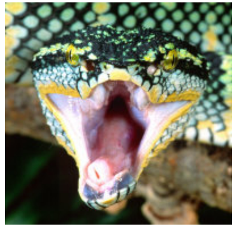
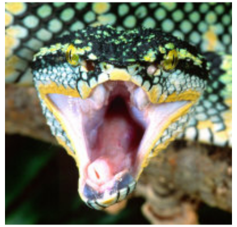


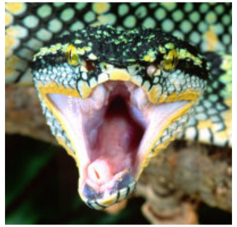
HDL Application Workers

HDL Worker Types



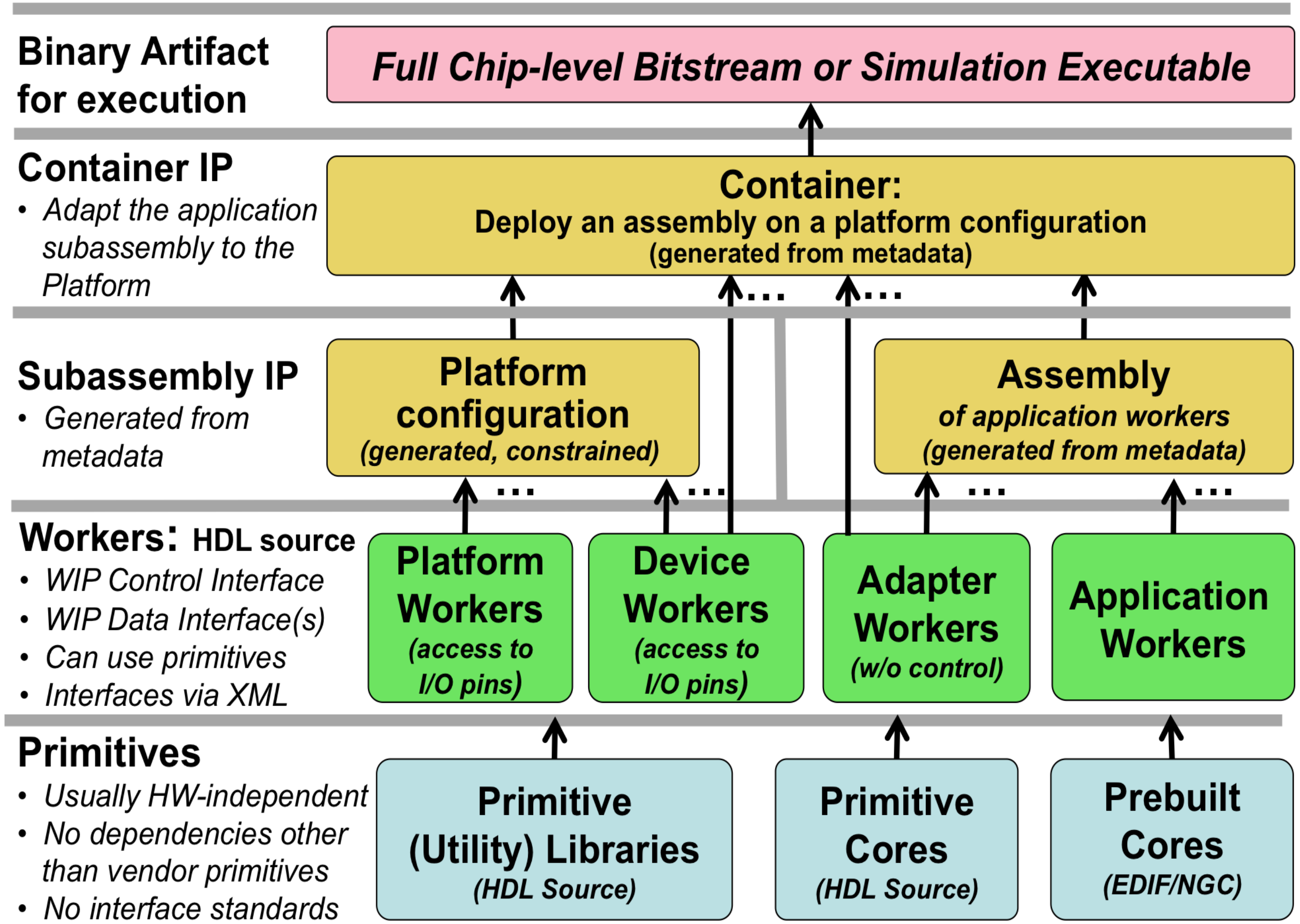
- Application Workers
  - Ideally portable and hardware independent
- Adapter Workers
  - Connects workers with incompatible data ports (*e.g.* protocols), as defined in OCS/OWD
  - No control plane access when OCS/NoControl=”true” (*i.e.* Configuration Properties)
    - Only *wci_clk* and *wci_reset* are provided by control interface
- Device Workers (Subdevices, Emulators)
  - Used to connect to the I/O pins of external hardware
- Platform Workers
  - Special type of Device Worker that performs platform-wide functions

HDL Build Flow Hierarchy



Generated above this point

No user VHDL above this point



Binary Artifact for execution

Full Chip-level Bitstream or Simulation Executable

Container IP

- *Adapt the application subassembly to the Platform*

Container:
Deploy an assembly on a platform configuration
(generated from metadata)

Subassembly IP

- *Generated from metadata*

Platform configuration
(generated, constrained)

Assembly
of application workers
(generated from metadata)

Workers: HDL source

- *WIP Control Interface*
- *WIP Data Interface(s)*
- *Can use primitives*
- *Interfaces via XML*

Platform Workers
(access to I/O pins)

Device Workers
(access to I/O pins)

Adapter Workers
(w/o control)

Application Workers

Primitives

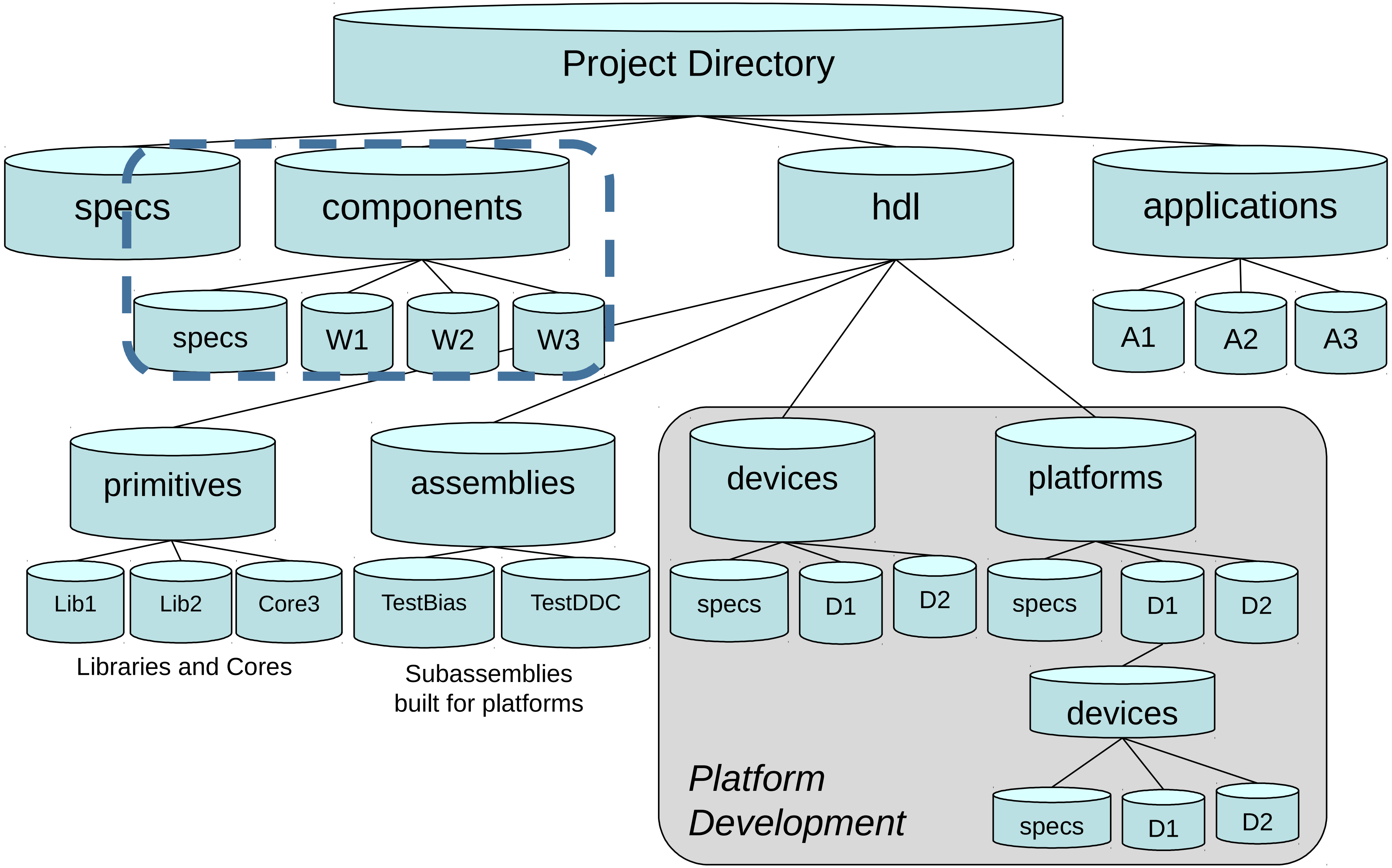
- *Usually HW-independent*
- *No dependencies other than vendor primitives*
- *No interface standards*

Primitive (Utility) Libraries
(HDL Source)

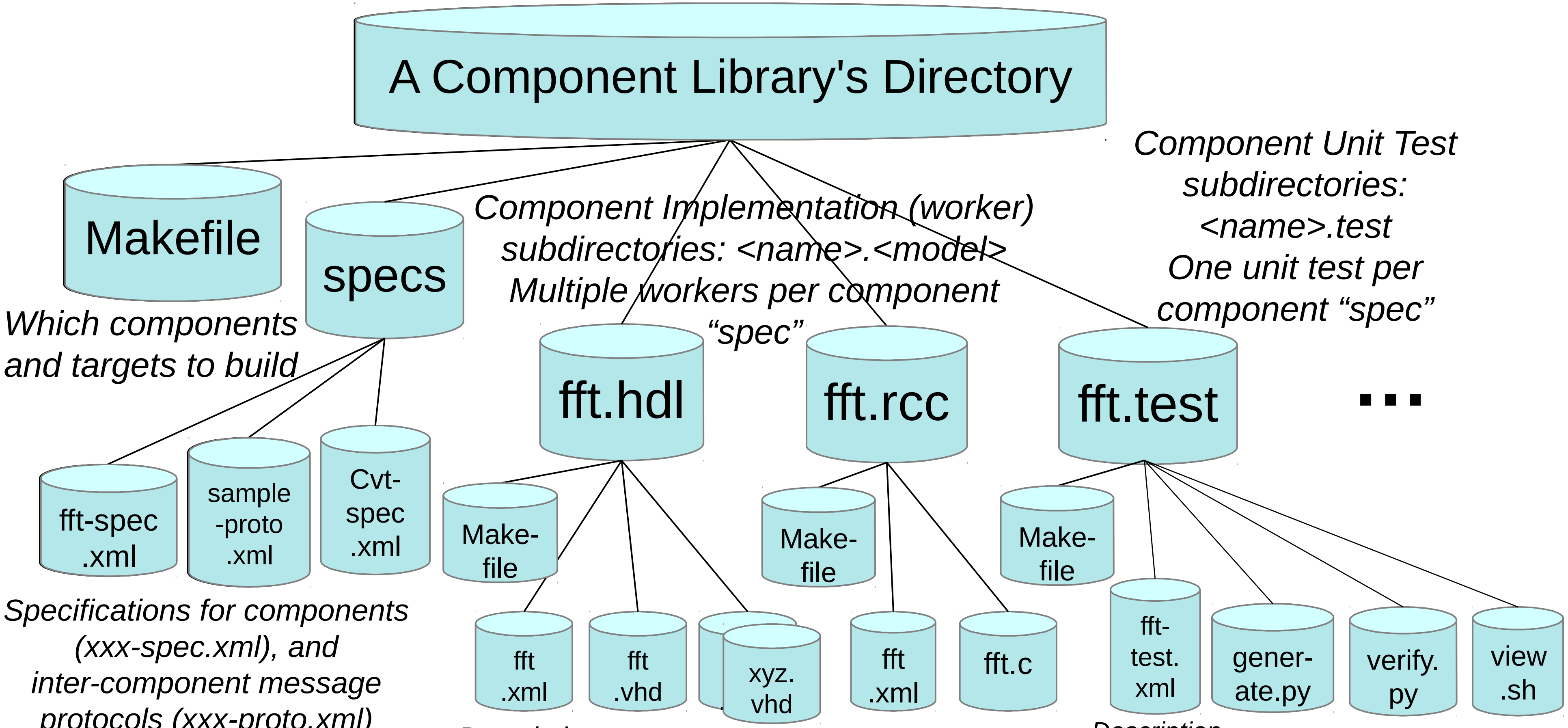
Primitive Cores
(HDL Source)

Prebuilt Cores
(EDIF/NGC)

Project Directory Layout



Component Directory Layout



A Component Library's Directory

Makefile

Which components and targets to build

specs

fft-spec .xml

sample -proto .xml

Cvt- spec .xml

Specifications for components (xxx-spec.xml), and inter-component message protocols (xxx-proto.xml) (all in XML)

Component Implementation (worker) subdirectories: <name>.<model>
Multiple workers per component "spec"

fft.hdl

Make- file

fft .xml

Description of Worker

fft .vhd

Main Worker VHDL Source Code File

xyz. vhd

Other VHDL Source Code Files

fft.rcc

Make- file

fft .xml

fft.c

Component Unit Test subdirectories: <name>.test
One unit test per component "spec"

fft.test

Make- file

fft- test. xml

Description of Unit Test

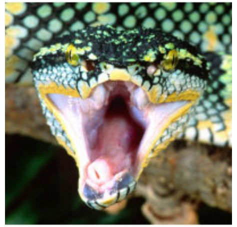
gener- ate.py

verify. py

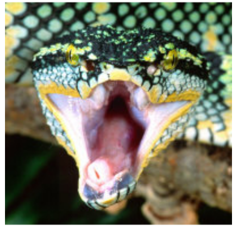
view .sh

User-chosen scripts to generate, verify, and view test data

...

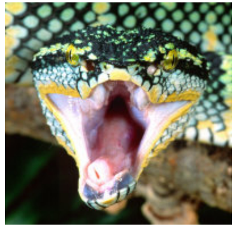


Vendor Tools & Environment Variables



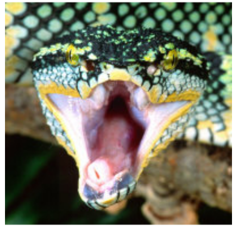
- Xilinx ISE 14.7, Isim
  - OCPI_XILINX_DIR, OCPI_XILINX_VERSION, OCPI_XILINX_TOOLS_DIR
  - OCPI_XILINX_LICENSE_FILE
- Altera Quartus-II 15.1
  - OCPI_ALTERA_DIR, OCPI_ALTERA_VERSION, OCPI_ALTERA_TOOLS_DIR
  - OCPI_ALTERA_LICENSE_FILE
- Mentor ModelSim 10.6a/10.4c **(Mixed-language license REQUIRED)**
  - OCPI_MODELSIM_DIR
  - OCPI_MODELSIM_LICENSE_FILE
- Vivado 2017.1, Xsim **(Vivado 2013.4 SDK is REQUIRED)**
  - Paths default to OCPI_XILINX_DIR and OCPI_XILINX_VERSION
  - OCPI_XILINX_VIVADO_DIR, OCPI_XILINX_VIVADO_VERSION, OCPI_XILINX_VIVADO_TOOLS_DIR, OCPI_XILINX_VIVADO_LICENSE_FILE
- **Default tool installation locations work for most**
- **MUST define license file location(s)**
- $ env | grep OCPI | sort

HDL Application Workers

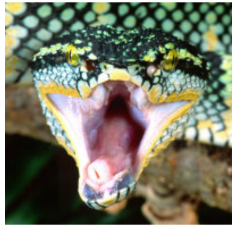


- Implement an OCS
- Described by an OWD, written in VHDL
- Intended to be hardware/platform agnostic
  - Ability to wrap vendor-specific IP to achieve vendor-neutral code
- Can be built for multiple FPGA/simulation targets
- Typically, organized in a component library
- Cannot instantiate other workers (no circular dependencies)
- Connected together in an OHAD to form an HDL Assembly

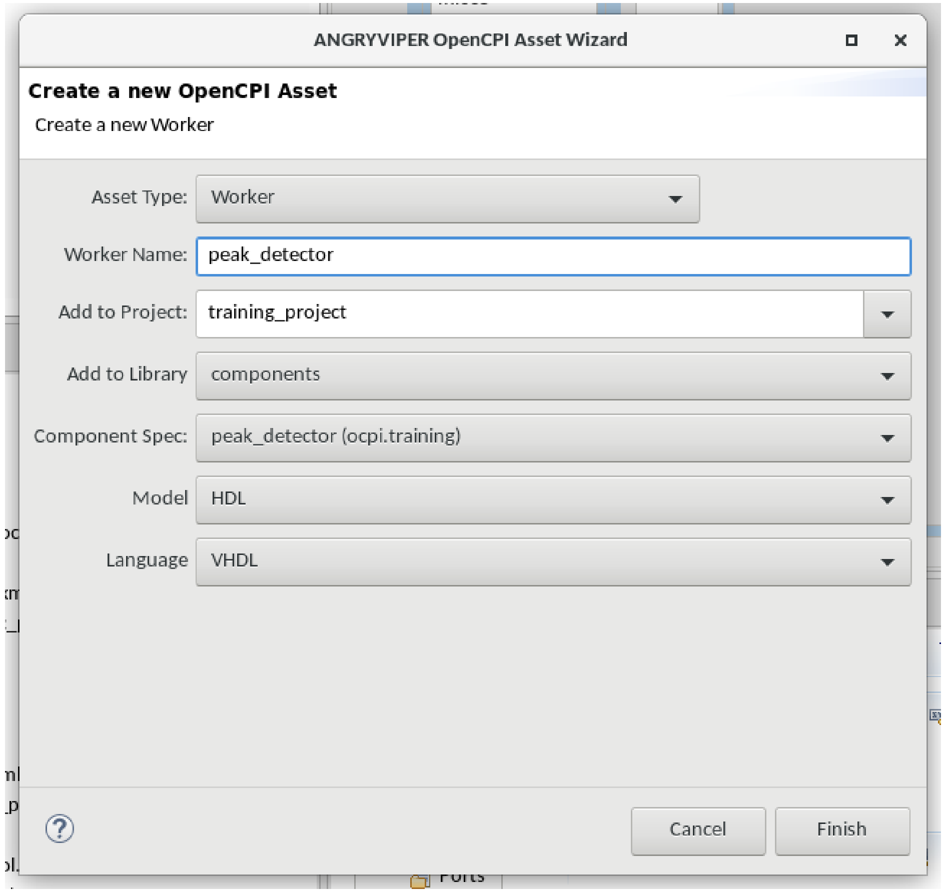
App Worker Development Flow



1) OPS: Use pre-existing or create new
2) OCS: Use pre-existing or create new
3) **Create new App Worker (Modify OWD, Makefile, and source HDL/RCC code)**
4) **Build the App Worker for target device(s)**
5) Create Unit Test ({component}-test.xml, generate, verify and view scripts)
6) Build Unit Test
7) Run Unit Test



Creating an HDL Application Worker



ANGRYVIPER OpenCPI Asset Wizard

Create a new OpenCPI Asset

Create a new Worker

Asset Type: Worker

Worker Name: peak_detector

Add to Project: training_project

Add to Library: components

Component Spec: peak_detector (ocpi.training)

Model: HDL

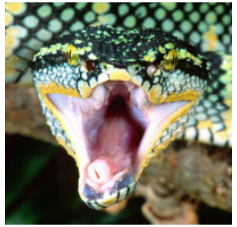
Language: VHDL

Key files:

```
peak_detector.hdl/
|-- gen
|   |-- peak_detector-build.xml     # Initial build config
|   |-- peak_detector-defs.vhd      # VHDL package defs
|   |-- peak_detector-impl.vhd      # Generated Shell & Entity
|   |-- peak_detector.mk            # Parameter definitions
|   `-- peak_detector-skel.vhd      # Initial Source "skeleton"
|-- Makefile                    # a.k.a "Worker" Makefile
|-- peak_detector.vhd           # Architecture (user code here)
`-- peak_detector.xml           # OpenCPI Worker Description (OWD)
```

Create via `ocpidev` utility:
ocpidev create worker <name>.hdl -S <OCS>

HDL Worker's generated VHDL



- Shell, Entity, and Package of records: *worker.hdl/gen/worker-impl.vhd*
  - Generated by framework based on OCS and OWD (Ports and Properties/Parameters)
  - Outward facing interfaces (subset of Open Core Protocol (OCP)) are "Normalized" to easily connect to other Workers in an HDL Assembly
  - Inward facing interfaces are available as VHDL records
    - Control Ports: *ctl_in.*, ctl_out.**
    - Configuration Properties: *props_in.*, props_out.**
    - Data Ports (I/O): *in_in.*, out_out.**
    - Service Ports (time): *time_in.*, time_out.**

      Port names above are the **defaults**, framework does supports ability to rename all ports
- Architecture: *worker.hdl/worker.vhd*
  - Upon initial creation of HDL workers, worker.hdl/gen/*worker-skel.vhd* is automatically copied and renamed worker.hdl/*worker.vhd*
  - "Business Logic" goes here

HDL Application “Worker” Makefile

- Generated by framework
- Typically not manually edited
- Minimally contains

```
include $(OCPI_CDK_DIR)/include/worker.mk
```

- Build Worker from <worker>.hdl/

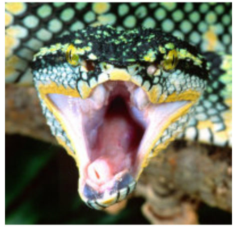
```
ocpidev build --hdl-target [target]
```

- Build worker from Parent library/

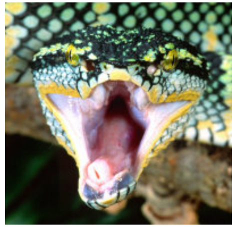
```
ocpidev build library [libname] --worker
[worker] --hdl-target [target]
```

- Subdirectories created with build artifacts
  - <worker>.hdl/target-[target]/

Variable in HDL Worker Makefile	Override / augment Component Library Makefile?	
SourceFiles	N	A list of additional source files for this worker (VHDL or Verilog)
Libraries	Y	A list of primitive libraries built elsewhere. If a name has no slashes, it will follow the HDL Search Path rules
OnlyTargets	Y	A list of targets for which this worker should be built
ExcludeTargets	Y	A list of targets for which this worker should NOT be built
XmlIncludeDirs	Y	A list of directories elsewhere for searching for xml files included from the OWD (in addition to the ../specs directory in the component library containing this worker)
Worker	N	Name of worker; the default is from the directory name
Cores	N	A list of HDL primitive cores built elsewhere
VerilogIncludeDirs	Y	Searchable directories for Verilog include files, in addition to the worker directory
HdlExactPart	N	A variable to override the default part within a family specified by HdlTarget(s)

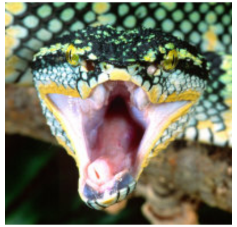


HDL Worker Description OWD XML



- Primary reasons to modify the OWD:
  - <u>ADD</u> implementation-specific configuration properties by using the “property” element
  - Increase accessibility to existing OCS properties
    - Use the “specproperty” element to <u>ADD</u> Readable, Writable, Volatile, Initial
      - **MUST** follow configuration property rules
      - “Readable” is special / advanced case
    - ***CANNOT*** <u>REMOVE</u> accessibility defined in the OCS. **Cannot break the “contract”!**
  - Limit properties to have **build-time** values with the **Parameter=”true”** attribute
    - May set a <u>default</u> value for Parameter or set in the <worker>-build.xml (generics/constants for HDL; const for RCC)
    - Specify an OCS property is a Parameter for the Worker
      - Applies only to OCS *Initial* properties (using specproperty) or OWD properties
  - Specify interface style and implementation attributes for Ports
    - stream interface, data path width, message abort support, etc.

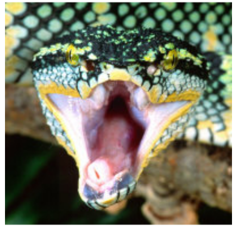
HDL OWD Top-level Attributes



- Specify which ControlOperations that worker will implement
  - *none* are required for HDL workers
- In addition to those attributes defined for all OWDs, an HDL Worker's OWD may configure multiple additional attributes

HDL OWD Attribute	Data Type	
DataWidth	unsigned	Default physical width of **ALL** data ports of worker. Otherwise, based on protocol of the port.
RawProperties	string	Indicates if worker will use raw property interface
FirstRawProperty	string	Indicates the name of the first property that requires the raw property interface.

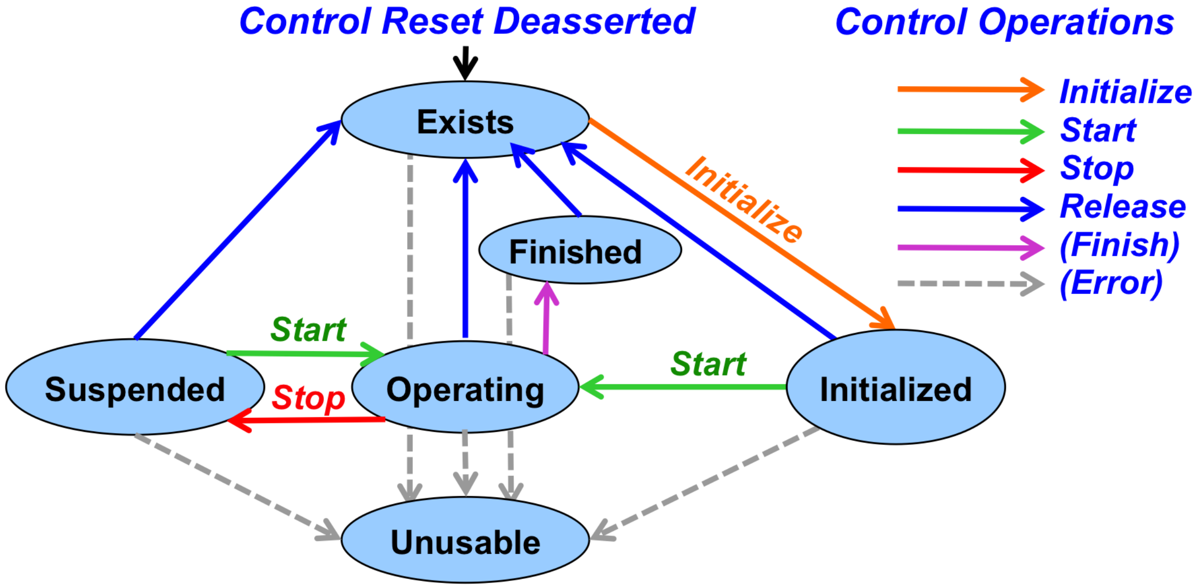
HDL Worker Control Interfaces



- Conveys Life-Cycle like **initialize**, **start**, or **stop**
  - Required when top-level **ControlOperations=”initialize, start, stop”** in OWD
- Provides a Control Clock *ctl_in.clk* and associated synchronous reset *ctl_in.reset*
  - Reset will be asserted synchronously for *at least* 16 clock cycles
  - Available to all types of Workers (including Adapters: OCS/NoControl=”true”)
- VHDL uses record ports *ctl_in* and *ctl_out* (default names)
  - Input: clk, reset, control_op, state, is_operating, abort_control_op, is_big_endian
  - Output: done, error, finished
- Optionally use *ctl_out.done* and *ctl_out.error* when control operations or property accesses take more than one clock cycle, where done is an indication to change state
- Optionally set *ctl_out.finished* if the worker has some semantic of being finished, *i.e.* transition the worker to the finished life-cycle state

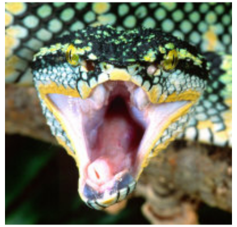
HDL Worker Control Interfaces

- Typically, HDL Workers have no need to implement any of the Control Operations
- Enters ***Exists*** State upon deassert of Worker's *ctl_in.reset*
- If ***Initialize*** Control Operation not implemented, then Worker is in ***Initialized*** State
- **MUST NOT** perform any data transactions with Data Ports unless *ctl_in.is_operating* is asserted
  - Indicates Worker has been started and provides Control Software ability to suspend or resume all or parts of an Application
  - ***Operating*** State
- ***Unusable*** state entered when Control Ops fail
- ***Finished*** state declared by worker without any Control Op



(Error): fatal error from control operation, other transitions based on success: non-fatal errors do not change states.
(Finish): is self initiated, not controlled from outside

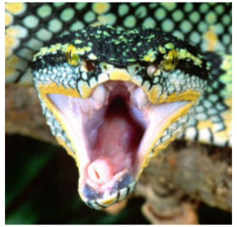
HDL Worker Property Interfaces



- VHDL records props_in and props_out
- Types defined in package ocpi.types
  - uchar_t, char_t, ushort_t, short_t, ulong_t, long_t, ulonglong_t, longlong_t
  - float_t, double_t, string_t, bool_t, enum foo_t
- Available signals depend on Type of properties declared in OCS and OWD
  - Scalar vs SequenceLength

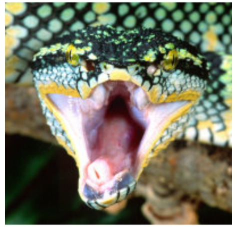
Property Access Options	VHDL signal names	Accessible when
props_in.	foo	Initial or Writable
	foo_length	Initial or Writable when SequenceLength is defined
	foo_written	Writable
	foo_any_written	Writable and SequenceLength is defined
	foo_read	Volatile
props_out.	foo	Volatile
	foo_length	Volatile when SequenceLength is defined

HDL Worker Property Behavior



- Writable Properties
  - Registered in the shell
  - Available on “props_in” port record
  - If not **Initial**, then **Writable** at run-time and given “props_in.<prop>_written” pulse
- Writable Arrays/Sequences
  - “props_in.<prop>_written” pulse only happens when value is *completely* updated
  - “props_in.<prop>_any_written” is pulsed when any part is written
- Readable Properties
  - If not **Volatile**, shell handles read-back
  - If **Volatile**, “props_out.<prop>” must be driven by “Business Logic” worker code

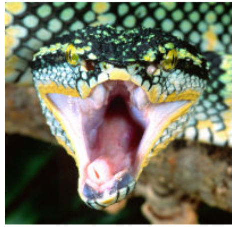
HDL Worker Raw Property Interfaces



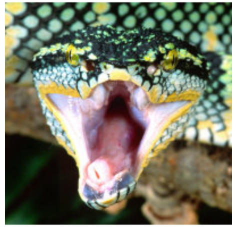
- Raw Property Interface
  - Used when the worker manages storage and addressing of property values
  - May avoid register duplication in both the outer shell and inner worker
  - Enabled by setting rawProperties (all) or firstRawProperty (named and later) attributes in the OWD
  - Often used when interfacing FPGA-external devices with memory mapped interfaces

Raw Property Access Options	**VHDL signal names**
Input	raw.address
	raw.byte_enable
	raw.is_read
	raw.is_write
	raw.data
Output	raw.data
	raw.done
	raw.error

HDL Worker Data Interfaces



- Convey message with an associated **opcode** which indicates message types that are defined in the **protocol**
  - Exception: When **protocol** contains ONE message type, NO **opcode** is used
- Convey message boundaries between messages
  - Opcode, Start-of-Message, End-of-Message, Length
- Implement flow control
  - Input data is explicitly accepted when offered from upstream
  - Output data cannot be produced unless permission is granted from downstream
- Special use case is when message is zero width
  - ALL messages in protocol have no Arguments, thus are ALL zero length
  - Therefore, message **opcodes** are all that is conveyed. Essentially a “pulse” or “event” interface.



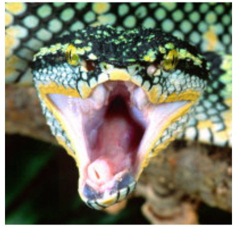
HDL Worker Data Interfaces

- Streaming Data Interfaces
  - FIFO-like interface with extra meta-bits for message boundaries and byte enables
    - Metadata includes **SOM, EOM, Valid, Abort** (optional)**, Byte_Enable** (optional)
  - Buffers consist of 1 or 2 pipeline registers with flow control
  - Output cannot be produced (**give**) unless permission is granted (output is **ready**; not full)
  - Worker explicitly accepts (**take**) data at input interfaces only when input is **ready** (not empty)

StreamInterface Attribute	Data Type	Description
DataWidth	unsigned	The width of the data path for this interface. The default is the smallest element in the message protocol indicated in the OCS, unless overridden by a default **datawidth** attribute at the top level of this OWD (**HdlWorker**)

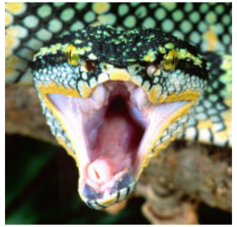
- **HDL Workers must**
  - **Respect backpressure**
  - **Convey message boundaries**

Message Payload vs Physical Data Width



- Each Message Payload has a serialized format as a sequence of bytes that, when used in software, are laid out in byte-addressed memory
- HDL Worker Data Interfaces have a Physical Width
  - Indicates number of physical wires over which the message is conveyed
  - Overridden by **DataWidth** attribute in the top-level of HDL OWD or per Port
  - **MUST** be a multiple of the smallest data value in the protocol **(DEFAULT is 1x)**
- If the Operation element in a protocol contains:
  <Argument Name='a1' Type='uchar'/> <!-- **SMALLEST DATA VALUE** -->
  <Argument Name='a2' Type='uShort' ArrayDimensions='2'/>
  <Argument Name='a3' Type='ulongLong'/>
- And the values of this payload are:
  a1: 1, *a2*: {0x2345,0x6789}, *a3*: 0xfedcba9876543210

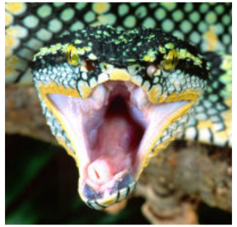
HDL Worker Data Interfaces:
Message Payload vs Physical Data Width



- Then the byte sequence, with proper alignment and encoded little-endian, is shown below (“x” values are padding for alignment)
- DataWidth=8 (DEFAULT because *a1* is of type 'uchar')

Sequence # ⇨	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Contents(hex)	01	x	45	23	89	67	x	x	10	32	54	76	98	ba	dc	fe
Arguments	a1		a2[0]		a2[1]				a3							
Contents	1		0x2345		0x6789				0xfedcba9876543210							

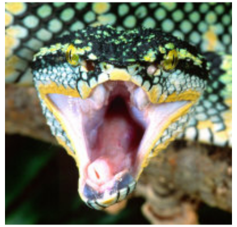
HDL Worker Data Interfaces:
Message Payload vs Physical Data Width



- Then the byte sequence, with proper alignment and encoded little-endian, is shown below (“x” values are padding for alignment)
- DataWidth=16

Sequence # ⇨	0	1	2	3	4	5	6	7
15 downto 8	x	23	67	x	32	76	ba	fe
7 downto 0	01	45	89	x	10	54	98	dc

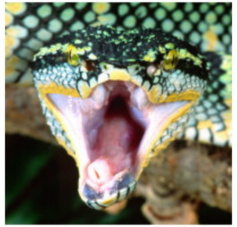
HDL Worker Data Interfaces:
Message Payload vs Physical Data Width



- Then the byte sequence, with proper alignment and encoded little-endian, is shown below (“x” values are padding for alignment)
- DataWidth=32

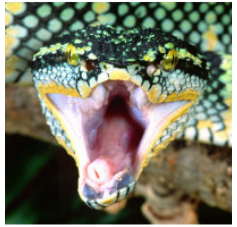
Sequence # ⇨	0	1	2	3
31 downto 24	23	x	76	fe
23 downto 16	45	x	54	dc
15 downto 8	x	67	32	ba
7 downto 0	01	89	10	98

HDL Worker Data Interfaces: byte_enable

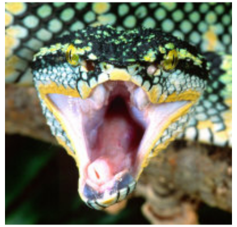


- Byte Enables on data interfaces are only present when needed
  - Determined by combination of protocol (smallest data value) and DataWidth
- Two values are inferred (can be overriden) from protocol
  - **DataValueWidth**: smallest data value in protocol
  - **DataValueGranularity**: least common multiple of data values among all messages in protocol; all message lengths are a multiple of this number of data values
- Therefore, the physical data width of the interface (DataWidth) must be a multiple of DataValueWidth
  - **DataWidth > DataValueWidth * DataValueGranularity**
    - byte enables are provided in the interface, because data words at the **end** of a message may be **partially** valid

HDL Worker Data Interfaces: byte_enable



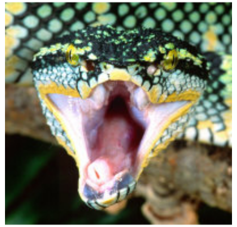
- Message is a sequence of short (16 bit) values, DataWidth is 16
  - DataValueWidth = 16
  - DataValueGranularity = 1
  - No byte enables required
- Message is a sequence of short (16 bit) values, DataWidth is 32
  - DataValueWidth = 16
  - DataValueGranularity = 1
  - Byte enables (2) are required since sequences might be an odd number of shorts
- Message is a sequence of pairs of short (16 bit) values, DataWidth is 32
  - DataValueWidth = 16
  - DataValueGranularity = 2
  - Byte enables not required since sequences are always a multiple of two shorts

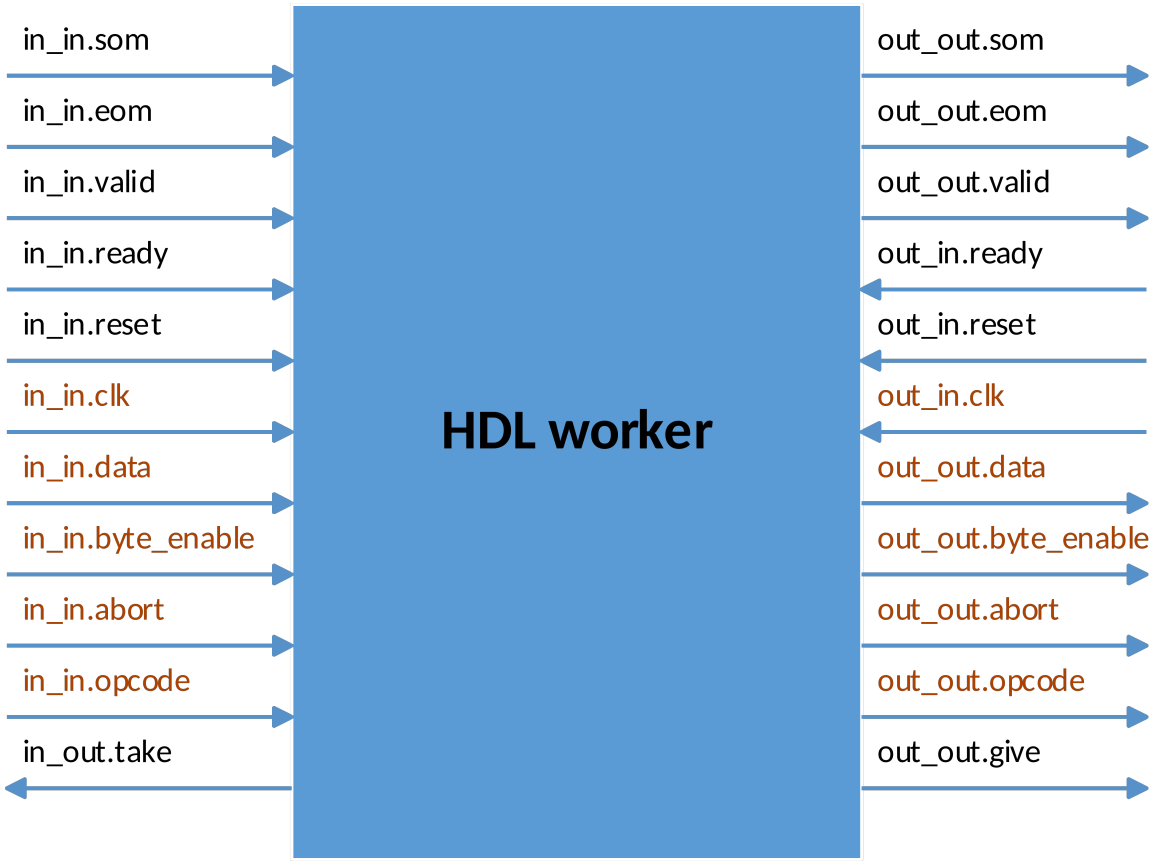


HDL Streaming Data Signals

- **som**: Start of message: indicates first word in message, *regardless of data present*
- **eom**: End of message: indicates last word in message, *regardless of data present*
- **valid**: indicates the validity of data in a message
- **abort** (optional): indicates this word is the end of an abort message
- **byte_enable** (optional): combined with **valid=true**, indicates which bytes in a data word are valid
  - All 1s except on the last **valid** word of a message

HDL Streaming Data Signals





HDL worker

in_in.som
in_in.eom
in_in.valid
in_in.ready
in_in.reset
in_in.clk
in_in.data
in_in.byte_enable
in_in.abort
in_in.opcode
in_out.take

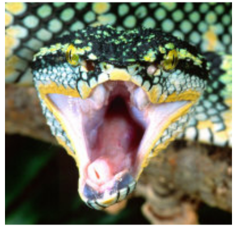
out_out.som
out_out.eom
out_out.valid
out_in.ready
out_in.reset
out_in.clk
out_out.data
out_out.byte_enable
out_out.abort
out_out.opcode
out_out.give

* optional signals

SOM	Valid	EOM	Signal Description
1	0	0	The start of a message, without any associated data
1	1	0	The start of a message, coincident with data
1	0	1	A zero-length message, with no data, in a single word
1	1	1	A single word message
0	0	0	*Reserved*
0	0	1	A trailing end of message, with no data
0	1	0	A data value in the middle of a message
0	1	1	A data value, coincident with the end of the message

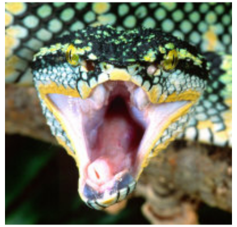
***When "abort" is not used, table is valid**

HDL Streaming Data Signals



- Rules for input interfaces:
  - **ready** indicates that metadata and perhaps the data signals are valid
  - If **ready** not asserted, *none of the metadata signals are valid or meaningful*
  - Worker *takes* input data when **ready** is asserted by asserting **take**
  - It is invalid to assert **take** if the **ready** input signal is not asserted
- Rules for output interfaces:
  - **ready** indicates that metadata and perhaps data can be produced
  - If **ready** not asserted, none of the metadata or data output are considered valid
  - Worker *gives* data when **ready** is asserted by asserting **give**
  - It is invalid to assert **give** if the **ready** input signal is not asserted

Ready (I/O), **take**, **give** control the flow of data and metadata words through an interface



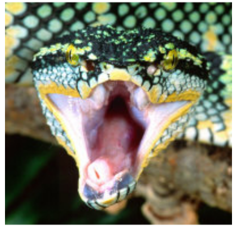
HDL Streaming Data Signals

- Data flows according to FIFO semantics
- Table of signal terminology comparison

Meaning	OpenCPI	Classic FIFO	AXI	Xilinx FIFO
Data is available to consume	ready	not_empty	valid	!empty
Consume Data	take	dequeue	ready	rd_en
Data can be produced	ready	not_full	ready	!full
Produce Data	give	enqueue	valid	wr_en

- In AXI interfaces, either signal (**`valid`** or **`ready`**) may be asserted early. The handshake (**`ready`**) can in fact be asserted early even when **`valid`** is not yet asserted.
- With OpenCPI it is invalid to assert **`take`** or **`give`** without **`ready`** being asserted.
- In Xilinx FIFO, **`rd_en`** and **`wr_en`** are ignored if the fifo is **`empty`** (input) or **`full`** (output).

HDL Application Worker Example

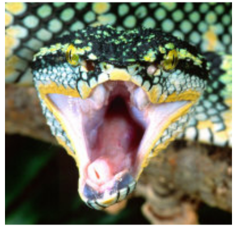


Example:

- No **opcode** or **byte_enable** is used since the protocol has a single operation
- ctl_in.is_operating reflects the reset condition
- Combinatorial logic: computation takes place in a single clock cycle: Simply adds a constant to every data value from input to output

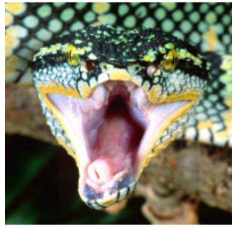
```
library IEEE; use IEEE.std_logic_1164.all; use ieee.numeric_std.all;
library ocpi; use ocpi.types.all; -- remove this to avoid all ocpi name collisions
architecture rtl of my_vhdl_worker is
  signal doit : bool_t;
begin

  doit       <= ctl_in.is_operating and in_in.ready and out_in.ready;
  in_out.take   <= doit;
  out_out.give  <= doit;
  out_out.data  <= std_logic_vector(unsigned(in_in.data) + 3);
  out_out.som   <= in_in.som;
  out_out.eom   <= in_in.eom;
  out_out.valid <= in_in.valid;

   -- loop-back property example
   props_out.my_prop <= props_in.my_prop;

end rtl;
```

HDL Worker Service Interfaces



- Time Service Interface
  - “time of day” provided to the precision of the OWD attributes within the TimeInterface element, in GPS time
    - SecondsWidth – 0 to 32 bits for reporting seconds field in time-of-day (**Optional**)
      - = 32 bits, absolute time
      - < 32bits, relative time truncated preserving the LSB, to that value and wraps.
    - FractionWidth – 0 to 32 bits for reporting fractions field in time-of-day (**Optional**)
      - = 32 bits, 2^-32 or ~233 ps.
      - < 32bits, MSB are preserved, such that MSB is always ½ second
    - AllowUnavailable – Indicates when time-of-day is valid (**Optional**)
  - VHDL uses record port time_in.*
    - Input: seconds, fraction, valid
    - Ex: time_now <= std_logic_vector(time_in.seconds) & std_logic_vector(time_in.fraction);

HDL Build Targets: HdlTargets vs. HdlPlatforms



HdlTargets

- Chips or chip families, and simulators
- Used to build HDL primitives, workers, and assemblies
- OnlyTargets/ExcludeTargets
- Smallest part in family chosen by default – to override, use the HdlExactPart Makefile variable
- Ex: xsim, modelsim, isim, stratix4, virtex6, zynq, zynq_ise

HdlPlatforms

- Actual FPGAs on specific boards, and simulators
- Used to build HDL containers (final bitstreams)
- OnlyPlatforms/ExcludePlatforms
- HdlTarget(s) implied (family and part) at all levels except final bitstream
- Ex: xsim, modelsim, isim, alst4, ml605, zed, zed_ise, matchstiq_z1