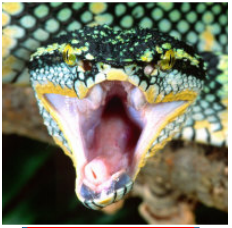


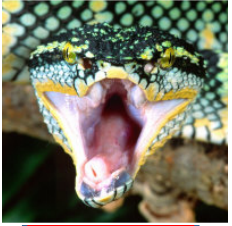
# OpenCPI Concepts

# Overview

- ♦ Terminology
  - ♦ Building blocks
    - ♦ Component, Worker, Application, Assembly etc.
  - ♦ Organizational
    - ♦ Projects, Libraries, etc.
- ♦ Who can develop using OpenCPI?
  - ♦ Three types of developers
    - ♦ Application
    - ♦ Component - *Primary focus of this training*
    - ♦ Platform



# Building Blocks Terminology



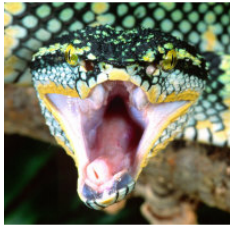
# Building Blocks Terminology: Component

**Term:** Component

**Definition:** A specific function with which to compose applications and a "contract" for workers

**Described by:** OpenCPI Component Specification XML (OCS)

**Example:** FIR Filter



## Features of Components

- Property – run time attribute used to control the component's operation
  - Variable in C/C++, Register in VHDL
  - Parameter – build time property used to control the way the component is built
    - static const in C/C++, generic in VHDL
- Port – an interface used to communicate with other components

## Example

```
<ComponentSpec>  
  <Property Name="taps" ArrayLength="numberOfTaps"/>  
  <Property Name="numberOfTaps" Parameter="true"/>  
  <Port Name="in" Producer="false"/>  
  <Port Name="out" Producer="true"/>  
</ComponentSpec>
```

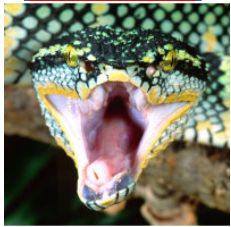
# Building Blocks Terminology: Worker

**Term:** Worker

**Definition:** A concrete implementation of a component

**Described by:** OpenCPI Worker Description XML (OWD), Makefile, build file, source code

**Example:** fir\_filter.rcc, fir\_filter.hdl



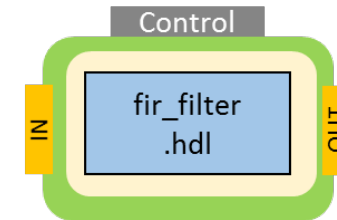
## Features of Workers

- ♦ Stored in directory indicating *authoring model* of worker
  - ♦ <worker>.rcc for C/C++, <worker>.hdl for VHDL
- ♦ Worker files
  - **Makefile** – Includes information for building worker
  - **Worker description XML** – May contain additional properties & port information to expand or refine OCS
  - **-build.xml file** – Declares worker build configurations (optional / future)
  - **Source code**

## Example

One OCS  $\Rightarrow$  Two Workers

1. fir\_filter.hdl
2. fir\_filter.rcc



```
├── fir_filter.hdl
│   ├── fir_filter-build.xml
│   ├── fir_filter.vhd      - Source code
│   ├── fir_filter.xml      - Worker Description XML
│   ├── gen/                - Generated code
│   └── Makefile
├── fir_filter.rcc
│   ├── fir_filter-build.xml
│   ├── fir_filter.cc       - Source code
│   ├── fir_filter.xml      - Worker Description XML
│   ├── gen/                - Generated code
│   └── Makefile
```

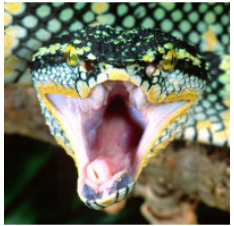
# Building Blocks Terminology: Protocol

**Term:** Protocol

**Definition:** Description of the set of messages that *may* flow between the ports of components

**Described by:** Protocol Specification XML

**Example:** iq\_with\_time-prot.xml



## Features of Protocols

- The protocol is specified in the Port element of an OCS

## Example Port Description

```
<ComponentSpec>

  <Port name="in"  protocol="iq_with_time.xml"/>
  <Port name="out" protocol="iq_with_time.xml"/>

</ComponentSpec>
```

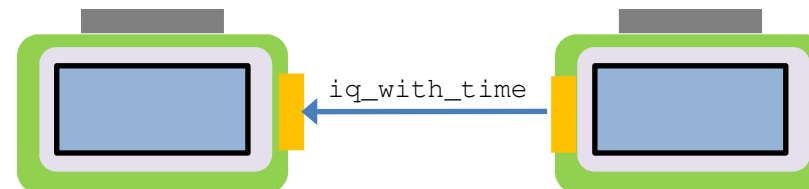
## Example Protocol Specification XML

```
<Protocol Name="iq_with_time">

  <Operation Name="iq" >
    <Argument Name="data" Type="Struct">
      <Member Name="I" Type="Short"/>
      <Member Name="Q" Type="Short"/>
    </Argument>
  </Operation>

  <Operation Name="Time">
    <Argument Name="time" Type="ULongLong"/>
  </Operation>

</Protocol>
```



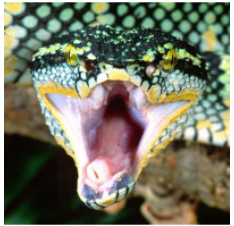
# Building Blocks Terminology: Application

**Term:** Application

**Definition:** Heterogeneous group of connected OpenCPI components

**Described by:** Application Specification XML ("App XML")

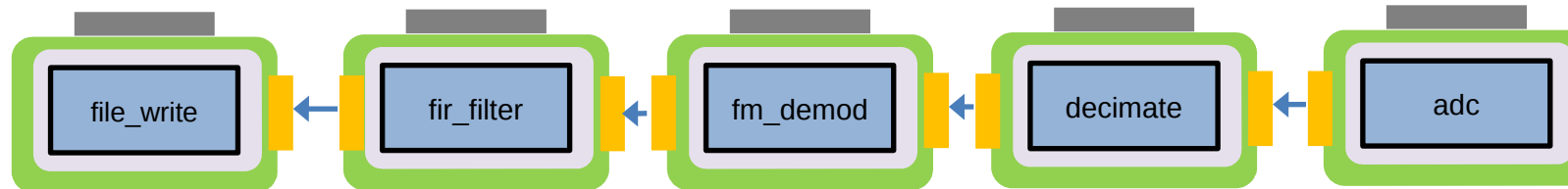
**Example:** FSK Demodulator



## Features of Applications:

- There can be multiple applications per OpenCPI Project
- XML-only applications do not need to be built

## Application



## Application Specification XML

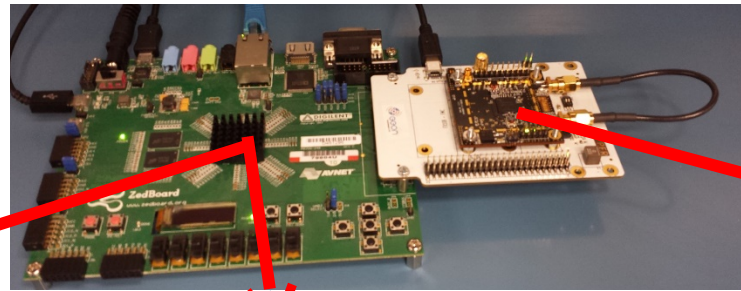
```
<application>
  <instance component='adc'           connect='decimate' />
  <instance component='decimate'      connect='fm_demod' />
  <instance component='fm_demod'      connect='fir_filter' />
  <instance component='fir_filter'    connect='file_write' />
  <instance component='file_write' />
</application>
```

# Building Blocks Terminology: Platform

**Term:** Platform

**Definition:** Physical "motherboard" housing one or more interconnected processors and associated I/O devices

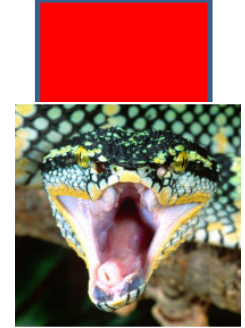
**Example:** ZedBoard with Myriad-RF 1/Zipper Daughtercards



GPP  
Zynq 7020  
ARM

FPGA  
Zynq 7020 Programmable Logic

XCVR  
Lime  
Micro  
LMS6002





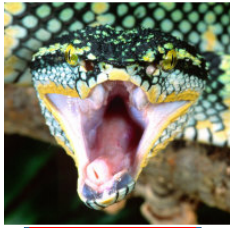
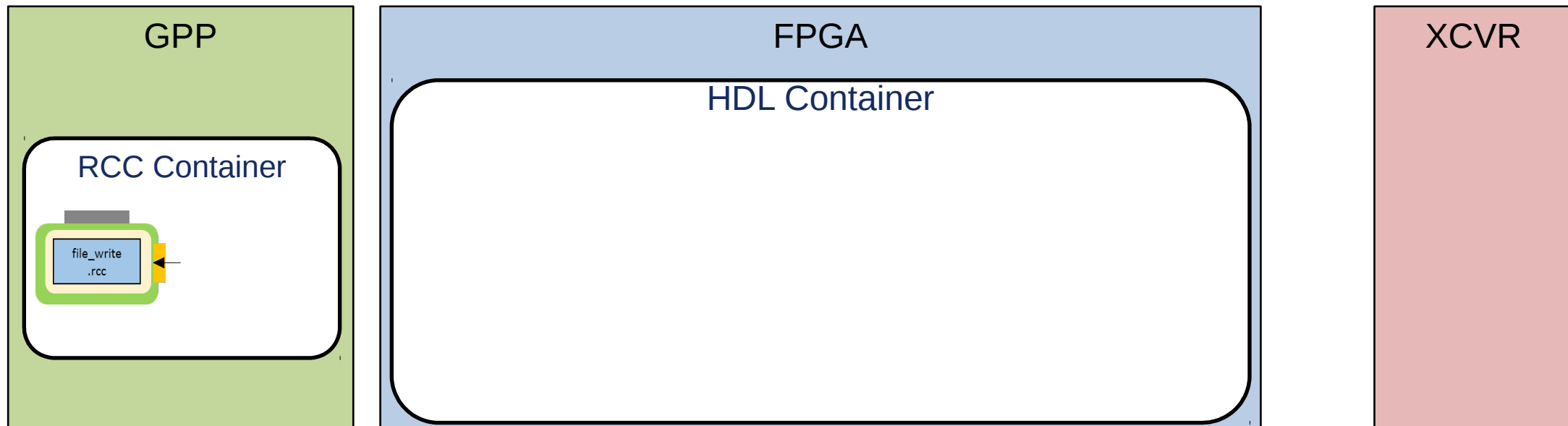
# Building Blocks Terminology: Container

**Term:** Container

**Definition:** the OpenCPI execution environment on some platform that will execute workers (i.e. where they execute)

**Description:**

- In HDL, the container is the complete design for an entire FPGA, including workers and infrastructure. Described by XML. Typically built inside of HDL assembly directories.
- In RCC, the container loads, executes, controls, and moves data to/from RCC workers.



# Building Blocks Terminology: HDL Assembly

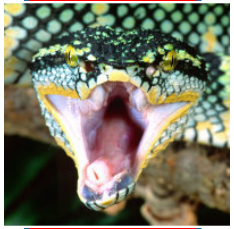
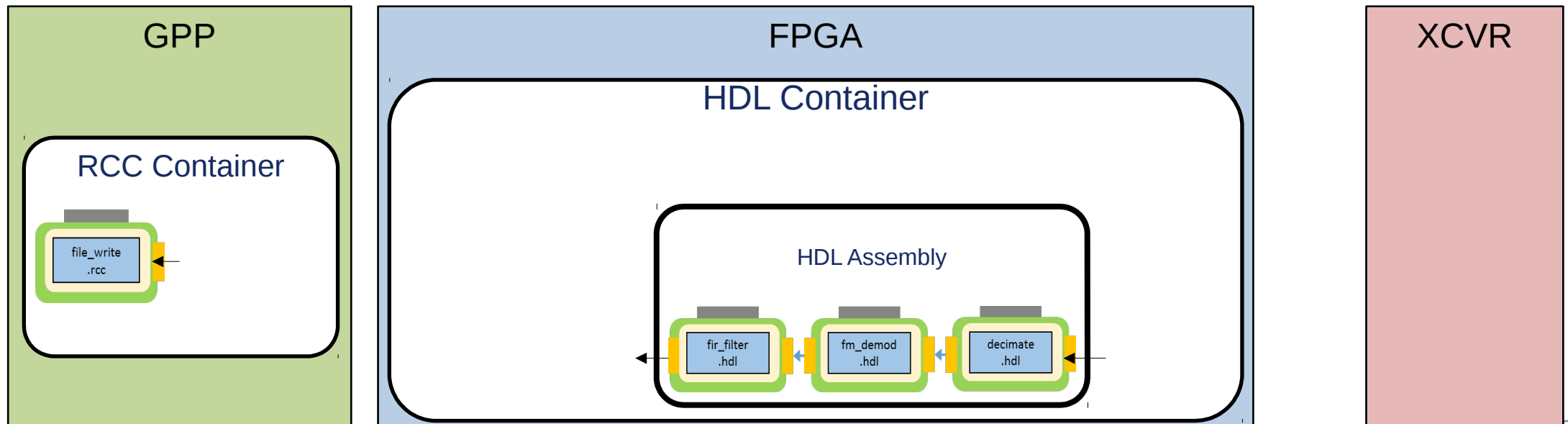
**Term:** HDL Assembly

**Definition:** A fixed composition of HDL workers that can act as subset of a heterogeneous OpenCPI application

**Described by:**

- HDL Assembly Description XML (OHAD)
- **NO VHDL**

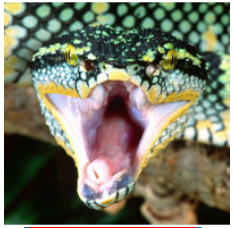
```
<HdlAssembly>
  <Connection name="in" external="consumer">
    <Port instance="decimate" name="in"/>
  </Connection>
  <Instance worker="decimate" connect="fm_demod"/>
  <Instance worker="fm_demod"/>
  <Instance worker="fir_filter"/>
  <Connection name="out" external="producer">
    <Port instance="fir_filter" name="out"/>
  </Connection>
</HdlAssembly>
```



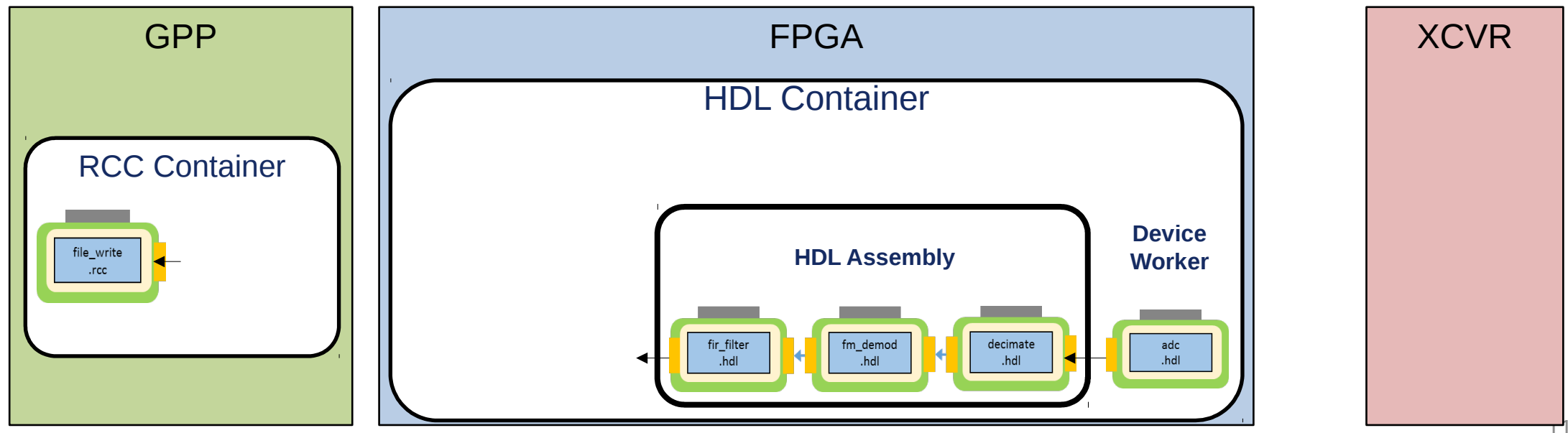
# Building Blocks Terminology: Device Worker

**Term:** Device Worker

**Definition:** Specific type of HDL worker connected to I/O devices external to the FPGA



Open  
CPI



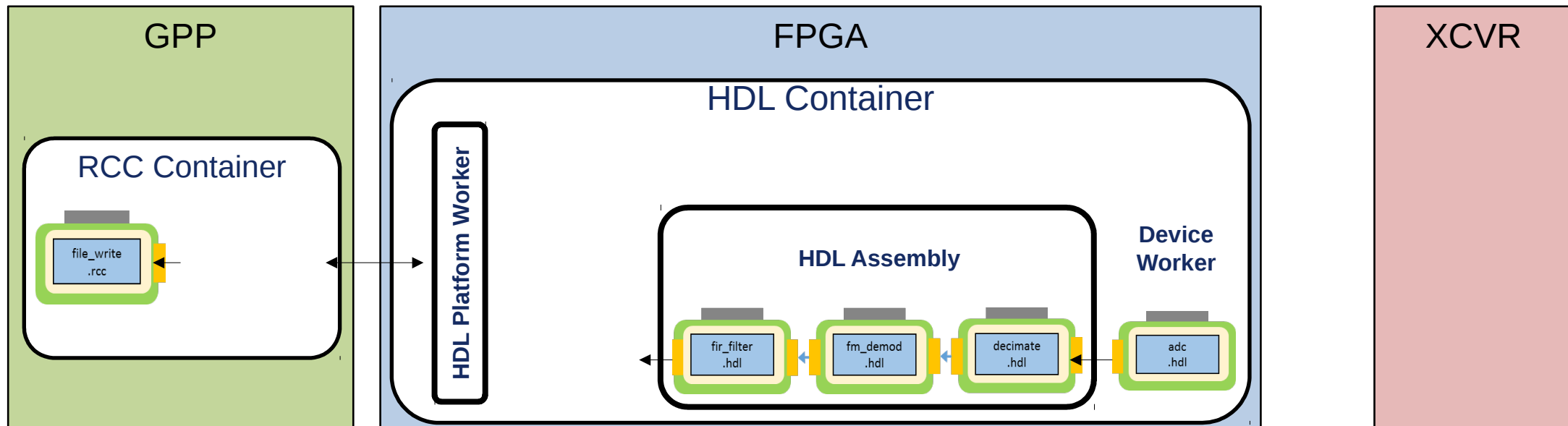
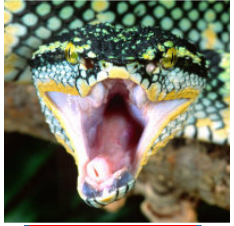
# Building Blocks Terminology: HDL Platform Worker

**Term:** HDL Platform Worker

**Definition:** Platform-specific type of HDL device worker providing infrastructure for implementing control/data interfaces to interconnects external to the FPGA. Somewhat analogous to Board Support Package.

**Described by:** Platform XML *and* HDL source code.

**Example:** matchstiq\_z1, zed, ml605, alst4 (located in hdl/platforms directories)

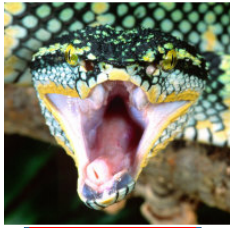


# Building Blocks Terminology: Control Plane

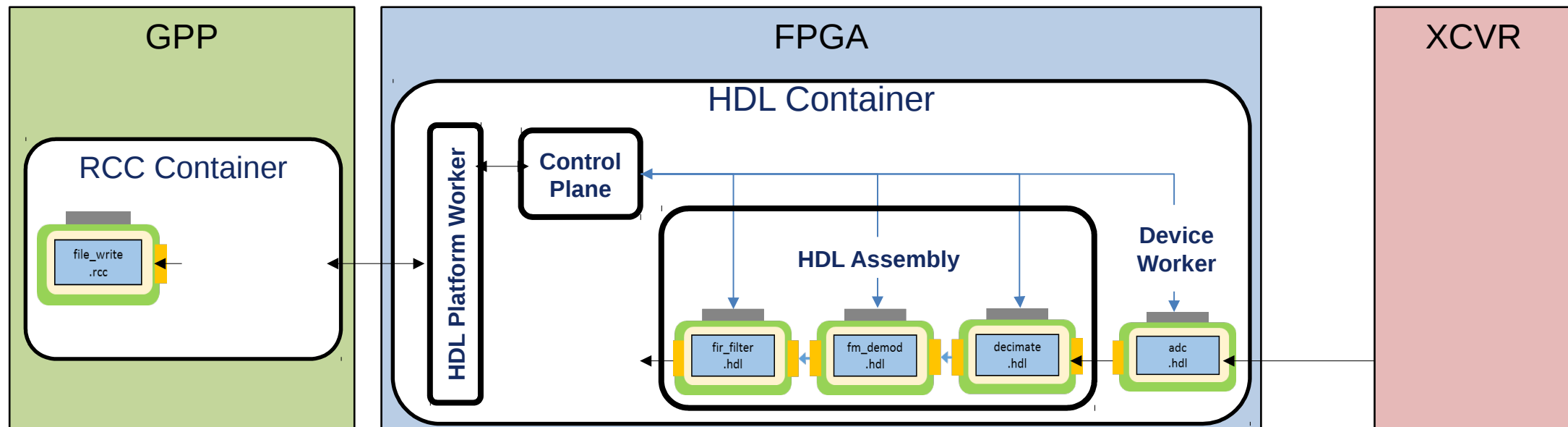
**Term:** Control Plane

**Definition:** Platform-independent HDL module for reading and writing properties of HDL workers

**Described by:** HDL source code



Open  
CPI

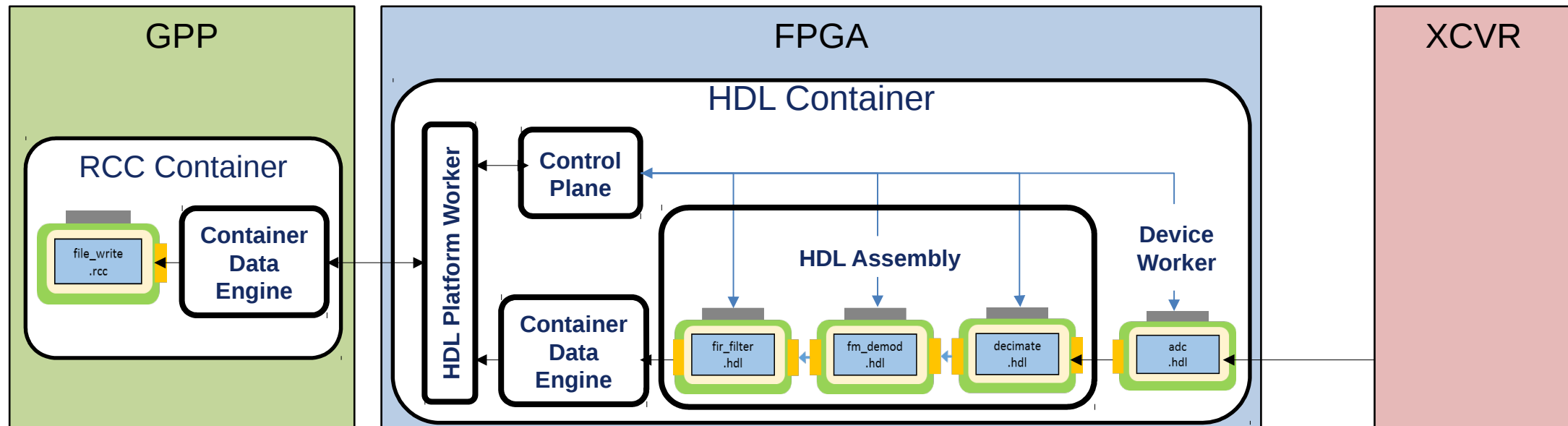
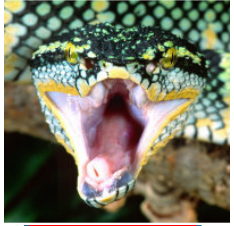


# Building Blocks Terminology: Container Data Engine

**Term:** Container Data Engine

**Definition:** Portable framework module for moving data to/from containers

**Described by:** C++ and HDL source code



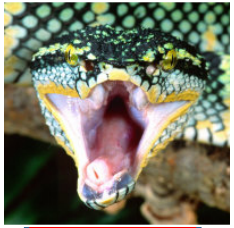
# Building Blocks Terminology: Artifact

**Term:** Artifact

**Definition:** A file containing executable code for one or more workers for a specific platform

**Described by:** XML embedded in the binary

**Example:** file\_write.so, fsk\_demodulator.bitz



## Features of Artifacts

- Output of build process
  - For RCC workers: .so
  - For HDL containers: .bitz
- XML appended to artifact describes everything OpenCPI needs to launch executable
  - For RCC workers: describes worker properties, ports, platform
  - For HDL containers: describes multiple workers and their connections & properties

## Examples

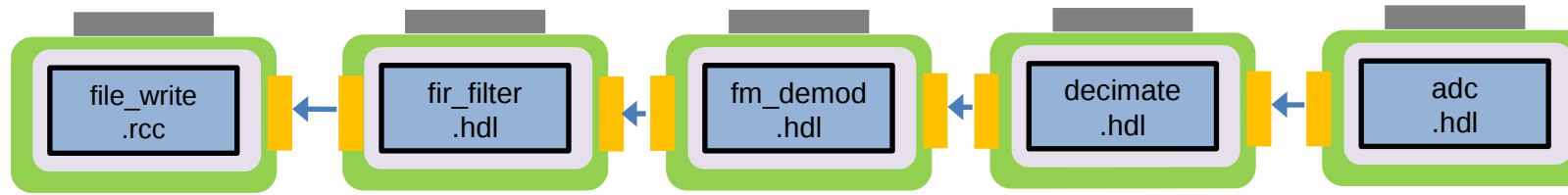
### ▪ RCC worker

```
file_write.rcc/  
├── file_write.c  
├── file_write.xml  
├── gen/  
├── Makefile  
└── target-linux-x13_3-arm/  
    ├── file_write_assy-art.xml - Embedded in .so  
    └── file_write.so - Binary file
```

### ▪ HDL container – multiple workers

```
hdl/assemblies/fsk_demodulator/  
├── container-fsk_demodulator_matchstiq_z1_base  
│   ├── gen/  
│   └── target-zynq/  
│       ├── fsk_demodulator_matchstiq_z1_base-art.xml - Embedded in .bitz  
│       └── fsk_demodulator_matchstiq_z1_base.bitz - Binary file  
├── fsk_demodulator.xml  
├── Makefile  
└── gen/
```

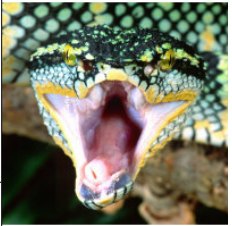
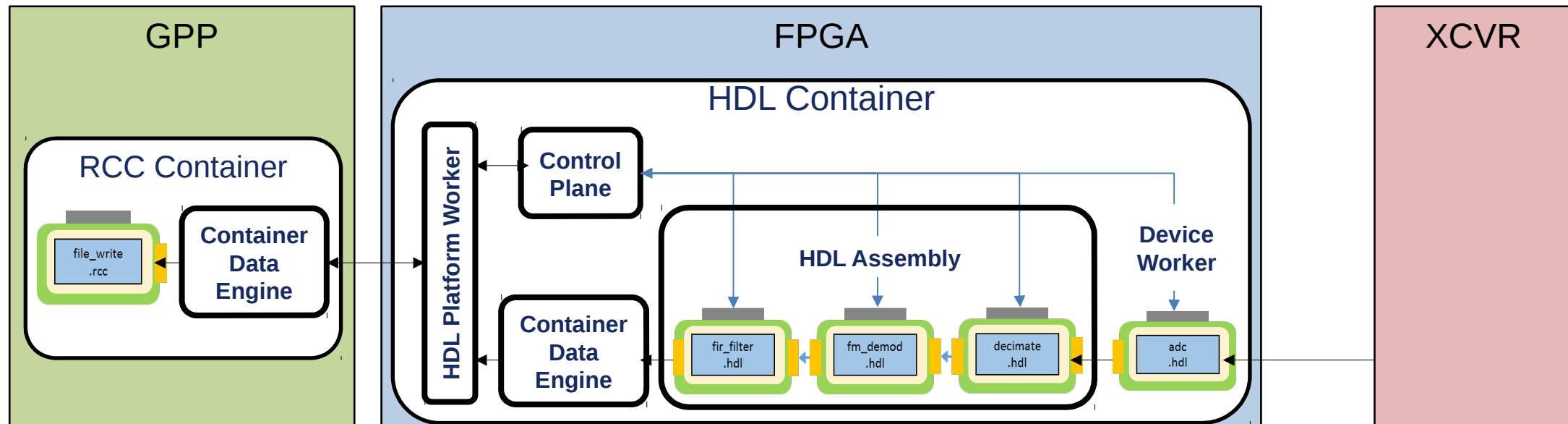
## Application



## Application Specification XML

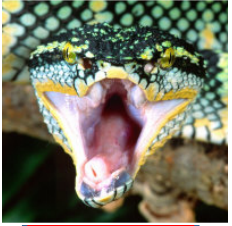
```
<application>
  <instance component='adc'           connect='decimate' />
  <instance component='decimate'       connect='fm_demod' />
  <instance component='fm_demod'       connect='fir_filter' />
  <instance component='fir_filter'     connect='file_write' />
  <instance component='file_write' />
</application>
```

## Deployment of Application – 2 Artifacts (file\_write.so and fsk\_demodulator.bitz)





# Comparison of FPGA Design Flows: Build Time



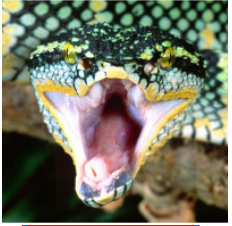
## Typical Design Flow

1. Write and simulate modules to implement algorithm using VHDL and vendor tools
2. Write structural code and glue logic to connect modules using VHDL
3. Write structural code to connect modules to platform interfaces using VHDL
4. Run FPGA vendor tools to create executable code

## OpenCPI Design Flow

1. Write module as OCPI worker using VHDL and OCPI framework (which uses vendor tools)
2. Use IDE (or write XML) to describe workers' connections
3. Use automated build engine to generate structural code and run FPGA vendor tools to create executable code

# Comparison of FPGA Design Flows: Switching between Xilinx and Altera



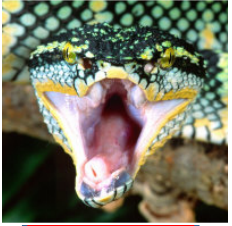
## Typical Design Flow

- Create separate project for Vivado/Quartus
  - Import source files
  - Generate and rebuild any required IP cores
  - Translate project settings and build options
  - Translate constraint files

## OpenCPI Design Flow (using OpenCPI supported platform)

- Run OpenCPI utilities (command-line or IDE) using different arguments

# Organizational Terminology

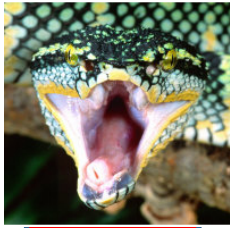


# Organizational Terminology: Project

**Term:** Project

**Definition:** A functionally-related set of Components, Assemblies, Applications, Platforms, etc. in a single location

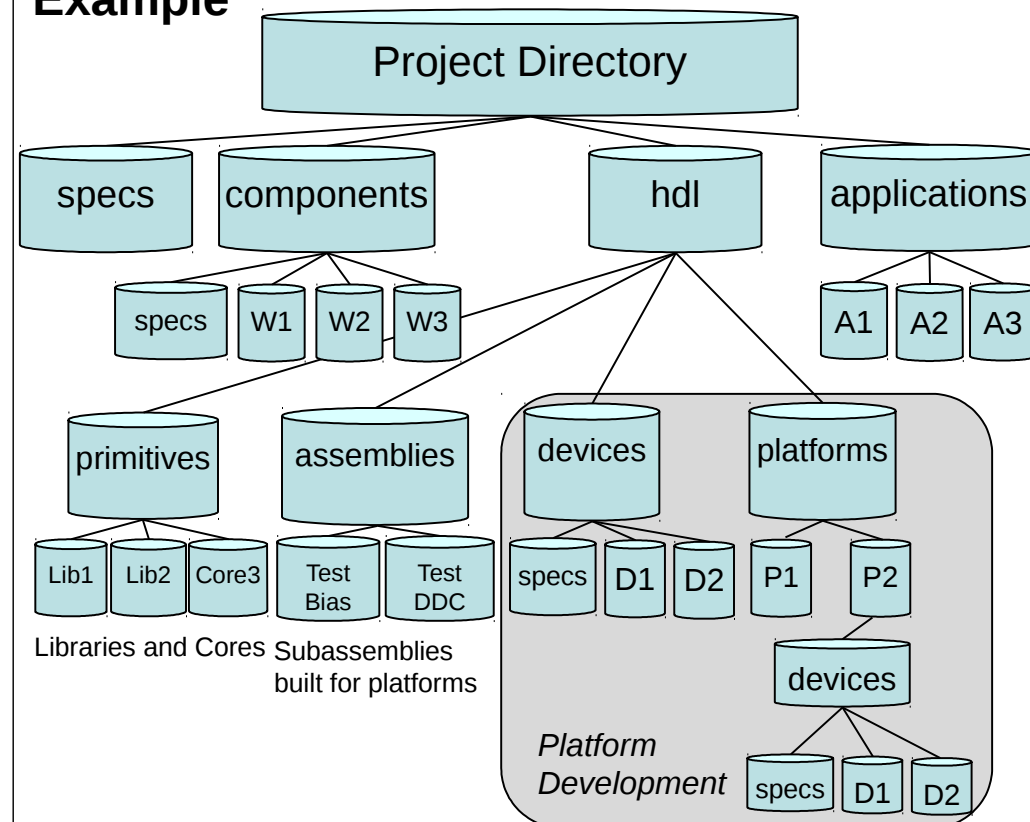
**Described by:** Directory structure and Makefiles



## Features of Projects

- Single directory tree containing all related source and artifacts to solve a specific problem
- Often top-level is stored in revision control SCM
- Provide an organizational hierarchy
- Can refer to other projects, e.g. the Core Project
- Single "exports" directory at top-level to be imported into other projects
- Can contain multiple applications

## Example

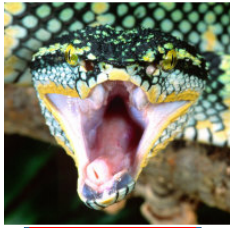


# Organizational Terminology: Namespace

**Term:** Namespace

**Definition:** A sequence of words that are used to organize objects of various kinds so that they may be uniquely referred to

**Describes:** Various "things" within OpenCPI



## Features of Namespaces

- Should start with a unique organizational designator designating the author, *e.g.* the OpenCPI team uses the prefix "ocpi"
- Second term is the Project name
- Used at various levels (not yet all)
- No official registration at this time, but internet domains are fairly unique
- Don't use ocpi for your own assets

## Examples

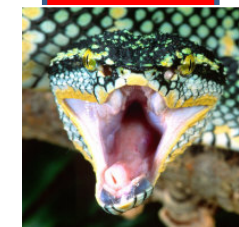
- ocpi.core is the "*Core Project*" provided by "The OpenCPI Team"
- ocpi.assets is the "Assets Project" provided by "The OpenCPI Team"
- ocpi.assets.dsp\_comps.complex\_mixer is the "Complex Mixer" found in the "DSP\_Comms" *Library* within the "Assets Project" provided by "The OpenCPI Team"
- (*Italics* indicate upcoming terms)

# Organizational Terminology: Project Registry

**Term:** Project Registry

**Definition:** A directory that contains references to projects in a development environment

**Described by:** Symbolic links



## Features of Project Registry

- By registering a project, a user is publishing their project so it can be referenced/searched by any user or project using that same project registry
- A project registry can be created, deleted, and updated using `ocpidev`
- Default registry location is `OCPI_CDK_DIR/./project-registry`

## Example

```
/opt/opencpi/project-registry/  
├─ ocp.assets -> /data/opencpi/projects/assets  
├─ ocp.cdk -> ../cdk  
└─ ocp.core -> /data/opencpi/projects/core
```

## To add add/remove project to a registry:

- `ocpidev [register|unregister] project [project]`

## To view registered projects:

- `ocpidev show registry`

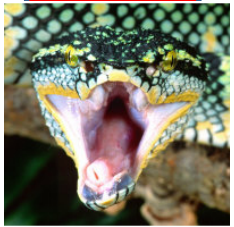
# Organizational Terminology: Library

**Term:** Library

**Definition:** A conceptually-related set of components within a single location

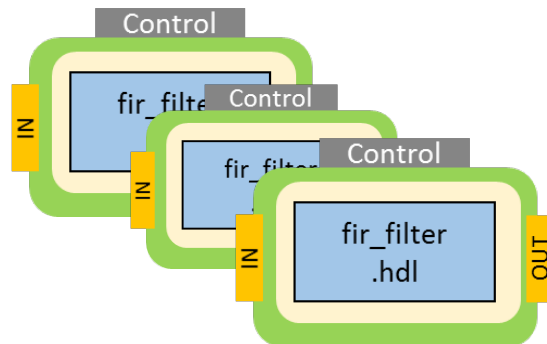
**Described by:** Directory structure and Makefile

**Example:** Utility Components

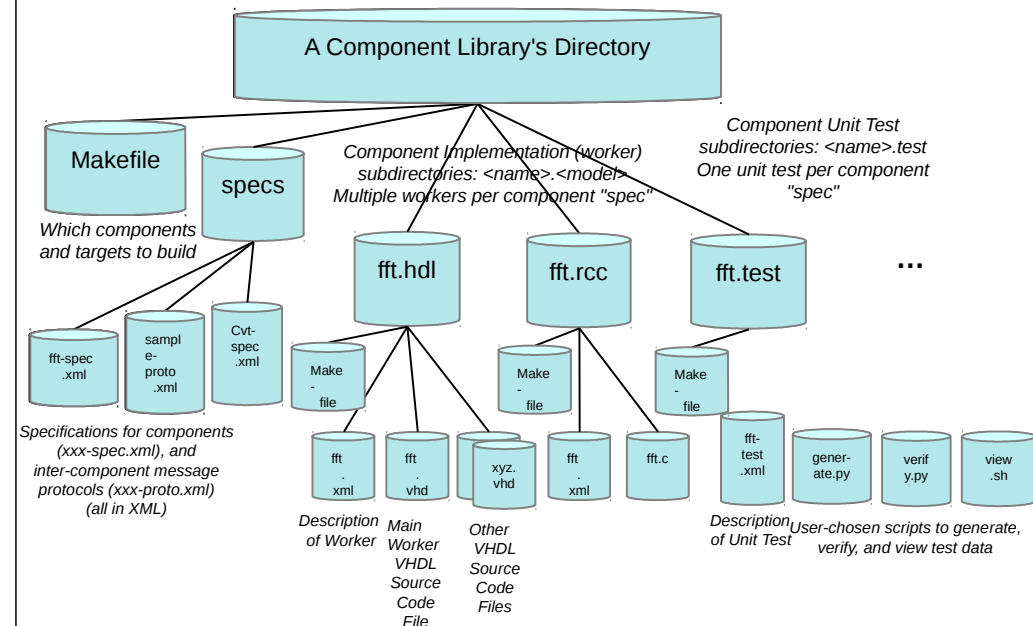


## Features of Libraries

- Provide an organizational hierarchy
- Reduces clutter
- Encourages reusability



## Example

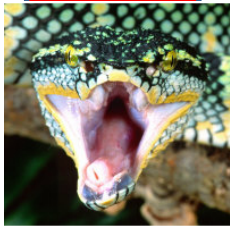


# Organizational Terminology: Core Project

**Term:** Core Project

**Definition:** The minimal set of Components, Assemblies, etc., required for the OpenCPI framework to operate

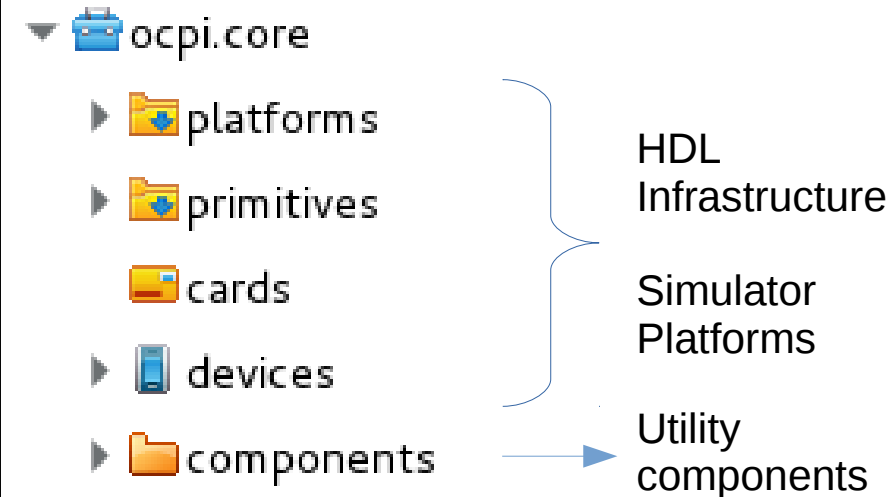
**Described by:** Directory structure and Makefiles



## Features of Core Project

- Provides utility components
  - file read/write capabilities for testing
  - Other unit test capabilities
- Provides all required board support infrastructure HDL for each target platform (e.g. AXI interface on Zed)
- Contains HDL simulation platforms
  - Modelsim
  - xsim
  - isim
- *Always* needs to be compiled for each platform the user wishes to support

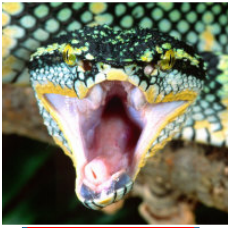
## IDE view of Core Project





# Who can develop using OpenCPI?

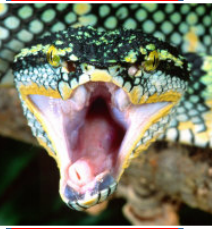
- Three types of developers
  - Application
  - Component: Primary focus of this training
  - Platform



# Application Developer

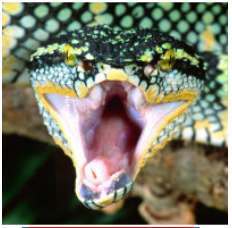
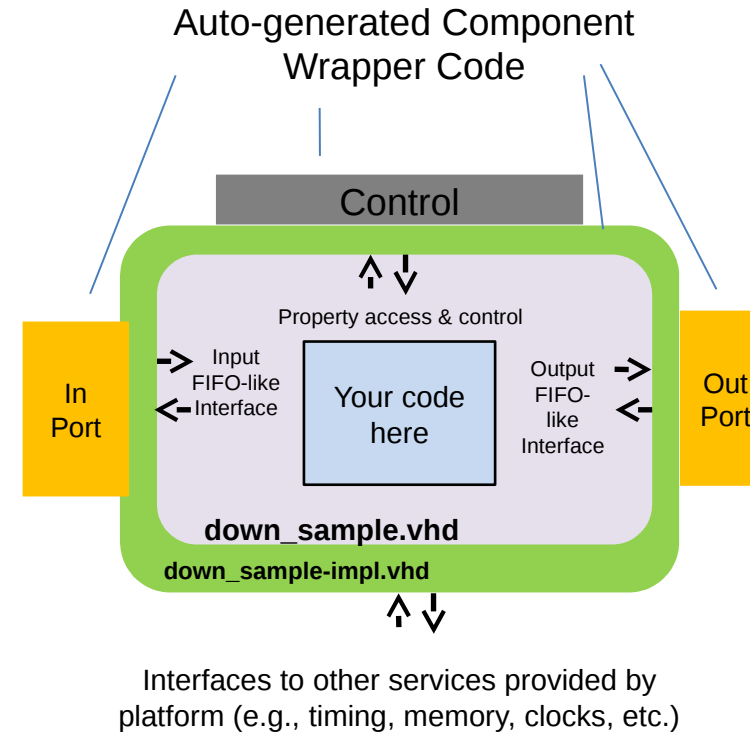
- XML or IDE driven
- Relies on existing libraries
- Requires no C/C++ or VHDL knowledge

```
<application>
  <instance component='adc'           connect='decimate' />
  <instance component='decimate'      connect='fm_demod' />
  <instance component='fm_demod'      connect='fir_filter' />
  <instance component='fir_filter'    connect='file_write' />
  <instance component='file_write' />
</application>
```

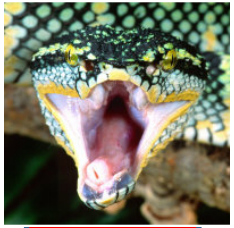


# Component Developer

- Requires C/C++ or VHDL knowledge
- Knowledge of hardware details of target platform is not required



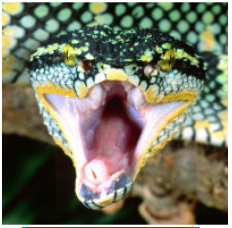
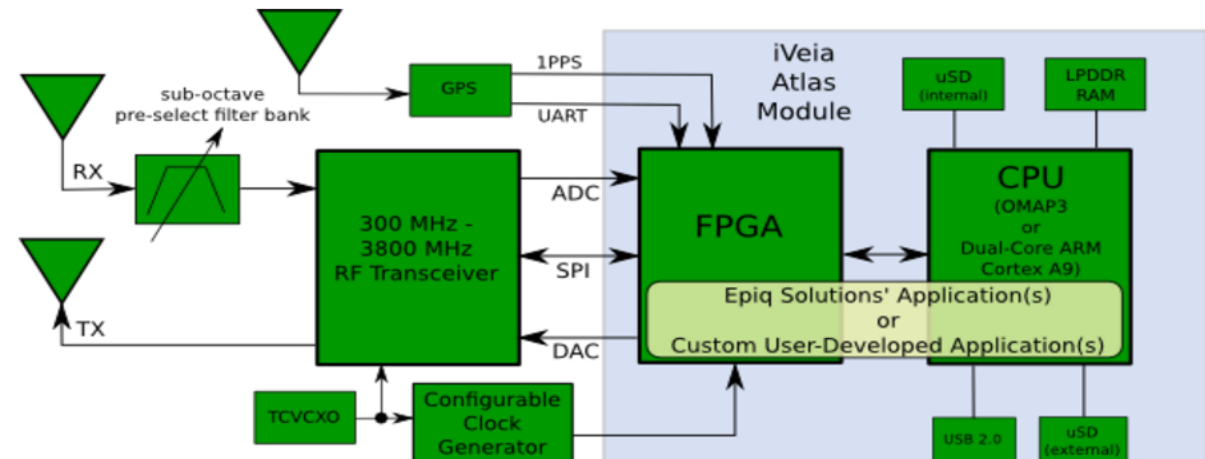
# Component Development Kit (CDK)



What's included?	
<ul style="list-style-type: none"><li>▪ Pre-built artifacts (ready to use) for commonly used software components</li></ul>	<ul style="list-style-type: none"><li>▪ File I/O</li><li>▪ Test infrastructure</li></ul>
<ul style="list-style-type: none"><li>▪ Makefiles required to build software and hardware workers</li></ul>	<ul style="list-style-type: none"><li>▪ Makefiles are used to:<ul style="list-style-type: none"><li>▪ Interpret XML and drive code generators</li><li>▪ Compile software workers</li><li>▪ Run FPGA synthesis software</li></ul></li></ul>
<ul style="list-style-type: none"><li>▪ Utility programs for running and debugging applications</li></ul>	<ul style="list-style-type: none"><li>▪ ocpirun – run applications using application XML</li><li>▪ ocpidev – create projects, libraries, shell code</li><li>▪ ocpihdl –peek/poke HDL container for debug</li><li>▪ ocpixml – examine XML associated with artifacts</li></ul>
<ul style="list-style-type: none"><li>▪ Headers for all C/C++ code to interface to OCPI framework</li></ul>	<ul style="list-style-type: none"><li>▪ Standalone programs</li><li>▪ Provides interface for other frameworks like REDHAWK</li></ul>

# Platform Developer

- Advanced
- Requires in-depth knowledge of platform
  - FPGA pinouts, Interface Control Documents, Schematics



# Summary of OpenCPI Development Roles

3 types of development with common Makefile, XML driven workflow

	Application Development	Component Development	Platform Development
<b>Objective</b>	<ul style="list-style-type: none"><li>• Create applications using components</li></ul>	<ul style="list-style-type: none"><li>• Create building blocks for applications</li></ul>	<ul style="list-style-type: none"><li>• Create infrastructure for running applications</li></ul>
<b>Examples</b>	<ul style="list-style-type: none"><li>• tb_bias</li><li>• FSK app</li></ul>	<ul style="list-style-type: none"><li>• bias</li><li>• FIR filter</li></ul>	<ul style="list-style-type: none"><li>• ZedBoard</li><li>• Matchstiq-Z1</li></ul>
<b>Key functions</b>	<ul style="list-style-type: none"><li>• Declare components and their connections and properties</li></ul>	<ul style="list-style-type: none"><li>• Process data and interface between other components</li><li>• Vendor agnostic (ideally)</li></ul>	<ul style="list-style-type: none"><li>• Provide interface to software and FPGA peripheral (devices workers)</li></ul>
<b>Skills Required</b>	<ul style="list-style-type: none"><li>• Familiarity with component library</li></ul>	<ul style="list-style-type: none"><li>• S/W: C, C++</li><li>• H/W: VHDL</li></ul>	<ul style="list-style-type: none"><li>• H/W: VHDL</li><li>• Strong knowledge of platform architecture and interfaces</li></ul>
	Knowledge of OpenCPI build flow		

