PHANTOM Documentation

Release 1.0

University of York

May 30, 2017

CONTENTS:

CHAPTER

ONE

ACCESSING IP CORES FROM LINUX

PHANTOM FPGA IP cores are accessed by Linux userland software through the following API.

1.1 Common Definitions

PHANTOM_OK
Returned by functions to indicate no error.

PHANTOM_FALSE
Returned by functions to indicate false.

PHANTOM_ERROR
Returned by functions to indicate an otherwise unspecified error.

PHANTOM_NOT_FOUND
Returned by functions to indicate something was not found.

phantom_data_t
A type guaranteed to be wide enough to store an entire word. On Zynq systems this is *uint32_t*, and on a Zynq UltraScale+ device it is *uint64_t*.

phantom_address_t
A type guaranteed to be wide enough to store an entire system address. On Zynq systems this is *uint32_t*, and on a Zynq UltraScale+ device it is *uint64_t*.

PHANTOM_DMA_TO_IP
Specify a DMA transfer from main memory into an IP core.

PHANTOM_DMA_FROM_IP
Specify a DMA transfer from an IP core's address space into main memory.

1.2 Configuring the FPGA

int **phantom_fpga_is_done**()
Check the FPGA's *done* pin, which is asserted when the FPGA is successfully programmed with a bitfile.

Returns

- *PHANTOM_OK* if the FPGA has been configured with a compatible bitfile
- *PHANTOM_FALSE* if the DONE pin is not asserted.

int **phantom_fpga_configure**(FILE **bitfile*)
Configure the FPGA fabric with the specified bitfile. bitfile is a configuration file as generated by the Xilinx tools for the appropriate FPGA part. This function assumes that the FPGA is not currently programmed. Note that this function does not block until programming is complete. *phantom_fpga_is_done()* should be called to determine that programming has completed successfully before the FPGA is used.

Parameters

- **bitfile** (*FILE**) – A FILE pointer to a compatible bitfile, opened in binary mode.

Returns

- *PHANTOM_OK* if configuration started successfully
- *PHANTOM_ERROR* if the bitfile could not be sent to the FPGA.

1.3 The *phantom_ip_t* structure

The PHANTOM IP cores in the current design are represented by instances of the *phantom_ip_t* structure which are obtained by calling *phantom_fpga_get_ips()*.

typedef struct {...} phantom_ip_t;

The structure contains at least the following members:

char ***idstring**
An identifying string.

uint32_t **id**
A unique identifier for the IP.

uint8_t **num_masters**
The number of AXI Masters in the IP.

phantom_address_t **base_address**
The base memory address of the IP core's AXI Slave.

phantom_address_t **address_size**
The size in bytes on the IP core's AXI Slave memory space.

1.4 Reading IP Core Information

int **phantom_fpga_get_num_ips** ()
If the FPGA is currently programmed, this gives the number of PHANTOM IP cores in the current design.

Returns the number of IP cores that are currently configured onto the FPGA. -1 if a problem is encountered.

phantom_ip_t ***phantom_fpga_get_ips** ()
If the FPGA is currently programmed, this fetches an array of *phantom_ip_t* that describe the PHANTOM IP cores in the design, as in the following example:

```
phantom_ip_t *ips = phantom_fpga_get_ips();

for(int i = 0; i < phantom_fpga_get_num_ips(); i++) {
        printf("%d: %s\n", i, ips[i].idstring);
}
```

Returns An array of the PHANTOM IPs in the currently-programmed design, or *NULL* if none exist.

int **phantom_fpga_ip_initialise** (phantom_ip_t **ip*, const char **idstring*)
If the FPGA is currently programmed, search for the PHANTOM IP core with the provided *idstring*. The *ip* struct is then initialised.

Parameters

- **ip** (*phantom_ip_t**) – A *phantom_ip_t* struct to be initialised.
- **idstring** (*char**) – The identifier string of the IP.

Returns

- *PHANTOM_OK* if the IP is successfully initialised.
- *PHANTOM_NOT_FOUND* if the FPGA is correctly programmed, but the requested core cannot be found.
- *PHANTOM_ERROR* if another error occurs.

int **phantom_fpga_ip_initialise_by_id**(phantom_ip_t *ip, uint32_t *id*)

As *phantom_fpga_ip_initialise()*, but initialises a core based on its unique id. This is useful for when multiple copies of the same IP core are present. These can be queried through the use of *phantom_fpga_get_ips()*.

Parameters

- **ip** (*phantom_ip_t**) – A *phantom_ip_t* struct to be initialised.
- **id** (*uint32_t*) – The unique identifier of the IP.

Returns

- *PHANTOM_OK* if the IP is successfully initialised.
- *PHANTOM_NOT_FOUND* if the FPGA is correctly programmed, but the requested core cannot be found.
- *PHANTOM_ERROR* if another error occurs.

1.5 IP Control

int **phantom_fpga_ip_start**(phantom_ip_t* *ip*)

Starts the specified IP. Has no effect if the core is already started.

Parameters

- **ip** (*phantom_ip_t**) – The IP core to control.

Returns *PHANTOM_OK* if the core started successfully, or *PHANTOM_ERROR* if not.

int **phantom_fpga_ip_is_done**(phantom_ip_t* *ip*)

Checks if the specified IP has completed its execution.

Parameters

- **ip** (*phantom_ip_t**) – The IP core to query.

Returns *PHANTOM_OK* if the core stopped successfully, or *PHANTOM_FALSE* if not.

int **phantom_fpga_ip_is_idle**(phantom_ip_t* *ip*)

Checks if the specified IP is idle and ready for I/O or to be started.

Parameters

- **ip** (*phantom_ip_t**) – The IP core to query.

Returns *PHANTOM_OK* if the core is idle, or *PHANTOM_FALSE* if not.

1.6 IP Data I/O

PHANTOM IPs are presented to the Linux kernel as Userland IO (UIO) devices, and so the entire address space can be mapped and accessed using simple get and set functions. This is not the most efficient way to move large volumes of data however. The IP core itself can read and write from main system memory through an AXI master interface. Therefore the UIO interface should be used to pass parameters and configuration data, and the device itself should fetch input data as required using AXI burst transfers.

It is also possible for the hardware design to include an AXI DMA controller to support automatic bulk data movement. This core can be controlled through the UIO interface through the simple helper functions provided here. More complex DMA transfers can be coded directly.

int **phantom_fpga_ip_set**(phantom_ip_t* *ip*, *phantom_address_t addr*, *phantom_data_t val*)

Set a value inside the AXI Slave address space of the IP. *addr* is based from 0 and will be automatically offset to the appropriate base address (*phantom_ip_t.base_address*).

Parameters

- **ip** (*phantom_ip_t**) – The IP core to query.
- **addr** (phantom_address_t) – The address, based at 0, inside the address space of the IP core.
- **val** (phantom_data_t) – The value to set.

Returns *PHANTOM_OK* if the value was set, or *PHANTOM_ERROR* if not.

phantom_data_t **phantom_fpga_ip_get**(phantom_ip_t* *ip*, *phantom_address_t addr*)

Get a value from the AXI Slave address space of the IP. *addr* is based from 0 and will be automatically offset to the appropriate base address (*phantom_ip_t.base_address*).

Parameters

- **ip** (*phantom_ip_t**) – The IP core to query.
- **addr** (phantom_address_t) – The address, based at 0, inside the address space of the IP core.

Returns The value of the argument specified by *addr*.

int **phantom_fpga_dma_transfer**(phantom_ip_t* *ip*, *phantom_address_t dma_core*, *phantom_address_t buffaddr*, *phantom_address_t length*, int *direction*)

Cause a DMA core in the specified IP core to initiate a DMA transfer. This function assumes that an AXI DMA IP core is located at the appropriate address in the memory space of the target IP. This function returns immediately and the transfer will begin a time after this. For more details consult the Xilinx DMA Core driver.

Parameters

- **ip** (*phantom_ip_t**) – The IP core to query.
- **dma_core** (phantom_address_t) – The offset, starting at 0, of the DMA core inside the address space of the IP.
- **buffaddr** (phantom_address_t) – The address in main memory to start the transfer from.
- **length** (phantom_address_t) – The length in bytes of the transfer.
- **direction** (*int*) – The direction of the transfer. Valid values are *PHANTOM_DMA_TO_IP* or *PHANTOM_DMA_FROM_IP*

Returns *PHANTOM_OK* if the transfer was started, or *PHANTOM_FALSE* if not.

int **phantom_fpga_dma_is_idle**(phantom_ip_t* *ip*, *phantom_address_t dma_core*, int *direction*)

Check if the DMA core in the specified IP core is idle in the given direction. Because the Xilinx AXI DMA core is bidirectional, it is possible for a transfer to be proceeding in one direction whilst the other is idle.

Parameters

- **ip** (*phantom_ip_t**) – The IP core to query.
- **dma_core** (phantom_address_t) – The offset, starting at 0, of the DMA core inside the address space of the IP.
- **direction** (*int*) – The direction of the transfer. Valid values are *PHANTOM_DMA_TO_IP* or *PHANTOM_DMA_FROM_IP*

Returns *PHANTOM_OK* if the core is idle, or *PHANTOM_FALSE* if not.

INDEX