OpenFPGA General API Specification 0.4 (DRAFT FOR COMMENT)

1. Goals and Assumptions

The general API specification is proposed as an industry standard API for high-level language access to Reconfigurable Computing (RC) resources in a portable manner. The API has several design goals:

- Portability and supportability across a wide range of available RC platforms.
- Similarity to existing common capabilities available within RC platforms.
- Minimal essential functionality to support general application acceleration with RC resources.
- Simple to use, understand and implement.

Several assumptions were made regarding the operational environment for processes incorporating the API including:

- A logical host software system - the host - exists able to handle basic communication and housekeeping tasks for the reconfigurable resource.
- A device is a generic term for a reconfigurable computing device such as an FPGA or series of connected FPGAs or even a heterogeneous device comprised of FPGAs and other non-reconfigurable devices.
- Calling process has configuration access to the reconfigurable resources.
- Calling process can assume control of the reconfigurable resource.
- Several reconfigurable devices may exist in the system.
- The system may be a multi-user system.
- Calling process can identify memory specific for communication with the RC device.
- The reconfigurable device may run asynchronously.
- The implementation of the API should be thread-safe.
- The API will be both “C-friendly” and “Fortran-friendly”.

Several assumptions were not made in creating the specification:

- All resources might not reside on the same host node.
- The operating system might not be multi-user.
- The operating system might not be thread-capable.
- Reconfigurable devices might not be homogenous.
- Methods of communication between host process and reconfigurable device are unspecified.
- Assumptions about memory hierarchies and memory address spaces shared between host system and reconfigurable resource are not made.

The general API specification does not attempt to provide all functionality for all aspects of reconfigurable computing use in high-level applications. Several aspects have been deferred to a later release or to be embodied as vendor specific extensions in the near term:

- Communication directly between RC devices.
- Continuously streaming data between host and RC devices.
- RC device-initiated communication to the host (interrupts)
- Remote management and remote configuration (including partial reconfiguration) of FPGAs.
- Communication and control among multiple RC devices.
- Full details on probing information and state of a RC device.
- Portability of specific core definitions

Specification Requirements:

- The OpenFPGA GenAPI interface specification provides a definition of a baseline portable API for interoperability at the source code level for applications incorporating reconfigurable device technology. Successful implementation of the OpenFPGA GenAPI requires the following:
  - All defined interface elements have been defined in a compilation environment
  - The state of the system behaves consistently, in accordance with the provided state diagram describing the macroscopic behavior of the heterogeneous system
  - Specified error messages and error handling are implemented according to specification.

- The OpenFPGA GenAPI does not preclude the vendor from offering compatible extensions of the API provided these extensions are clearly delineated.

- The OpenFPGA GenAPI specification is copyright by OpenFPGA, Inc. Permission for use is granted with the provision OpenFPGA Inc. and the specific version of the API are referenced.

Literature reference:

OpenFPGA GenAPI specification version 0.4. Available for download at **www.openfpga.org**. 2008.

2. Data Dictionary

The data dictionary provides a working glossary of terms utilized in the definition of the API specification.

Host:	A host is the computer system where the run time process or program invoking the API at an application level resides.
Device:	A device is a reconfigurable computing resource that can be configured dynamically. Usually a physical device containing an FPGA or other reconfigurable hardware.
Instance:	An instance is the set of instructions required to dynamically configure a device. For individual FPGA devices, this is most commonly the bitstream.
Context:	A context is the data associated with a collection of devices (as defined above) available to a running process.
Buffer:	A buffer is a range of physical memory that is commonly accessible by a device and the process.
Property:	A property is defined as a name-value pair.
`device_id:`	A label for a structure for a particular accelerator. It holds the state of the accelerator including hardware error states and can be queried by **`oa_get_device_property`**. At present, only RC device types are supported.
`context_id:`	A label for an allocation context state storage location. The structure handles software error states and holds other state for the group of allocated accelerators associated with the context. It can be queried by **`oa_get_context_property`**.
`instance_id:`	A label for a structure for a particular instance. It holds the state of the instance and can be queried by **`oa_get_instance_property`**.
`buff_id:`	A label for a structure for an allocated buffer. It holds the state of the buffer, including its association with a particular memory location of a reconfigurable resource.
msglvl:	Property defining the message level of output generated by invoked methods
status**:**	The status returned by an invoked method

3. Implementable data types

Defined constants and other values are defined in the following section.

3.1 Tracing and debugging information

msglvl implemented as defined integer constants

Definition	Value	Behavior
OA_NO_OUTPUT	0	No output produced from invoked method
OA_TRACE	1	Low-level tracing provided including method name and timing sequence
OA_TRACE_DATA	2	OA_TRACE plus signatures of data passed.

Content and format of the specific information when displayed is vendor defined.

3.2 Success status

status implemented as defined integer constants

Definition	Value	Behavior
OA_SUCCESS	1	successful method invocation
OA_FAILURE	0	unsuccessful method invocation

3.3 Memory types

When referring to types of memory available on the system, the following declarations and definitions are utilized.

mem_types implemented as defined integer constants

Definition	Value	Description
OA_MEMT_UNKOWN	0	Type of attached memory bank is unkown
OA_MEMT_SRAM	1	Attached memory bank is of SRAM type
OA_MEMT_DRAM	2	Attached memory bank is of DRAM type
OA_MEMT_QDRAM	3	Attached memory bank is of QDRAM type

4. Method Summary Overview

Consolidated interface method index:

Calls to initialize the API, close the API and handle errors

- oa_init(i_version_str, o_context_id);
- oa_end(i_context_id);
- oa_get_errno (i_context_id, o_errno_int);
- oa_perror (i_context_id, i_errno_int, i_txt_str);

Atomic allocation of devices

- oa_find_device(i_context_id, i_prev_device_id, o_next_device_id);
- oa_alloc_device_add (i_context_id, i_device_id);
- oa_alloc_device_commit (i_context_id);
- oa_free_device (i_context_id, i_device_id);
- oa_list_alloc_device (i_context_id, i_prev_device_id, o_next_device_id);

Querying properties and states of the API, FPGAs and instances.

- oa_get_context_property (i_context_id, i_property_id_str, o_property_str);
- oa_set_context_property (i_context_id, i_property_id_str, i_property_str);
- oa_get_device_property (i_context_id, i_device_id, i_property_id_str, o_property_str);
- oa_set_device_property (i_context_id, i_device_id, i_property_id_str, i_property_str);
- oa_get_instance_property (i_context_id, i_instance_id, i_property_id_str, o_property_str);
- oa_set_instance_property (i_context_id, i_instance_id, i_property_id_str, i_property_str);

Allocating instances, configuring and resetting FPGAs

- oa_alloc_instance (i_context_id, i_bitstream_filename, o_instance_id);
- oa_free_instance (i_context_id, i_instance_id);
- oa_use_instance (i_context_id, i_bitstream_ptr, i_size_int, o_instance_id);
- oa_configure_device (i_context_id, i_device_id, i_instance_id);
- oa_init_device (i_context_id, i_device_id);

Allocating transfer and receive buffers

- oa_malloc_buffer (i_context_id, i_device_id, i_port_pos_str, i_size_int, i_mode_int, o_buff_id);
- oa_reuse_buffer (i_context_id, i_device_id, i_port_pos_str, i_size_int, i_mode_int, o_buff_id);
- oa_get_buffer_ptr (i_context_id, i_buff_id, o_buffer_ptr);
- oa_free_buffer (i_context_id, i_device_id, i_buff_id);

Initiating execution, waiting for normal termination and performing premature abortion of execution

- oa_run (i_context_id, i_device_id);
- oa_wait (i_context_id, i_device_id);
- oa_abort (i_context_id, i_device_id);
- oa_status(i_context_id, i_device_id, o_value_int32);

Transferring data sets and single values to and from devices

- oa_send (i_context_id, i_buff_id);
- oa_receive (i_context_id, i_buff_id);
- oa_write_parameter{32|64} (i_context_id, i_device_id, i_parameter_pos_str, i_value_int{32|64})
- oa_read_parameter{32|64} (i_context_id, i_device_id, i_parameter_pos_str, o_value_int{32|64})

5. OpenFPGA GenAPI Function Definitions

Note: All proposed functions have the argument **`context_id`** to hold state across function calls. For example, error state will be held by **`context_id`**. In the tables of the argument description, this argument is usually not described for simplicity.

All proposed functions return a **`success`** flag. The value of the flag will be 1 to indicate successful completion of the function call, and 0 to indicate failure (See section 3.2). Use the functions **`oa_get_errno`** and **`oa_perror`** to query the error state (See section 6.0 for error details).

All arguments are passed by reference.

Error conditions are not yet identified for each method.

5.1 Discovery and Initialization

These methods provide a means for processes to discover, initialize, allocate and query reconfigurable resources available to the process.

Function: **`oa_init`**
Usage: `success = oa_init( i_version_str, o_context_id );`

Parameter	I/O	Description
i_version_str	I	The OpenFPGA GenAPI version that the system must be compatible with to correctly execute the program. If the system is not capable of supporting the version specified by i_version_str, the oa_init call will flag failure. Regardless of success, the context_id will be in a state usable with oa_get_errno and oa_perror. The version string could be two numbers separated by a dot (i.e. "1.3"), but it could any other string too, including free-form name strings (i.e. "gutsy"). By specifying a particular version, the programmer is certifying that all uses of the API will be compatible with that version; the program should fail if API calls are made that are incompatible with the stated version.
o_context_id	O	The new context_id. This structure is used by all other oa_functions to hold state

Description: Initialize the API. This also returns a **`context_id`** for use in all other functions. The **`version_str`** parameter is required, and is used to verify that the system running the program is compatible with the version of the OpenFPGA GenAPI that is assumed in the program.

Error conditions:

Function: **`oa_end`**
Usage: `success = oa_end( i_context_id );`

Parameter	I/O	Description
`i_context_id`	I	The **`context_id`** to free

Description: Free the **`context_id`** structure associated with the corresponding **`oa_init`** call. Will also free any other reconfigurable resources associated with the **`context_id`**, including any data structures created by other OpenFPGA GenAPI calls using this **`context_id`**. Memory buffer allocation is unaffected. User is responsible for previously releasing any user allocated buffers.

Function: **oa_get_errno**
Usage: `success = oa_get_errno ( i_context_id, o_errno_id );`

Parameter	I/O	Description
i_context_id	I	
o_errno_id	O	The error, for use with **oa_perror**

Description: If a call to a function in the API fails (returns zero), the **errno_id** parameter can be used with **oa_perror** to print a human readable error message.

Error conditions:

Function: **oa_perror**
Usage: `success = oa_perror ( i_context_id, i_errno_id, i_user_str );`

Parameter	I/O	Description
i_context_id	I	
i_errno_id	I	An error that has previously been reported.
i_user_str	I	A string of text that is written to stderr before the error message.

Description: Prints a human-readable error message on stderr that describes the error associated with **i_errno_id**. Any text in **i_user_str** is printed first, before the error message, and the whole message is terminated by a new-line.

Error conditions:

Function: **oa_find_device**
Usage: `success = oa_find_device( i_context_id, i_prev_device_id, o_next_device_id );`

Parameter	I/O	Description
i_context_id	I	
i_prev_device_id	I	Vendor & platform specific ID or OA_NO_DEVICE
o_next_device_id	O	Vendor & platform specific ID or OA_NO_DEVICE

Description: The function finds an available device supported though the API and returns a handle structure to the device. If called with a non-null argument, the following device in a vendor-defined order is returned. By repeatedly calling this function with the previous call's result as input argument, all available devices supported by the API in the system can be traversed.

Error conditions:

Function: **oa_alloc_device_add**
Usage: `success = oa_alloc_device_add ( i_context_id, i_device_id );`

Parameter	I/O	Description
i_context_id	I	The allocation context to add a device to
i_device_id	I	The **device_id** to add to the allocation context

Description: Add a specific device to an allocation context. The allocation context, **context_id**, is given by a call to **oa_init**, and the **device_id** is given by calls to **oa_find_device**. Note that the device are not actually allocated until a call to **oa_alloc_device_commit** has been made.

Error conditions:

Function: **oa_alloc_device_commit**
Usage: `success = oa_alloc_device_commit ( i_context_id );`

Parameter	I/O	Description
i_context_id	I	

Description: Allocates all devices that have been added to the **context_id** by calls to **oa_alloc_device_add** as an atomic operation. This prevents race conditions between multiple processes competing for device resources.

Error conditions:

Function: **oa_list_alloc_device**
Usage:
```
success = oa_list_alloc_device ( i_context_id,
i_prev_device_id, o_next_device_id );
```

Parameter	I/O	Description
i_context_id	I	
i_prev_device_id	I	Vendor & platform specific ID or OA_NO_DEVICE
o_next_device_id	O	Vendor & platform specific ID or OA_NO_DEVICE

Description: Iterate through the devices that have previously been added to the **context_id.**

Error conditions:

Function: **oa_free_device**
Usage: `success = oa_free_fpga ( i_context_id, i_device_id );`

Parameter	I/O	Description
i_context_id	I	
i_device_id	I	The device to remove from the **context_id** and free from the common resource pool

Description: Remove a specific device from the allocation context and, if allocated, free it.

Error conditions:

Function: **`oa_get_context_property`**, **`oa_get_device_property`**, **`oa_get_instance_property`**

Usage:
```
success = oa_get_context_property ( i_context_id,
i_property_id_str, o_property_str );
success = oa_get_fpga_property ( i_context_id, i_device_id,
i_property_id_str, o_property_str );
success = oa_get_instance_property ( i_context_id,
i_instance_id, i_property_id_str, o_property_str );
```

Parameter	I/O	Description
`i_context_id`	I	
`i_device_id`	I	The device to query, only available with **`oa_get_device_property`**
`i_instance_id`	I	The instance to query, only available with **`oa_get_instance_property`**
`o_property_str`	O	The value of the property queried

Description: Query the properties of a resource. All properties are strings, and resulting values are strings, separated by space if multiple values are returned. Properties and property values may consist of underscore, numbers and lowercase letters in the English alphabet. Default properties are `undefined` and `properties`. The `properties` property will return a list of all valid properties for the device.

See Section 5.9 for definitions of properties available at each level.

Error conditions:

Function: **`oa_set_context_property`**, **`oa_set_device_property`**, **`oa_set_instance_property`**

Usage:
```
success = oa_set_context_property ( i_context_id,
i_property_id_str, i_property_str );
success = oa_set_device_property ( i_context_id, i_device_id,
i_property_id_str, i_property_str );
success = oa_set_instance_property ( i_context_id,
i_instance_id, i_property_id_str, i_property_str );
```

Parameter	I/O	Description
`i_context_id`	I	
`i_device_id`	I	The device to set a property in, only available with **`oa_set_device_property`**
`i_instance_id`	I	The instance to set a property in, only available with **`oa_set_instance_property`**
`i_property_id_str`	I	The name of the property to set
`i_property_str`	I	The value to which the property should be set

Description: Change the properties of resources. All properties are strings, and settable values are strings, separated by space if multiple values are set. Properties and property values may consist of underscore, numbers and lowercase letters in the English alphabet. Default properties are `undefined` and `properties`.

Error conditions:

5.2 Instance Configuration and Initiation

These methods provide a means for the process to load hardware instances for future execution, configure devices with loaded instances and reset the configured device.

Function: **oa_alloc_instance**
Usage:
```
success = oa_alloc_instance ( i_context_id,
i_configuration_filename_str, o_instance_id );
```

Parameter	I/O	Description
i_context_id	I	
i_config_filename_str	I	The file name, including full path to the configuration file (bitstream in the case of FPGA devices)
o_instance_id	O	A handle to the allocated instance

Description: Load file contents containing a device configuration instance from disk into primary memory and assign a instance id.

Error conditions:

Function: **oa_free_instance**
Usage:
```
success = oa_free_instance ( i_context_id, i_instance_id );
```

Parameter	I/O	Description
i_context_id	I	
i_instance_id	I	The instance to remove from memory

Description: Free the resources associated with a particular instance from primary memory. This does not remove the instance from a configured device if it has previously been used to configure a device.

Error conditions:

Function: **oa_use_instance**
Usage:
```
success = oa_use_instance ( i_context_id, i_bitstream_ptr,
i_size_int, o_instance_id );
```

Parameter	I/O	Description
i_context_id	I	
i_config_ptr	I	Pointer to a memory space containing configuration data for a device
i_size_int	I	Number of bytes of configuration data stored at location of **i_config_ptr**
o_instance_id	O	A handle to the allocated instance

Description: If a bitstream has been loaded into memory through other means than **oa_alloc_instance**, this function can be used to associate a **instance_id** with that memory space. Note: The memory space containing the bitstream will not be reclaimed when performing **oa_free_instance**, **oa_end**, or any other function that would normally have de-allocated the memory space for the bitstream. Responsibility for reclaiming that memory space lies with the allocator of that space.

Error conditions:

Function: **`oa_configure_device`**
Usage:
```
success = oa_configure_device ( i_context_id, i_device_id,
i_instance_id );
```

Parameter	I/O	Description
`i_context_id`	I	
`i_device_id`	I	The device to configure
`i_instance_id`	I	The instance to configure the device with

Description: Configure a device with the specified instance and, if necessary, reset the device and the loaded instance. After this call, the device will be configured with the instance and made ready for execution. If the system allows it, the call is non-blocking, so other operations may be performed while the device is being configured. Any subsequent API call that requires the device to have completed its configuration will block until configuration is completed.

Error conditions:

Function: **`oa_init_device`**
Usage:
```
success = oa_init_device ( i_context_id, i_device_id );
```

Parameter	I/O	Description
`i_context_id`	I	
`i_device_id`	I	The device to initialize

Description: Initialize a device to its initial state. This puts a configured device in a state ready for execution and signals the loaded instance to reset. This call does not clear the instance from the device.

Error conditions:

5.3 Buffer Allocation

These API calls will allocate, free and re-assign buffers for communication of data to and from the RC devices.

Function: **`oa_malloc_buffer`**
Usage:
```
success = oa_malloc_buffer ( i_context_id, i_device_id,
i_port_pos_str, i_size_int, i_mode_int, o_buff_id );
```

Parameter	I/O	Description
`i_context_id`	I	
`i_device_id`	I	The device to communicate with on this buffer
`i_port_pos_str`	I	The memory port on the device that this buffer shall be associated with
`i_size_int`	I	The minimum number of bytes to allocate in the buffer
`i_mode_int`	I	The direction of communication for this buffer `INPUT`, `OUTPUT` or `INOUT`
`o_buff_id`	O	A structure representing the allocated buffer

Description: Allocate memory to use when communicating with a device. The buffer is associated with a particular device and a particular memory port on the device, along with the direction of communication that will take place on this buffer. The memory may be aligned to be optimal for data transfers, e.g. on memory page boundaries. The size argument is a minimal size only; for some kinds of allocation, memory can only be allocated in chunks. The buffer may be allocated in memory-mapped memory directly attached to the device on some platforms, such as the Cray XD1. On other platforms, such as SGI and Nallatech, the buffer will be used message-passing style.

Error conditions:

Function: **`oa_reuse_buffer`**
Usage:
```
success = oa_reuse_buffer ( i_context_id, i_device_id,
i_port_pos_str, i_size_int, i_mode_int, o_buff_id );
```

Parameter	I/O	Description
`i_context_id`	I	
`i_device_id`	I	The new device to communicate with on this buffer
`i_port_pos_str`	I	The memory port on the new device that this buffer shall be associated with
`i_mode_int`	I	The direction of communication for this buffer `INPUT`, `OUTPUT` or `INOUT`
`i_buff_id`	I	The old buffer that is being re-used
`o_buff_id`	O	A new structure representing the buffer in its new use

Description: In some situations the same buffer needs to be used together with several different devices. One example is when the output of one device is to be used as input to another device. To avoid memory copying in such situations, a buffer can be re-used for communication with two different devices.

Error conditions:

Function: **oa_get_buffer_ptr**
Usage:
```
success = oa_get_buffer_ptr ( i_context_id, i_buff_id,
o_buffer_ptr );
```

Parameter	I/O	Description
i_context_id	I	
i_buff_id	I	A buffer id referring to the memory

Description: Returns a pointer to the buffer referred to by the **i_buff_id**. Memories that are returned from **oa_get_buffer_ptr** will not be automatically freed by **oa_close**. This allows the program to use buffer spaces even after the use of the OpenFPGA GenAPI has ceased, without first performing a memory copy. After a call to **oa_get_buffer_ptr** it is the responsibility of the calling program to eventually free the buffer, either through **oa_free_buffer**, or, if the API has been closed, through commonly available means, such as free().

Error conditions:

Function: **oa_free_buffer**
Usage:
```
success = oa_free_buffer ( i_context_id, i_buff_id );
```

Parameter	I/O	Description
i_context_id	I	
i_buff_id	I	The buffer to free

Description: Free the resources associated with this buffer. If the memory for the buffer has been re-used for use with several devices, all **buff_id**:s that refer to the same memory must be freed before the memory itself will be freed. If the **buff_id** is currently in use by a running FPGA, the function will fail. This will also free buffers have been accessed through **oa_get_buffer_ptr**.

Error conditions:

5.4 Execution

These API calls provide the process abilities to execute and communicate with a instance running in a device

Function: **oa_run**
Usage: `success = oa_run ( i_context_id, i_device_id );`

Parameter	I/O	Description
i_context_id	I	
i_device_id	I	The device to run

Description: Runs the instance configured on the device. The call to **oa_run** is non-blocking, i.e. the call returns immediately. **oa_send** should be performed to transfer data to device memories before **oa_run** is called. After a subsequent call to **oa_wait**, **oa_receive** should be used to retrieve data that has been computed on the device. **oa_malloc_buffer** must have been performed on both send and receive buffers before **oa_run** is called to allow for shared-memory models.

Error conditions:

Function: **oa_wait**
Usage: `success = oa_wait ( i_context_id, i_device_id );`

Parameter	I/O	Description
i_context_id	I	
i_device_id	I	The running device to wait for completion on

Description: A blocking wait for an RC device to complete its run. The function will return when the run of the instance on the RC device has completed.

Error conditions:

Function: **oa_status**
Usage: `success = oa_status ( i_context_id, i_device_id, o_status);`

Parameter	I/O	Description
i_context_id	I	
i_device_id	I	The device to test status on
O_status	O	The status of the device

Description: A method to test the status for a configured device. The function returns immediately with the latest status of the device. Status states include:

OA_STATUS_UNDEFINED:= device resource prior to any initialization.

OA_STATUS_CONFIGURED := returned for a device that has been configured but has not yet been started

OA_STATUS_RUNNING := returned for a device that is currently running

OA_STATUS_STOPPED := returned for a device that has been started but is no long running.

Error conditions:
OA_STATUS_UNDEFINED will result in the success state to be set to error.

Function: **`oa_abort`**
Usage: `success = oa_abort ( i_context_id, i_device_id );`

Parameter	I/O	Description
`i_context_id`	I	
`i_device_id`	I	The device to abort

Description: Forcibly aborts execution on a currently running device before normal completion. After the call, buffers that are declared `OUTPUT` or `INOUT` will be in an unpredictable state. Flushing of buffer contents is not required in this version

Error conditions:

5.5 Data Transfer

Function: **`oa_send`**
Usage:
```
success = oa_send ( i_context_id, i_buff_id );
```

Parameter	I/O	Description
`i_context_id`	I	
`i_buff_id`	I	The id of the buffer where the data resides before sending. During the malloc call, the buff_id was associated with full information about what device to send data to, the mode of transfer, etc.

Description: Send data to a device. On systems with message-passing, such as Nallatech or SGI, the function actually performs the data transmission: The function sends the data in the buffer to memories connected to the device, where they will be available to the instance once it is run. On systems with memory mapping, such as the Cray XD1, the function takes the role of synchronization: The function call indicates that the shared memory region is now available to the device and will no longer be modified by the host process.

Error conditions:

Function: **`oa_receive`**
Usage:
```
success = oa_receive ( i_context_id, o_buff_id );
```

Parameter	I/O	Description
`i_context_id`	I	
`i_buff_id`	I	The id of the buffer to receive data in. During the malloc call, the buff_id was associated with full information about what device to get data from, the mode of transfer, etc.

Description: Receives a data block from a device. On systems with message-passing, such as Nallatech or SGI, the function actually performs the data transmission: The function transfers data from the device to the buffer. On systems with memory mapping, such as the Cray XD1, the function takes the role of synchronization: The function call indicates that the shared memory region is now available to the host process and will no longer be modified by the device. The function will block, on message-passing systems until the data transfer from the device has completed, on shared-memory systems until the buffer has been released by the device.

Error conditions:

Function: **`oa_write_parameter{32|64}`**
Usage:
```
success = oa_write_parameter{32|64} ( i_context_id,
i_device_id, i_parameter_pos_str, i_value_int{32|64} )
```

Parameter	I/O	Description
`i_context_id`	I	
`i_device_id`	I	The device to write the parameter to
`i_parameter_pos_str`	I	The parameter port on the device to write the parameter to
`i_value_int{32|64}`	I	The parameter value as 32 bit-wide or 64 bit-wide reprentation

Description: Writes a parameter to the device. Only one value can be sent per parameter port during the run.

Error conditions:

Function: **`oa_read_parameter{32|64}`**
Usage: `success = oa_read_parameter{32|64} ( i_context_id, i_device_id, i_parameter_pos_str, o_value_int{32|64} )`

Parameter	I/O	Description
`i_context_id`	I	
`i_device_id`	I	The device to read the parameter from
`i_parameter_pos_str`	I	The parameter port on the device to read the parameter from
`o_value_int{32|64}`	O	The resulting value in a 32 bit-wide or 64 bit-wide representation

Description: Receives a single parameter from the device. Only one value can be received per parameter port during the run.

Error conditions:

5.9 Property Definitions

This section defines the minimal properties required by the standard specification for each level of abstraction.

Methods: oa_get_context_property(), oa_set_context_property()

Name	Values	Description
Version	0.4	Valid versions of the GenAPI currently implemented or to be utilized for subsequent execution

Methods: oa_get_device_property(), oa_set_device_property()

Name	Values	Description
Example: Vendor	Text less than 32 characters	Manufacturer of the device
Model	Text less than 32 characters	Model number of the device (vendor defined and registered name)
N_Mem_Banks	Integer	Number of memory banks attached to the device
N_Mem_Types	Integer	Number of different types of memory banks attached to the device (QDRAM, SRAM, DRAM, …)
Mem_Bank_Sizes	Where supported, structure of integers presented as a comma separated value list. E.g. "SRAM=10,DRAM=20" provides the information on two memory banks, SRAM of size 10MB and DRAM of size 20MB.	Ordered list of the size of memory banks in Mbyte (1M=2^20)
Optional: Link_Bandwith	Integer	Theoretical peak bandwidth between device and host system in Gbyte/s (1G = 2^30)
Optional: Link_Latency	Integer	Theoretical lantency of the link between device and host system in ns.

Methods: oa_get_instance_property(), oa_set_instance_property()

Name	Values	Description
Example: Max_Clock_Rate	Text less than 32 characters	The maximum frequency that the given instance may be executed
MD5_Checksum	Text less than 128 characters	The MD5 checksum of the bitstream used to create the instance
SHA1_Checksum	Text less than 128 characters	The SHA1 checksum of the bistream used to create the instance

6.0 System Behavior

Significant operational states of the system are described below. Expected behavior within these states is described for each.

1.0 Pre-initialization: The state of the environment prior to execution of the **`oa_init`** function that creates an accelerator context.

2.0 Initialized Environment: The state of the environment following initialization and creation of an accelerator context and prior to committing RC resources for use. In this state, RC devices (FPGAs) may be readily added and removed.

3.0 Device Committed: The state of the system following allocation and commitment of RC devices (FPGAs) for the application.

4.0 Instances Allocated: The state of the system following the commitment of RC devices and first addition of a instance for configuring resources. Instances may be added and freed in this state without limit.

5.0 Device Configured: The state of the system immediately following the first commitment of a instance to a specific RC device. Additional RC devices may be configured and initialized in this state. Furthermore, instances may be added and eliminated in this state. Errors occur if unavailable instances are removed, or RC devices are attempted for configuration with non-existing instances.

6.0 Device Running: At least one RC device is running on the system. Instances may be added or removed when in this state. RC devices may be configured and started in this state provided RC device is not running.

Clarifications:

- Buffer allocations must follow configuration of the RC devices and prior to commencement of RC device execution.
- Send and receive functions are only successful if the requested RC resource is either prepared for the run state (send) or is in the run state (receive).
- Query and set functions do not transition the major system states.
- Normal validation of parameters for each invoked method is assumed.
- RC resources on the system are discovered as part of the initialization process

Primary Error Message Summary

Error Message List	Explanation
OA_ENVIRONMENT_NOT_READY	Environment not initialized
OA_DEVICE_COMMITTED	Attempting to commit a previously committed device
OA_DEVICE_RUNNING	Attempting to change allocation of a running device
OA_NO_COMMITTED_DEVICES	There were no devices committed prior to adding new instances
OA_NO_AVAILABLE_INSTANCE	The given instance is not allocated
OA_DEVICE_NOT_CONFIGURED	The given reconfigurable resource is not configured
OA_DEVICE_NOT_RUNNING	The given reconfigurable device is not running
OA_INVALID_DEVICE	The device handle passed is not valid
OA_INVALID_INSTANCE	The instance handle passed is not valid
OA_INVALID_CONTEXT	The context handle passed is not valid

The following state-transition table defines the dominant system behavior for successful execution and changes in the predominant system states when using the General API.

Significant system changes are highlighted in green.
Environment re-initialization transitions are highlighted in yellow.
Peach color highlights calls that are not errors yet do not cause a major state transition.

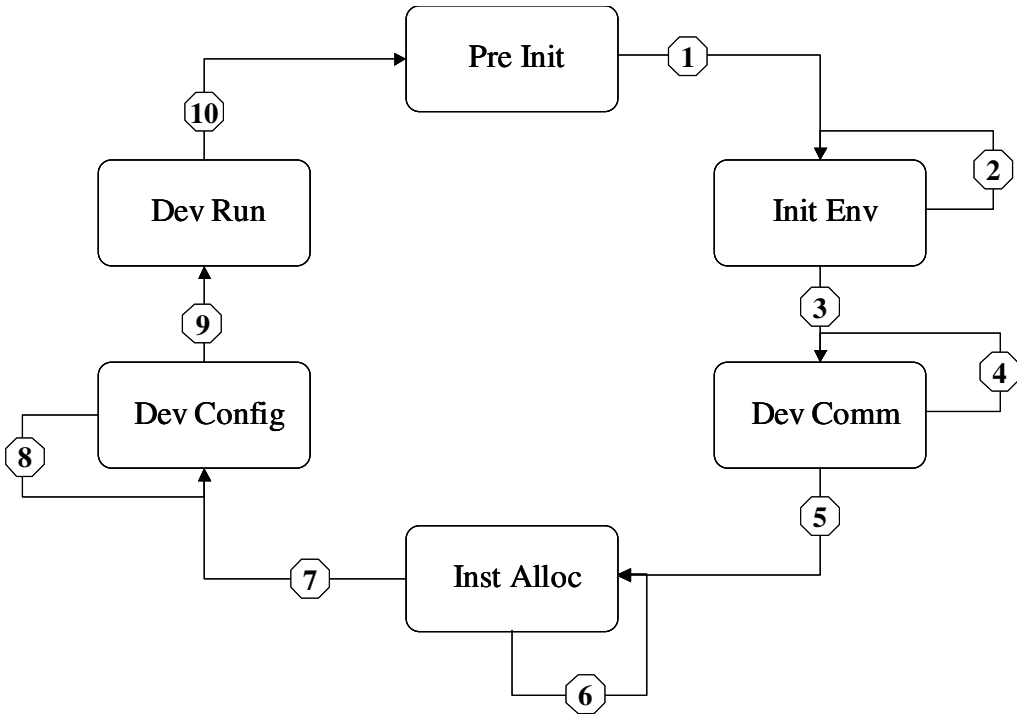
Methods in highlighted in bold are state changing methods used in a normal execution. Additional problem specific memory allocations are required for most implementations.

State / Method	Pre-initialization (PI) 1.0	Initialized Environment (IE) 2.0	Devices Committed (DC) 3.0	Instances Allocated (IA) 4.0	Devices Configured (DD) 5.0	Devices Running (DR) 6.0
oa_init	**[IE]** Action: Create new context	**[IE]** Action: Create new context	**[IE]** Action: Create new context	**[IE]** Action: Create new context	**[IE]** Action: Create new context	**[IE]** Action: Create new context
oa_alloc_device_add	Error (Environment not initialized)	**No Change/ A: individual device added**	Error (if device already committed)	Error (devices already committed)	Error (devices already committed)	Error (devices already running)
oa_alloc_device_free	Error (Environment not initialized	No Change/ A: individual device freed	Error (if device already committed)	Error (devices already committed)	Error (devices already committed)	Error (devices already running)
oa_alloc_device_commit	Error (Environment not initialized)	**[DC] A: devices committed**	No change/ No Error	Error (devices already committed)	Error (devices already committed)	Error (devices already running)
oa_alloc_instance	Error (Environment not initialized)	Error (resources must be committed before adding instances)	**[IA] A: instance is added to available instances**	No change/ A: Instance added	No Change/ A: Instance added	No Change/ A: Instance added
oa_use_instance	Error (Environment not initialized)	Error (resources must be committed before adding instances)	[IA] A: instance is added to available instances	No change/ A: Instance added	No Change/ A: Instance added	No Change/ A: Instance added
oa_free_instance	Error (Environment not initialized)	Error (no available instances)	Error (no available instances)	No change (Error if instance is not in allocated set)	No change/ A: instance is freed (Error if instance is not in allocated set)	No change/ A: instance is freed (Error if instance is not in allocated set)
oa_configure_device	Error (Environment not initialized)	Error (no available instances)	Error (no available instances)	**[DC] A: configure device**	No change/ A: configure device with instance	Error (if on currently running device)
oa_init_device	Error (Environment not initialized)	No change	No Change	No change/ A: device initialized	No change/ A: device instance is initialized	Error (if on currently running device)
oa_run	Error (Environment not initialized)	Error (device not configured)	Error (device not configured)	Error (device not configured)	**[DR]**	No Change
oa_wait	Error (Environment not initialized)	Error (device not configured)	Error (device not configured)	Error (device not yet configured)	Error (if request for non-running device)	**[DD] A: wait for decice to complete**
oa_abort	Error (Environment not initialized)	Error (device not configured)	Error (no running devices)	Error (device not yet configured)	Error (if request for non-running device)	[DD] A: Abort requested device
oa_end	[PI]Action: Release context	[PI] A: Release context	[PI] A: Release context	[PI] A: Release context	**[PI]** A: Release context	[PI] A: abort devices

State Diagram Transition Description

Required standard behavior for the system using primitive functions defined in the API for a single created context.

1: oa_init
2: oa_init (another version arg), oa_alloc_device_*, oa_free_device, oa_list_device, oa_{get/set}_property
3: oa_alloc_device_commit
5: oa_{alloc/free/use}_instance,
7: oa_configure_device,
9: oa_run
10: oa_end



8.0 Revision History

July 1, 2008 – Added revised methods and definitions (S Mohl)

July 7, 2008 – Added error condition placeholders, state descriptions and corresponding transition table. (E Stahlberg)

July 9, 2008 – fpga_id replaced with device_id to support generic accelerators including groups of fpgas working as a unit (E Stahlberg)

July 14, 2008 – Revisions to clarify state behavior and rename methods for context and instance relating to run-time contexts and design instances. (T Steinke/E Stahlberg) Create version 0.4.

July 16, 2008 – Final cleanup and editing.

9.0 Comments and Questions

July 9, 2008 OpenFPGA Forum

Q: How are environment variables handled in the get_property and set_property methods?

A: standard updated to indicate that accelerator specific execution environment variables are handled in the get and set property methods in the first version of the standard. General environment variable inquiry and setting are not included in the first version of the standard. (Modeled after OpenMP specification behavior)

Q: Consider revising term 'design' in favor of another term.

A: Recommend replacing design with term instance

Q: Why is there a new memory allocation method?

A: A new memory allocation method is required to support vendor implementation-dependent handling of hierarchical memory, such as memory common between FPGA and calling process

Q: Should instance pool be limited to a process or be shared among processes?

A: An implementation dependent choice. Each application should be responsible to see that each rc bitstream is loaded for the given application.

Q: What about a test FPGA runtime status capability?

A: This functionality will be added as oa_get_device_status()

Q: What about accelerator initiated data transfers?

A: These are an implementation dependent choice for possible better performance. Receive buffers have been previously allocated and are available to the accelerator to write when data is available. Therefore, the use of this technique is not precluded in the definition, and not required.

Q: Be consistent in naming methods. Either be open accelerator or open fpga but not both.

A: Methods named consistently relative to resource involved in the result. Where possible, method names will be generic.

Q. Which methods are high-level and which are low-level?

Recommend which methods are to be used to create libraries.

What methods should be used in a high-level application development?

What methods should be used in a low-level application development?

A: Following the lead for OpenMP, methods will be provided across levels. Several examples will be provided with best practices in each instance. The choice of how methods are used within the application remains the purview of the developer.

Q: What about a single method to wait for all accelerators?

A: A constant will be defined to use in place of device_id to indicate all devices within a working context.

Q: Can oa_init being called multiple times?

A: Yes. Successful return only if called with a supported version of the interface when environment is in the pre-initialized state. Calls with an unsupported version number will result in an error status.

Q. What is the behavior if an allocated device becomes no longer available before the commit is called?

A: The commit returns an error status if all allocated devices in the environment cannot be committed.

Q: Is the state information thread local or process global in nature?

A: State information is globally accessible by context _id.

