FPGA Shell Interface Specification (For xvu9p)

Revision 0.13

Describes the Customer Logic interfaces, and design specification.

1 Revision History

Revision	Data	Content
0.9	7/8/2016	• Initial Alpha release
0.10	9/30/2016	• Add PCIe tag restriction • Removed PCIe link information • Added MSI-X • Update block diagram • Added CTL/ID/Status interfaces
0.11	10/10/2016	• Added AXI-4 address restriction
0.12	10/14/2016	• Added reference for exact architecture version (xvu9p) • Changed HL→shell • Clarified role of PCI FLR in user PF • Add error counter and status place holder • Increased MSI-X to 48, with 32 typically allocated for DMA and the rest are for the user • Increase AXI bus to PCIe AxUSER from 10-bit to 11-bit • Refined BAR2 size and added BAR4 • Defined two generic control register and two generic status registers between Shell and CL that the user can access through the management PF • Explicit description of the features of UserPF vs ManagementPF • Error handling on AXI buses • Clarify how AXI4 will present access from BAR0 and BAR2 • ManagementPF have fixed Amazon PCIe VenID/DevID while UserPF have programmable VenID/DevID • Changed AXI-Lite to AXI-4 from Shell to CL • Add byte enable capability on CL PCIe master interface • Added DDR4 Self Refresh (future)
0.13	10/18/2016	• Clarify which interfaces are in CL • Add clock creation example • Clarify Management PF is independent of CL • Implementation note on BAR size • Added pwr_state from HL to CL

2 Overview

The AWS FPGA instance provides FPGA acceleration capability to AWS compute instances. Each FPGA is divided into two partitions:

- Shell (SH) – AWS platform logic responsible for taking care of the FPGA external peripherals, PCIe, and Interrupts.
- Customer Logic (CL) – Custom acceleration logic created by an FPGA Developer

At the end of the development process, the Shell and CL will become an Amazon FPGA Image (AFI)

This document specifies the hardware interface and functional behavior between the Shell and the CL.

While there are multiple versions and multiple generations of the FPGA-accelerated EC2 instances, the rest of this document focuses on the Shell design for xvu9p architecture used in EC2 F1 instance.

For full details of the available FPGA enabled instances please refer to:

http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/FPGA.html [TBD - subject to change]

2.1 Architecture and version

This specification applies to the Xilinx Virtex Ultrascale Plus platform, referred to in AWS APIs and the HDK release as FpgaImageArchitecture=xvu9p.

The Shell is tagged with a revision number. Note while AWS tries to keep the revision constant, sometimes it is necessary to update the revision due to discovered issues or added functionality. The HDK release includes the latest Shell version.

New shell versions require updated CL implementation.

2.2 Convention

CL – Customer's Logic: the Logic to be provided by the developer and integrated with the Shell.

DW – Doubleword: referring to 4-byte (32-bit) data size

AXI – ARM Advanced eXtensible Interface.

AXI Stream – ARM Advanced eXtensible Stream Interface

M – Typically refers to the Master side of an AXI bus

S – Typical refers to the Slave side of AXI bus

3 CL (for xvu9p architecture as in EC F1 instances)

The F1 FPGA platform includes the following interfaces available to the CL:

- One x16 PCIe Gen 3 Interface
- Four DDR4 RDIMM interfaces (with ECC)
- Optional: Two 2GB half width (8-lane) HMC interfaces @ 12.5Gb/s per lane [Subject to change]

- Optional: Four inter-FPGA links (not available on all F1 instance sizes)

3.1 CL/Shell Interfaces (AXI-4)

All interfaces except the inter-FPGA links use the AXI-4 protocol. The AXI-4 interfaces in the Shell have the following restrictions:

- AxSIZE – All transfers must be the entire width of the bus. While byte-enables bitmap are supported, it must adhere to the interface protocol (i.e. PCIe contiguous byte enables on all transfers larger than 64-bits)
- AxBURST – Only INCR burst is supported.
- AxLOCK – Lock is not supported.
- AxCACHE – Memory type is not supported.
- AxPROT – Protection type is not supported
- AxQOS – Quality of Service is not supported
- AxREGION – Region identifier is not supported

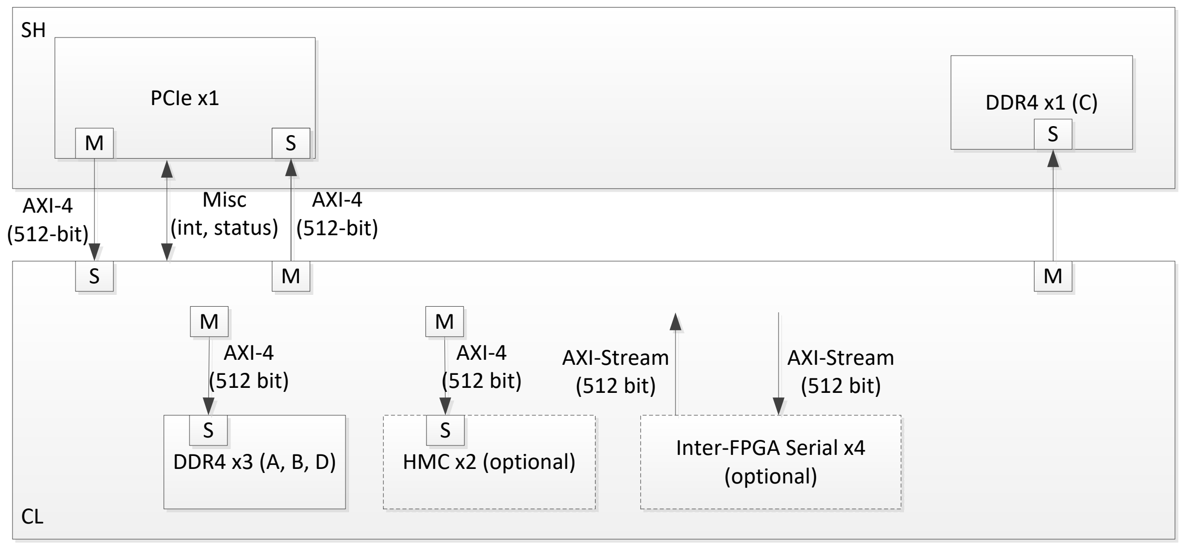


Figure 1 Interfaces (note not all interfaces are available in all instances)

3.1.1 Interfaces implemented in CL

Some of the interfaces are implemented in the CL rather than the HL. For those interfaces, the designs are provided by AWS and instantiated in the CL. Note this is due to physical constraints.

There are four DDR interfaces labeled A, B, C, and D. Interfaces A, B, and D are in the CL and interface C is implemented in the CL. A design block (sh_ddr.sv) instantiates the three DDR interfaces in the CL (A, B, D).

The optional HMC interfaces are implemented in the CL. There is a design block (hmc_wrapper.sv) that instantiates the HMC interfaces.

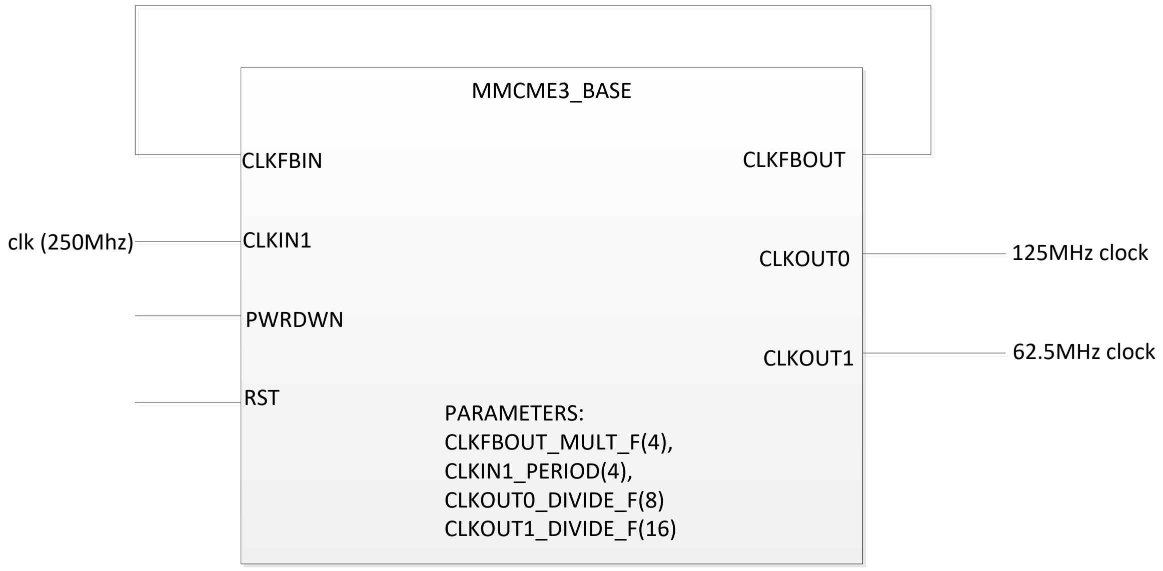
The optional inter-FPGA serial links are implemented in the CL. There is a design block (aurora_wrapper.sv) that instantiates the inter-FPGA links.

For interfaces that are implemented in the CL, the AXI-4 (or AXI-Stream) interfaces do not connect into the SH, but connect locally inside the CL to the AWS provided blocks. There are also statistics interfaces that must be connected from SH to the interface blocks.

3.2 Clocking/Reset

A single 250MHz clock, and associated asynchronous reset is provided to the CL. All Shell interfaces are synchronous to the 250MHz clock. The CL can derive clocks off of the 250MHz clock (i.e. use an MMCM to create clocks of other frequencies/phases). The reset signal combines the board reset and PCIe reset conditions.

Below is an example of generating slower clocks using the Xilinx Mixed Mode Clock Manager (MMCM) IP. In this example the supplied 250MHz clock is divided down into a 125MHz clock and a 62.5MHz clock. Please refer to the Xilinx documentation (ug974) for more information.



3.2.1 FLR

FLR is supported for the User Physical Function using a separate FLR interface:

- sh_cl_flr_assert – Level signal that is asserted when FLR has been requested
- cl_sh_flr_done – Asserted for a single clock to acknowledge the FLR. This must be asserted in response to sh_cl_flr_assert. Note due to pipeline delays it is possible sh_cl_flr_assert is asserted for some number of clocks after cl_sh_flr_done.

3.3 PCIe Endpoint Presentation to Instance

There are two physical functions (PFs) presented to the user's operating system:

- FPGA Management PF –This PF allows for management of the FPGA including tracking FPGA state and loading CL images onto the FPGA.
- User PF – Customer's PF for Customer specific logic (CL)

3.3.1 Management PF

The management PF is a separate PF from the CL PF. Details are provided for reference. The Management PF exposes:

a) Amazon's specific and fixed PCIe VendorID and DeviceID
b) Two BARs with 4KB size
c) Single MSI-X interrupt
d) No BusMaster support
e) A range of 32-bit addressable registers

The Management PF is persistent throughput the life of the instance, and it will not be reset or cleared (even during the AFI attach/detach process).

3.3.2 User PF

The User PF exposes the following:

a) Developer's specific PCIe VendorID, DeviceID, SystemID and SubsystemID as registered through ***aws ec2 fpgaImageCreate*** command parameters.
b) PCIe BAR0 as a 64-bit prefetchable BAR sized as 64GB (*note the BAR size is subject to change, goal is 64GB, but will be no smaller than 128MB)*. This BAR may be used to map the entire External/Internal memory space to the instance address space if desired, through mmap() type calls.
c) PCIe BAR4 as a 64-bit prefetchable BAR sized as 32MB. This BAR is allows for mapping a different software/driver process than is mapped through BAR0. For example, this could be used to map to a DMA controller.
d) PCIe BAR2 as a 64-bit prefetchable BAR sized as 4KB for the MSI-X interrupt tables
e) FLR capability that will reset the CL
f) BusMaster capability to allow CL to master transactions towards the instance memory
g) Up to 48 MSI-X vectors

The Developer can write a drivers for the User PF or can leverage the reference driver provided in the HDK (driver included in Amazon Linux).

3.3.2.1 BAR0 Size implementation note

The CL design determines the mapping of BAR0 to resources in the CL. It is possible to use any amount of the BAR0 address space in the CL logic. The BAR0 size is large to allow for implementations that desire to directly map a large portion of the CL RAM space directly to the instance address space (up to 64GB of DDR4 and 4GB of HMC).

3.4 PCIe Interface between Shell and CL

The PCIe interface between the Shell and CL is accessed over two AXI-4 interfaces:

3.4.1 AXI-4 for inbound PCIe transactions (Shell is master, CL is slave)

This AXI-4 bus is for PCIe transactions mastered by the Instance and targeting User PF BAR0 or BAR4 memory space.

It is a 512-bit wide AXI-4 interface that supports 32-bit transactions only.

AxUSER[0] on this interface signals whether the address targeted BAR0 (AxUSER[0] == 0), or BAR4 (AxUSER[0] == 1).

A read or write request on this AXI-4 bus that is not acknowledged by the CL within a certain time window, will be internally terminated by the Shell. If the time-out error happens on a read, the Shell will return 0xDEADBEEF data back to the instance. This error is reported to the Management functions.

Note in future revisions this interface will support larger burst sizes (up to the Maximum Payload Size).

3.4.2 AXI-4 for outbound PCIe transactions (CL is master, Shell is slave)

This is a 512-bit wide AXI-4 Interface for the CL to master cycles to the PCIe bus. This is used, for example, to DMA data to/from instance memory.

The following PCIe interface configuration parameters are provided from the Shell to the CL, and the CL logic must respect these maximum limits:

- sh_cl_cfg_max_payload[1:0] – PCIe max payload size:
    - 2'b00 – 128 Byte
    - 2'b01 – 256 Byte (Most probable value)
    - 2'b10 – 512 Byte
    - 2'b11 – Reserved
- sh_cl_cfg_max_read_req[2:0]
    - 3'b000 – 128 Byte
    - 3'b001 – 256 Byte
    - 3'b010 – 512 Byte (Most probable value)
    - 3'b011 – 1024 Byte
    - 3'b100 – 2048 Byte
    - 3'b101 – 4096 Byte

The PCIe CL to Shell AXI-4 interfaces implement “user” bits on the address channels (AxUSER[18:0]).

- AxUSER[10:0] – DW length of the request. This is 1-based (0-zero DW, 1-one DW, 2-two DW, etc…)
- AxUSER[14:11] – First Byte enable for the Request
- AxUSER[18:15] – Last Byte enable for the Request

PCIe AXI-4 interface restrictions:

- Transfers must not violate PCIe byte enable rules (see byte enables below)
- Transfers must not cross a 4Kbyte Address boundary (PCIe restriction)
- Transfers must not violate Max Payload Size

- Read requests must not violate Max Read Request Size
- A read request transaction must not be issued using the same ARID (AXI4 Read ID), if that ARID is already outstanding. <u>The Shell does not force ordering between individual read transactions</u>.
- The PCIe interface supports 5-bit ARID (32 outstanding read transactions maximum). Note PCIe extended tag is not supported on the PCIe interface.
- The address on the AXI-4 interface must reflect the correct DW address of the transfer. The logic does not support using a 64B aligned address, and using STRB to signal the actual starting DW.
- The first/last byte enables are determined from the AxUSER bits, but for writes the WSTRB signal must be correct and reflect the appropriate valid bytes.

3.4.3 Byte Enable Rules

All PCIe transactions must adhere to the PCIe Byte Enable rules (see PCI Express Base specification). Rules are summarized below:

- All transactions larger than two DW must have contiguous byte enables
- Transactions that are less than two DW may have non-contiguous byte enables

3.4.4 AXI4 Error handling

Transaction on AXI4 interface will be terminated and reported as SLVERR on the RRESP/BRESP signals and will not passed to the instance in the following cases:

- PCIe BME (BusMaster Enable) is not set in the PCIe configuration space
- Illegal transaction address (Addressing memory space that's not supported by the instance)
- Transaction crossing 4KB boundaries
- Illegal byte-masking
- Illegal length
- Illegal address alignment
- Illegal ARID (ARID is already been used for an outstanding read transaction)

If the CL initiates a PCIe read-request through the AXI4 bus, and the read response suffers a time-out or an uncorrectable PCIe error, the Shell will complete the request and signal SLVERR on the RRESP signal.

3.4.5 Interrupts (MSI-X)

Only MSI-X interrupts are supported. Up to 48 vectors are supported. It is up to software to determine how many vectors have been allocated and to configure the CL logic appropriately. There is a simple interface to generate MSI-X interrupts:

- cl_sh_msix_int – Pulse this signal to generate an MSI-X interrupt. (CL to Shell)
- cl_sh_msix_vec[7:0] – Vector number to generate the MSI-X interrupt. Note only values 0-47 are valid, all other values are reserved. This is sampled with cl_sh_msix_int. (CL to Shell)
- shell_cl_msix_int_ack – CL will receive a pulse on this signal when the interrupt request has been processed. There can only be a single request outstanding at a time (when CL pulses cl_sh_msix_int, it must wait for sh_cl_msix_int_sent before pulsing cl_sh_msix_int again even if for a different vector).

- sh_cl_msix_int_sent – CL will receive this status with sh_cl_msix_int_ack. If ‘1’ it means the MSI-X was successfully transmitted on the PCIe bus. If ‘0’, it means the MSI-X was not transmitted on the PCIe bus (for example if MSI-X was not enabled, or the vector/function was masked).

3.5 DDR-4 Interface

Each DDR-4 interface is accessed over an AXI-4 interface:

- AXI-4 (CL Master) – 512-bit AXI-4 interface to read/write DDR

There is a single status signal that he DDR is trained and ready for access. The addressing uses ROW/COLUMN/BANK mapping of AXI address to DDR Row/Col/Bank. The Read and Write channels are serviced with round robin priority (equal priority).

The DDR-4 interface uses the Xilinx DDR-4 controller. The AXI-4 interface adheres to the Xilinx specification. User bits are added to the read data channel to signal ECC errors with the read data.

3.5.1 DDR Preservation (Future)

In future Shell versions a DDR preservation feature will be implemented. This feature allows the DDR state to be preserved when dynamically changing CL logic. The current Shell version will not guarantee preservation of DDR contents if the CL logic is re-loaded.

3.6 HMC Interface

Each HMC interface is accessed over an AXI-4 interface (details TBD in a future version of this specification):

- AXI-4 (CL Master) – 512-bit AXI-4 interface to read/write HMC

3.7 Inter-FPGA Link

In some of the instances with multiple FPGA, a bi-directional ring runs between the FPGAs. There are 4 inter-FPGA QSFP links. These links are x4 links (4 bi-directional lanes per link) running at 25Gb/s per lane. *Goal is 25Gb/s, based on signal integrity actual rate may be less than 25Gb/s.* The Shell takes care of abstracting most of the complexities of the serial links from the CL. The Shell implements:

- Clock compensation
- Lane/Data alignment
- Data Encapsulation
- Flow Control
- Bit error detection

AXI-Stream interfaces are used to transmit/receive data. The receive AXI-Stream interface adds a CRC_ERROR signal to signal there were bit errors in a packet and that the packet should be discarded. Note the Shell only implements bit error detection, it does not implement bit error correction (i.e. retry protocol for errored packets). If a reliable transport is desired, that must be implemented in the CL.

The inter-FPGA links are connected in a ring topology. Any “routing” between the FPGAs is also implemented in the CL. For example if FPGA#0 needs to send data to FPGA#3, then FPGA#1 and FPGA#2 will pass the data along the ring, and FPGA#3 will consume the data. Arbitration for the serial links for pass through and locally generated traffic is also implemented in the CL.

3.8 Miscellaneous signals

There are some miscellaneous generic signals between Shell and CL.

3.8.1 PCIe IDs

There are some signals that must have the PCIe IDs of the CL:

- cl_sh_id0
  - [15:0] – Vendor ID
  - [31:16] – Device ID
- cl_sh_id1
  - [15:0] – Subsystem Vendor ID
  - [31:16] – Subsystem Device ID

3.8.2 General control/status

The functionality of these signals is TBD.

- cl_sh_status0[31:0] – CL to Shell status
- cl_sh_status1[31:0] – CL to Shell status
- sh_cl_ctl0[31:0] – Shell to CL control information
- sh_cl_ctl1[31:0] – Shell to CL control information
- sh_cl_pwr_state[1:0] – This is the power state of the FPGA. 0x0
  - 0x0 – Power is normal
  - 0x1 – Power level 1
  - 0x2 – Power level 2
  - 0x3 – Power is critical and FPGA is subject to shutting off clocks or powering down

3.9 Memory address map for Management PF

TBD.