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
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

Unoptimized workload → Software optimizations → Co-design

Application

General Purpose Software Stack

General Purpose Hardware Stack

Optimized Application

Optimized Software Stack

General Purpose Hardware Stack

Optimized Application

Optimized Software Stack

Optimized Hardware Stack
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
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.

CoDes Lab @ Red Hat - BU Collaboratory

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

[Diagram showing programmability and performance comparison between microcontrollers, FPGAs, and application specific integrated circuits (ASICs).]
Why are FPGAs so difficult to use?

Inefficient development flow between software developer and hardware developer
Dynamic Infrastructure Services Layer (DISL)
import DISL
sys = DISL.new(name='test')
sys.add_softcore(type='riscv')
sys.add_memory(size='32kB')
sys.add_debug(baud=115200)
sys.add_timer(resolution='us')
sys.build(board='xyz')
import DISL
sys = DISL.new(name='test')
sys.add软核(type='riscv')
sys.add_memory(size='32kB')
sys.add_debug(baud=115200)
sys.add_timer(resolution='us')
sys.build(board='xyz')
HW interface: IP Configuration and DISL Component Library

- Component library
- Board-specific IP block
- Board-specific HDL
- Generic HDL + interfaces for board/system HDL
- Board-specific parameters and default values
- Generic parameters and default values
- Template for interfacing generic HDL
- DISL Component Library
- IP Configuration
- Parameter mapping function
- Abstracted parameters and default values
IP block for a different board

Board-specific HDL

Template for interfacing generic HDL

DISL Component Library

IP Configuration

Parameter mapping function

Abstracted parameters and default values

Board-specific parameters and default values
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
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)

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

<table>
<thead>
<tr>
<th>Major proprietary tooling and IP blocks</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Vivado</td>
<td>For synthesis + Place &amp; Route - open source tooling currently does not support certain PHYs</td>
</tr>
<tr>
<td>FPGA PHYs</td>
<td>Vivado MIG PHY for DDR3, BSCANE2 PHY for JTAG support, PLLs for clock generation</td>
</tr>
</tbody>
</table>
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
What you’ll see in this demo: multiple hardware types

- Microcontroller (ESP32-S3)
- FPGA (Cmod A7, Arty A7)
- ASIC (ArduCam)
Demo overview

Part A: Application co-design

Part B: Secure wireless management
Part A: Generating and optimizing the hardware design
Hardware setup

- ArduCam Mini 2MP
- BME280 Temperature Sensor
- Power Meter
- Cmod A7-35T FPGA
- Host CPU
- Arty A7-35T FPGA
- SPI/I²C
- USB
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

- JPEG capture (1s)
  - modify softcore code
- RGB565 capture (7s, 7x slower)

Runtime:

- 1s for JPEG capture
- 7s for RGB565 capture
- 0.99 s for capture + process + transmit (RGB565)
- 6.96 s for capture + process + transmit (RGB565)
Result: Running edge detection

- **JPEG capture**
  - Runtime: 1s
  - Modify softcore code

- **RGB565 capture**
  - Runtime: 7s
  - 7x slower
  - Modify softcore code

- **Edge detection in software**
  - Runtime: 24s
  - 24x slower
Reducing the processing overhead using custom hardware offload
Result: Edge detection in SW

Runtime: 24s

Edge detection in software
Result: Edge detection in HW

Edge detection in software

Runtime: 24s

Offload processing to custom hardware

Edge detection in hardware

Runtime: 7.5s

3.2x faster
Result: Binary image transmission

Runtime: 24s
Edge detection in software

Runtime: 75s
3.2x faster
Offload processing to custom hardware

Runtime: 7s
3.4x faster
Modify softcore code

Transmit binary image

Runtime: 6.78 s
Frame Rate: 0.15 frames/s

Runtime: 7.46 s
Frame Rate: 0.13 frames/s

Runtime: 23.78 s
Frame Rate: 0.04 frames/s
Reducing the image capture overhead through more complex offloads
Result: Offload processing only

Capture + Transmit in SW

Runtime: 7s

Runtime: 6.78 s
Frame Rate: 0.15 frames/s
Result: Offloading all three parts

Capture + Transmit in SW

Offload capture + transmit to custom hardware as well

Capture + Process + Transmit in hardware

Runtime: 7s

23x faster

Runtime: 0.3s

Result: Offloading all three parts
Result: Capturing JPEG

- **Capture + Transmit in SW**
  - Runtime: 7s
  - offload capture + transmit to custom hardware as well

- **Capture + Process + Transmit in hardware**
  - Runtime: 0.3s (23x faster)
  - Add JPEG decoding to custom hardware

- **Capture JPEG instead of RGB565**
  - Runtime: 0.13s (54x faster)

**Runtime:**
- 0.3s
- 0.13s
- 7s
- 0.30s
- 0.13s
Adding person detection using a Convolutional Neural Network
Training and deploying the Convolutional Neural Network

320x240 binary image -> 240 x 40 bytes where 1 byte = 8 consecutive pixels in a row

Training (Tensorflow)

Total trainable params 4534 (17.71 KB) -> Train -> int8 quantization -> Model size 7.59 KB

Inference (FPGA)

Camera -> Edge detection -> Frame buffer -> Softcore -> Debug
Result: CNN running on the RISC-V softcore
Deploying the hardware on a different FPGA board

Custom Hardware:
- Host CPU
- Power Meter
- FTD2232H
- ArduCam Mini 2MP
- BME280 Temperature Sensor
- SPI Bus
- SPI
- JPEG Decoder
- Reorder Buffer
- Systolic Array
- FSM
- PicoRV32 Softcore
- Cache
- Cache Line Builder
- DDR3
- Cache Line Builder
- GPIO
- I2C Bus
- Programmer
- Timer
- Chip Manager
- Debug (UART)
- JTAG TAP

Channel A
Channel B
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

VPN

Users

Admins

Status UI

Backend Application

Database

File Server

Secured Infrastructure Network

Admin UI

TCP

Outward Facing Network

Outward Facing Network

Broker

TCP

Devices

https

https

https

https

Users

Admins

Secured Infrastructure Network

Admin UI

TCP

Outward Facing Network

Outward Facing Network
Demo/PoC Architecture

- Devs
  - Tkinter Control UI
  - MQTT Explorer
  - DB Adapter
- PC
  - Postgresql Database
  - File Server
  - Mosquitto MQTT Broker
- Devices
- MQTT
- http

Red Hat Research
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>”}
```

Comms Processor sends <MESSAGE> to /DEVICE_NAME>/<TOPIC_NAME>
Controller GUI
Demo Board

- ArduCam
- CMOD A7
- ESP32-S3
- OLED Display
- μSD Card
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
Remote Pico RV32 Programming with Temperature Sensor Demo

FPGA bitstream supporting BME280 on I²C 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
Remote Person Detection Demo

FPGA bitstream supporting Arducam on SPI/I²C per Part A

We expect to see detection status from the device: Detected or Not Detected
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.
Keep up with Red Hat Research!

For more information, contact us at Ahmed Sanaullah (sanaullah@redhat.com) and Jason Schlessman (jschless@redhat.com)

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