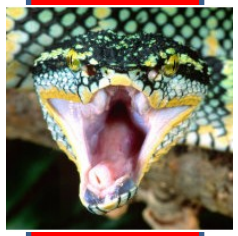
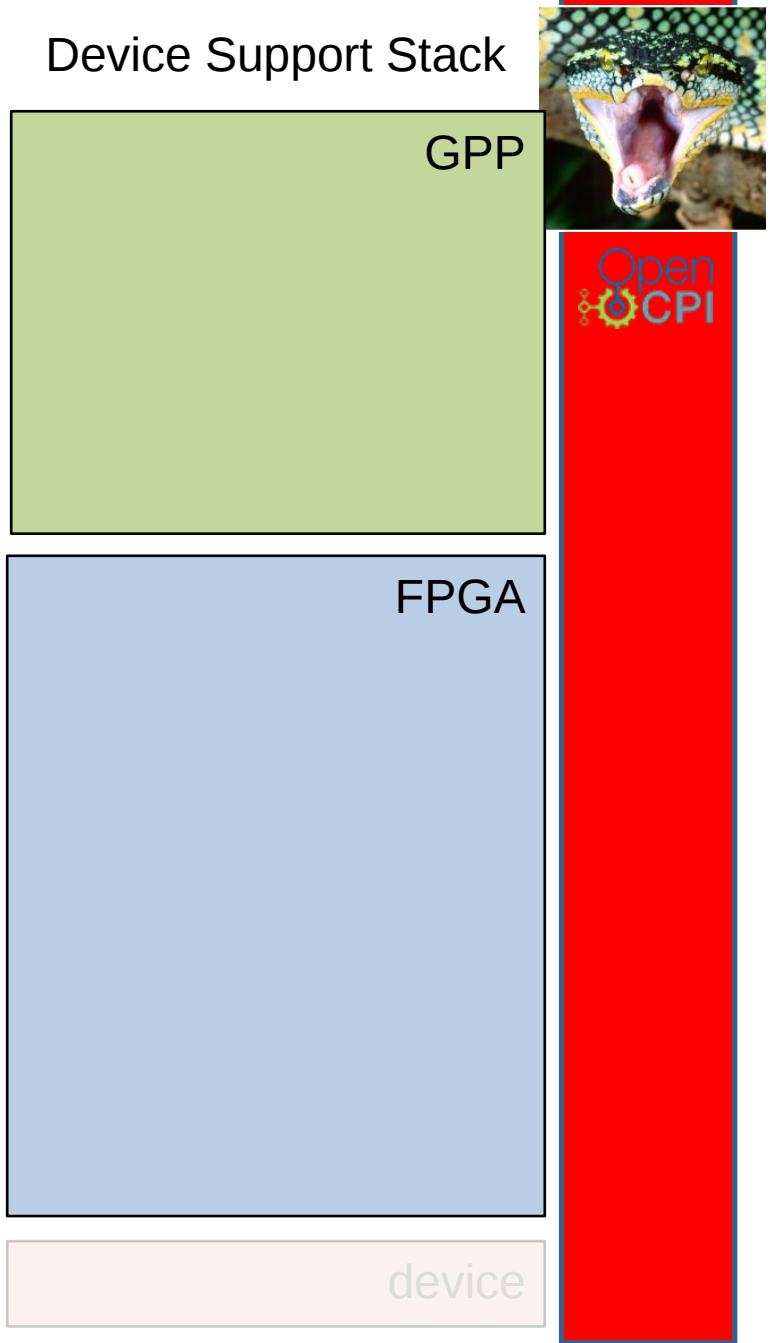


Device Support for FPGA Platforms

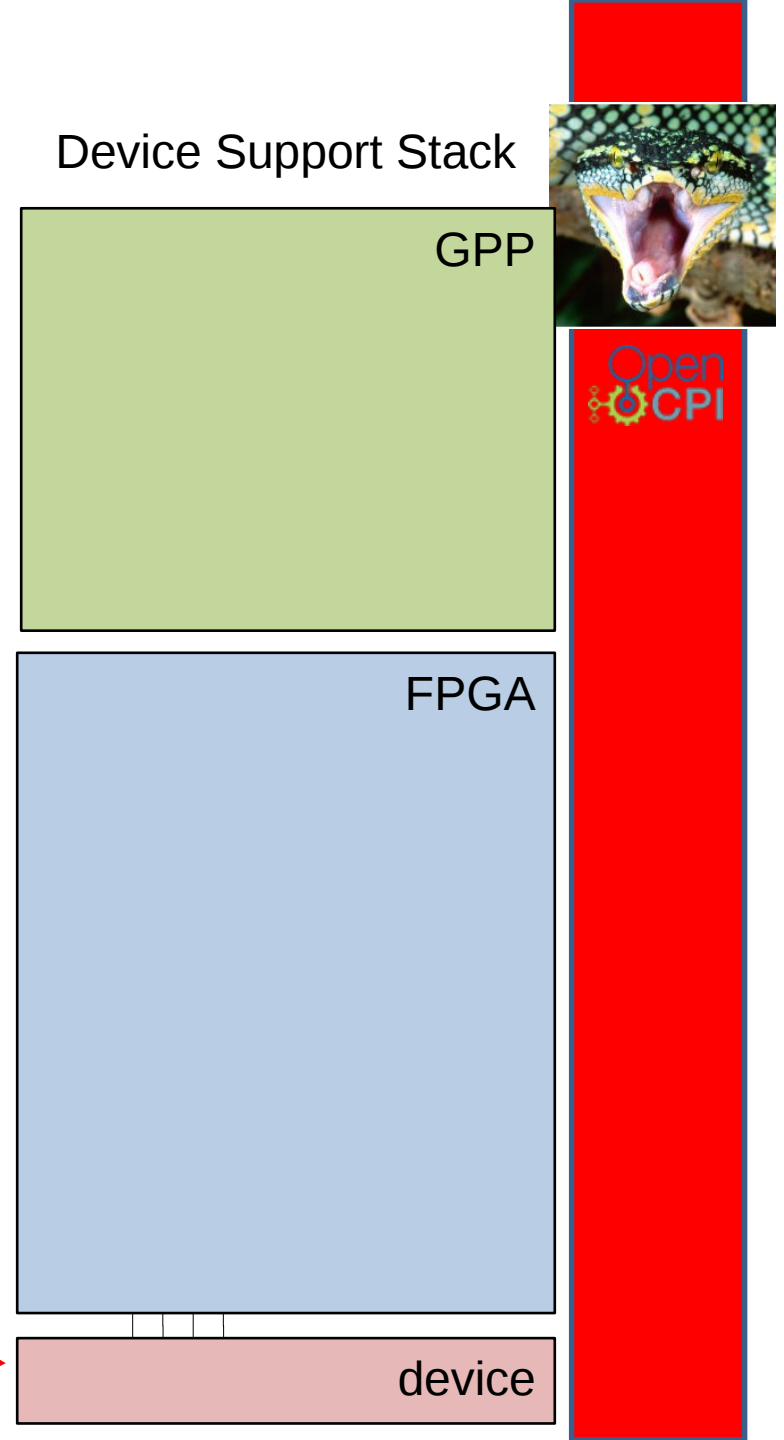
Device Support Concepts

- Endpoint Proxy
  - A software worker that typically have multiple device proxy “slaves”
- Device Proxy
  - A software worker (RCC/C++) specifically paired with a device worker 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



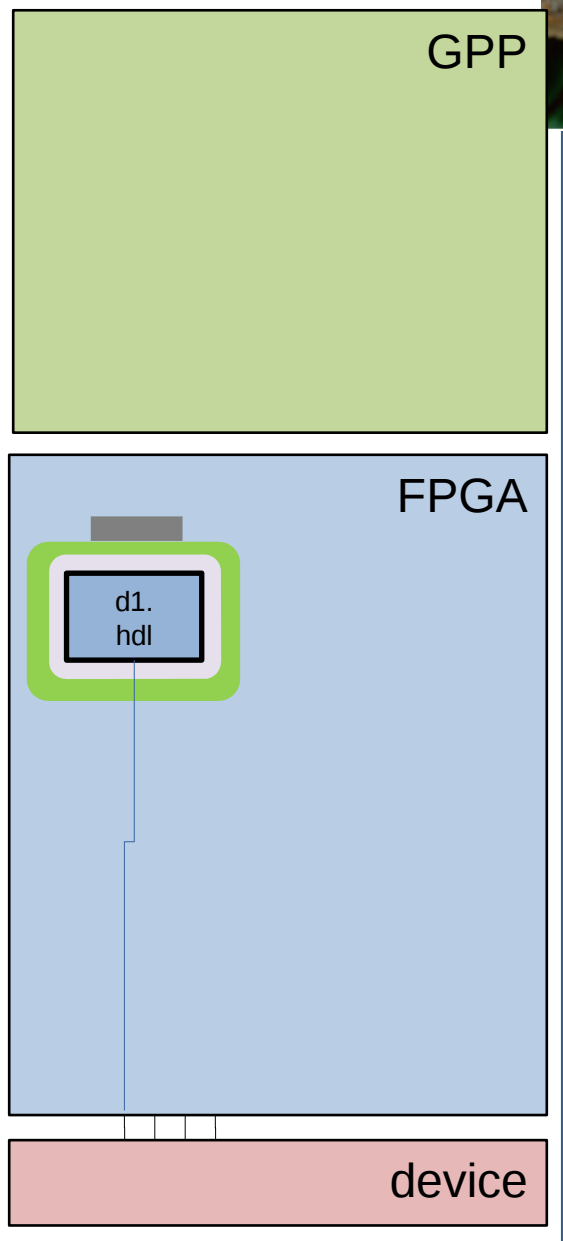
Device Support Concepts

- Endpoint Proxy
  - A software worker that typically have multiple device proxy “slaves”
- Device Proxy
  - A software worker (RCC/C++) specifically paired with a device worker 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



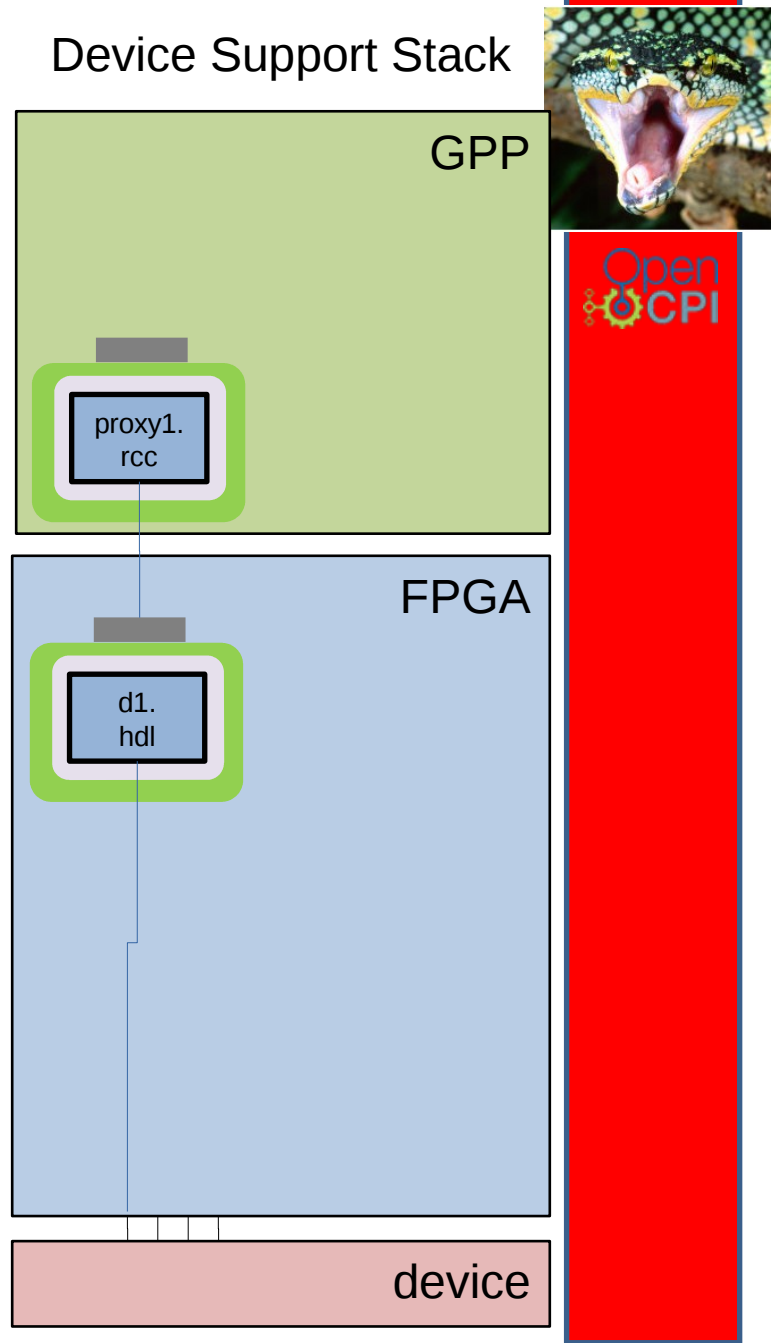
Device Support Concepts

- Endpoint Proxy
  - A software worker that typically have multiple device proxy “slaves”
- Device Proxy
  - A software worker (RCC/C++) specifically paired with a device worker 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



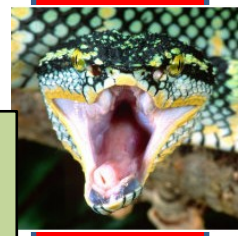
Device Support Concepts

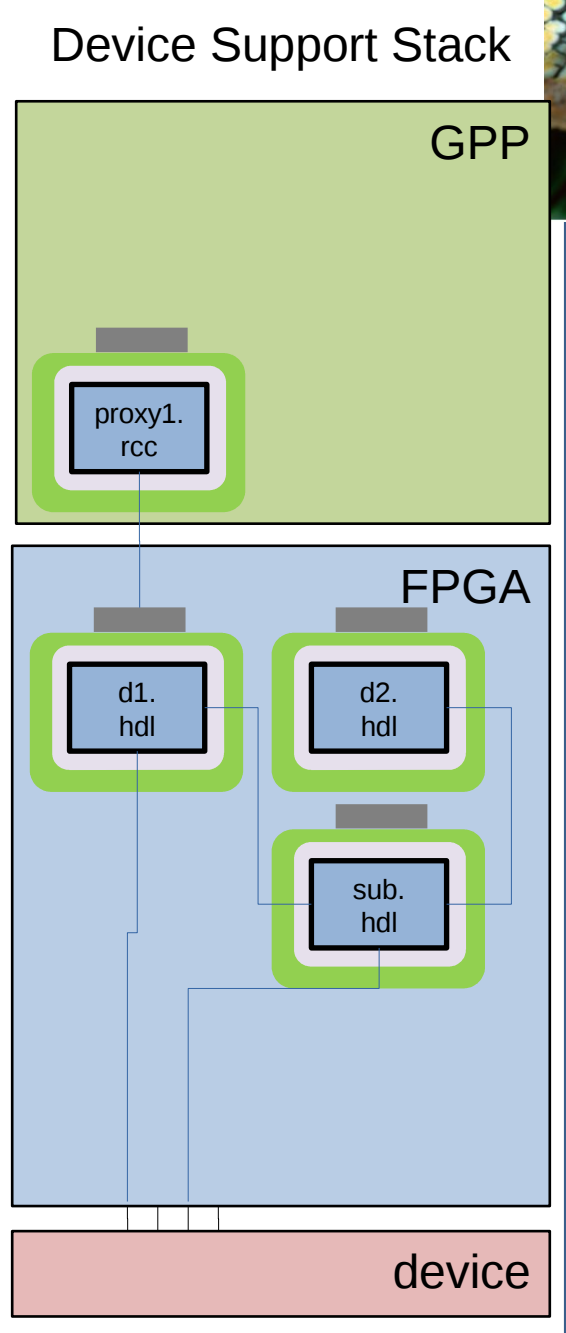
- Endpoint Proxy
  - A software worker that typically have multiple device proxy “slaves”
- Device Proxy
  - A software worker (RCC/C++) specifically paired with a device worker which, when required, provides a higher-level control interface
- Device Worker
  - A specialized HDL Worker intended for direct 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 Support Concepts

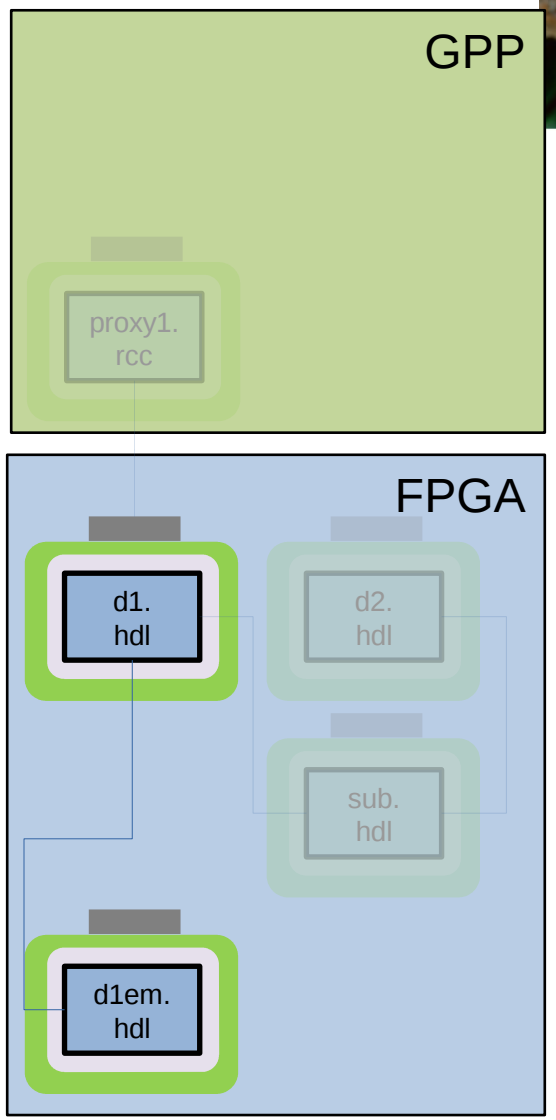
- Endpoint Proxy
  - A software worker that typically have multiple device proxy “slaves”
- Device Proxy
  - A software worker (RCC/C++) specifically paired with a device worker which provides a high-level control interface
- Device Worker
  - A specialized HDL Worker intended for direct connection to device pins
- Subdevice Worker
  - Specialized Device Worker which implements required sharing of low level hardware between Device Workers, e.g. a shared I2C bus
- Emulator Worker
  - Emulates a device for test purposes
- Device
  - Hardware elements that are locally attached to the processor of the platform

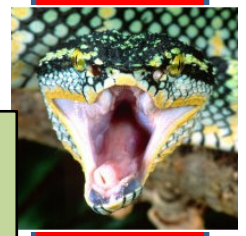




Device Support Concepts

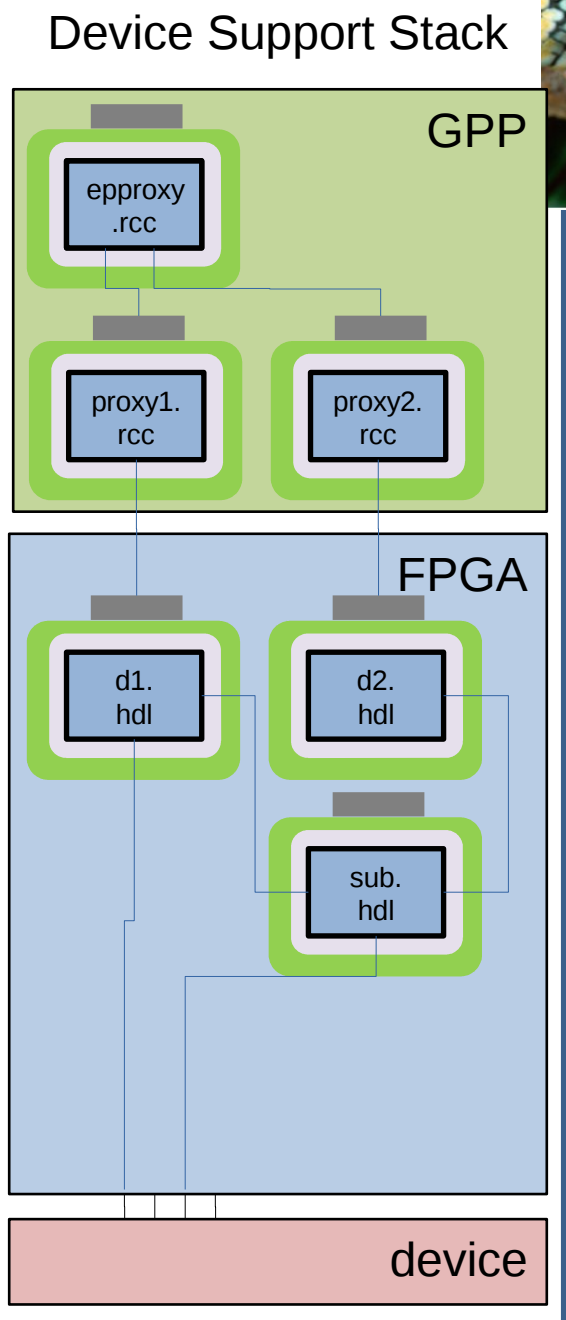
- Endpoint Proxy
  - A software worker that typically have multiple device proxy “slaves”
- Device Proxy
  - A software worker (RCC/C++) specifically paired with a device worker which provides a high-level control interface
- Device Worker
  - A specialized HDL Worker intended for direct connection to 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





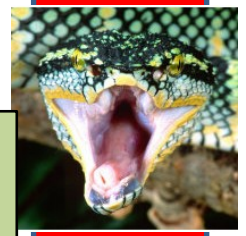
Device Support Concepts

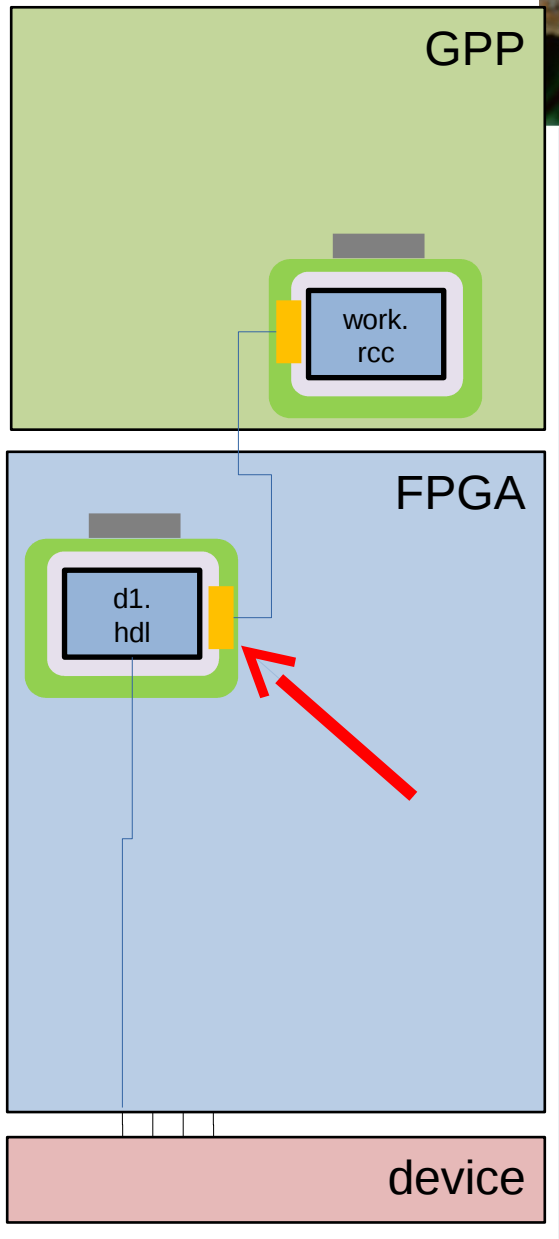
- Endpoint Proxy
  - A software worker that typically has multiple device proxy “slaves”
- Device Proxy
  - A software worker (RCC/C++) specifically paired with a device worker which provides a high-level control interface
- Device Worker
  - A specialized HDL Worker intended for direct connection to 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

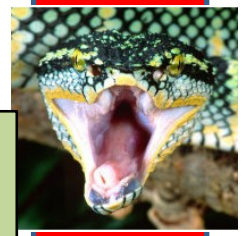
- Device Workers can have ports just the same as any other HDL worker
- RawProp Port
  - HDL worker port containing raw property signals (used to delegate property accesses to lower-level bus/memory protocols)
- Devsignal
  - An HDL signal (literally a wire on the FPGA) shared between device workers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

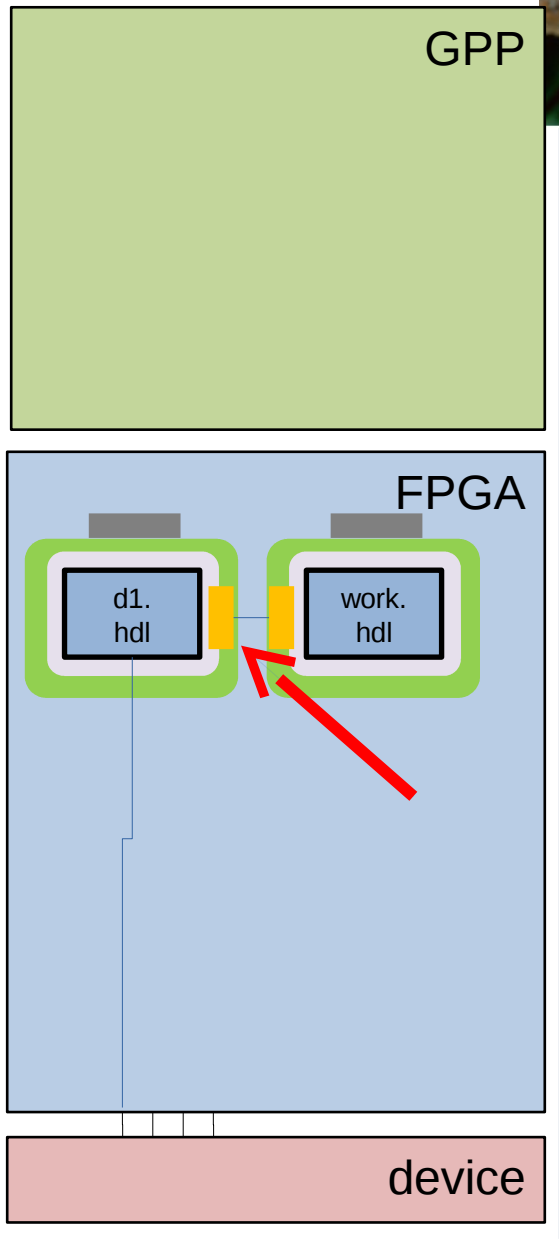




Device/Subdevice Port Types

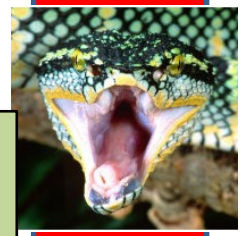
- Device Workers can have ports just the same as any other HDL worker
- RawProp Port
  - HDL worker port containing raw property signals (used to delegate property accesses to lower-level bus/memory protocols)
- Devsignal
  - An HDL signal (literally a wire on the FPGA) shared between device workers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

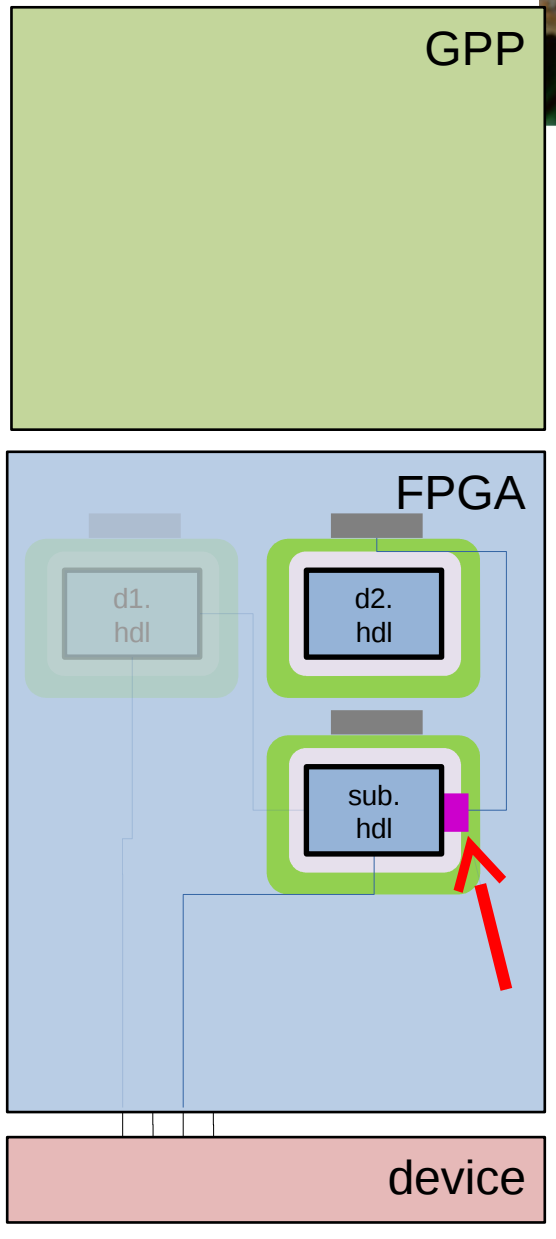




Device/Subdevice Port Types

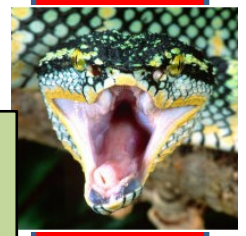
- Device Workers can have ports just the same as any other HDL worker
- RawProp Port
  - HDL worker port containing raw property signals (used to delegate property accesses of supported device worker to lower-level bus/memory protocols)
- Devsignal
  - An HDL signal (literally a wire on the FPGA) shared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





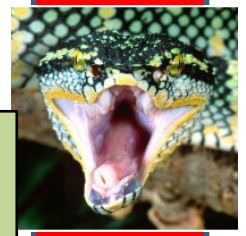
Device/Subdevice Port Types

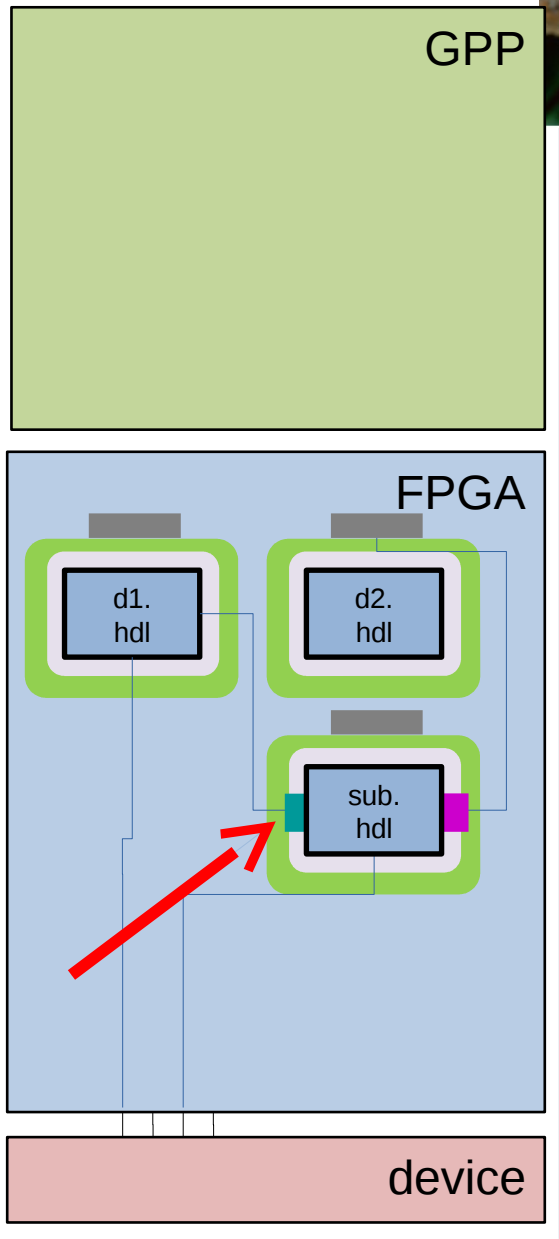
- Device Workers can have ports just the same as any other HDL worker
- RawProp Port
  - HDL worker port containing raw property signals (used to delegate property accesses of supported device worker to lower-level bus/memory protocols)
- Devsignal
  - An HDL signal (literally a wire on the FPGA) shared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



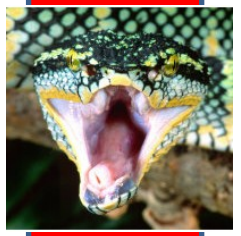
Device/Subdevice Port Types

- 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)



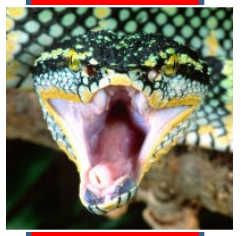


Case Study – AD9361 Support



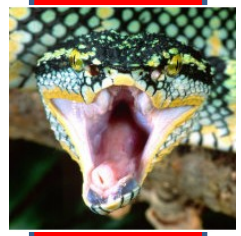
- AD9361 is a complex, multi-function “transceiver on a chip”
  - 2x RF-to-digitized-baseband RX channels
  - 2x digital baseband-to-RF TX channels
  - SPI bus access to large register set
  - Multimodal to support many interfacing devices (I/O standards, FDD/TDD, full/half duplex, small/large data bus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

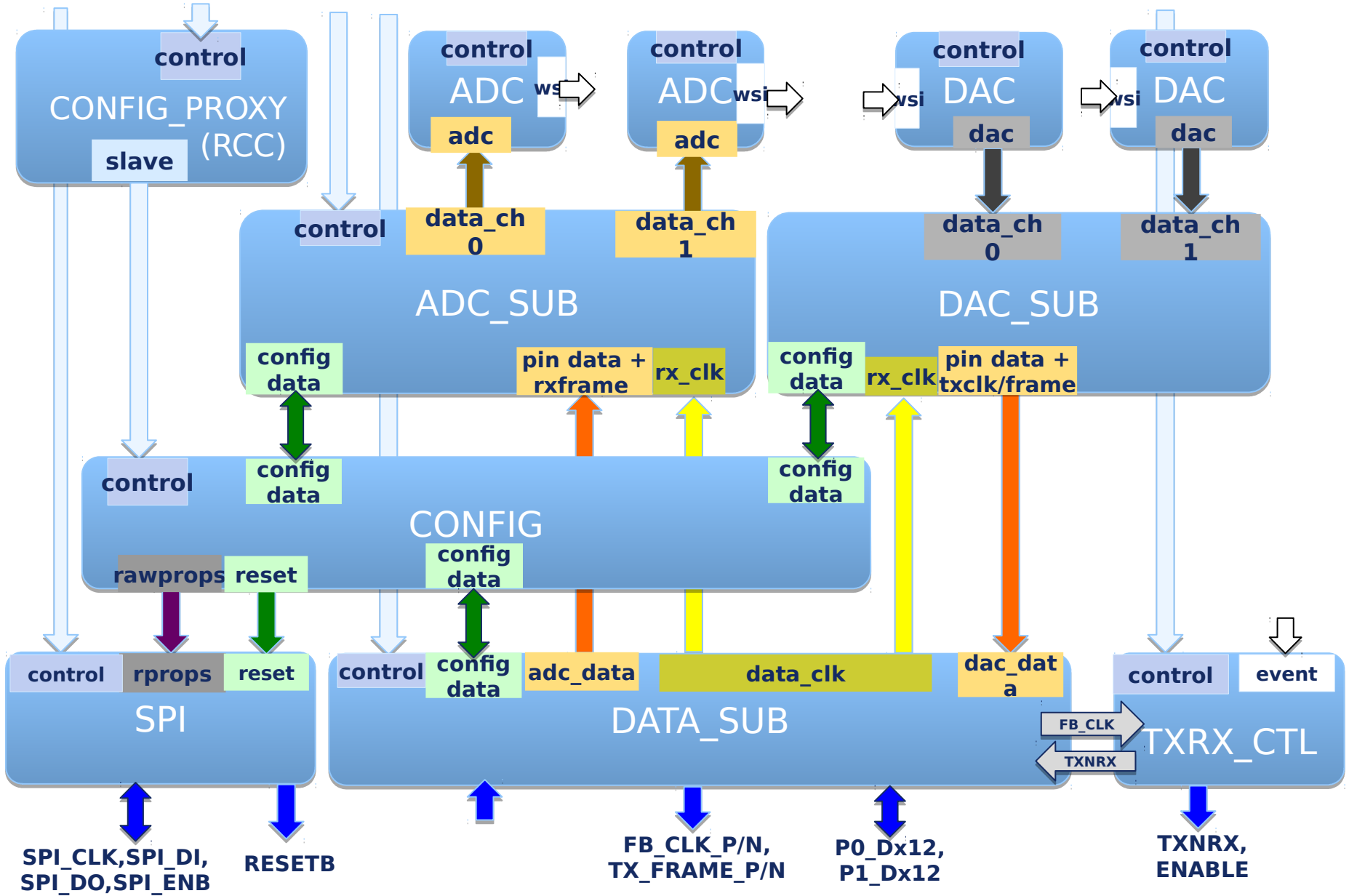
Case Study – AD9361 Support



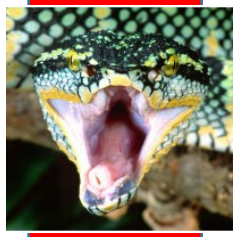
- OpenCPI design approach supports complex AD9361 functionality
  - Device workers separated by function
    - ad9361_adc.hdl (2x) - single RX data port
    - ad9361_adc_sub.hdl - data de-interleaving for all RX channels
    - ad9361_dac.hdl (2x) - single TX data port
    - ad9361_dac_sub.hdl - data interleaving for all TX channels
    - ad9361_config.hdl - register set exposed as properties
    - ad9361_data_sub.hdl - supports various modes (iostandards, etc)
    - ad9361_spi.hdl - SPI bus access
    - ad9361_config_proxy.rcc - exposes high-level software API
  - AD9361 modes implemented via worker parameter properties (I/O standards, FDD/TDD, full/half duplex, small/large data bus widths)

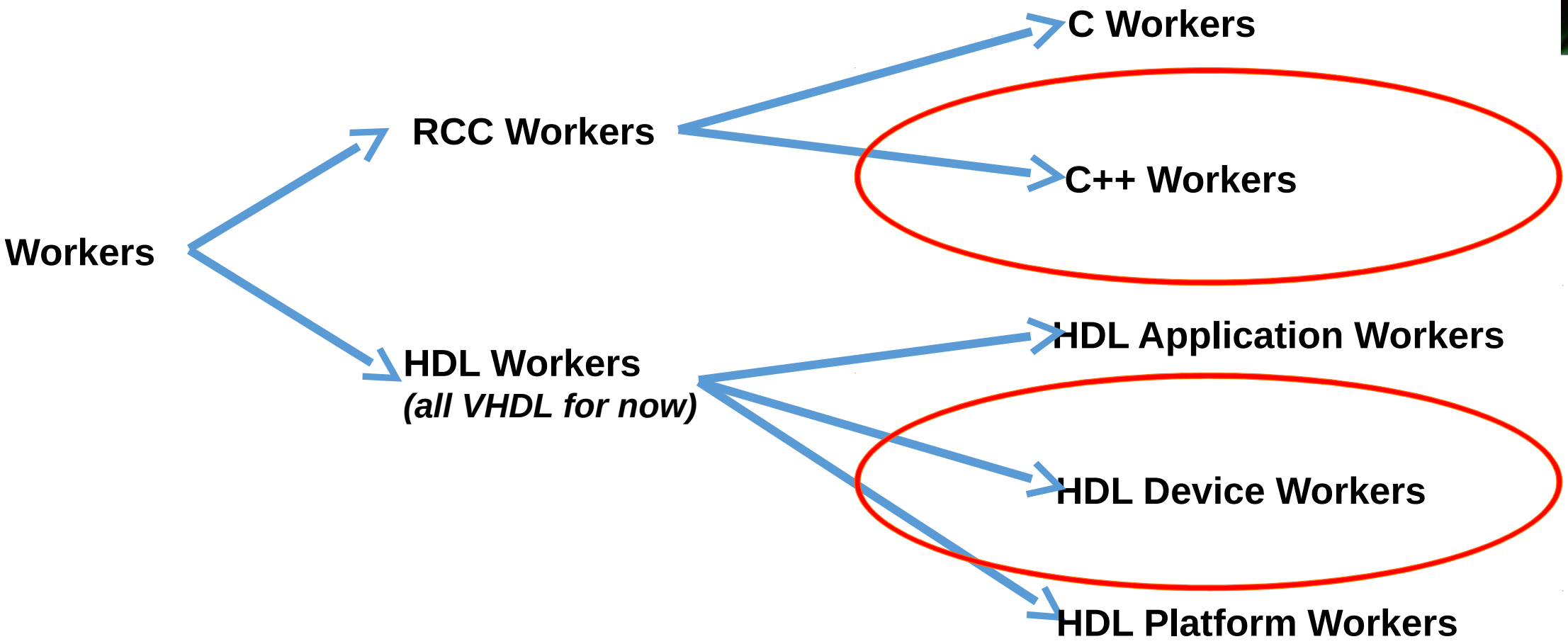
Case Study – AD9361 Support



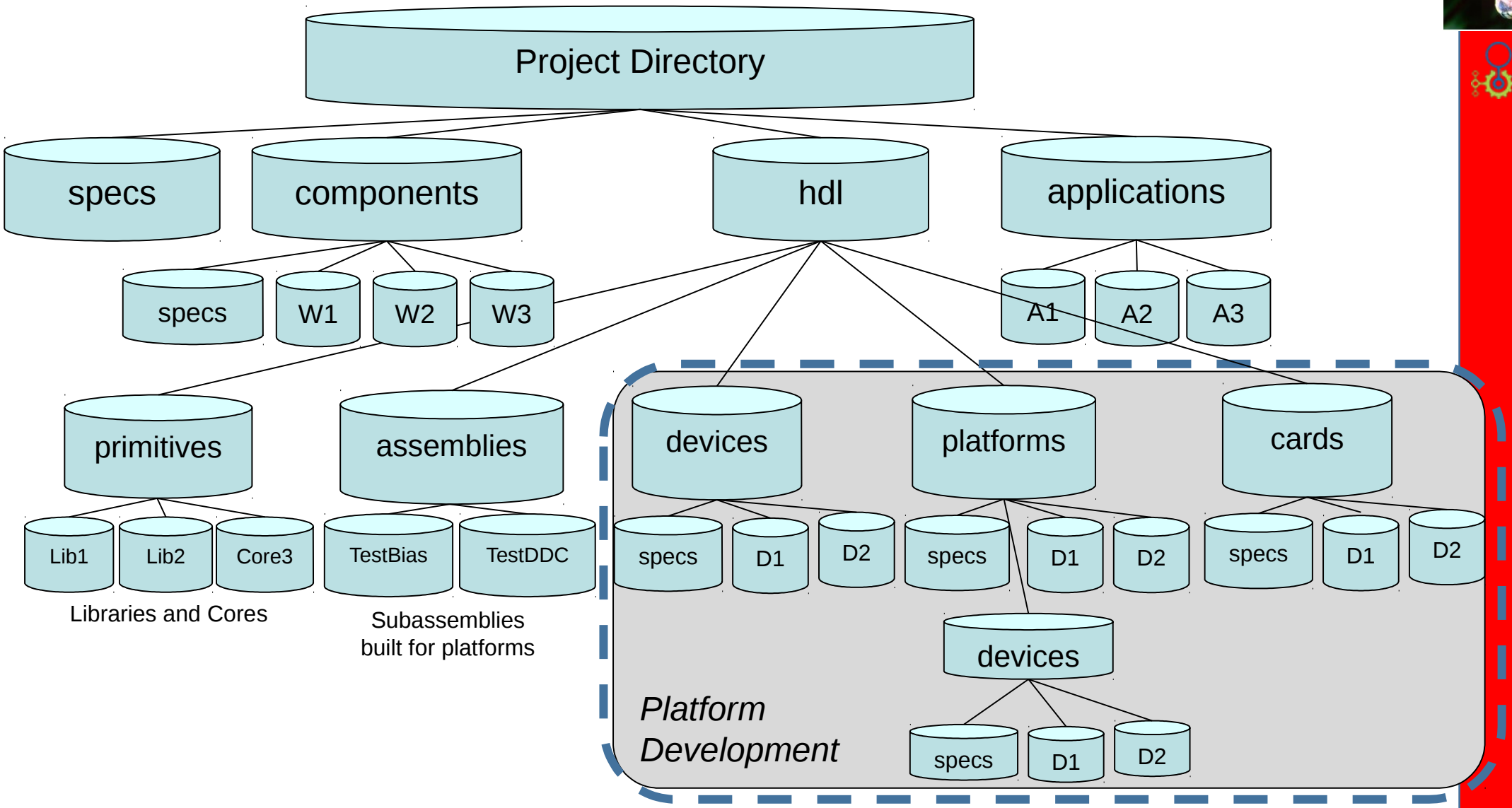


Types of OpenCPI Workers

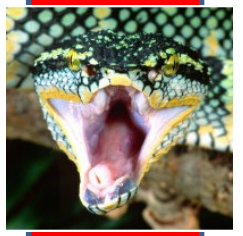




OpenCPI Project Directory Layout



Device Support Usage



Device/subdevice workers are either:

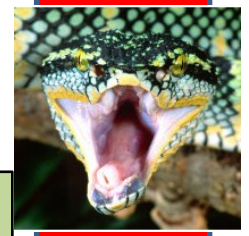
- Simulated (and used alongside an emulator worker), or
- Used in FPGA bitstream alongside the physical device
  - Devices always exist on a card or a platform
  - Device/subdevice workers exist in and OpenCPI card or platform definition

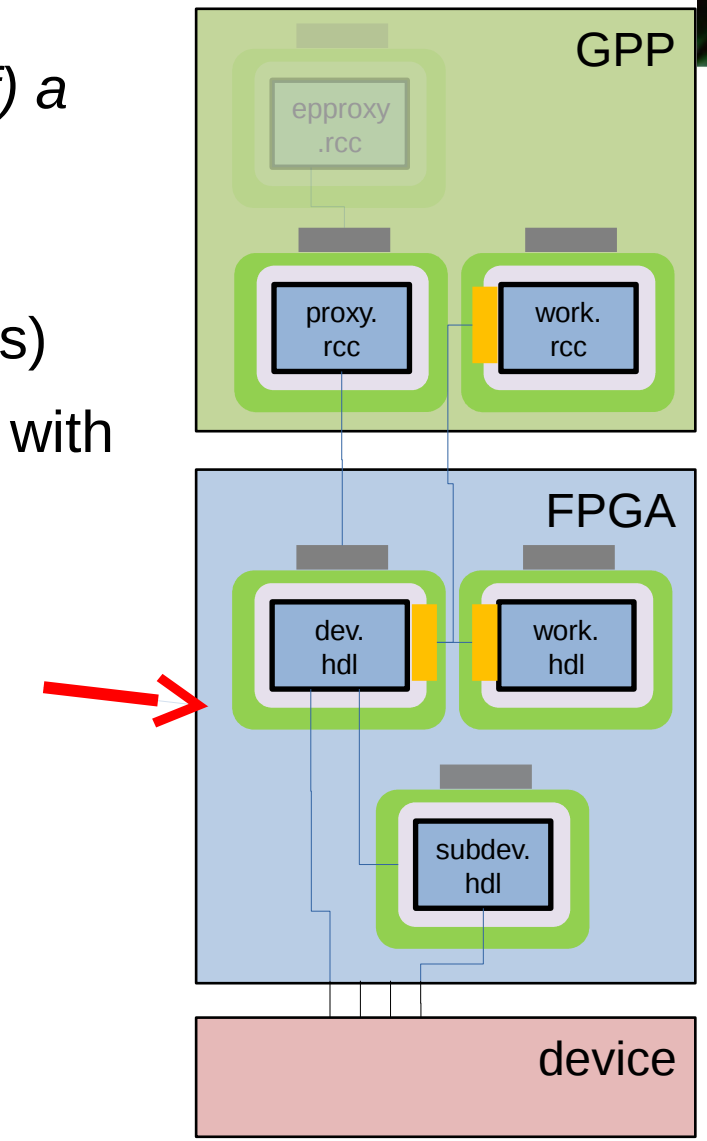
Quick Intro - OpenCPI Platforms/Slots/Cards

- 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)

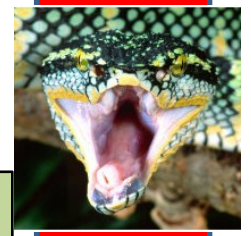
Device Worker Overview

- *"A device worker implements (all or some subset of) a data sheet"*
- HDL authoring model (VHDL only)
- One-to-one relationship with device proxy (if it exists)
- (all or some subset of) device pins map one-to-one with device worker signals

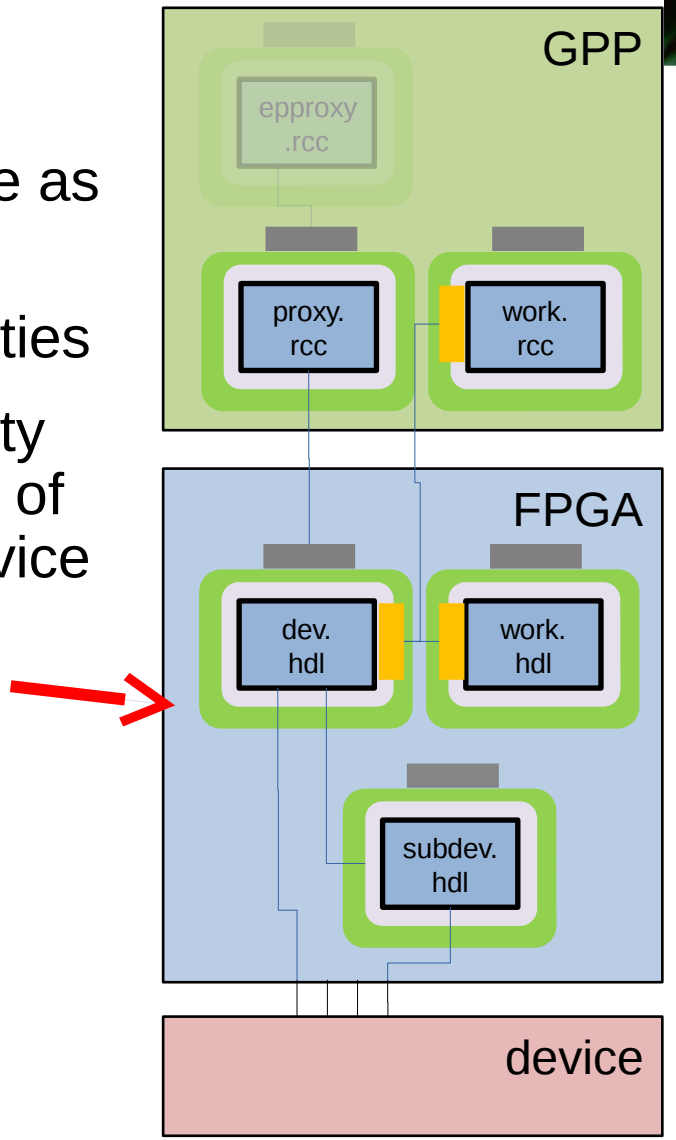




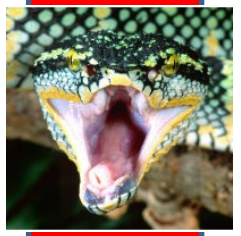
Device Worker Overview



- Best practices:
  - Use datasheet's signal and register names as close as possible.
  - One-to-one mapping between registers and properties
  - Multi-function devices need modularity for optionality (allows saving FPGA resources when using subset of functions), e.g. the following would be separate device workers (a:
    - <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
    - Direct control of enable pin(s)



Device Worker Creation



- 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

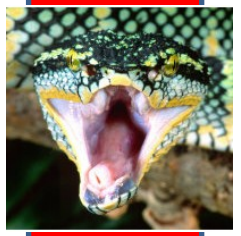
```
ocpidev create hdl device my_device.hdl  -S <OCS> -P <platform>
```

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

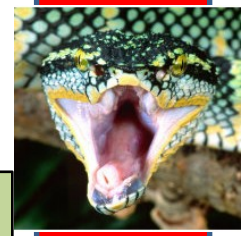
Device Worker Creation

- OWD XML example

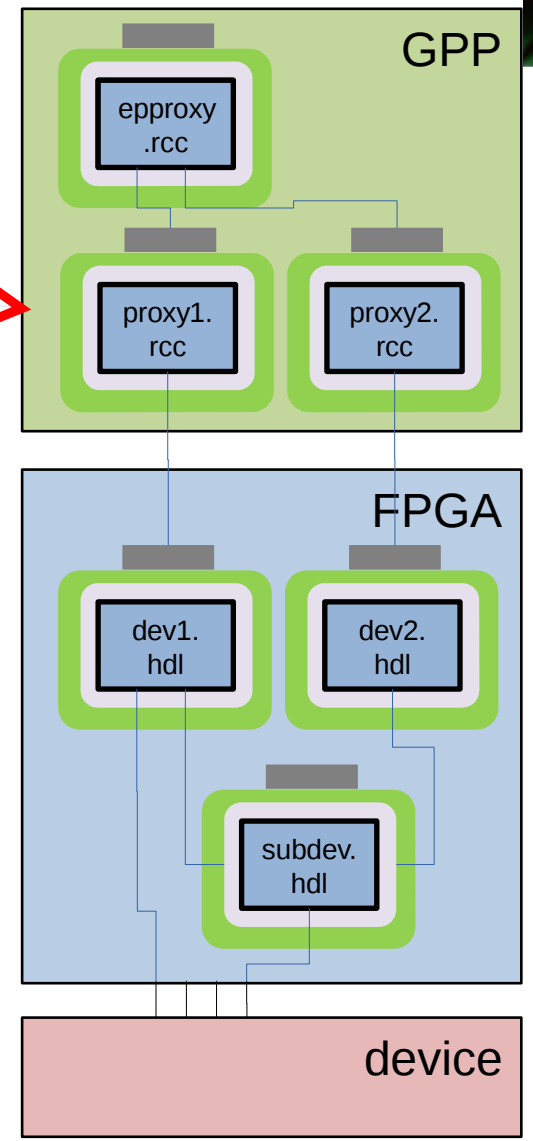
```
<HdlDevice Language='vhdl' spec='dac-spec'>
      <StreamInterace Name='in' DataWidth='32'/>
      <signal output='valid'/>
      <signal output='data' width='8'/>
      <signal input='ready'/>
</HdlDevice>
```



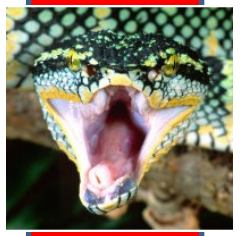
Device Proxy Overview



- A device proxy is used when a higher-level property interface than the device worker is necessary
- RCC authoring model (C++ only)
- One-to-one relationship with device worker (cannot have one-to-many)
- Skeleton is implemented as C++ class w/ “slave” object, which is used to access slave device worker's properties
- Best practices:
  - **Provides a higher-level property interface than the device worker, e.g. floating point center/tuning frequency proxy property**
  - **Each property represents a device “setting” (which may span multiple device registers), e.g. device has divider setting which spans 3 registers, device worker has 3 properties, but device proxy exposes all 3 as a single value**



Device Proxy Creation



- 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
```

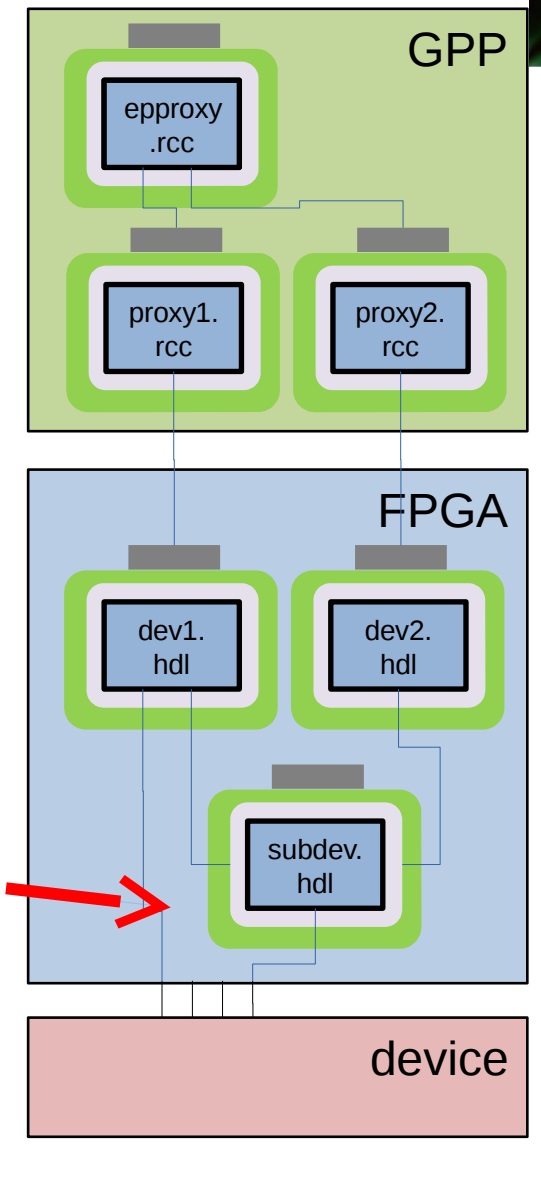
  - Platform library

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

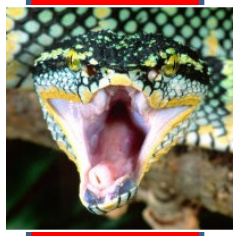
- Device Proxy directory contents:
  - my_device.cc – initial C++ skeleton
  - my_device.xml – OWD for the device proxy
  - Makefile

Subdevice Worker Overview

- Implements required sharing of low level hardware between 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 bus access as properties



Subdevice Worker Creation



- Create from top level of project
  - Devices library

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

  - Cards library

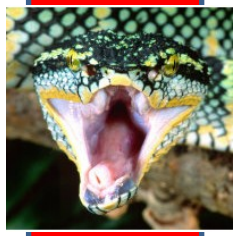
```
ocpidev create hdl device my_device_sub.hdl  -S <OCS> -h cards
```

  - Platform library

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

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

Subdevice Worker Creation

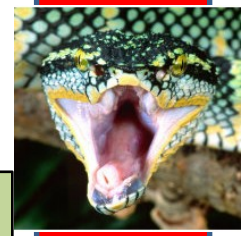


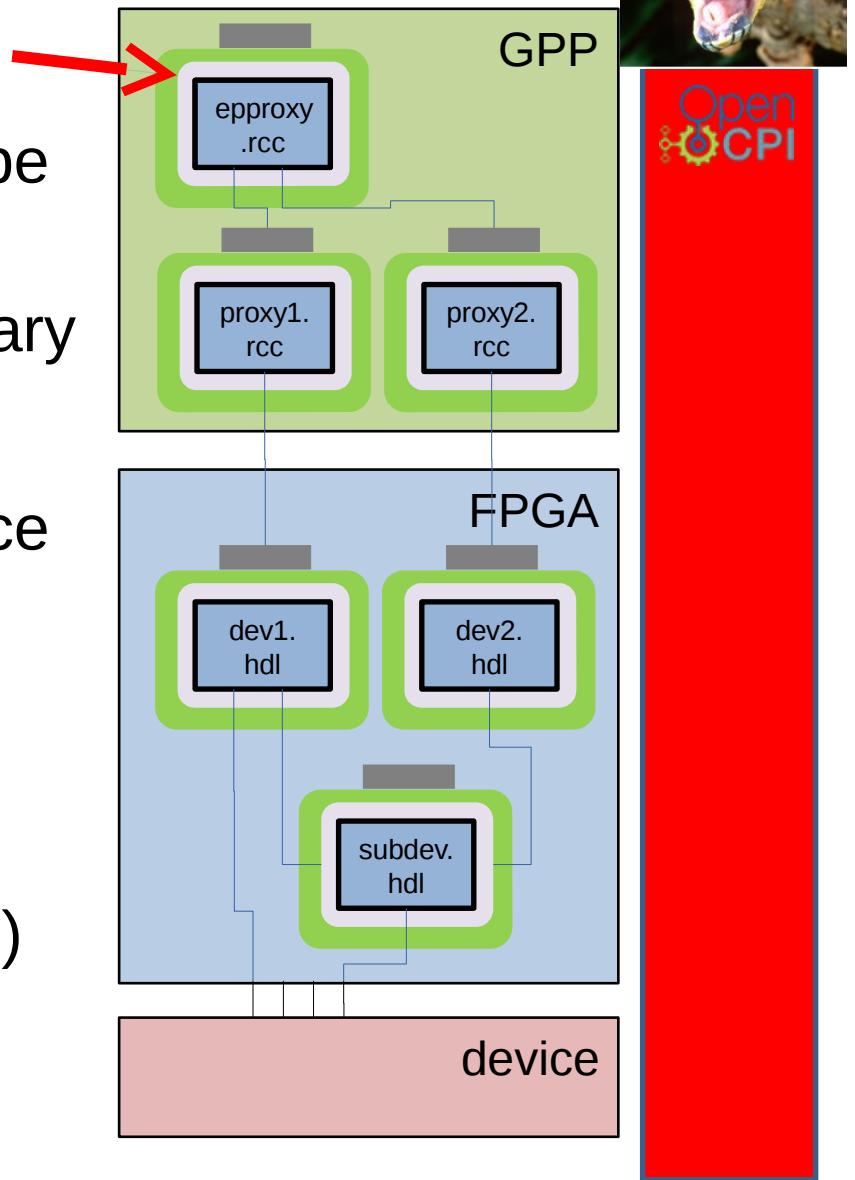
- OWD XML example

```
<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>
```

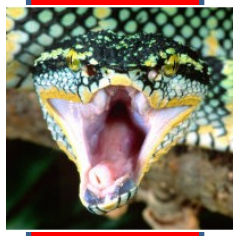
Endpoint Proxy Overview

- 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
- RCC authoring model
- One-to-one or one-to-many relationship with device proxy(ies) and/or device worker(s)
- Best practices
  - Component spec should be hardware (platform/device)-agnostic (but worker implementations are often card/platform-specific)





Endpoint Proxy Overview



- Create from top level of project
  - Devices library

```
ocpidev create worker my_endpoint_proxy.rcc -S <OCS> -h devices
```

  - Cards library

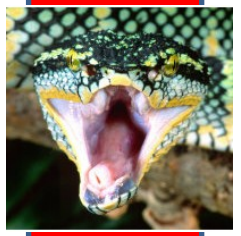
```
ocpidev create worker my_endpoint_proxy.rcc -S <OCS> -h cards
```

  - Platform library

```
ocpidev create worker my_endpoint_proxy.rcc -S <OCS>  -P <platform>
```

- Endpoint Proxy directory contents:
  - my_endpoint_proxy.cc – initial C++ skeleton (must add ACI calls to access slave device proxy properties)
  - my_endpoint_proxy.xml – OWD for the endpoint proxy
  - Makefile

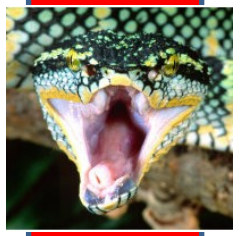
Endpoint Proxy Creation



- OWD XML example

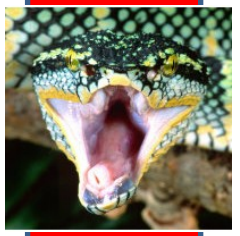
```
<RccWorker language='c++' spec='rx_spec'>
      <SpecProperty Name=”rf_gain_dB” WriteSync=”true”/>
      <Property Name=”SMA_channel” Type=”ushort” Default=”0”/>
</HdlDevice>
```

Device Support Development Process



1) Acquire device datasheet
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

Device Support Development Process



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)**