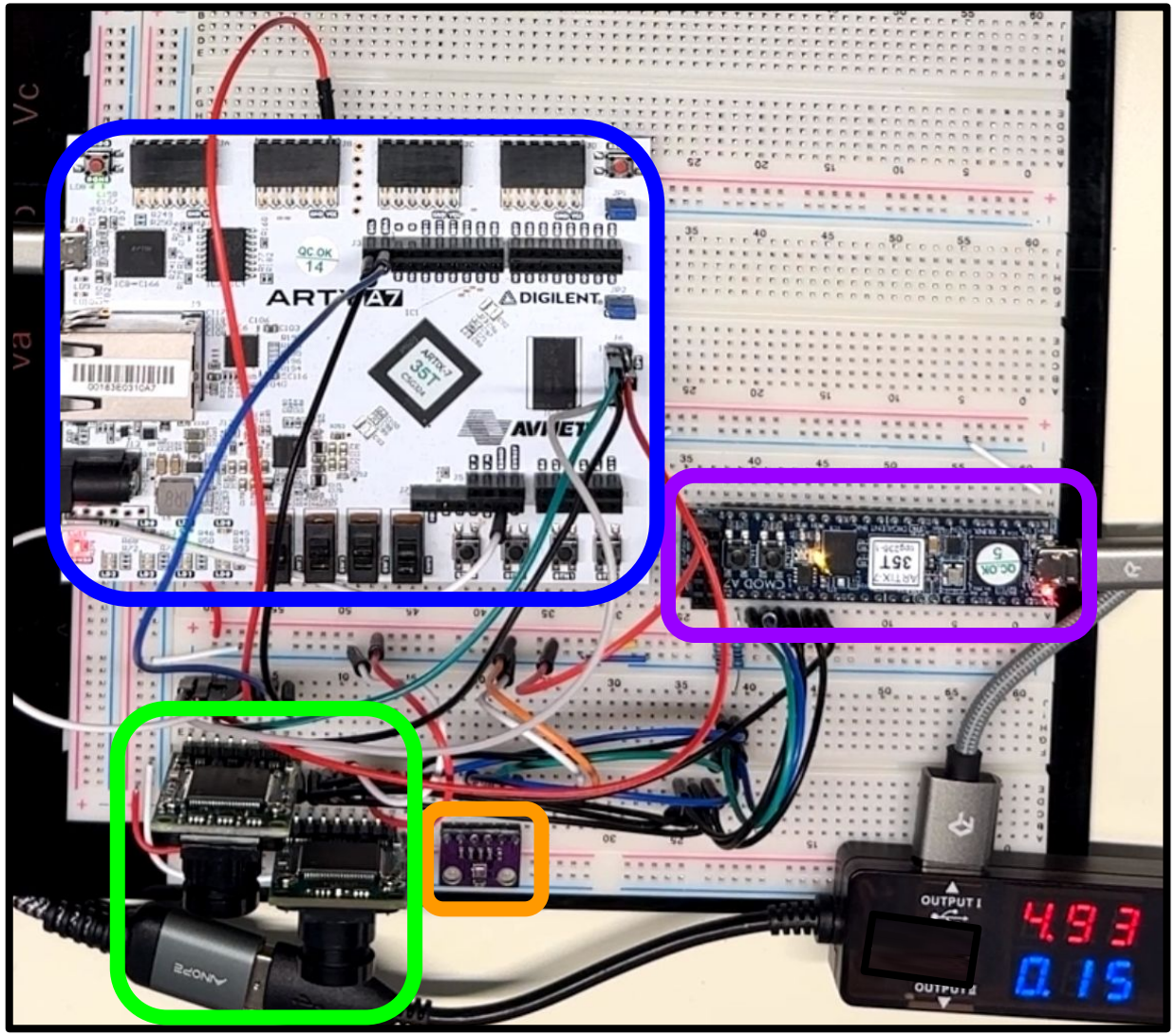
Part B: Secure wireless management



Part A: Generating and optimizing the hardware design

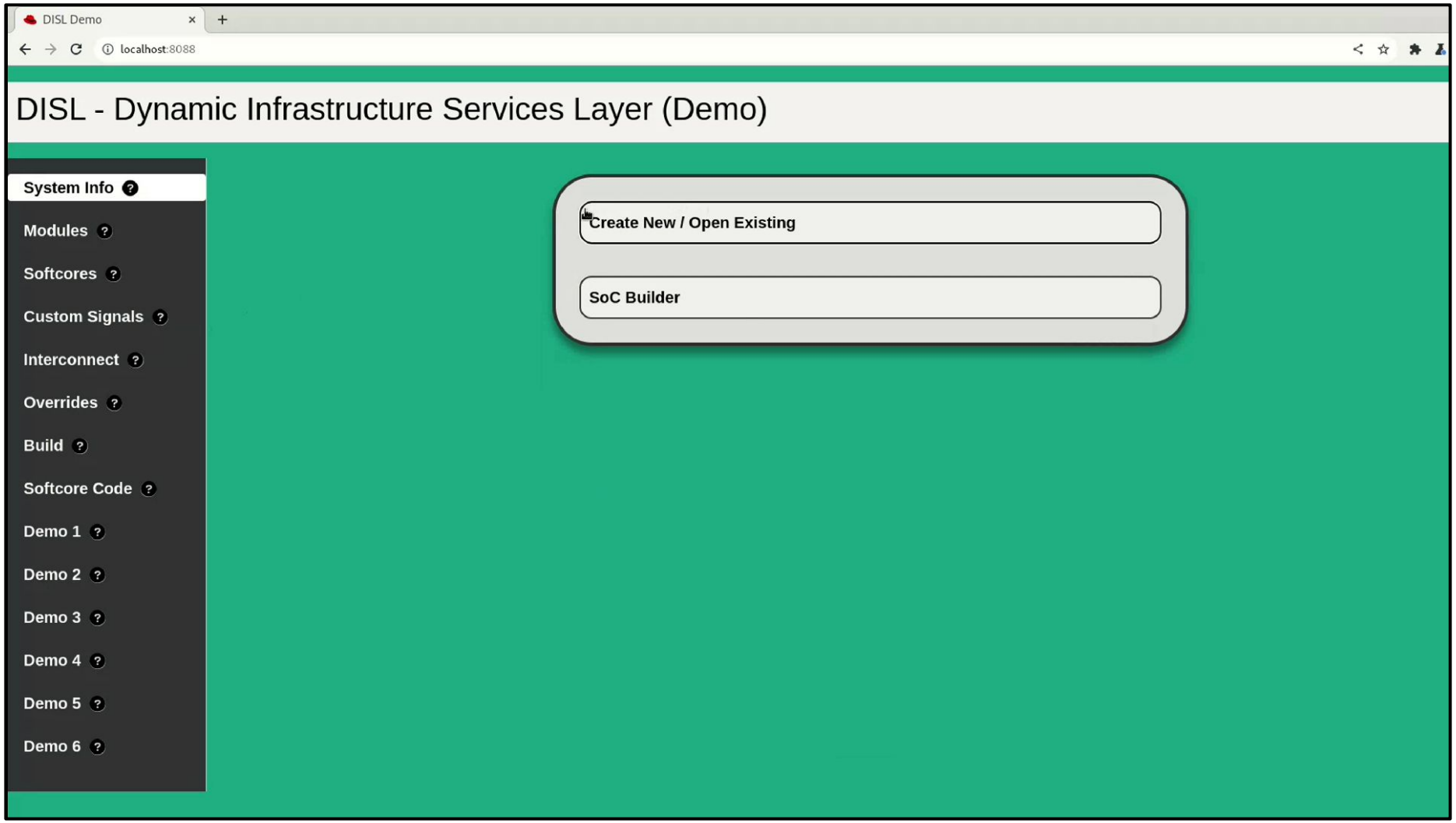
Hardware setup



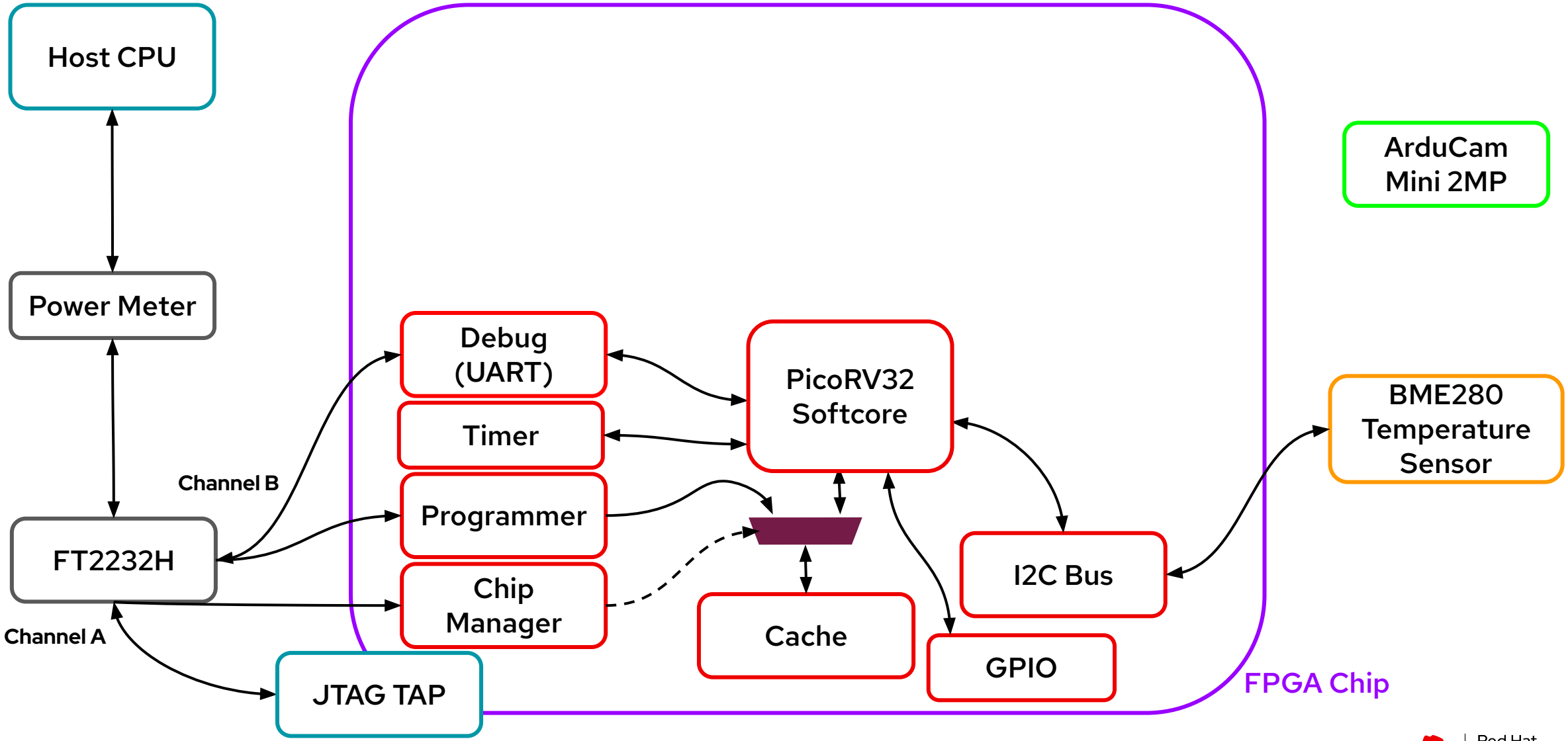


Red Hat Research

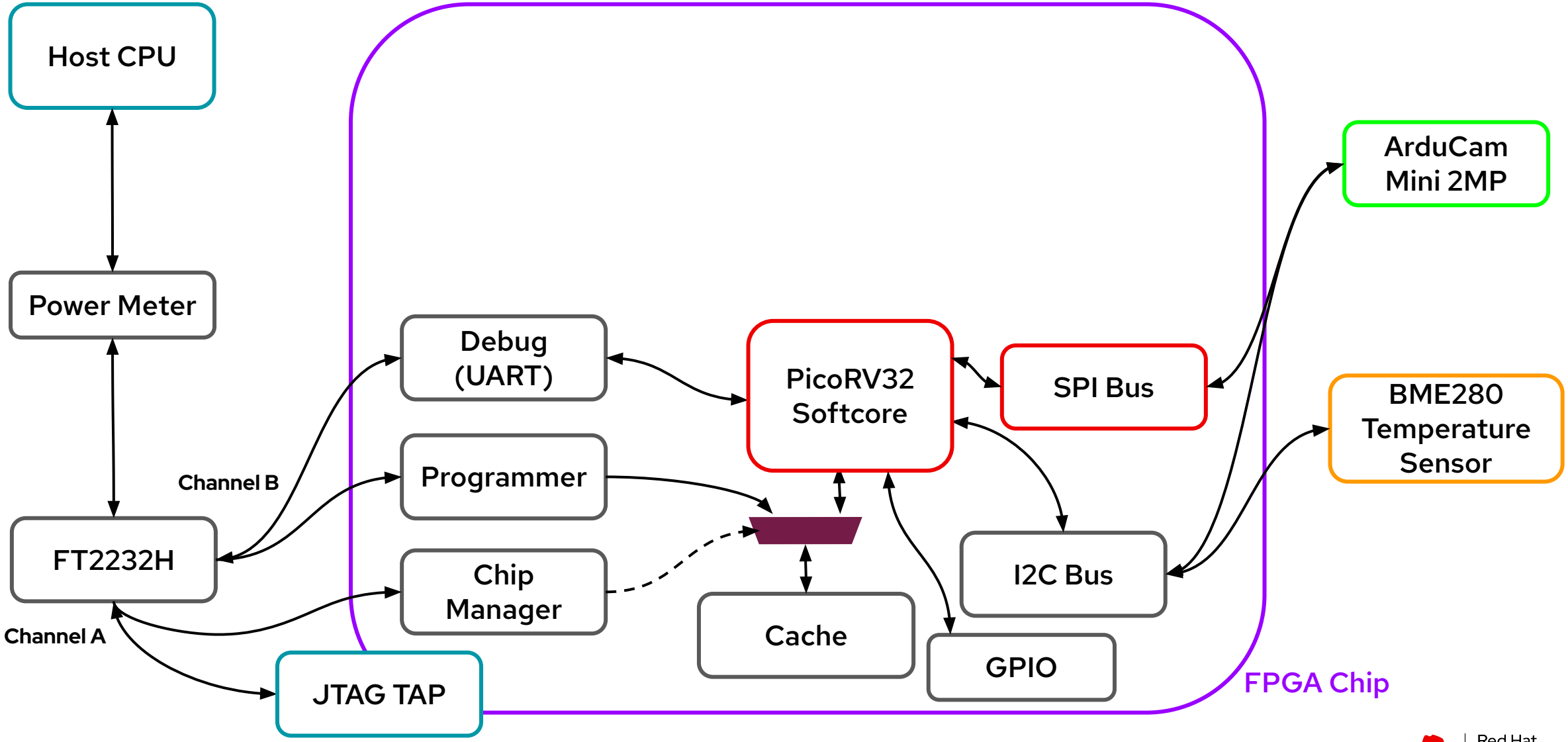
Web application



Generating and testing a simple System on Chip (SoC)



Adding support for the camera module using software libraries



Result: Capturing JPEG

Runtime: 1s

JPEG capture

Result: Capturing RGB565

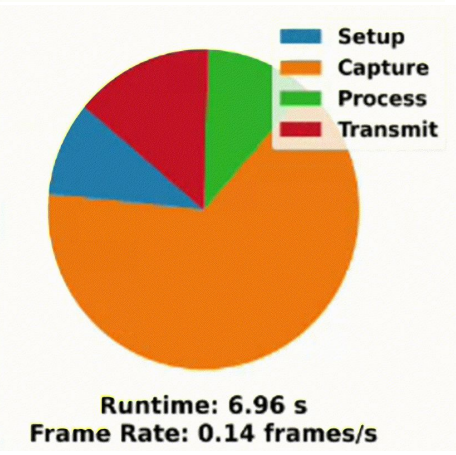
Runtime: 1s

JPEG capture

modify softcore code

Runtime: 7s
7x slower

RGB565 capture



Result: Running edge detection

Runtime: 1s

JPEG capture

modify softcore code

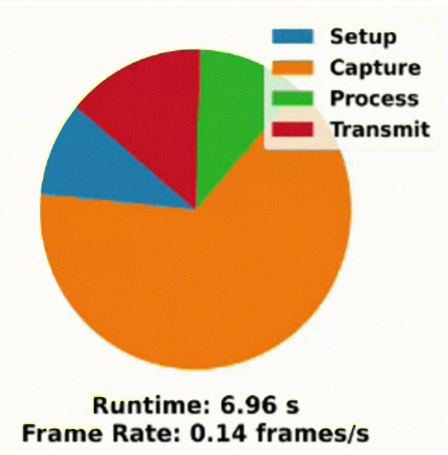
Runtime: 7s
7x slower

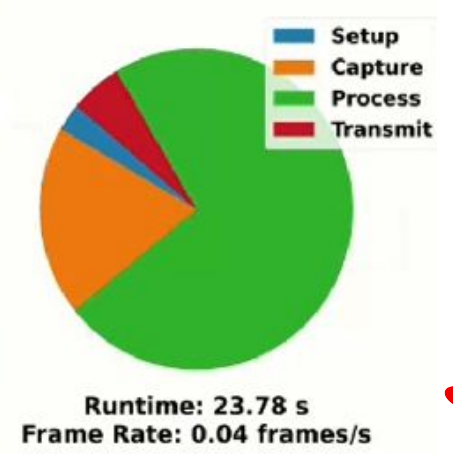
RGB565 capture

modify softcore code

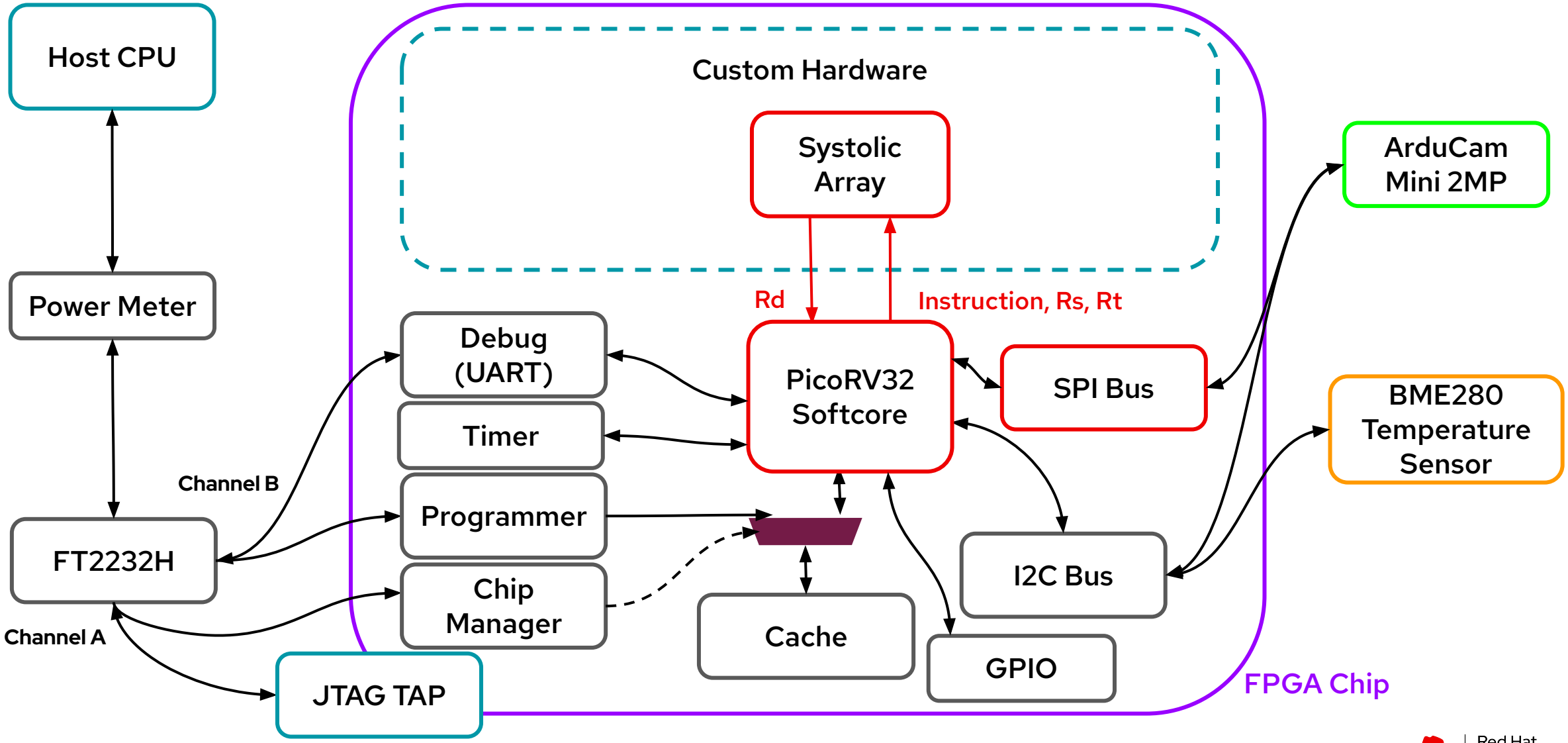
Runtime: 24s
24x slower

Edge detection in software





Reducing the processing overhead using custom hardware offload

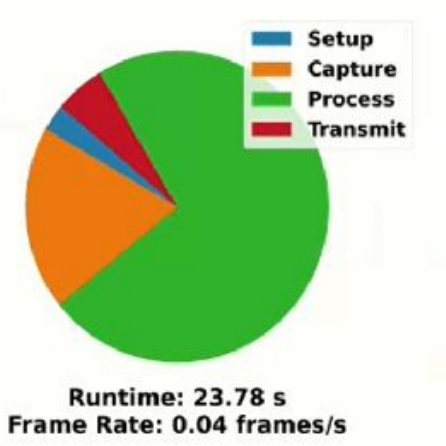


Red Hat Research

Result: Edge detection in SW

Runtime: 24s

Edge detection in software



Result: Edge detection in HW

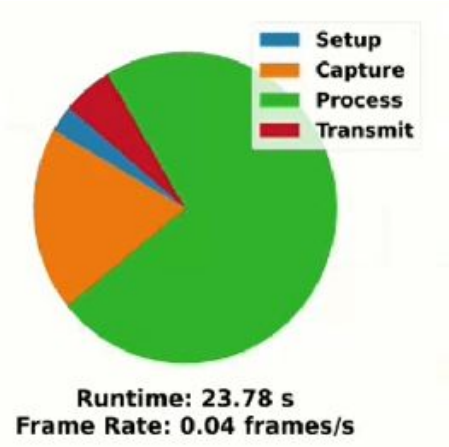
Runtime: 24s

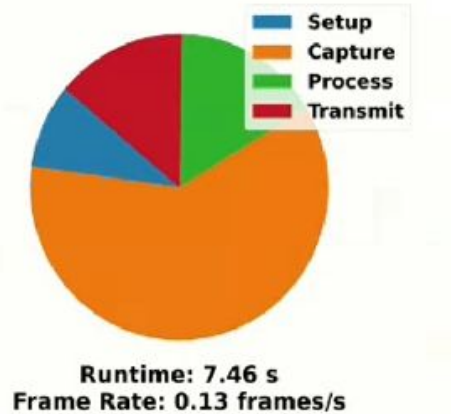
Edge detection in software

offload processing to custom hardware

Runtime: 7.5s
3.2x faster

Edge detection in hardware





Red Hat Research

Result: Binary image transmission

Runtime: 24s

Edge detection in software

offload processing to custom hardware

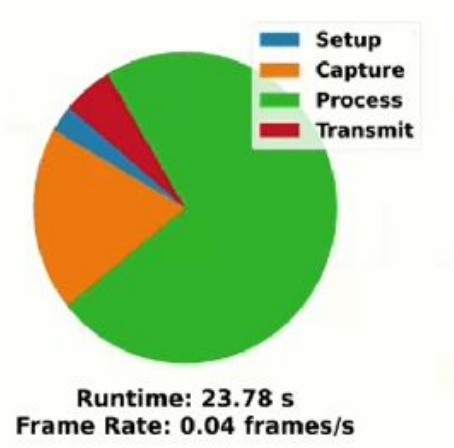
Runtime: 7.5s
3.2x faster

Edge detection in hardware

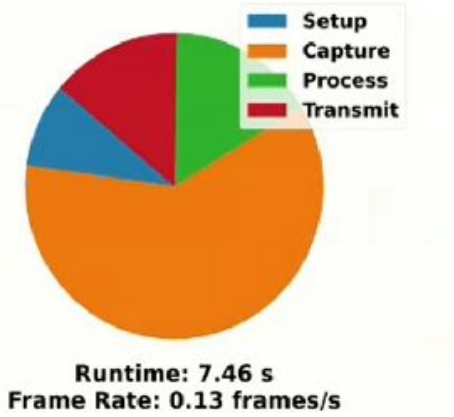
modify softcore code

Runtime: 7s
3.4x faster

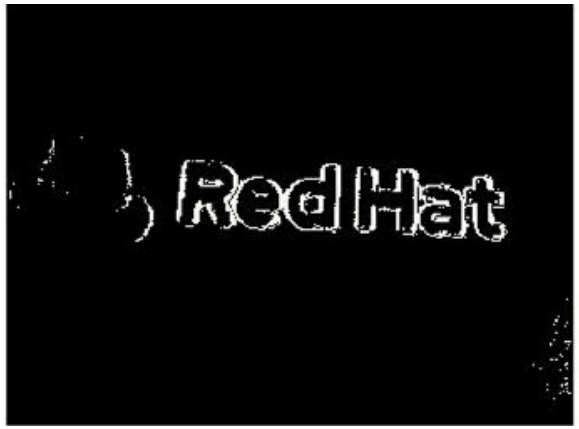
Transmit binary image

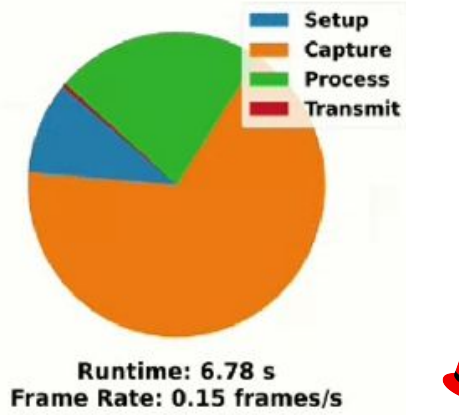


Runtime: 23.78 s
Frame Rate: 0.04 frames/s



Runtime: 7.46 s
Frame Rate: 0.13 frames/s

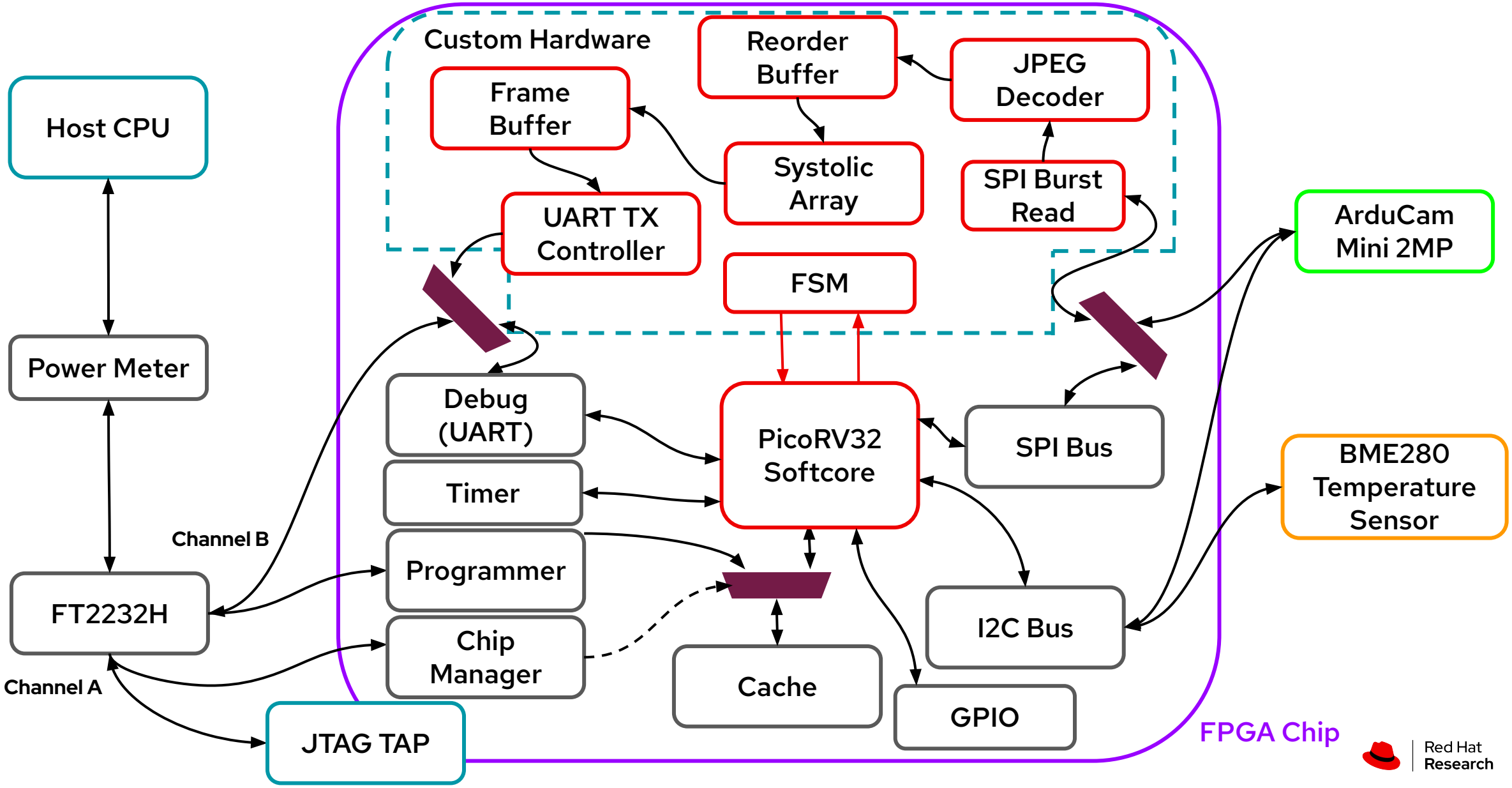




Runtime: 6.78 s
Frame Rate: 0.15 frames/s

Red Hat Research

Reducing the image capture overhead through more complex offloads

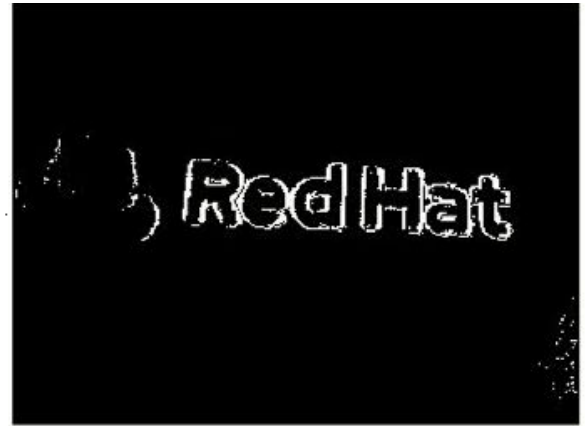


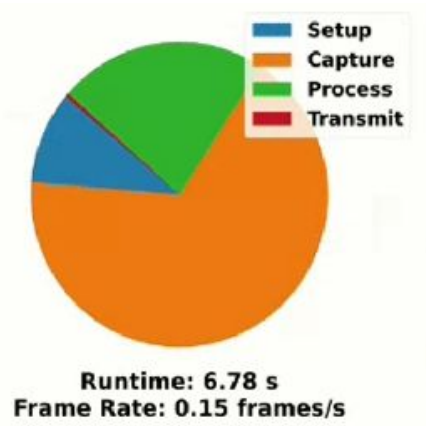
Red Hat Research

Result: Offload processing only

Runtime: 7s

Capture +Transmit in SW





Result: Offloading all three parts

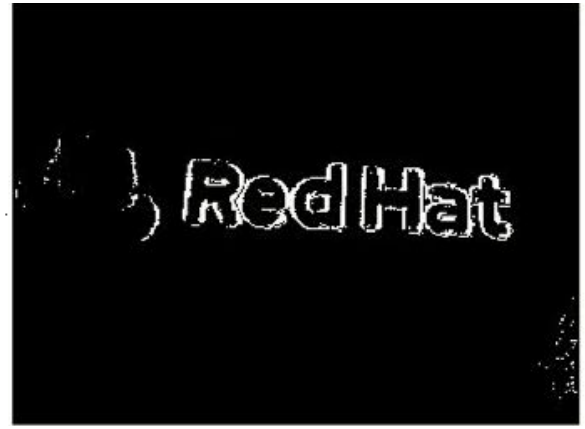
Runtime: 7s

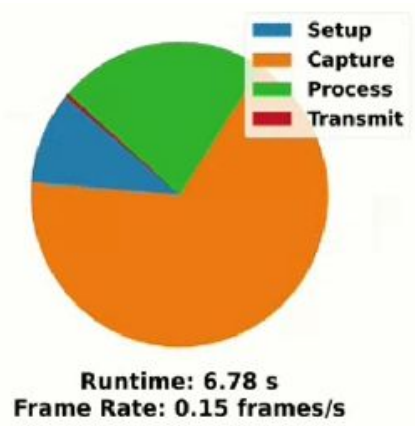
Capture +Transmit in SW

offload capture + transmit to custom hardware as well

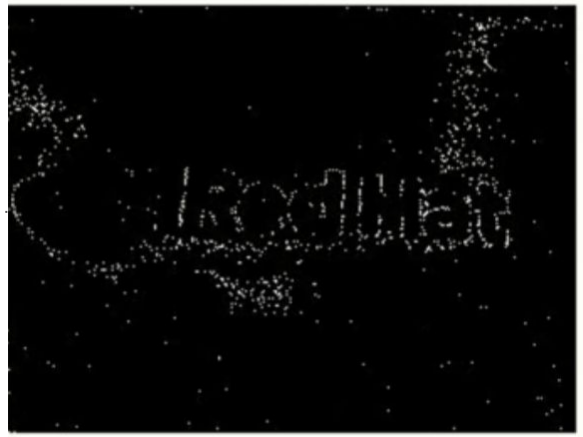
Runtime: 0.3s
23x faster

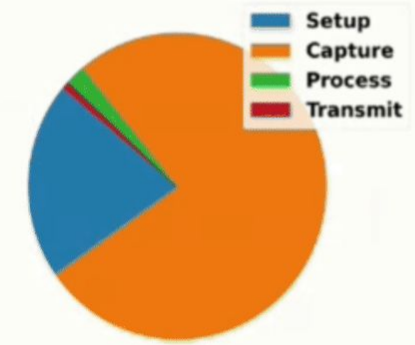
Capture + Process + Transmit in hardware





Runtime: 6.78 s
Frame Rate: 0.15 frames/s



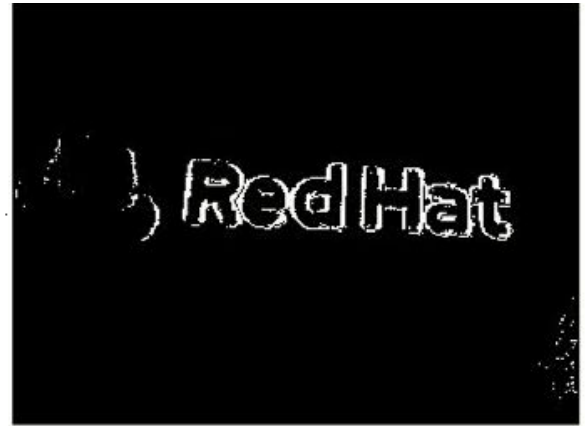


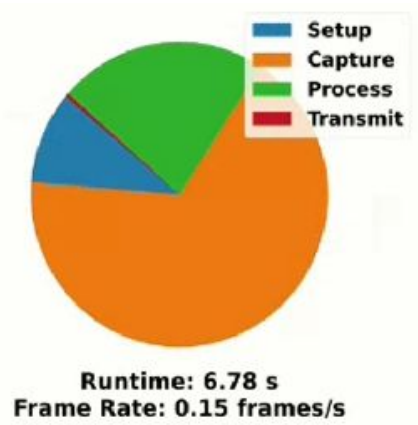
Runtime: 0.30 s
Frame Rate: 3.30 frames/s

Result: Capturing JPEG

Runtime: 7s

Capture +Transmit in SW



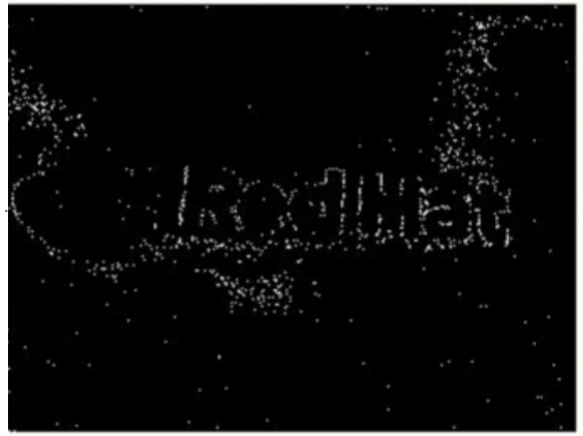


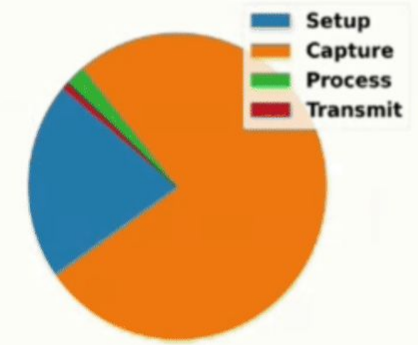
Runtime: 6.78 s
Frame Rate: 0.15 frames/s

offload capture + transmit to custom hardware as well

Runtime: 0.3s
23x faster

Capture + Process + Transmit in hardware



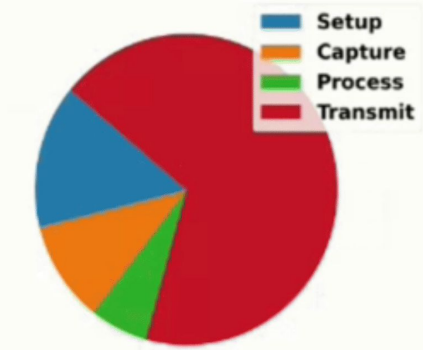


Runtime: 0.30 s
Frame Rate: 3.30 frames/s

Add JPEG decoding to custom hardware

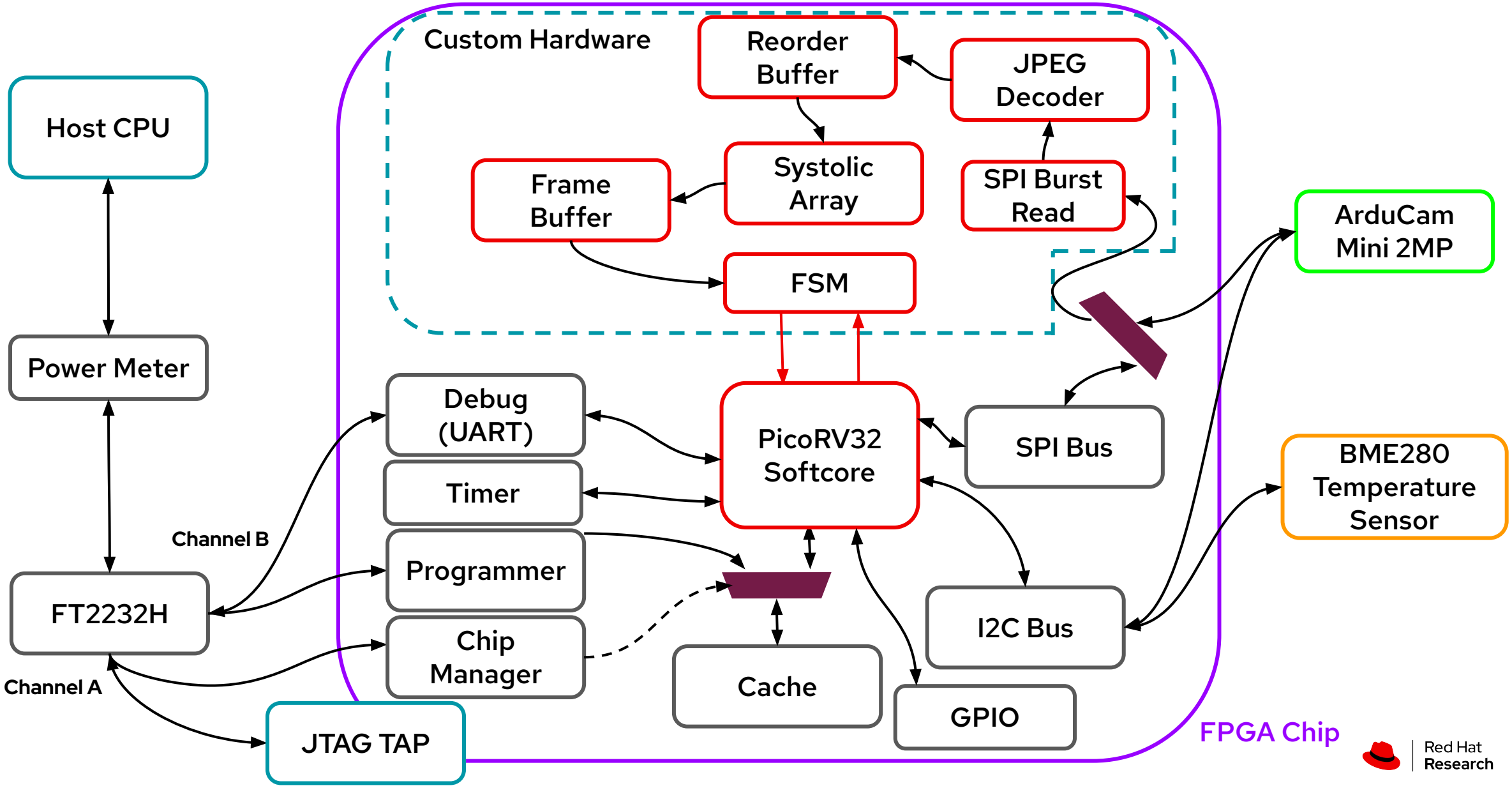
Runtime: 0.13s
54x faster

Capture JPEG instead of RGB565



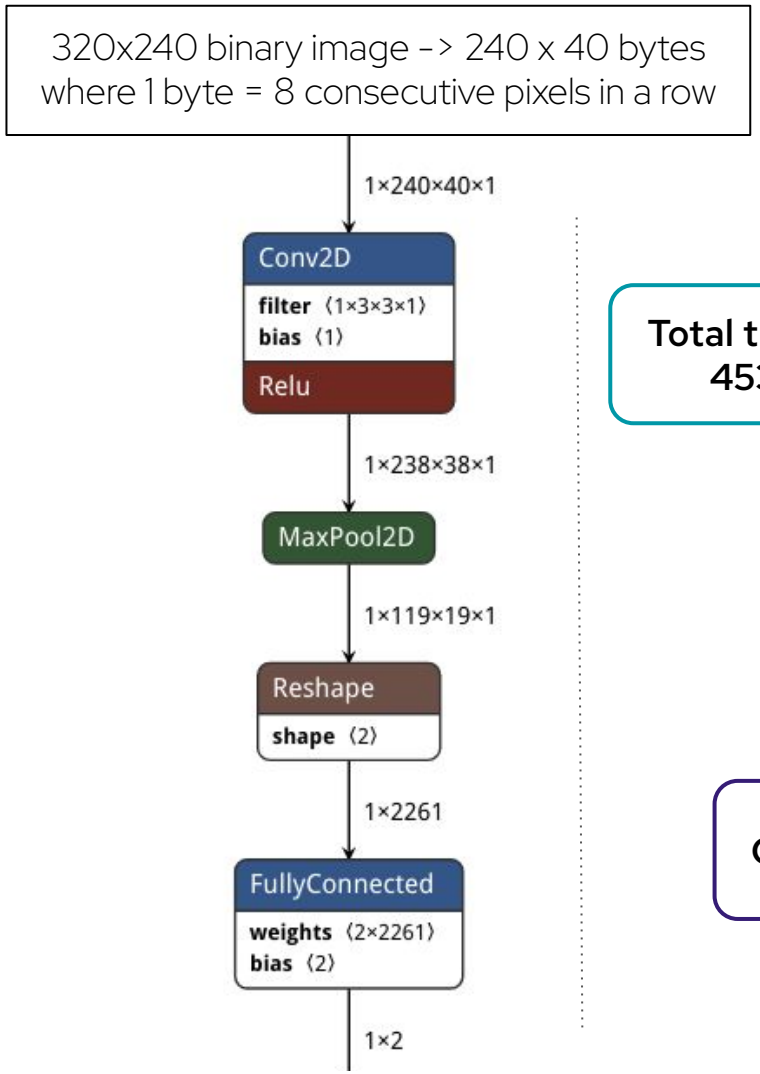
Runtime: 0.13 s
Frame Rate: 7.75 frames/s

Adding person detection using a Convolutional Neural Network



Red Hat Research

Training and deploying the Convolutional Neural Network



Training (Tensorflow)

Total trainable params
4534 (17.71 KB)

Train

int8
quantization

Model size
7.59 KB

Camera

Edge
detection

Frame
buffer

Softcore

Debug

Inference (FPGA)

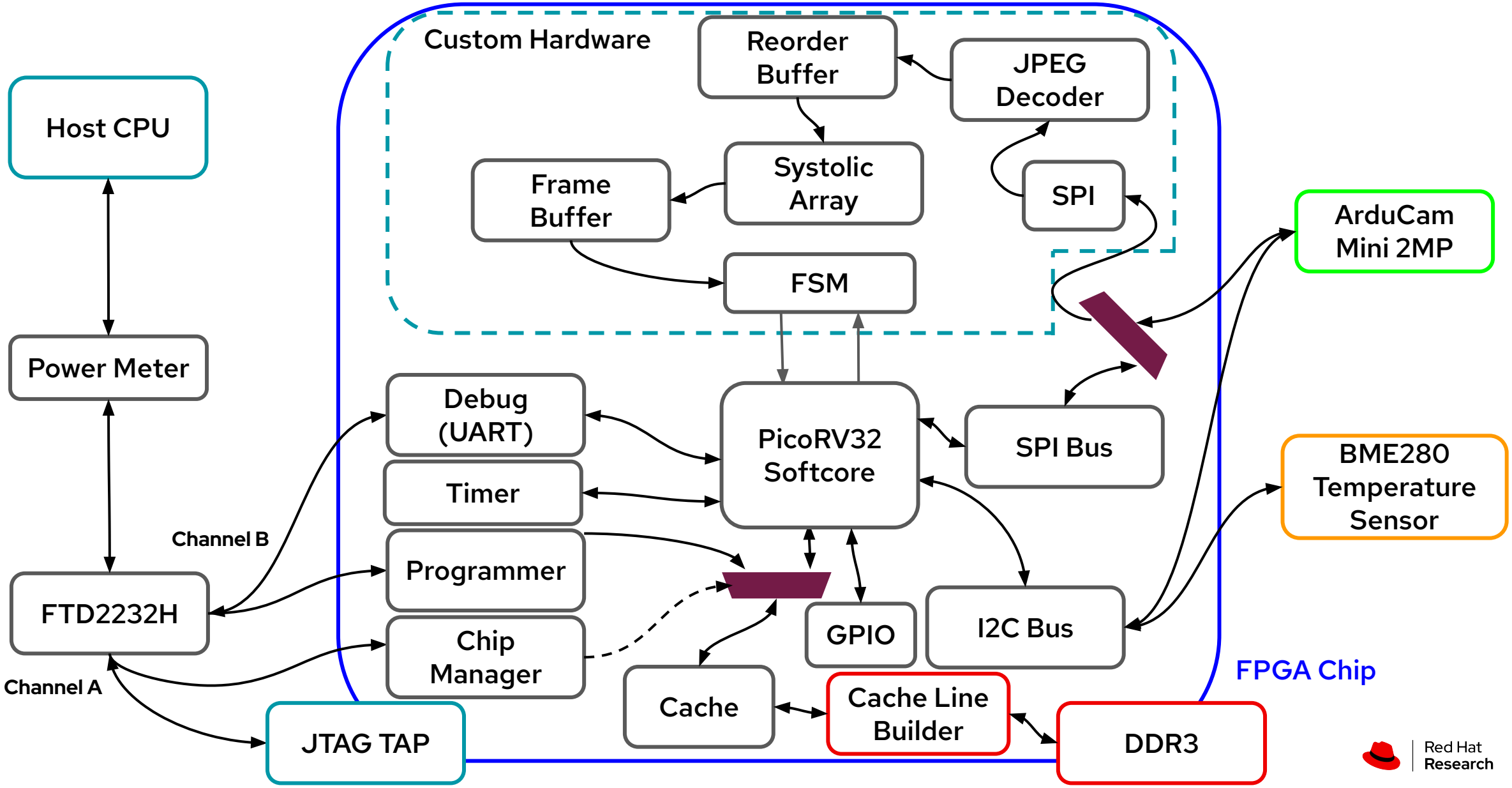
Red Hat
Research

Result: CNN running on the RISC-V softcore

Not a person

Person

Deploying the hardware on a different FPGA board



Red Hat Research

Result: Softcore @ 12 MHz

Person detection on Cmod A735T

Result: Softcore @ 83 MHz*

Person detection on Cmod A735T

optional modification to the SoC

Person detection on Arty A735T

*Can also do this on the Cmod board with a PLL

Result: Transmit on detection

Person detection on Cmod A735T

optional modification to the SoC

Person detection on Arty A735T

modify softcore code

Transmit 10 frames on detection without processing during this time

Part B: Remote management of the FPGA

Adding Remote Capabilities

Motivations:

Scaling up devices

Scaling down resources

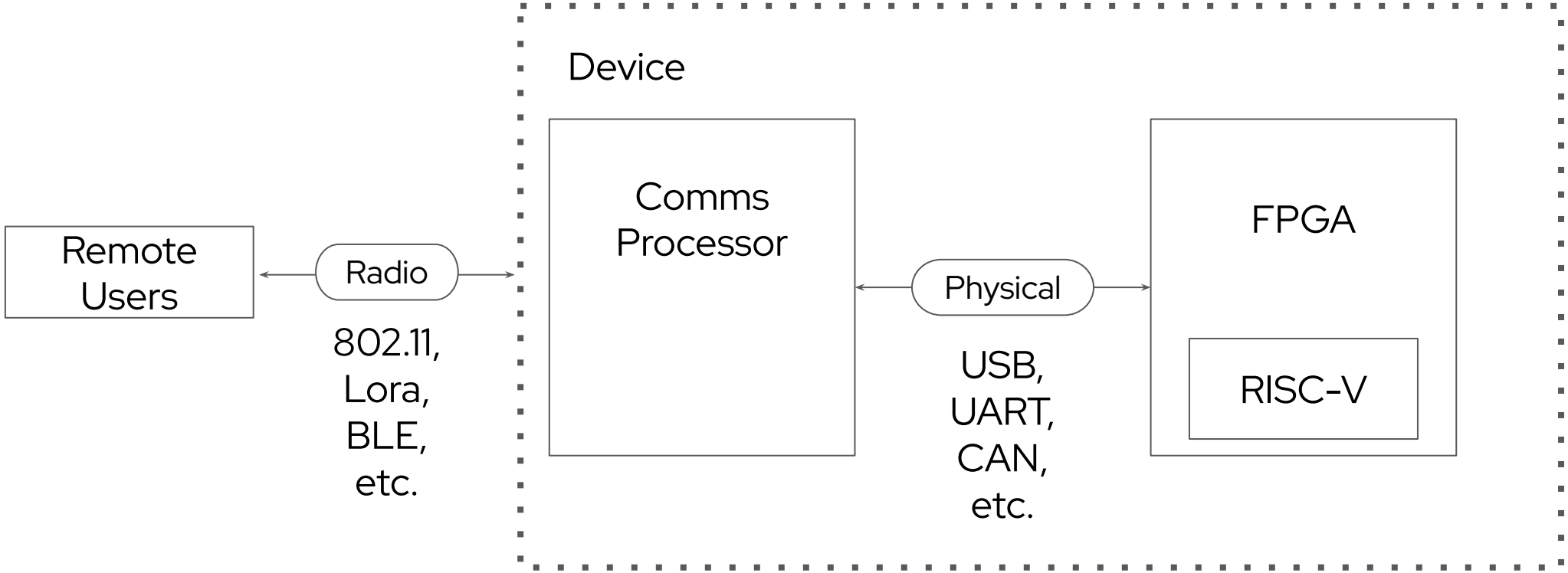
Goals:

Remote FPGA reconfiguration

Remote softcore flashing

Data communication

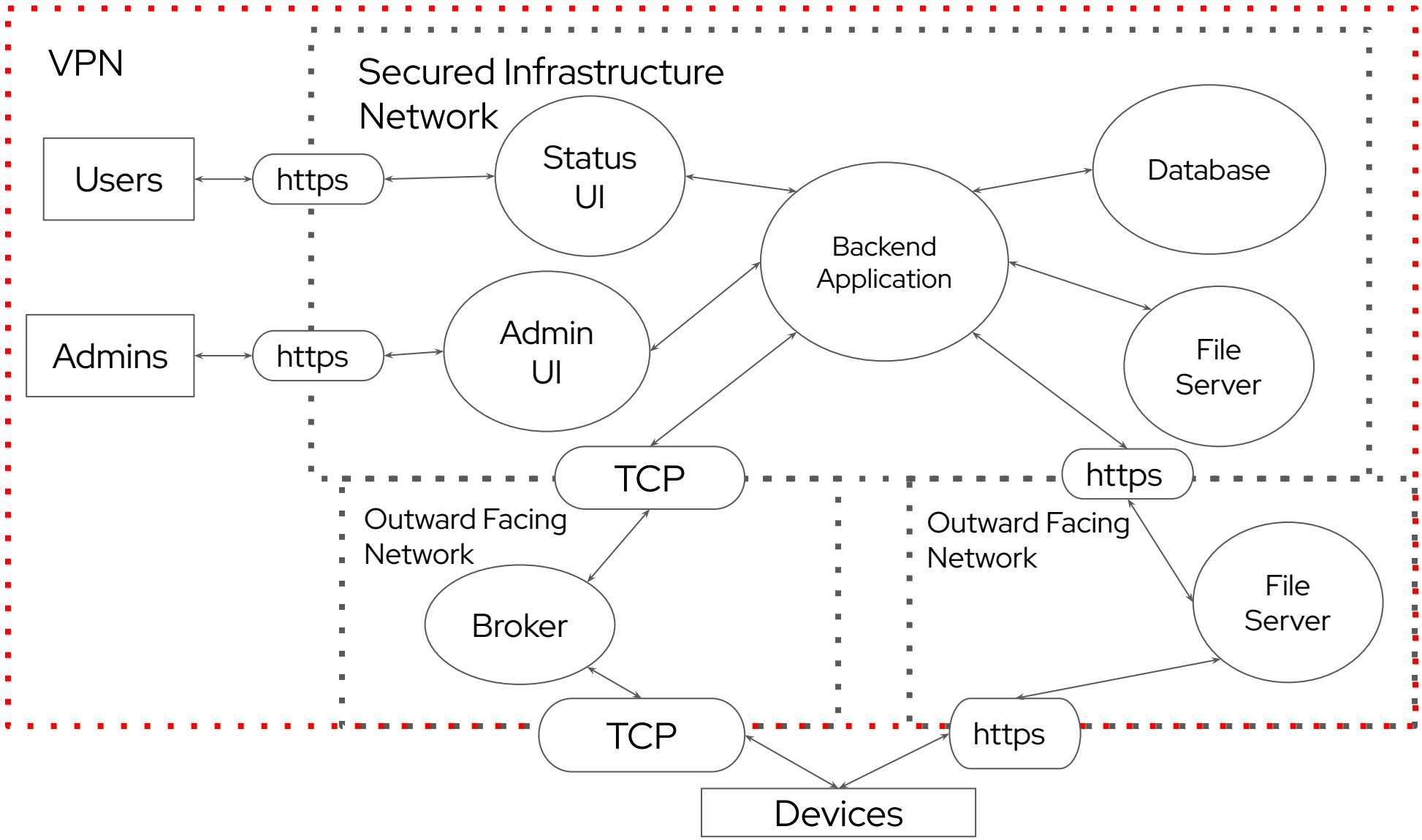
Device Architecture with Comms Processor



Comms currently ESP32-S3 MCU

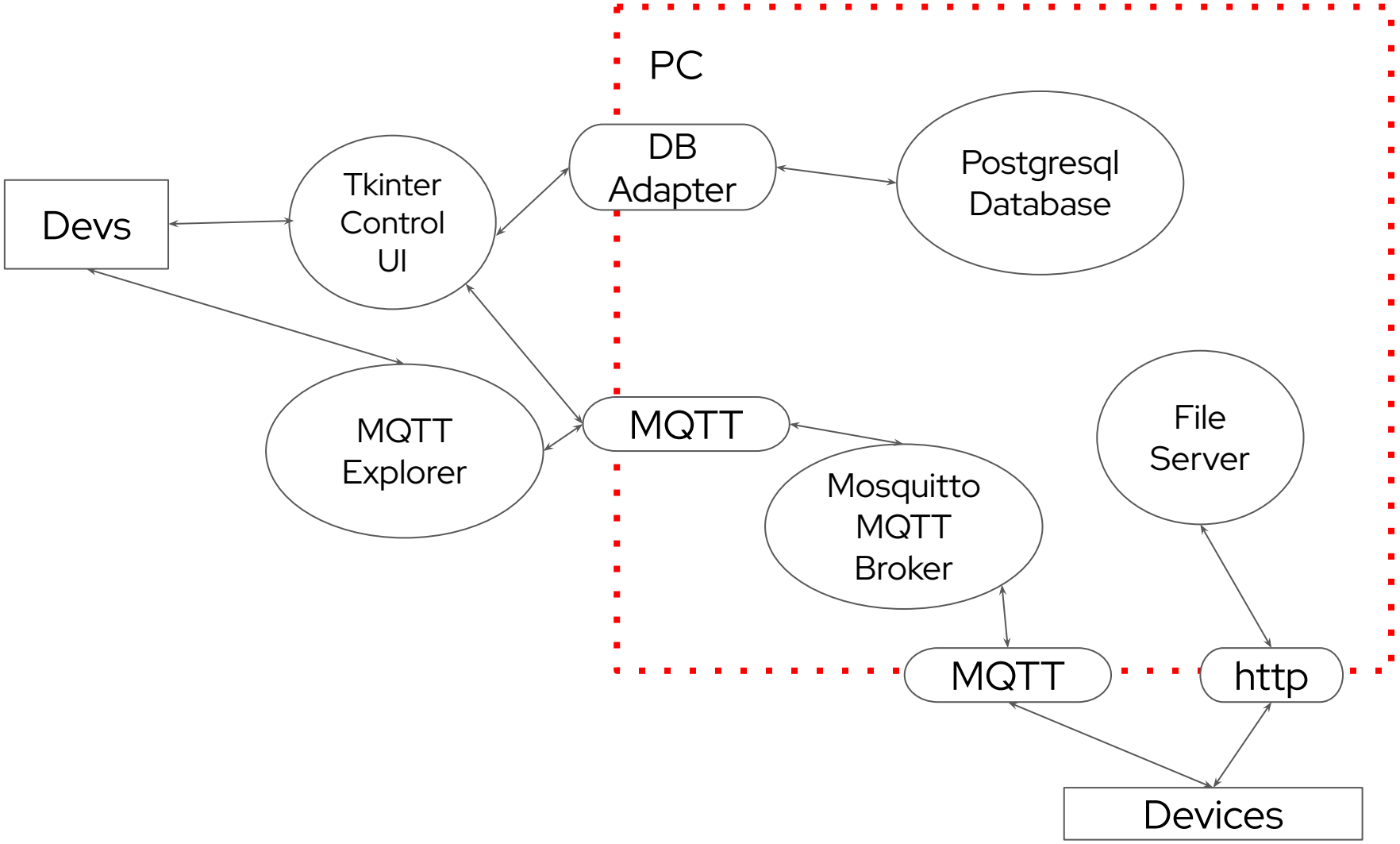
Comms intended to be swappable (e.g. Pico W)

Envisaged Enterprise Architecture



Red Hat Research

Demo/PoC Architecture



Comms Processor Provisions

MQTT topics

/<DEVICE_NAME>/heartbeat - uptime message

/<DEVICE_NAME>/in-command - commands to device

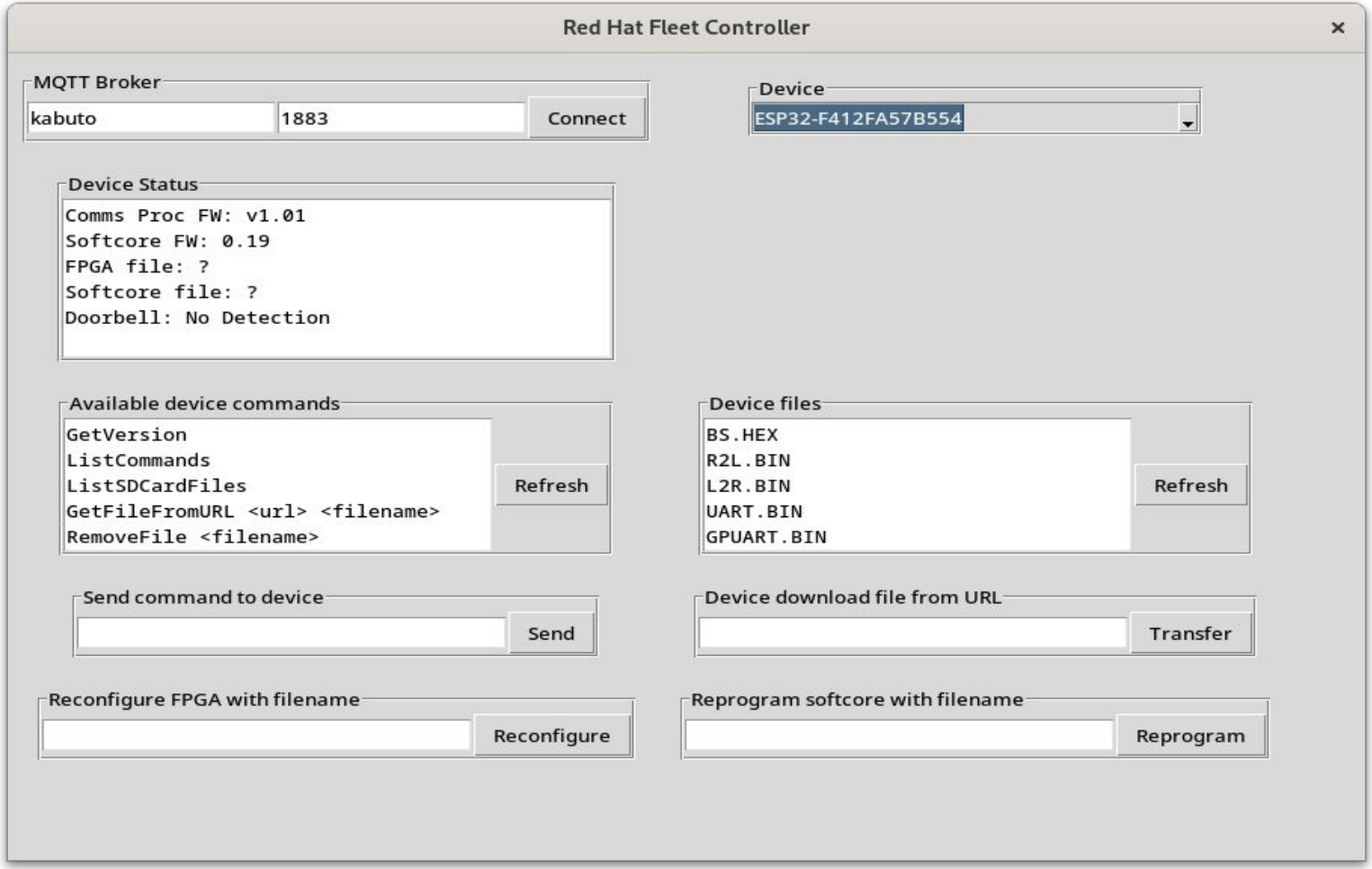
/<DEVICE_NAME>/out-command - responses from device

FPGA data communication via UART

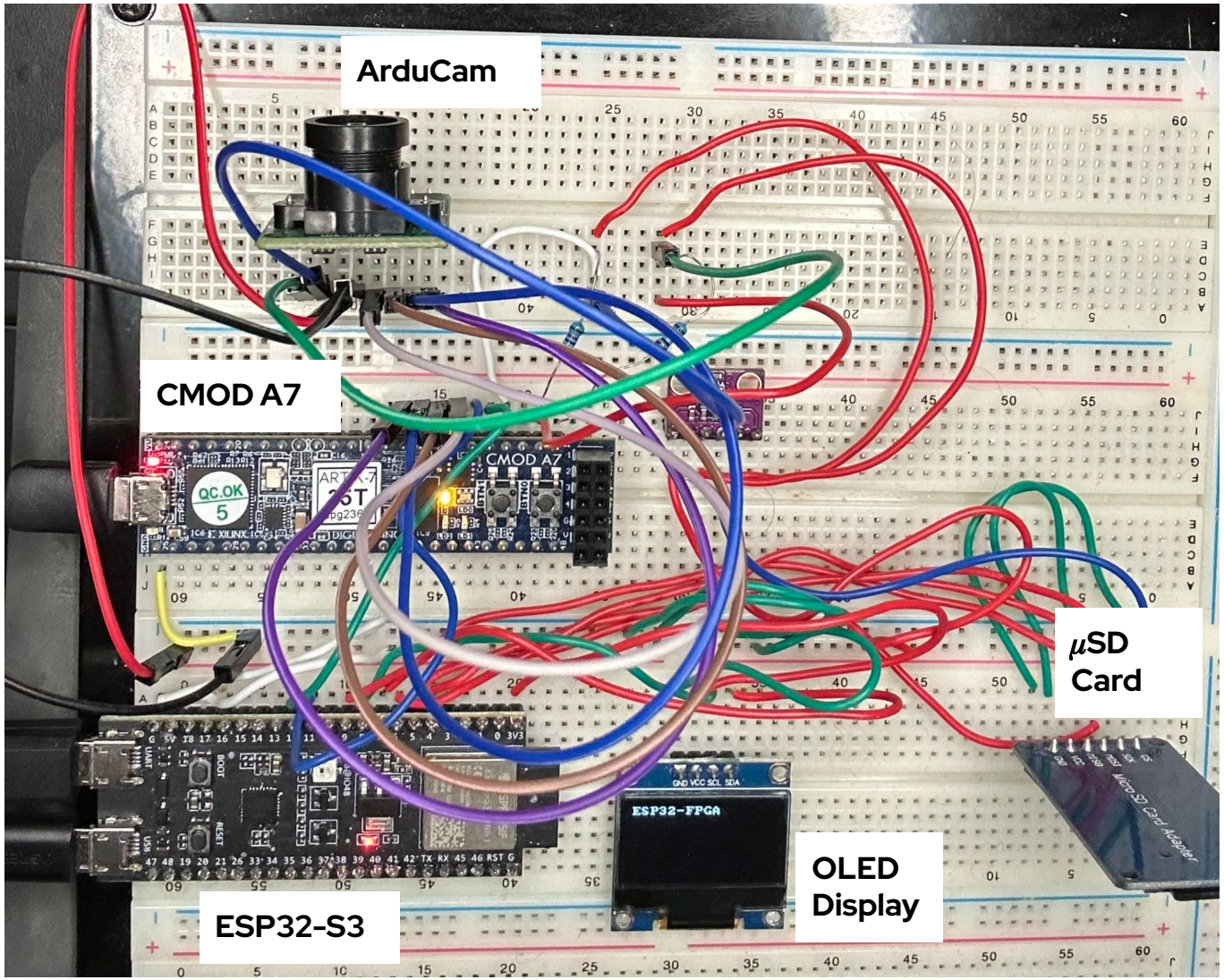
Given data of format {“topic”:”<TOPIC_NAME>”,”message”:”<MESSAGE>”}\r

Comms Processor sends <MESSAGE> to /<DEVICE_NAME>/<TOPIC_NAME>

Controller GUI



Demo Board



Remote FPGA Programming Demo

Given single and double LED blink bitstreams

Use controller UI to reconfigure the CMOD A7 FPGA with each bitstream

We expect to see single or double LED blinks on the CMOD A7 board

Red Hat Fleet Controller

MQTT Broker

localhost 1883 Connect

Device

None Selected

Device Status

Available device commands

Refresh

Device files

Refresh

Send command to device

Send

Device download file from URL

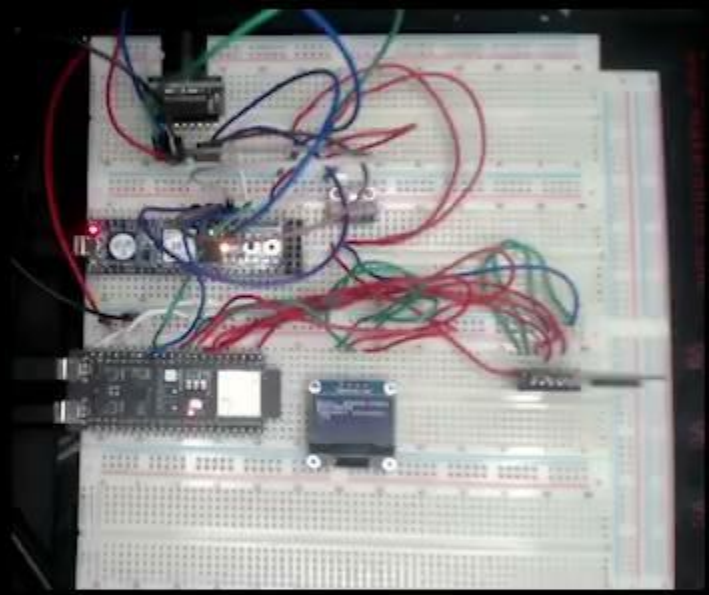
Transfer

Reconfigure FPGA with filename

Reconfigure

Reprogram softcore with filename

Reprogram



Remote Pico RV32 Programming with Temperature Sensor Demo

FPGA bitstream supporting BME280 on I^2C per Part A

Remote Pico RV32 programming to report sensor readings and FW version

We expect to see temperature readings and version 0.13 or 0.14

Red Hat Fleet Controller

MQTT Broker

localhost 1883 Connect

Device

None Selected

Device Status

Available device commands

Refresh

Device files

Refresh

Send command to device

Send

Device download file from URL

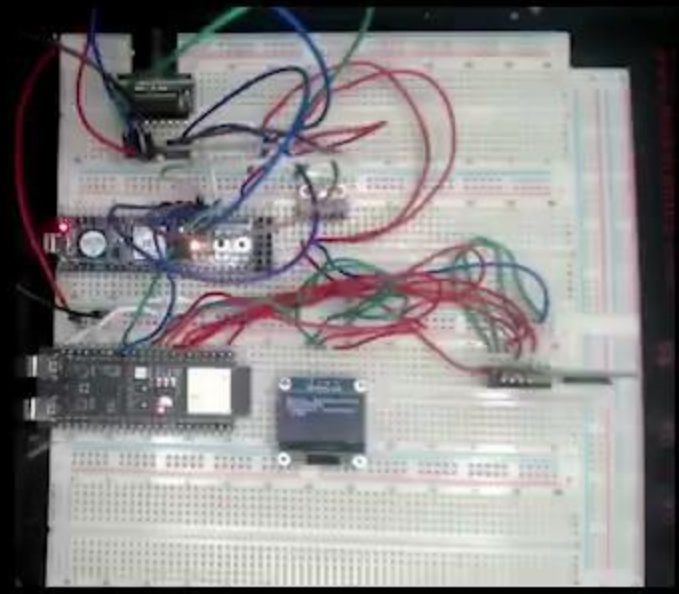
Transfer

Reconfigure FPGA with filename

Reconfigure

Reprogram softcore with filename

Reprogram



Remote Person Detection Demo

FPGA bitstream supporting Arducam on SPI/I^2C per Part A

We expect to see detection status from the device: Detected or Not Detected

Red Hat Fleet Controller

MQTT Broker

localhost | 1883 | Connect

Device

None Selected

Device Status

Available device commands

Refresh

Device files

Refresh

Send command to device

Send

Device download file from URL

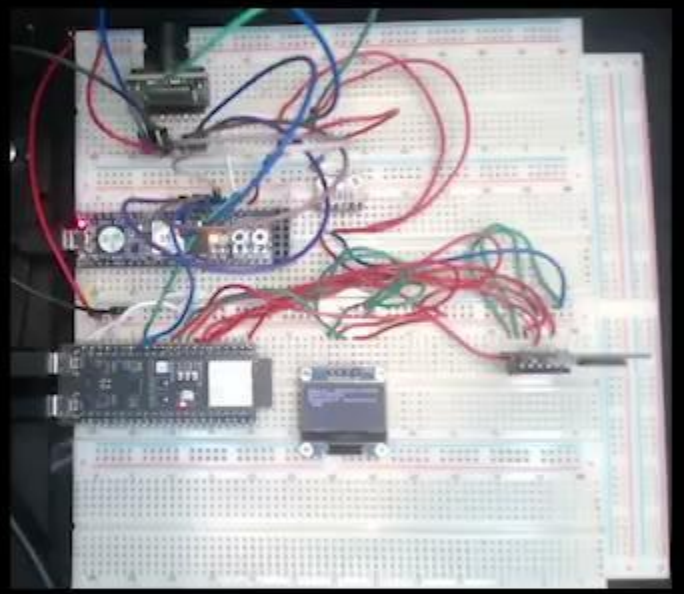
Transfer

Reconfigure FPGA with filename

Reconfigure

Reprogram softcore with filename

Reprogram



Innovations

DISL makes it possible to codesign hardware and software for an FPGA

DISL lets you focus on your application development without using low level hardware description language

This demo showed an example of how to manage devices in the field with wireless access

This demo showed example UIs for simplified system generation, device access and management

The demonstration code is available. Contact us.

For more information, contact us at

Ahmed Sanaullah (sanaullah@redhat.com) and
Jason Schlessman (jschless@redhat.com)

Keep up with Red Hat Research!

Come to an upcoming RHR event

Visit research.redhat.com/events

Sign up for Red Hat Research Quarterly

Scan the QR code or visit: research.redhat.com/quarterly/

Join a Research Interest Group

Learn more at research.redhat.com/rigs

Red Hat Research