



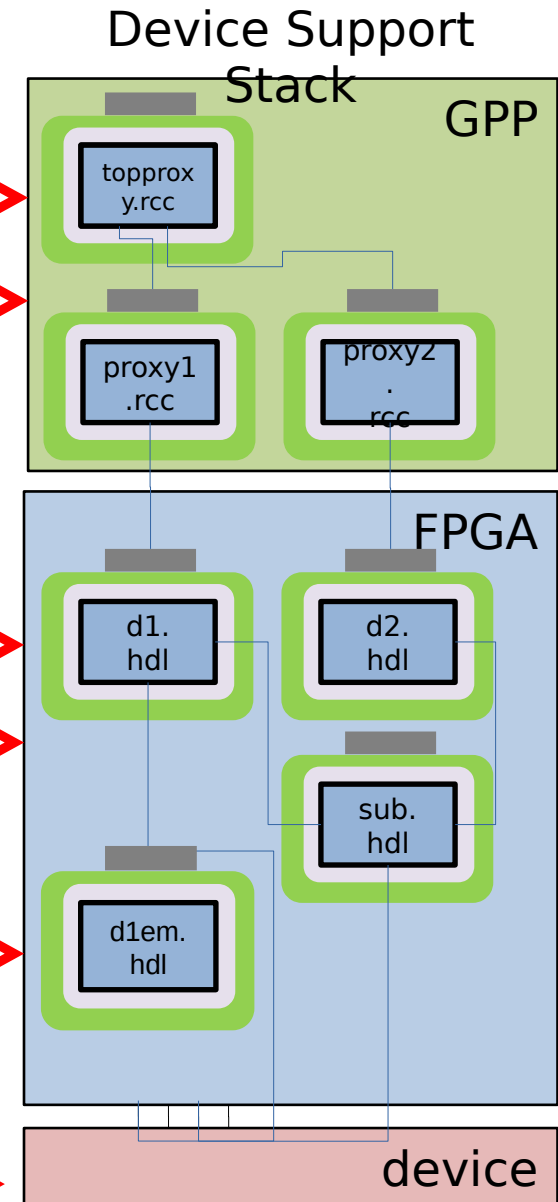
OpenCPI - Open Component Portability Infrastructure

**An Open Source Infrastructure for
Heterogeneous/Embedded Component-
Based Systems and Applications**

Device Support for FPGA Platforms

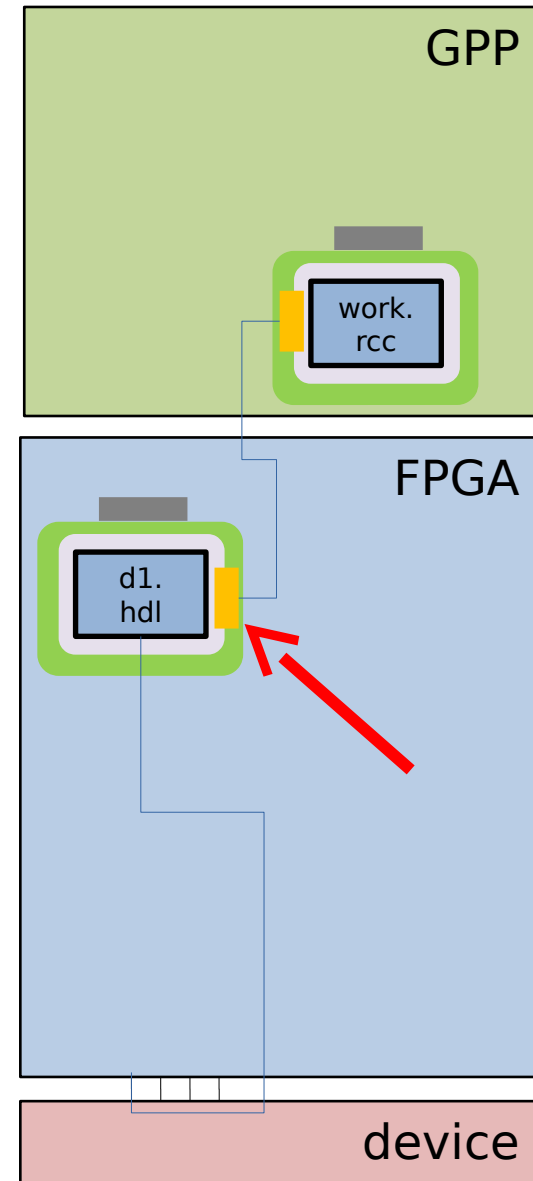
Device Support Concepts

- Top-Level Proxy
 - A software worker that typically has multiple device proxy ~~"slaves"~~
- Device Proxy
 - A software worker (RCC/C++) specifically paired with one or more device workers which provides a high-level control interface
- Device Worker
 - A specialized HDL Worker intended for direct connection to ~~device pins~~
- Subdevice Worker
 - Specialized Device Worker which implements required sharing of low level hardware between Device Workers
- Emulator Worker
 - Emulates a device for test purposes
- Device
 - Hardware elements that are ~~locally attached to the processor of the platform~~



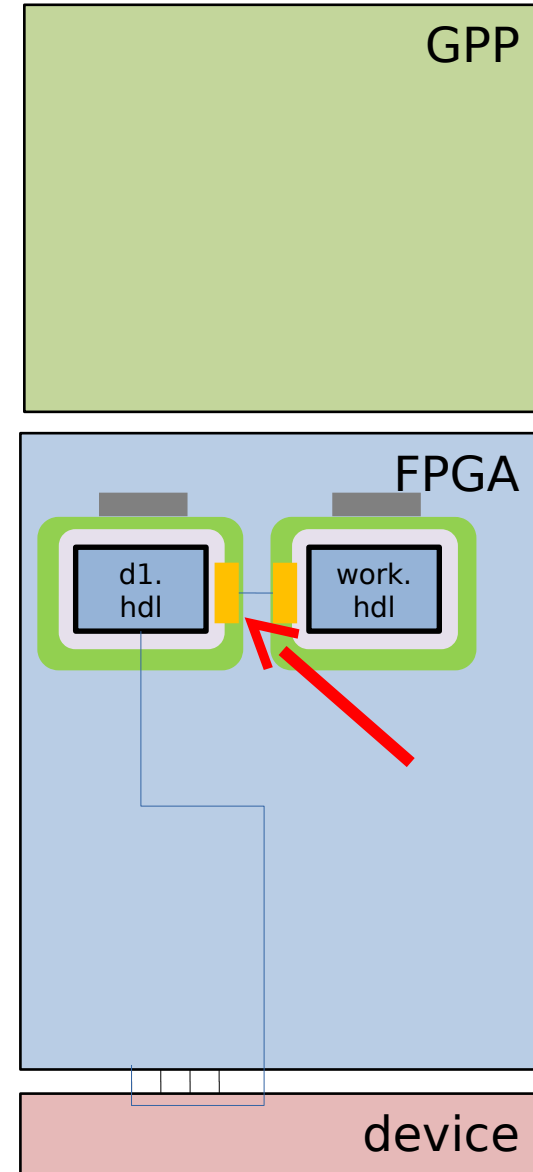
Device/Subdevice Port Types

- Device Workers can have ports just the same as any other HDL worker
- RawProp Port
 - HDL worker port containing raw property signals (used to delegate property accesses to lower-level bus/memory protocols)
- Devsignal
 - An HDL signal (literally a wire on the FPGA) shared between device workers
- Devsignal Port
 - A collection of one or more devsignals (necessary to make devsignal connections)



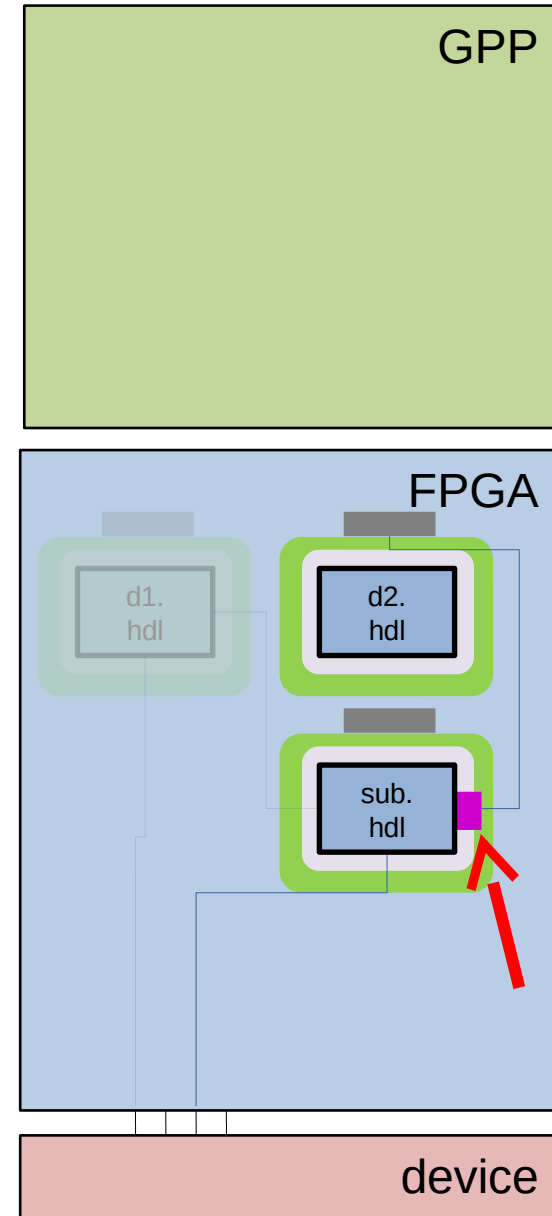
Device/Subdevice Port Types

- Device Workers can have ports just the same as any other HDL worker
- RawProp Port
 - HDL worker port containing raw property signals (used to delegate property accesses to lower-level bus/memory protocols)
- Devsignal
 - An HDL signal (literally a wire on the FPGA) shared between device workers
- Devsignal Port
 - A collection of one or more devsignals (necessary to make devsignal connections)



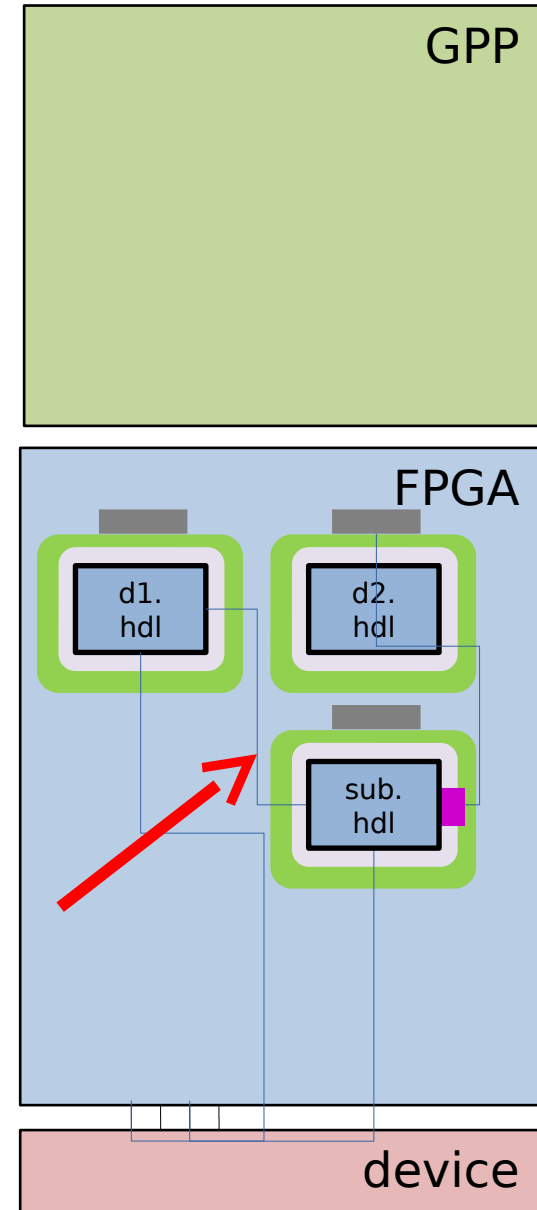
Device/Subdevice Port Types

- Device Workers can have ports just the same as any other HDL worker
- RawProp Port
 - HDL worker port containing raw property signals (used to delegate property accesses of supported device worker to lower-level bus/memory protocols)
- Devsignal
 - An HDL signal (literally a wire on the FPGA) shared between device workers
- Devsignal Port
 - A collection of one or more devsignals (necessary to make devsignal connections)



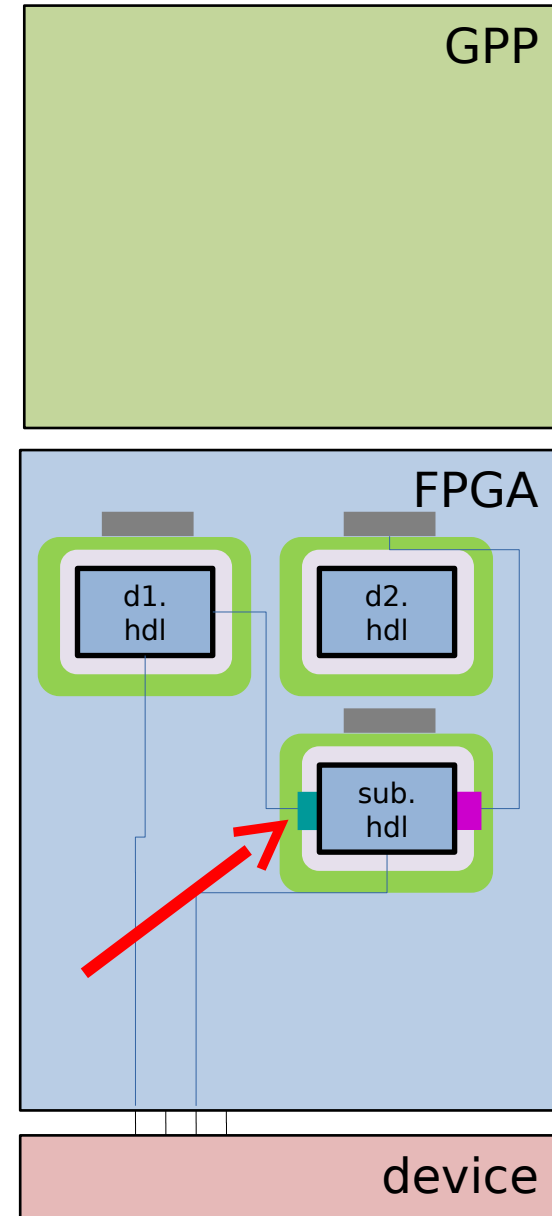
Device/Subdevice Port Types

- Device Workers can have ports just the same as any other HDL worker
- RawProp Port
 - HDL worker port containing raw property signals (used to delegate property accesses of supported device worker to lower-level bus/memory protocols)
- Devsignal
 - An HDL signal (literally a wire on the FPGA) shared between device workers
- Devsignal Port
 - A collection of one or more devsignals (necessary to make devsignal connections)



Device/Subdevice Port Types

- Device Workers can have ports just the same as any other HDL worker
- RawProp Port
 - HDL worker port containing raw property signals (used to delegate property accesses of supported device worker to lower-level bus/memory protocols)
- Devsignal
 - An HDL signal (literally a wire on the FPGA) shared between device workers
- Devsignal Port
 - A collection of one or more devsignals (necessary to make devsignal connections)

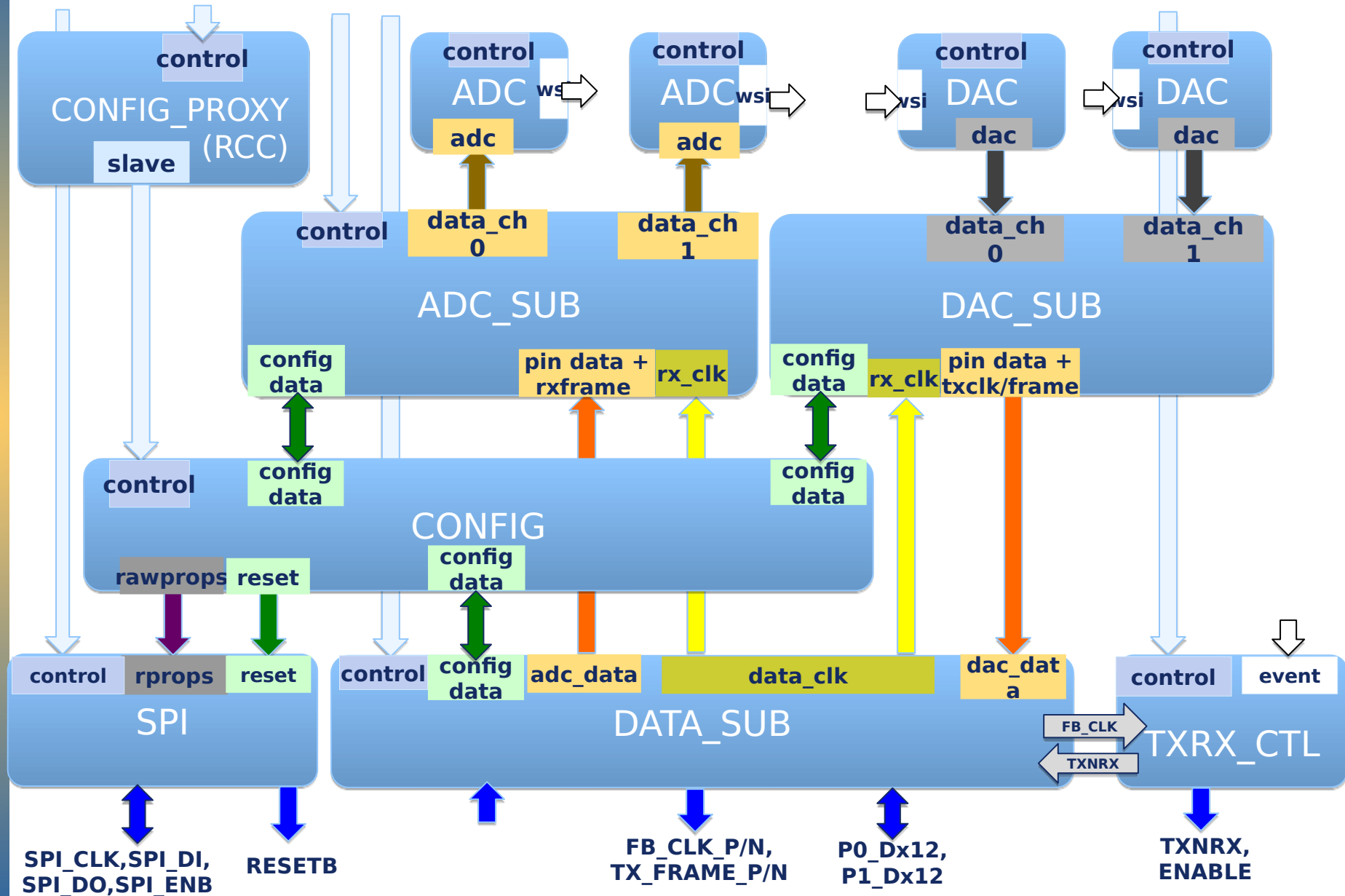


- AD9361 is a complex, multi-function "transceiver on a chip"
 - 2x RF-to-digitized-baseband RX channels
 - 2x digital baseband-to-RF TX channels
 - SPI bus access to large register set
 - Multimodal to support many interfacing devices (I/O standards, FDD/TDD, full/half duplex, small/large data bus widths)

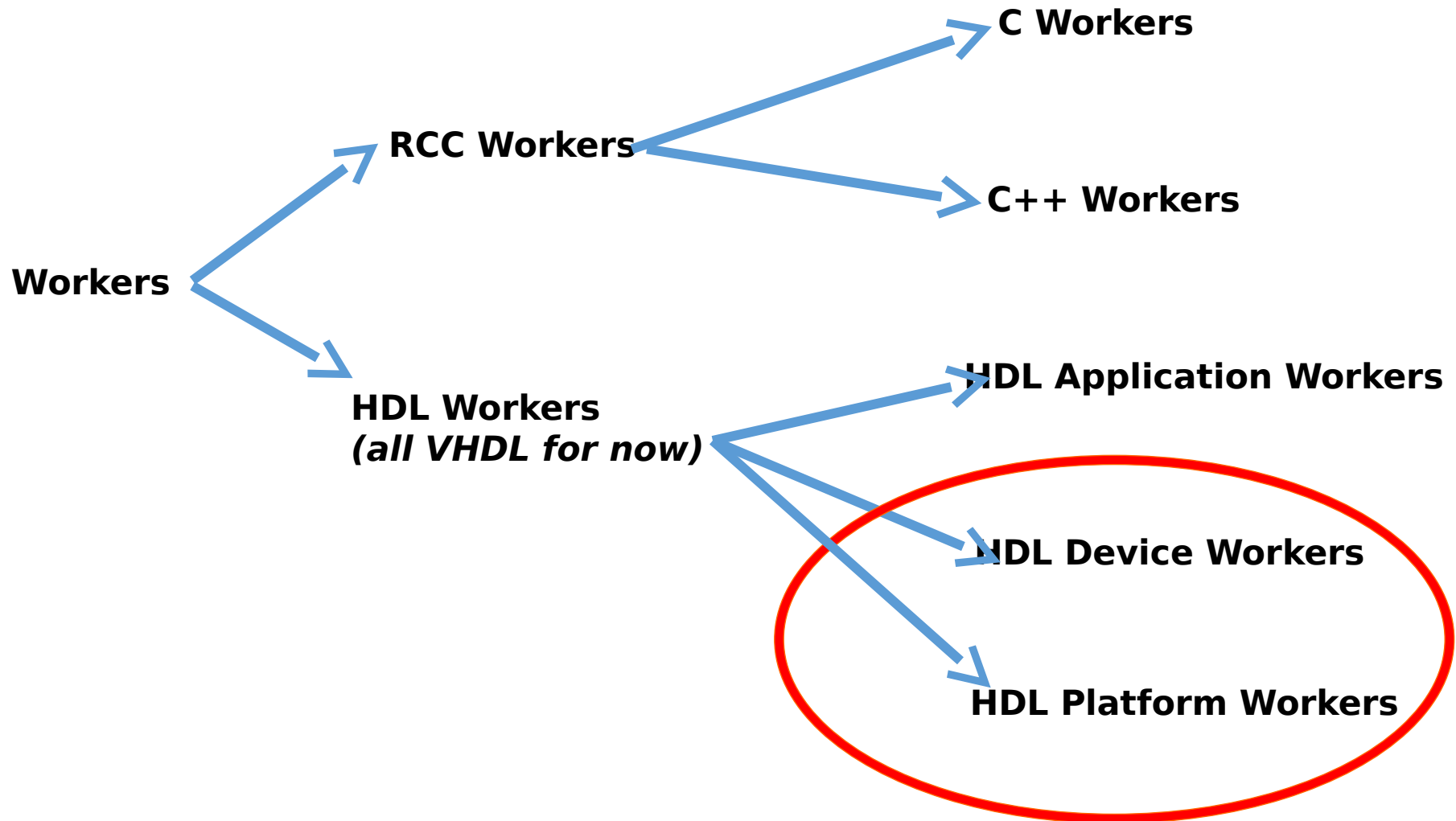
Case Study: AD9361 Support

- OpenCPI design approach supports complex AD9361 functionality
 - Device workers separated by function
 - ad9361_adc.hdl (x2) - single RX data port
 - ad9361_adc_sub.hdl channels - data de-interleaving for all RX channels
 - ad9361_dac.hdl (x2) - single TX data port
 - ad9361_dac_sub.hdl channels - data interleaving for all TX channels
 - ad9361_config.hdl properties - register set exposed as properties
 - ad9361_data_sub.hdl (iostandards, etc) - supports various modes
 - ad9361_spi.hdl - SPI bus access
 - ad9361_config_proxy.rcc - exposes high-level software API

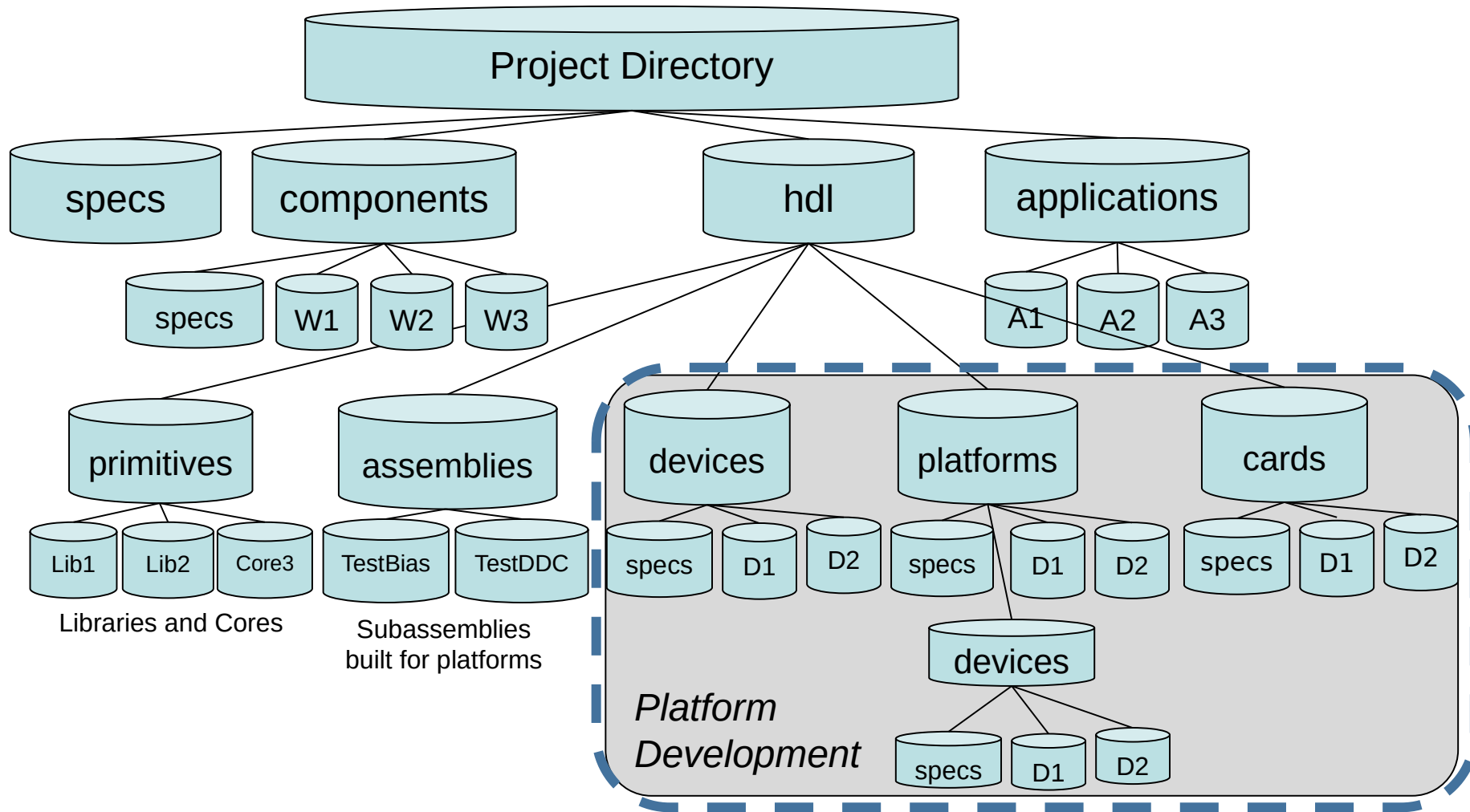
Case Study: AD9361 Support



Types of OpenCPI Workers



Project Directory Layout



HDL device/subdevice workers are either:

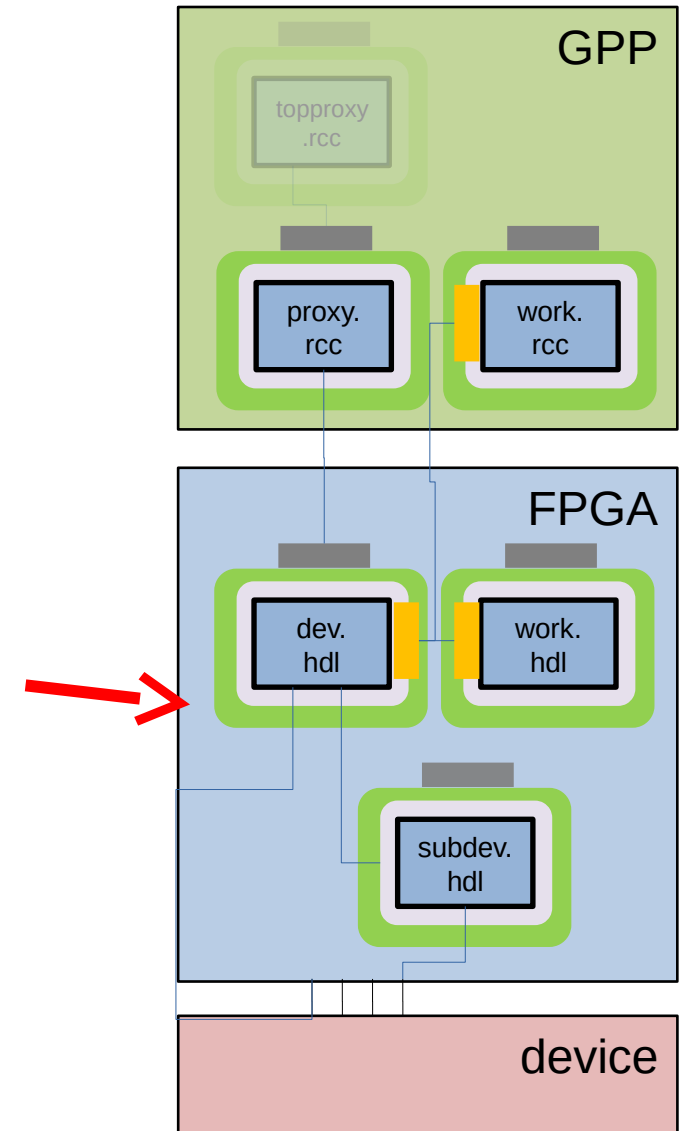
- Simulated (and used alongside an emulator worker)
- Connected to pins on the FPGA that the device is connected to
 - Devices always exist on a platform or daughtercard ("card")
 - HDL device/subdevice workers exist in an OpenCPI platform or card definition

Quick Intro: OpenCPI Platforms, Slots, Cards

- A *platform* is a PCB with:
 - An FPGA which has some connection to a GPP (GPP may be external to PCB)
 - Zero or more devices
 - Zero or more *slots* for connecting *daughtercards* with their own devices
- An OpenCPI platform is supported by:
 - A *platform worker*
 - Zero or more *device workers*
 - Zero or more *slot specifications* (definition of slot pins)
- An OpenCPI platform may be expanded by
 - Zero or more *card specifications* (groups of *device workers* connected to *slot*)

HDL Device Worker Overview

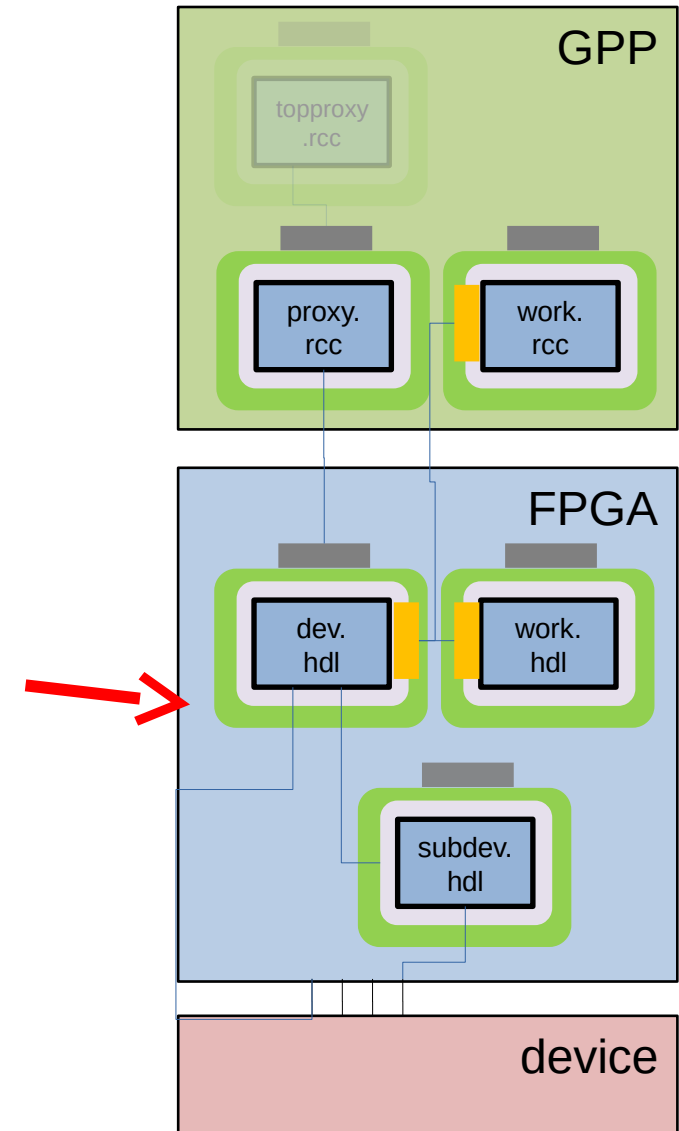
- "An HDL device worker implements (all or some subset of) a data sheet"
- HDL authoring model (VHDL only)
- Often one-to-one relationship with device proxy
 - Device proxy optional
- (All or some subset of) device pins map one-to-one with device worker signals



HDL Device Worker Overview

(cont.)

- Best practices:
 - Use data sheet's signal and register names as close as possible.
 - One-to-one mapping between registers and properties
 - Multi-function devices need modularity (allows saving FPGA resources when using subset of functions), e.g., the following would be separate device workers:
 - `<device>_config.hdl`: Expose register set via properties
 - `<device>_adc.hdl`: Egress ADC data via a port
 - `<device>_dac.hdl`: Ingress DAC data via a port



Creating HDL Device Workers

- Create from top level of project
 - Devices library

```
ocpidev create hdl device my_device.hdl -S <OCS> -h devices
```

- Cards library

```
ocpidev create hdl device my_device.hdl -S <OCS> -h cards
```

- Platform library

```
ocpidev create hdl device my_device.hdl -S <OCS> -P <platform>
```

- Device Worker directory contents:
 - my_device.vhd – initial VHDL skeleton
 - my_device.xml – initial OWD for the device worker

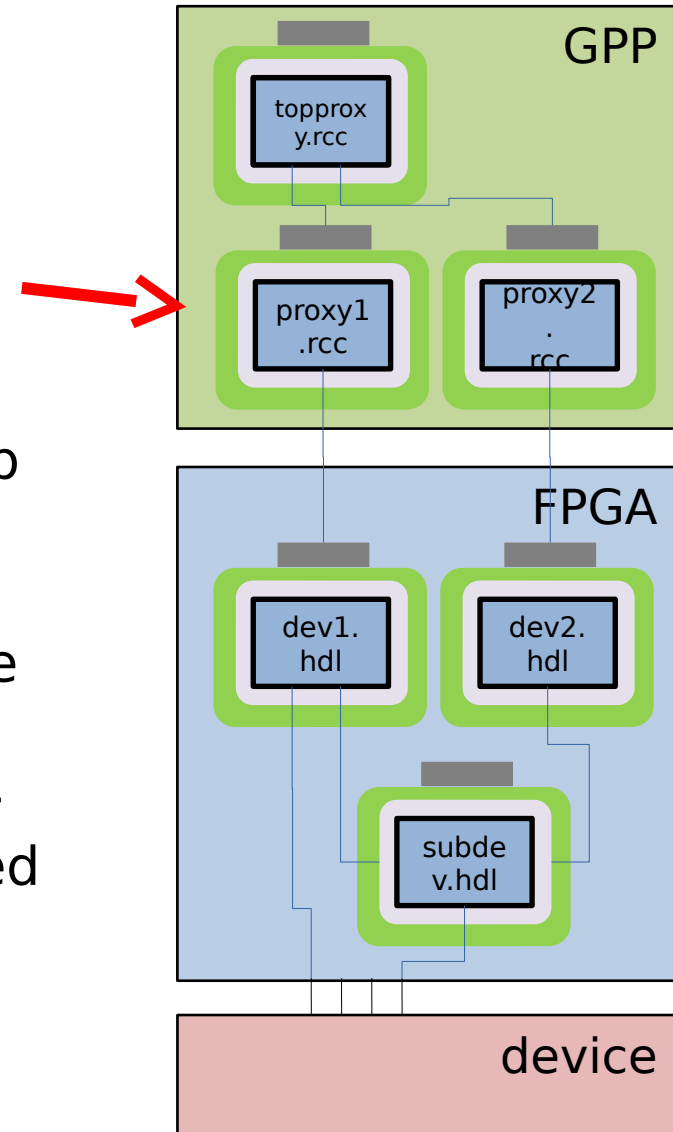
Creating HDL Device Workers (cont.)

- OWD XML example

```
<HdlDevice Language='vhdl' spec='dac-spec'>  
  <StreamInterface Name='in' DataWidth='32'/>  
  <signal output='valid'/>  
  <signal output='data' width='8'/>  
  <signal input='ready'/>  
</HdlDevice>
```

Device Proxy Worker Overview

- Used when a higher-level view (*i.e.* property interface) of the device worker(s) is necessary
- RCC authoring model (C++ only)
 - No restrictions on slave's authoring model
- One-to-one *or* one-to-many relationship with slave worker(s)
- In C++ proxy worker class, given a slave object matching the worker name to access its properties
 - Names can be customized via XML or GUI, and multiple slaves are supported
 - First slave aliased to special name "slave" for compatibility with older user source code (< 1.4)



- Provide a higher-level property interface than the device worker, e.g. floating point tuning frequency proxy property
- Each property represents a device "setting" (which may span multiple devices and/or registers), e.g.:
 - Device has a higher-level conceptual setting which requires configuration of 3 registers across 2 device workers, e.g. swap TX and RX physical media ports
 - One device worker has 2 of the required properties
 - Second device worker has 1 of them
 - ... device proxy worker *exposes all 3 as a single property*

- Create from top level of project
 - Devices library

```
ocpidev create worker my_device_proxy.rcc -S <OCS> -V my_device.hdl -h devices
```

- Cards library

```
ocpidev create worker my_device_proxy.rcc -S <OCS> -V my_device.hdl -h cards
```

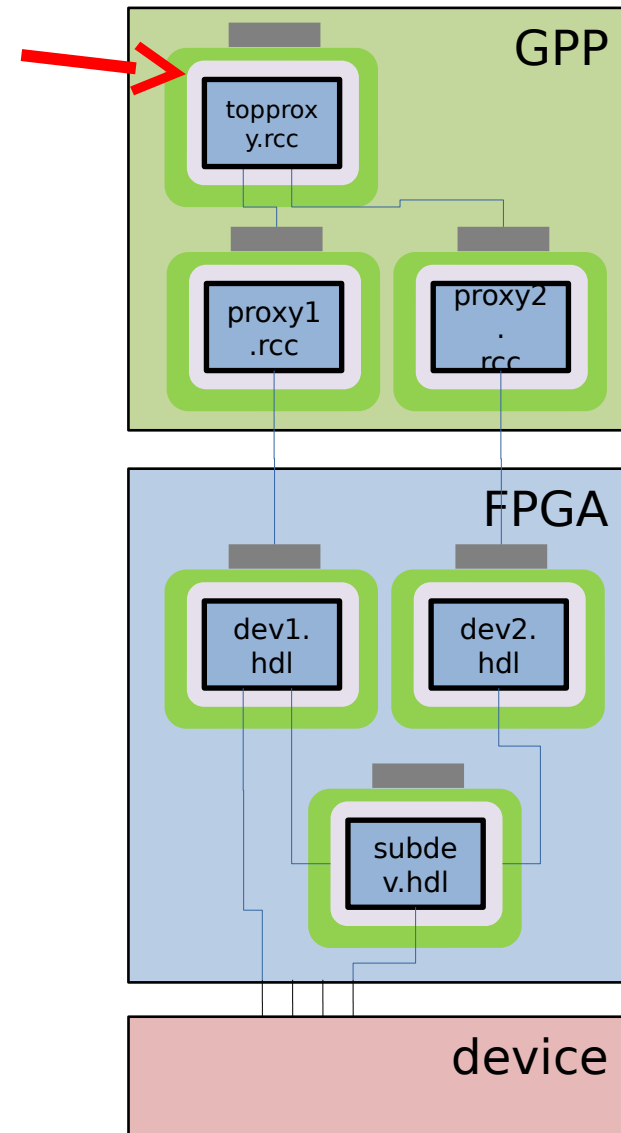
- Platform library

```
ocpidev create worker my_device_proxy.rcc -S <OCS> -V my_device.hdl -P <platform>
```

- Only supports single slave from CLI at this time
- Device Proxy Worker directory contents:
 - my_device.cc – initial C++ skeleton
 - my_device.xml – OWD for the device proxy worker

Top-level Proxy Overview

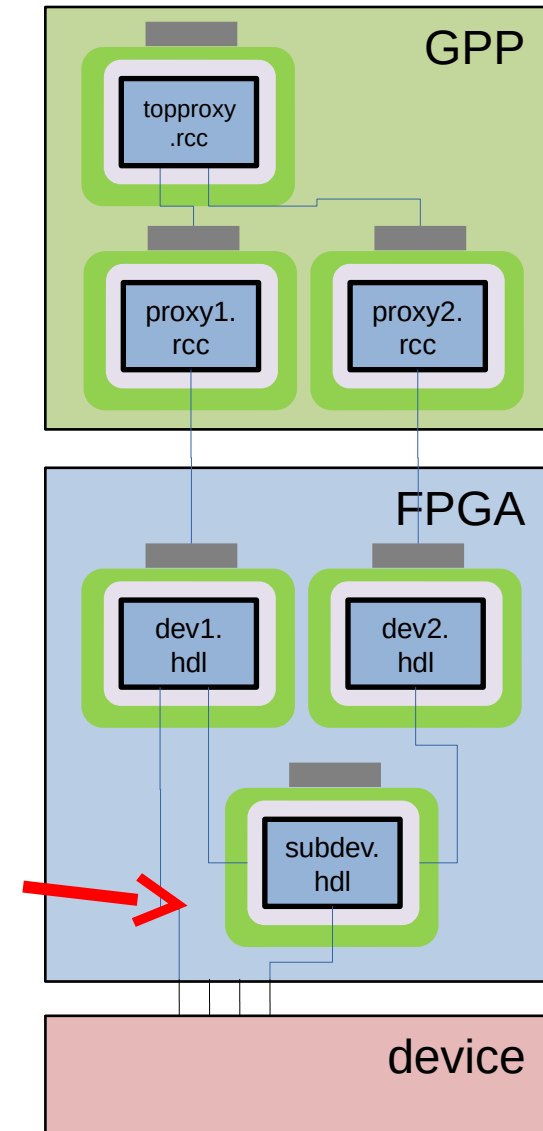
- Intended to standardize app-level interaction with one or more devices, allowing for applications to be highly portable across devices
- Intermediate proxies may or may not be necessary
- Slaves can be *any* model (e.g. `topproxy.rcc`)
- Specialized usage of device proxy worker
 - No difference from the framework's POV
- Best practices
 - Component spec should be hardware (platform/device)-agnostic (but worker implementations are often card/platform-specific)



- AKA "Endpoint proxy"

HDL Subdevice Worker Overview

- Implements required sharing of low-level hardware between HDL device workers
 - Common example: bus protocols (I²C/SPI)
- Common device pins *-map-to→* subdevice signals
- HDL authoring model (VHDL only)
- One-to-one or one-to-many relationship with device workers
- *Supports connections* expose functionality to supported device workers
 - *DevSignals ports* expose signals (literally wires in FPGA)
 - *RawProp ports* expose low-level



- Create from top level of project
- Devices library

```
ocpidev create hdl device my_device_sub.hdl -S <OCS> -h devices
```

- Cards library

```
ocpidev create hdl device my_device_sub.hdl -S <OCS> -h cards
```

- Platform library

```
ocpidev create hdl device my_device_sub.hdl -S <OCS> -P <platform>
```

- Subdevice Worker directory contents:
 - my_device_sub.hdl – initial VHDL skeleton
 - my_device_sub.xml – OWD for the subdevice (must add supports connections w/ devsignal/RawProp ports)

Creating HDL Subdevice Workers

(cont.)

OWD XML
example

```
<HdlDevice Language='vhdl' spec='dac-spec'>
  <signal output='valid'/>
  <signal input='ready'/>

  <!-- RawProp Ports -->
  <rawprop name="rprops"/>

  <!-- Devsignal Ports -->
  <devsignal name="dev_data" signals="data_signals.xml" master="true"/>

  <supports worker="ad9361_dac"/>
    <connect port="dev_dac" to="dev_data"/>
    <connect port="rawprops" to="rprops"/>
  </supports>
</HdlDevice>
```

Device Support Development Process

- 1) Acquire device data sheet
- 2) Plan and develop HDL workers
 - a) Break down device capabilities by function and plan accordingly
 - **Does the device have a bus access to a register set?**
 - **Create device worker w/ register set as properties**
 - **Create subdevice worker for bus access**
 - **Does the device send data to an FPGA (e.g. DAC interface)?**
 - **Create device worker w/ input port(s)**
 - **Does the device send data from an FPGA (e.g. ADC interface)?**
 - **Create device worker w/ output port(s)**
 - b) Add devices to card/platform definitions
 - c) Build bitstream using card/platform support for new device
- 3) Develop RCC workers
 - **Does the current collection of workers expose standard, portable properties for the device?**
 - **Create device proxy (single device worker slave) or endpoint proxy (multiple slaves)**