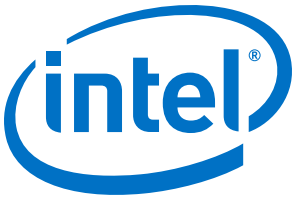


DMA Accelerator Functional Unit (AFU) User Guide

Updated for Intel® Acceleration Stack for Intel® Xeon® CPU with FPGAs: **2.0**

Subscribe
Send Feedback

UG-20207 | 2019.08.05
Latest document on the web: **PDF** | **HTML**

Contents

1. About this Document

1.1. Intended Audience

The intended audience comprises hardware or software developers that require an Accelerator Function (AF) to buffer data locally in memory connected to the Intel FPGA device.

1.2. Conventions

Table 1. Document Conventions

Convention	Description
#	Precedes a command that indicates the command is to be entered as root.
$	Indicates a command is to be entered as a user.
`This font`	Filenames, commands, and keywords are printed in this font. Long command lines are printed in this font. Although long command lines may wrap to the next line, the return is not part of the command; do not press enter.
<variable_name>	Indicates the placeholder text that appears between the angle brackets must be replaced with an appropriate value. Do not enter the angle brackets.

1.3. Acronyms

Table 2. Acronyms

Acronyms	Expansion	Description
AF	Accelerator Function	Compiled Hardware Accelerator image implemented in FPGA logic that accelerates an application.
AFU	Accelerator Functional Unit	Hardware Accelerator implemented in FPGA logic which offloads a computational operation for an application from the CPU to improve performance.
API	Application Programming Interface	A set of subroutine definitions, protocols, and tools for building software applications.
CCI-P	Core Cache Interface	CCI-P is the standard interface AFUs use to communicate with the host.
DFH	Device Feature Header	Creates a linked list of feature headers to provide an extensible way of adding features.
FIM	FPGA Interface Manager	The FPGA hardware containing the FPGA Interface Unit (FIU) and external interfaces for memory, networking, etc.

continued...

Acronyms	Expansion	Description
		The Accelerator Function (AF) interfaces with the FIM at run time.
FIU	FPGA Interface Unit	FIU is a platform interface layer that acts as a bridge between platform interfaces like PCIe*, UPI and AFU-side interfaces such as CCI-P.
MPF	Memory Properties Factory	The MPF is a Basic Building Block (BBB) that AFUs can use to provide CCI-P traffic shaping operations for transactions with the FIU.

1.4. Acceleration Glossary

Table 3. Acceleration Stack for Intel® Xeon® CPU with FPGAs Glossary

Term	Abbreviation	Description
Intel® Acceleration Stack for Intel Xeon® CPU with FPGAs	Acceleration Stack	A collection of software, firmware, and tools that provides performance-optimized connectivity between an Intel FPGA and an Intel Xeon processor.
Intel FPGA Programmable Acceleration Card (Intel FPGA PAC)	Intel FPGA PAC	PCIe FPGA accelerator card. Contains an FPGA Interface Manager (FIM) that pairs with an Intel Xeon processor over the PCIe bus.

2. DMA AFU Description

2.1. Introduction

The Direct Memory Access (DMA) AFU example shows how to manage memory transfers between the host processor and the FPGA. You can integrate the DMA AFU into your design to move data between the host memory and the FPGA local memory.

The DMA AFU comprises the following submodules:

- Memory Properties Factory (MPF) Basic Building Block (BBB)
- Core Cache Interface (CCI-P) to the Avalon® Memory-Mapped (Avalon-MM) Adapter
- DMA Test System which contains the DMA BBB

These submodules are described in more detail in the *DMA AFU Hardware Components* topic below.

Related Information

- The DMA AFU Hardware Components on page 6
- Avalon Interface Specifications
  For more information about the Avalon-MM protocol, including timing diagrams for read and write transactions.

2.2. The DMA AFU Software Package

The Intel Acceleration Stack for Intel Xeon CPU with FPGAs package file (`*.tar.gz`), includes the DMA AFU example. This example provides a user space driver. The host application uses this driver such that the DMA moves data between host and FPGA memory. The hardware binaries, sources, and the user space driver are available in the following directory: *$OPAE_PLATFORM_ROOT*`/hw/samples/dma_afu` .

Before experimenting with the DMA AFU, you must install the Open Programmable Acceleration Engine (OPAE) software package. Refer to *Installing the OPAE Software Package* in the *Intel Acceleration Stack Quick Start Guide for Intel FPGA Programmable Acceleration Card D5005* for installation instructions. This *Quick Start Guide* also includes basic information about the Open Programmable Acceleration Engine (OPAE) and configuring an AFU.

After installing the Open Programmable Acceleration Engine (OPAE) software package, a sample host application and the DMA AFU user space driver are available in the following directory: *$OPAE_PLATFORM_ROOT*`/hw/samples/dma_afu/sw`. To run the sample host application, `fpga_dma_test` on your Intel FPGA PAC D5005 hardware, refer to the steps in section *Running the DMA AFU Example*.

Related Information

- Intel Acceleration Stack Quick Start Guide for Intel FPGA Programmable Acceleration Card D5005
- Installing the OPAE Software Package

2.3. The DMA AFU Hardware Components

The DMA AFU interfaces with the FPGA Interface Unit (FIU) and FPGA memory.

Refer to the *FPGA Interface Manager (FIM) Data Sheet* for detailed specifications of the FPGA memory.

Note: The currently available hardware dictates this memory configuration. Future hardware may support different memory configurations.

You can use the DMA AFU to copy data between the following source and destination locations:

- The host to device FPGA memory
- Device FPGA memory to the host

A Platform Designer system, `<installation path>/hw/samples/dma_afu/hw/rtl/TEST_dma/<device>/dma_test_system.qsys` implements most of the DMA AFU.

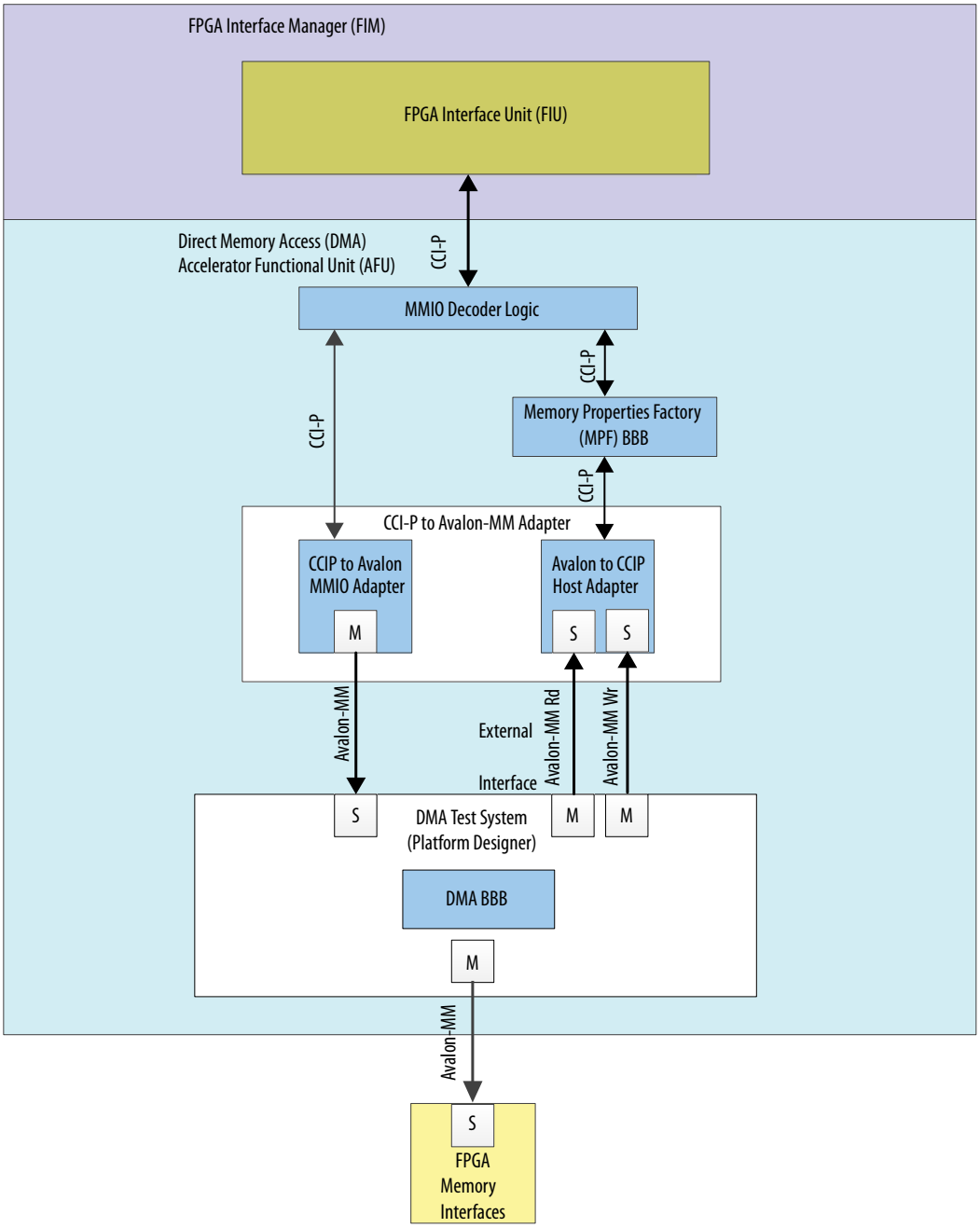
Part of the DMA AFU implemented in the Platform Designer system can be found in the following location:

```
<installation path>/hw/samples/dma_afu/hw/rtl/TEST_dma/<device>
```

You can find the DMA BBB in the following location:

```
<installation path>/hw/samples/dma_afu/hw/rtl/dma_bbb
```

Figure 1. DMA AFU Hardware Block Diagram



The DMA AFU includes the following internal modules to interface with the FPGA Interface Unit (FIU):

- Memory-Mapped IO (MMIO) Decoder Logic: detects MMIO read and write transactions and separates them from the CCI-P RX channel 0 that they arrive from. This ensures that MMIO traffic never reaches the MPF BBB and is serviced by an independent MMIO command channel.
- Memory Properties Factory (MPF): This module ensures that read responses from the DMA return in the order that they were issued. The Avalon-MM protocol requires read responses to return in the correct order.
- CCI-P to Avalon-MM Adapter: This module translates between CCI-P and Avalon-MM transactions, as follows:
  - CCI-P to Avalon-MMIO Adapter: This path translates CCI-P MMIO transactions into Avalon-MM transactions.
  - Avalon to CCI-P Host Adapter: These paths create separate read-only and write-only paths for the DMA to access host memory.
- DMA Test System: This module serves as a wrapper around the DMA BBB to expose the DMA masters to the rest of the logic in the AFU. It provides the interface between the DMA BBB and the CCI-P to Avalon Adapter. It also provides the interface between the DMA BBB and the local FPGA SDRAM banks.

Related Information

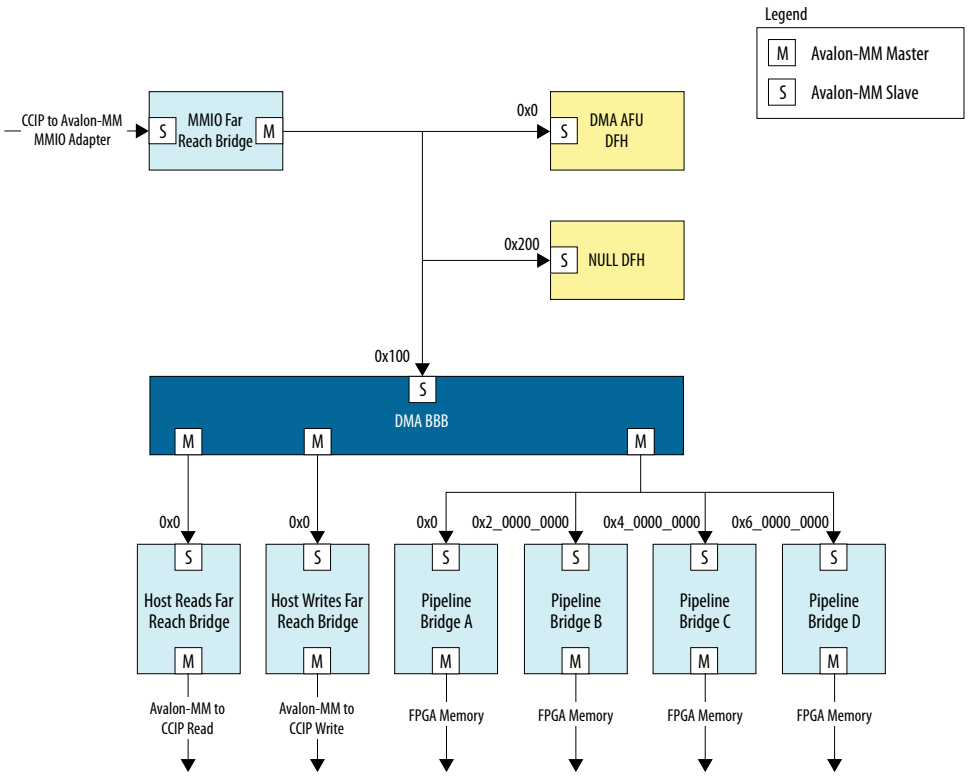
FPGA Interface Manager Data Sheet for Intel FPGA Programmable Acceleration Card D5005

2.3.1. DMA Test System

The DMA test system connects the DMA BBB to the rest of the FPGA design including CCI-P adaptation and the local FPGA memory.

Figure 2. DMA Test System Block Diagram

This block diagram shows the internals of the DMA test system. The DMA test system is shown as a monolithic block in Figure 1 on page 7.



The DMA test system includes the following internal modules:

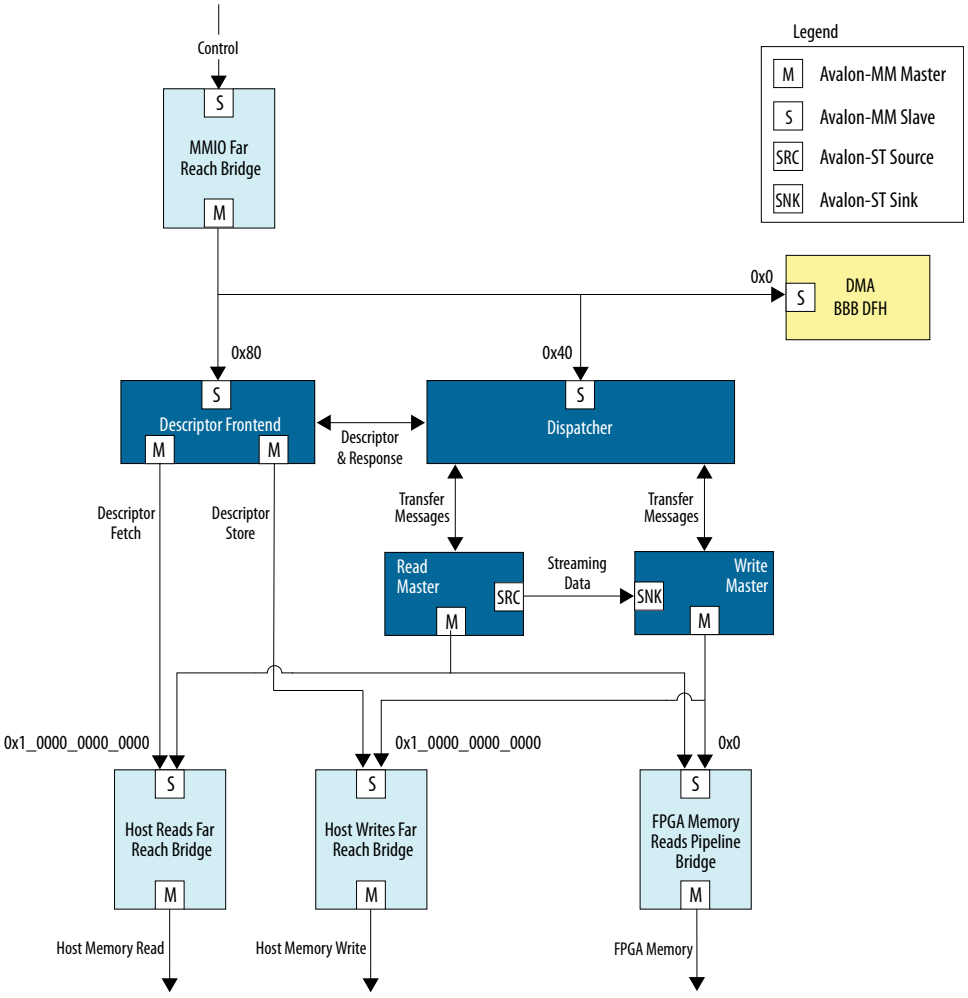
- Far Reach Bridge/Pipeline Bridge: A pipeline bridge with adjustable latency included to control topology and improve the design F_{max}.
- DMA AFU Device Feature Header (DFH): This is a DFH for the DMA AFU. This DFH points to the next DFH located at offset 0x100 (DMA BBB DFH).
- Null DFH: This component terminates the DFH linked-list. If you add more DMA BBBs to the design, ensure that the null DFH base address is located at the end of the DFH linked-list.
- DMA Basic Building Block (BBB): This block moves data between the host and the local FPGA memory. It also accesses host memory to access descriptor chains.

2.3.2. DMA BBB

The DMA BBB subsystem transfers data from source to destination addresses using Avalon-MM transactions. The DMA driver controls the DMA BBB by accessing the control and status register of the various components inside the system. The DMA driver also controls the DMA BBB by using shared memory to communicate transfer descriptors. The DMA BBB accesses data in FPGA memory at offset 0x0. The DMA BBB accesses data and descriptors in host memory at offset 0x1_0000_0000_0000.

Figure 3. DMA BBB Platform Designer Block Diagram

This block diagram excludes some internal Pipeline Bridge IP cores.



The components in the DMA BBB Platform Designer implement the following functions:

- Far Reach Bridge/Pipeline Bridge: A pipeline bridge with adjustable latency included to control topology and improve the design F_{max}.
- DMA BBB DFH: This is a device feature header for the DMA BBB. This DFH points to the next DFH located at offset 0x100 (Null DFH).
- Descriptor Frontend: Responsible for fetching descriptors and transferring them to the Dispatcher. When a DMA transfer completes the frontend receives status information from the Dispatcher and overwrites the descriptor in host memory.
- Dispatcher: This block schedules DMA transfers requests to the Read and Write Master.
- Read Master: This block is responsible for reading data from host or local FPGA memory and sending it as streaming data to Write Master.
- Write Master: This block is responsible for receiving streaming data from the Read Master and writing the contents to host or local FPGA memory.

3. Register Map and Address Spaces

The DMA AFU supports two memory views: The DMA view and the host view.

The DMA view supports a 49-bit address space. The lower half of the DMA view maps to the local FPGA memory. The upper half of the DMA view maps to host memory.

The host view includes all the registers accessible through MMIO accesses such as the DFH tables, and the control/status registers of the various IP cores used inside the DMA AFU.

The MMIO registers in the DMA BBB and AFU support 32- and 64- bit access. The DMA AFU does not support 512-bit MMIO accesses. Accesses to the Dispatcher registers inside the DMA BBB must be 32 bits (Descriptor frontend implements 64-bit registers).

3.1. DMA AFU Register Map

The DMA register map provides the absolute addresses of all the locations within the unit. These registers are in the host view because it's only the host that can access them.

Table 4. DMA AFU Memory Map

Byte Address Offsets	Name	Span in Bytes	Description
0x0	DMA AFU DFH	0x40	Device feature header for the DMA AFU. The ID_L is set to 0x9081f88b8f655caa and ID_H is set to 0x331db30c988541ea. The DMA AFU DFH has been parameterized to point to offset 0x100 to find the next DFH (DMA BBB DFH). You must not modify the base address of the DMA AFU DFH since it must be located at address 0x0 as defined by the CCIP specification.
0x100	DMA BBB	0x100	Specifies DMA BBB control and status register interface. You can refer to the DMA BBB register map for more information. Within the DMA BBB at offset 0 the DMA BBB includes it's own DFH. This DFH has been set to find the next DFH at offset 0x100 (NULL DFH). If you add more DMA BBBs, space them 0x100 apart and ensure the NULL DFH follows the last DMA by 0x100.
0x200	NULL DFH	0x40	Terminates the DFH linked-list. The ID_L is set to 0x90fe6aab12a0132f and ID_H is set to 0xda1182b1b3444e23. The NULL DFH has been parameterized to be the last DFH in hardware. For this reason the NULL DFH is located at address 0x200. If you add additional DMA BBBs to the system, you need to increase the NULL DFH base address accordingly so that it remains at the highest address. The DMA driver and test application do not use this hardware.

Table 5. DMA BBB Memory Map

The following byte addresses are relative offsets from the DMA BBB base address in the DMA AFU system (0x100).

Byte Address Offsets	Name	Span in Bytes	Description
0x0	`DMA BBB DFH`	0x40	Device feature header for the DMA AFU. The `ID_L` is set to 0xa9149a35bace01ea and `ID_H` is set to 0xef82def7f6ec40fc . The DMA BBB DFH has been parameterized to point to 0x100 for the next DFH offset. This next offset can be another DMA BBB, another DFH (not included in this design), or the NULL DFH.
0x40	`Dispatcher`	0x40	Control port for the dispatcher. The DMA driver uses this location to control the DMA or query its status.
0x80	`Descriptor Frontend`	0x40	The descriptor frontend is a custom component that reads descriptors from host memory and overwrites the descriptor when the DMA transfer completes. The driver instructs the frontend where the first descriptor lives in host memory and then the frontend hardware communicates with the driver primarily though descriptors stored in host memory.

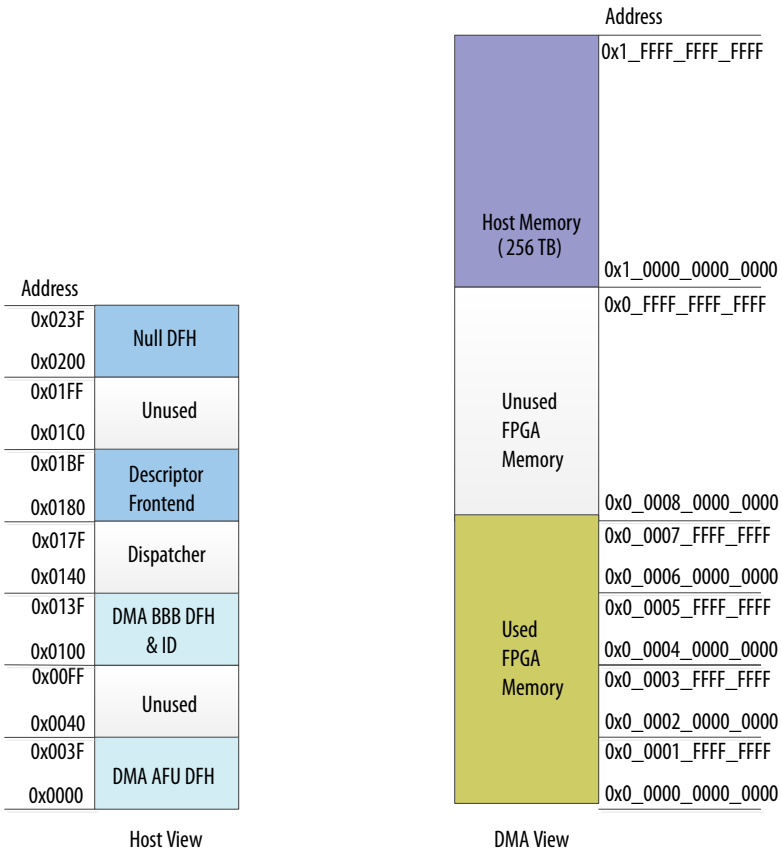
3.1.1. DMA AFU Address Space

The host can access registers listed in the Table 4 on page 12 and the Table 5 on page 13.

The DMA BBB subsystem has access to the full 49-bit address space. The lower half of this address space includes the local FPGA memories. The upper half of this address space includes the 48-bit host address memory.

The following figure shows the host and DMA views of memory.

Figure 4. The DMA AFU and Host Views of Memory

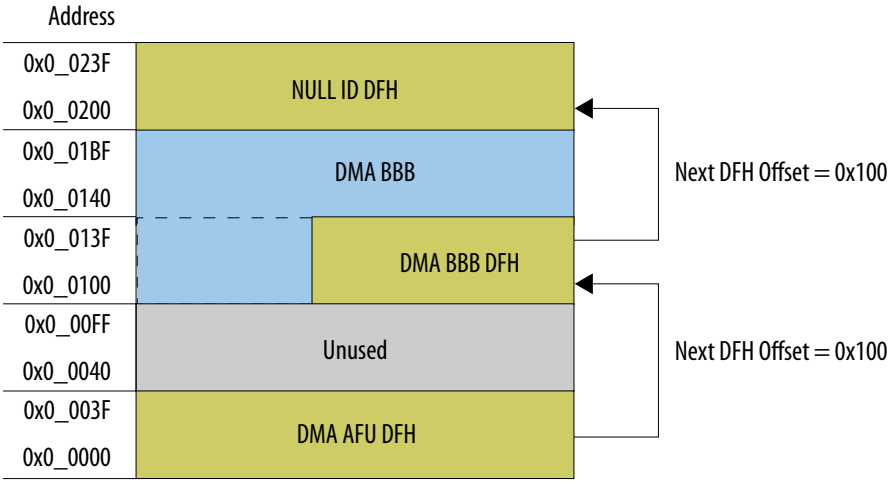


3.1.2. Device Feature Header Linked-List

The DMA AFU design example contains three device feature headers (DFH) that form a linked list. This linked list allows the sample application to identify the DMA AFU as well as the driver to identify the DMA BBB.

The DFH list includes a NULL DFH at the end. The inclusion of the null DFH at the end of the linked list allows you to add more DMA BBBs to your design. You simply need to move the NULL DFH to an address after the other BBBs. Each DMA BBB expects the next DFH to be located 0x100 bytes from the base address of the BBB. The following figure depicts the linked-list for the DMA AFU design example.

Figure 5. DMA AFU Device Feature Header (DFH) Chaining



4. Software Programming Model

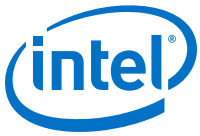
The streaming DMA AFU includes a software driver that you can use in your own host application. The `fpga_dma.cpp` and header file `fpga_dma.h` located at the following location implement the software driver:

```
$OPAE_PLATFORM_ROOT/hw/samples/dma_afu/sw
```

This driver supports the following functions:

API	Description
`fpgaCountDMAChannels`	Scans the device feature chain for DMA BBBs and count all available channels.
`fpgaDMAOpen`	Opens a handle to the DMA channel.
`fpgaDMAClose`	Closes a handle to the DMA channel.
`fpgaDMATransferInit`	Initializes an object that represents the DMA transfer.
`fpgaDMATransferReset`	Resets the DMA transfer attribute object to default values.
`fpgaDMATransferDestroy`	Destroys the DMA transfer attribute object.
`fpgaDMATransferSetSrc`	Set source address of the transfer. This address must be 64 byte aligned.
`fpgaDMATransferSetDst`	Set destination address of the transfer. This address must be 64 byte aligned.
`fpgaDMATransferSetLen`	Set transfer lengths in bytes. For non-packet transfer, you must set the transfer length to a multiple of 64 byte. In packet transfer, this is not a limitation.
`fpgaDMATransferSetTransferType`	Set type of the transfer. Legal values are: • `HOST_MM_TO_FPGA_MM` (Host to AFU) • `FPGA_MM_TO_HOST_MM` (AFU to host)
`fpgaDMATransferSetTransferCallback`	Registers callback for notification on asynchronous transfer completion. If you specify a callback, `fpgaDMATransfer` returns immediately (asynchronous transfer). If you don not specify a callback, `fpgaDMATransfer` returns after the transfer is complete (synchronous/blocking transfer).
`fpgaDMATransferSetLast`	Indicates the last transfer so the DMA can start processing the prefetched transfers. The default value is 64 transfers in the pipeline before the DMA starts to work on the transfers.
`fpgaDMATransfer`	Performs a DMA transfer.

For more information about the API, input, and output arguments, refer to the header file located at `$OPAE_PLATFORM_ROOT/hw/samples/dma_afu/sw/fpga_dma.h`



To know more about software driver use model, refer to the README file located at `$OPAE_PLATFORM_ROOT/hw/samples/dma_afu/README`

5. Running DMA AFU Example

Before you begin:

- You should be familiar with the examples in the *Intel Acceleration Stack Quick Start Guide for Intel FPGA Programmable Acceleration Card D5005*.
- You must define an environment variable. The environment variable is dependent on the Intel Acceleration Stack version you are using:
  - For current version, set the environment variable to $OPAE_PLATFORM_ROOT
- You must install the Intel Threading Building Blocks (TBB) library since the DMA driver relies on it.
- You must also set up two 1 GB hugepages to run the sample application.

```
$ sudo sh -c "echo 2 > /sys/kernel/mm/hugepages/hugepages-1048576kB/
nr_hugepages"
```

Perform the following steps to download the DMA Accelerator Function (AF) bitstream, to build the application and driver, and to run the design example:

1. Change to the DMA application and driver directory:

```
cd $OPAE_PLATFORM_ROOT/hw/samples/dma_afu/sw
```

2. Build the driver and application:

```
make
```

3. Download the DMA AFU bitstream:

```
fpgaconf ../bin/dma_afu.gbs
```

4. Execute the host application to write 100 MB in 1 MB portions from host memory to FPGA device memory and read it back:

```
./fpga_dma_test -s 104857600 -p 1048576 -r mtom
```

Related Information

Intel Acceleration Stack Quick Start Guide for Intel FPGA Programmable Acceleration Card D5005

6. Simulating the AFU Example

Intel recommends you refer to the *Intel Accelerator Functional Unit (AFU) Simulation Environment (ASE) Quick Start Guide* for your Intel FPGA PAC to be familiar with simulating similar examples and to setup your environment. Before you proceed through the following steps, verify that the OPAE_PLATFORM_ROOT environment variable is set to the OPAE SDK installation directory.

Complete the following steps to setup the hardware simulator for the DMA AFU:

1. Change to the DMA AFU sample directory:

```
cd $OPAE_PLATFORM_ROOT/hw/samples/dma_afu
```

2. Create an ASE environment in a new directory and configure it for simulating an AFU:

```
afu_sim_setup --source hw/rtl/filelist.txt build_ase_dir
```

3. Change to the ASE build directory:

```
cd build_ase_dir
```

4. Build the driver and application:

```
make
```

5. `make sim`

Sample output from the hardware simulator:

```
[SIM]  ** ATTENTION : BEFORE running the software application **
[SIM]  Set env(ASE_WORKDIR) in terminal where application will run (copy-and-
paste) =>
[SIM]  $SHELL   | Run:
[SIM]  ---------+-----------------------------------------------------
[SIM]  bash/zsh | export ASE_WORKDIR=$OPAE_PLATFORM_ROOT/hw/samples/dma_afu/
ase_mkdir/work
[SIM]  tcsh/csh | setenv ASE_WORKDIR $OPAE_PLATFORM_ROOT/hw/samples/dma_afu/
ase_mkdir/work
[SIM]  For any other $SHELL, consult your Linux administrator
[SIM]
[SIM]  Ready for simulation...
[SIM]  Press CTRL-C to close simulator...
```

Complete the following steps to compile and execute the streaming DMA AFU software in the simulation environment:

1. Open a new terminal window.
2. Change directory to:

```
cd $OPAE_PLATFORM_ROOT/hw/samples/dma_afu/sw
```

3. Copy environment setup string (choose string appropriate for your shell) from the steps above in the hardware simulation to the terminal window. See the following lines in the sample output from the hardware simulator.

```
[SIM]  bash/zsh | export ASE_WORKDIR=$OPAE_PLATFORM_ROOT/hw/samples/dma_afu/
build_ase_dir/work
[SIM]  tcsh/csh | setenv ASE_WORKDIR $OPAE_PLATFORM_ROOT/hw/samples/dma_afu/
build_ase_dir/work
```

4. `$ make USE_ASE=1`
5. Execute the host application to write 4 KB in 1 KB portions from the host memory back to FPGA device memory in the loopback mode:

```
./fpga_dma_st_test -s 4096 -p 1024 -r mtom
```

Related Information

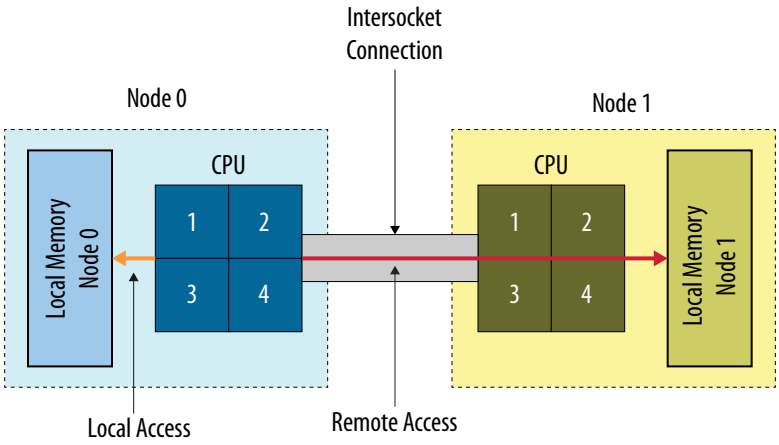
Intel Accelerator Functional Unit (AFU) Simulation Environment (ASE) Quick Start User Guide

7. Optimization for Improved DMA Performance

Implementation of NUMA (non-uniform memory access) optimization in the `fpga_dma_test.cpp` (application) allows processor to access it's own local memory faster than accessing the non-local memory (memory local to another processor).

A typical NUMA configuration is shown in the diagram below. The orange arrow represents access from a core to memory local of the same core. The red arrow illustrates the path taken when a core on Node 0 access memory that resides in local memory of Node 1.

Figure 6. Typical NUMA Configuration



Use the following code to implement NUMA optimization in your test application:

```
// Set up proper affinity if requested
    if (cpu_affinity || memory_affinity) {
        unsigned dom = 0, bus = 0, dev = 0, func = 0;
        fpga_properties props;
        int retval;
        #if(FPGA_DMA_DEBUG)
                char str[4096];
        #endif
        res = fpgaGetProperties(afc_token, &props);
        ON_ERR_GOTO(res, out_destroy_tok, "fpgaGetProperties");
        res = fpgaPropertiesGetBus(props, (uint8_t *) & bus);
        ON_ERR_GOTO(res, out_destroy_tok, "fpgaPropertiesGetBus");
        res = fpgaPropertiesGetDevice(props, (uint8_t *) & dev);
        ON_ERR_GOTO(res, out_destroy_tok, "fpgaPropertiesGetDevice");
        res = fpgaPropertiesGetFunction(props, (uint8_t *) & func);
        ON_ERR_GOTO(res, out_destroy_tok, "fpgaPropertiesGetFunction");

        // Find the device from the topology
        hwloc_topology_t topology;
        hwloc_topology_init(&topology);
        hwloc_topology_set_flags(topology,
                    HWLOC_TOPOLOGY_FLAG_IO_DEVICES);
```

```
        hwloc_topology_load(topology);
        hwloc_obj_t obj = hwloc_get_pcidev_by_busid(topology, dom, bus, dev,
func);
        hwloc_obj_t obj2 = hwloc_get_non_io_ancestor_obj(topology, obj);
        #if (FPGA_DMA_DEBUG)
            hwloc_obj_type_snprintf(str, 4096, obj2, 1);
            printf("%s\n", str);
            hwloc_obj_attr_snprintf(str, 4096, obj2, " :: ", 1);
            printf("%s\n", str);
            hwloc_bitmap_taskset_snprintf(str, 4096, obj2->cpuset);
            printf("CPUSET is %s\n", str);
            hwloc_bitmap_taskset_snprintf(str, 4096, obj2->nodeset);
            printf("NODESET is %s\n", str);
        #endif
        if (memory_affinity) {
            #if HWLOC_API_VERSION > 0x00020000
                retval = hwloc_set_membind(topology, obj2->nodeset,
                                HWLOC_MEMBIND_THREAD, HWLOC_MEMBIND_MIGRATE |
HWLOC_MEMBIND_BYNODESET);
            #else
                retval =
                hwloc_set_membind_nodeset(topology, obj2->nodeset,
                                HWLOC_MEMBIND_THREAD,
                                HWLOC_MEMBIND_MIGRATE);
            #endif
            ON_ERR_GOTO(retval, out_destroy_tok, "hwloc_set_membind");
        }
        if (cpu_affinity) {
            retval = hwloc_set_cpubind(topology, obj2->cpuset,
HWLOC_CPUBIND_STRICT);
            ON_ERR_GOTO(retval, out_destroy_tok, "hwloc_set_cpubind");
        }
    }
```

8. Document Revision History for the DMA Accelerator Functional Unit (AFU) User Guide

Document Version	Intel Acceleration Stack Version	Changes
2019.08.05	2.0 (supported with Intel Quartus Prime Pro Edition 18.1.2)	Initial release.