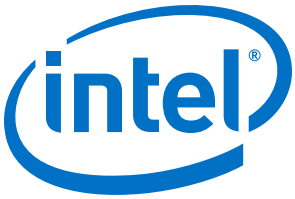


Streaming DMA Accelerator Functional Unit (AFU) User Guide

Updated for Intel® Acceleration Stack for Intel® Xeon® CPU with FPGAs: **1.2**

Subscribe
Send Feedback

UG-20171 | 2018.12.04
Latest document on the web: **PDF** | **HTML**

Contents

1. About this Document

This document describes the streaming direct memory access (DMA) Accelerator Functional Unit (AFU) implementation using the Platform Designer.

1.1. Intended Audience

This document is intended for hardware or software developer who requires an Accelerated Function (AF) that accesses the data buffered in memory and provides it to an accelerator as a serial stream of data. Intel recommends you gain familiarity with Platform Designer before using this design example.

Related Information

Platform Designer User Guide

1.2. Conventions

Table 1. Document Conventions

Convention	Description
#	If this symbol precedes a command, enter the command as a root.
$	If this symbol precedes a command, enter the command as a user.
`This font`	Indicates file names, commands, and keywords. The font also indicates long command lines. For long command lines, press Enter only if the next line starts a new command, where the # or $ character denotes the start of the next command.
<variable_name>	Indicates placeholder text that you must replace with appropriate values. Do not include 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.
ASE	AFU Simulation Environment	Co-simulation environment that allows you to use the same host application and AF in a simulation environment. ASE is part of the Intel Acceleration Stack for FPGAs.

continued...

Acronyms	Expansion	Description
CCI-P	Core Cache Interface	CCI-P is the standard interface AFUs use to communicate with the host.
CL	Cache Line	64-byte cache line
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. 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 Programmable Acceleration Card with Intel Arria® 10 GX FPGA	Intel PAC with Intel Arria 10 GX FPGA	PCIe accelerator card with an Intel Arria 10 FPGA. Programmable Acceleration Card is abbreviated PAC. Contains an FPGA Interface Manager (FIM) that pairs with an Intel Xeon processor over PCIe bus.
Intel Xeon Scalable Platform with Integrated FPGA	Integrated FPGA Platform	Intel Xeon plus FPGA platform with the Intel Xeon and an FPGA in a single package and sharing a coherent view of memory via the Ultra Path Interconnect (UPI).
`OPAE_PLATFORM_ROOT`		A Linux shell environment variable set up during the process of installing the OPAE SDK delivered with the Acceleration Stack.

2. Streaming DMA AFU Description

The streaming DMA AFU design example shows how to transfer data between the memory and Avalon®-ST sources and sinks. Most commonly, a streaming DMA is utilized to transfer data from host memory into a hardware accelerator and stream the results back to host memory without using the local FPGA memory as a temporary buffer. These streams typically operate in parallel mode and reduce the latency of a hardware accelerator by removing the additional memory copy operations.

The streaming DMA AFU comprises of the following sub-modules:

- Memory Properties Factory (MPF) Basic Building Block (BBB)
- Core Cache Interface (CCI-P) to Avalon-MM Adapter
- Streaming DMA Test System, which includes:
  - Memory-to-Stream (M2S) DMA BBB
  - Steam-to-Memory (S2M) DMA BBB
  - Streaming Pattern Checker and Generator

The streaming DMA AFU design example includes a user space driver as well as a host application that performs data transfer between host memory and the FPGA pattern checker and generator. You can use this design example as a starting point to implement streaming data transfers in your own AFU design by replacing the pattern checker and generator with your hardware accelerator and modifying the host application accordingly.

Both M2S and S2M DMA BBBs support packetized data, therefore the streaming data includes the start-of-packet (SOP), end-of-packet (EOP), and empty signals. You can use this packet support to transfer a hardware driven payload size. For example, a compression accelerator typically receives a known payload size; and the compression results have an unknown length until the accelerator completes this task. The compression accelerator simply issues a packet to the S2M DMA BBB and the driver provides the host application a continuous buffer that contains the compressed results and buffer length.

Related Information

Avalon Interface Specifications

2.1. Hardware Subsystems

The streaming DMA AFU interfaces with the FPGA Interface Unit (FIU) and two banks of local external SDRAM. The streaming DMA BBBs can address up to 256TB of memory connected to the FPGA. The streaming DMA design example reduces this memory span down to 8GB, which is split into two memory banks. If you are using the streaming DMA BBBs in a design that targets a platform with a different local memory hierarchy or density, then you can adjust the local memory pipeline bridges in the streaming DMA test system using Platform Designer.

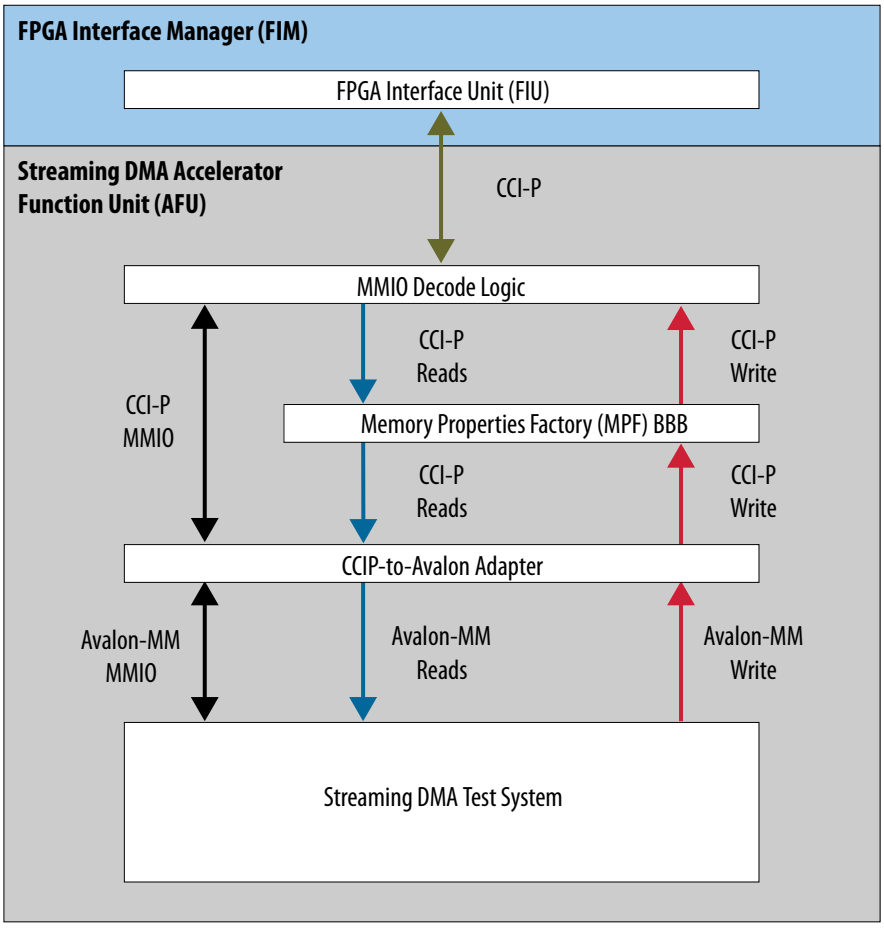
You can use the streaming DMA AFU to perform the following data transfer:

- Host memory to FPGA stream
- FPGA stream to host memory
- Local FPGA memory to FPGA stream[1]
- FPGA stream to local FPGA memory[1]

The streaming DMA AFU, M2S and S2M DMA BBBs are implemented as Platform Designer systems. Each of these systems can be found in the following location:

```
$OPAE_PLATFORM_ROOT/hw/samples/streaming_dma_afu/hw/rtl/<device>/
```

Figure 1. High Level System Diagram



The streaming DMA AFU includes the following modules that connect to the FIU:

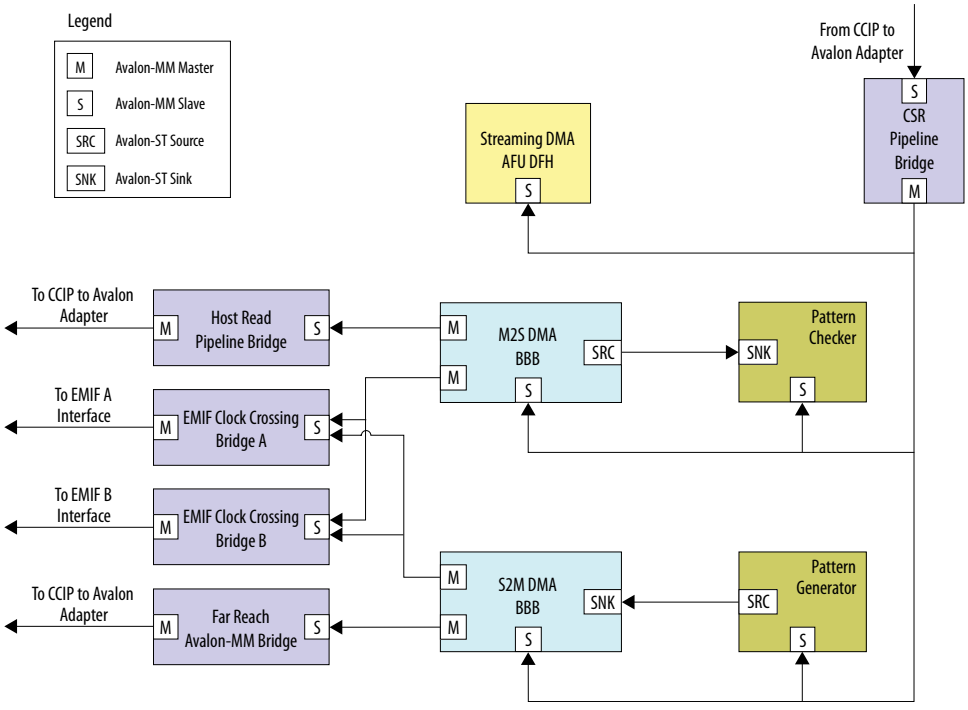
(1) Currently, the driver does not support this feature.

- MMIO Decode 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.
- MPF—ensures that read issued by the M2S DMA BBB are returned in the order that they were issued. The streaming DMA BBBs use the Avalon-MM protocol which requires the read data to return in-order.
- CCI-P to Avalon-MM Adapter—translates MMIO accesses to Avalon-MM read and write transactions. This module also receives Avalon-MM read and write transactions from the streaming DMA BBBs and converts them to CCI-P transactions that are issued to the host.
- Streaming DMA Test System—a wrapper around the two streaming DMA BBBs and includes pattern checker and generator components. This module exposesAvalon-MM master and slave interfaces that connect to the CCI-P to Avalon-MM adapter.

2.2. Streaming DMA Test System

The streaming DMA test system is a Platform Designer system that connects the streaming DMA BBBs to other IP in the system.

Figure 2. Streaming DMA Test System Block Diagram



The streaming DMA test system includes the following modules:

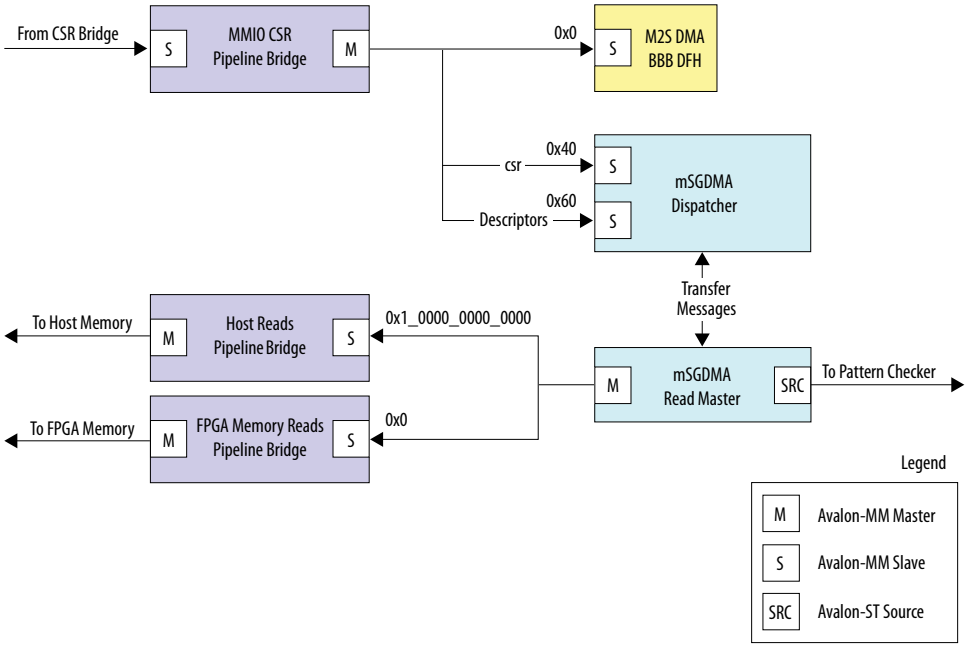
- AFU DFH—stores the 64-bit device feature header (DFH) for the streaming DMA AFU. The host driver scans the hardware that is searching for the AFU DFH and various BBBs used to identify the hardware. The AFU DFH is setup to point to the next DFH at offset 0x100.
- M2S DMA BBB—reads buffers from memory and provides the data as a serial stream to the Avalon-ST source port. In this design example, the streaming data is sent to the pattern checker.
- S2M DMA BBB—accepts a serial stream of data from its Avalon-ST port and writes the data to buffers in memory. In this design example, the streaming data is sent from the pattern generator.
- Pattern Checker and Generator—this module is programmed by the host with a pattern. The supplied host software configures each component with a pattern that increments by one for every increasing byte.
- Clock Crossing Bridge—this module has been added between the streaming DMAs and the local FPGA external memory to operate the streaming DMA AFU in the `pClk` clock domain.
- Pipeline Bridge—this module has been added between the M2S DMA BBB and host read interface of the CCI-P to Avalon-MM adapter to improve the maximum operating frequency (Fmax) of the streaming DMA AFU.
- Far Reach Avalon-MM Bridge—this module has been added between the S2M DMA BBB and host write interface of the CCI-P to Avalon-MM adapter to improve the maximum operating frequency (Fmax). It also sends write responses from the CCI-P interface to the S2M DMA.

2.3. Memory-to-Stream DMA BBB

The Memory-to-Stream (M2S) DMA BBB reads data from a buffer stored in memory and converts it into an Avalon-ST source stream. The buffer must be aligned to 64-bytes which is guaranteed by the driver for locations in host memory. The M2S DMA BBB is configured to handle up to a 1GB transfer size, but the driver divides the large transfers into smaller ones with a maximum size of 2MB.

The M2S DMA BBB streaming interface supports packet generation by exposing the start-of-packet (SOP), end-of-packet (EOP), and empty signals. Your host application can optionally instruct the streaming DMA driver to generate packetized data. If you enable the packetized data, then the empty signal conveys the number of bytes at the end of a transfer that are valid when the EOP signal is asserted. For example, a DMA transfer of 4100 bytes (with packet support) contains 64 full beats (each beat is 64 bytes) of streaming data with SOP asserted during the first beat. The empty signal is set to 60 during the last beat of data with EOP asserted, because only four bytes out of the 64 are valid.

Figure 3. M2S DMA BBB Platform Designer System



The components in the M2S DMA BBB Platform Designer system implement the following functions:

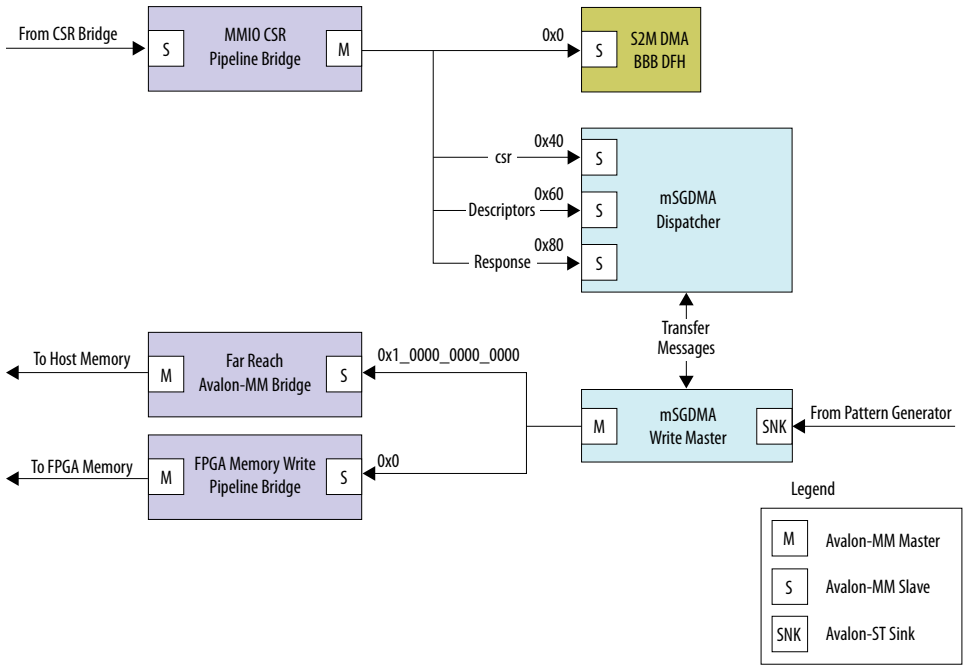
- M2S DMA BBB DFH—stores the 64-bit device feature header (DFH) for the M2S DMA BBB. The host driver scans the hardware that is searching for the AFU DFH and various BBBs used to identify the hardware. The M2S DMA BBB DFH is setup to point to the next DFH at offset 0x100.
- mSGDMA Dispatcher—buffers descriptors sent from the host to the BBB
- mSGDMA Read Master—accepts commands from the dispatcher and reads from memory and converts the data to an Avalon-ST stream. The data leaving the streaming port can be accompanied by streaming sideband signaling for SOP, EOP, and empty signals. If you require the stream to support non-multiples of 64 bytes, then you must request the driver to send packetized data. Therefore, if the last beat is not 64 bytes in size, then the empty signal informs your downstream hardware about the invalid bytes. Only the last beat can contain invalid bytes, all other beats must be 64 bytes in size which is defined by the Avalon-ST specification.
- Pipeline Bridge—this component has been added between the mSGDMA read master and host/local FPGA memory to improve the maximum operating frequency (Fmax) of the M2S DMA BBB. If your design does not require the M2S DMA BBB to connect to local FPGA memory, then export that interface and ground all its master inputs. All the mSGDMA dispatcher slave interfaces connect to a pipeline bridge which spans an address range of 0x100.

2.4. Stream-to-Memory DMA BBB

The Steam-to-Memory (S2M) DMA BBB accepts Avalon-ST data and transfers it to a buffer in memory. The buffer must be aligned to 64-bytes which is guaranteed by the driver for locations in host memory. The S2M DMA BBB is configured to handle up to a 1GB transfer size, but the driver divides the large transfers into smaller ones with a maximum size of 2MB.

The S2M DMA BBB streaming interface supports receiving packetized data by exposing the SOP, EOP, and empty signals. Your host application instructs the streaming DMA driver to use the packet signaling when it requests a streaming transfer. By using the packetized data, the hardware accelerator that provides the data can determine when transfer will complete. For example, if a data compression engine is connected to the S2M DMA BBB, the host application does not know how much data might stream until the compression operation is complete. Instead of dividing this data into frames, your hardware accelerator simply notifies the start and end of the payload via asserting SOP and EOP respectively. The DMA transfers the entire payload to memory and DMA driver instructs the host application of the payload length when it is complete.

Figure 4. S2M DMA BBB Platform Designer System



The components in the S2M DMA BBB Platform Designer system implement the following functions:

- S2M DMA BBB DFH—stores the 64-bit device feature header (DFH) for the S2M DMA BBB. The host driver scans the hardware that is searching for the AFU DFH and various BBBs used to identify the hardware. The S2M DMA DMA BBB DFH points to the next DFH at offset 0x100.
- mSGDMA Dispatcher—buffers descriptors sent from the host to the BBB. The dispatcher includes a response interface that the host driver reads to determine how much data was transferred when the data is packetized (non-deterministic payload length). This component is included with the design example because it is a slightly modified version of the component that is available in Intel Quartus® Prime Pro Edition.
- mSGDMA Write Master—accepts commands from the dispatcher and writes the data accepted by the Avalon-ST sink interface to memory. The data arriving at the streaming port can be accompanied by streaming sideband signaling for SOP, EOP, and empty signals. This component is included with the design example because it is a slightly modified version of the component that is available in Intel Quartus Prime Pro Edition.
- Pipeline Bridge—this component has been added between the mSGDMA write master and local FPGA memory to improve the maximum operating frequency (Fmax) of the S2M DMA BBB. If your design does not require the S2M DMA BBB to connect to local FPGA memory, then export that interface and ground all its master inputs. All the mSGDMA dispatcher slave interfaces connect to a pipeline bridge which spans an address range of 0x100.
- Far Reach Avalon-MM Bridge—this component has been added between the mSGDMA write master and host write interface of the CCI-P to Avalon-MM adapter to improve the maximum operating frequency (Fmax) of the S2M DMA BBB. It also forwards write responses to the write master . The streaming DMA driver instructs S2M DMA BBB to wait for all write responses to return before it sends an interrupt to the host ensuring that there are no write synchronization issues.

3. Memory Map and Address Spaces

The streaming DMA AFU has two memory views:

- DMA view
- Host view

The DMA view supports a 49-bit address space. The lower half of the DMA view maps to the local FPGA memory. Only the streaming DMA BBBs have connectivity to the local FPGA memory, the host cannot access 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 DFH tables, and control/status registers of the various components that are used inside the streaming DMA AFU.

The MMIO registers in both streaming DMA BBBs and the streaming DMA AFU support 32- and 64-bit access. The streaming DMA AFU does not support 512-bit MMIO accesses. The mSGDMA registers inside each streaming DMA BBB must be accessed using 32-bit accesses except for descriptor and response accesses.

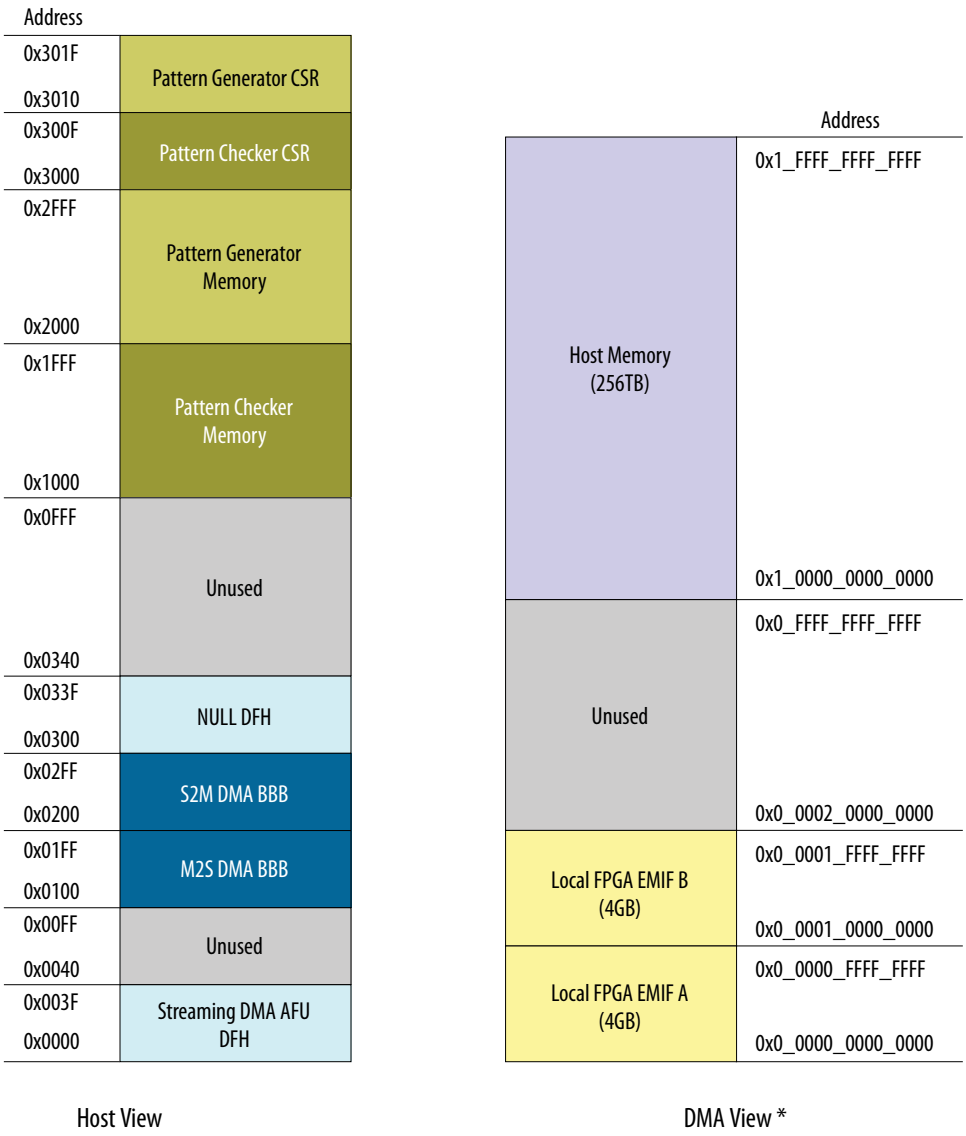
3.1. Streaming DMA AFU Memory Map

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

Table 4. Streaming DMA AFU Memory Map

Byte Address	Register Name	Span in Bytes	Description
0x0000	Streaming DMA AFU DFH	0x40	Device feature header for the streaming DMA AFU. This DFH points to 0x100 as the next DFH offset.
0x0100	M2S DMA BBB	0x100	Memory-to-stream DMA BBB.
0x0200	S2M DMA BBB	0x100	Stream-to-memory DMA BBB.
0x0300	NULL DFH	0x40	Null device feature header terminating the DFH linked list.
0x1000	Pattern Checker Memory Slave	0x1000	Pattern checker memory populated by the host application.
0x2000	Pattern Generator Memory Slave	0x1000	Pattern generator memory populated by the host application
0x3000	Pattern Checker CSR Slave	0x10	Pattern checker control and status registers
0x3010	Pattern Generator CSR Slave	0x10	Pattern generator control and status registers.

Figure 5. Streaming DMA AFU Memory Views



* You can adjust the local FPGA memory addressable space in the DMA AFU platform designer system. The S2M and M2S DMAs are designed to address upto 256 TB of FPGA memory.

3.2. Memory-to-Stream DMA BBB Memory Map

The M2S DMA BBB memory map provides the address offsets of all the locations within the BBB. The following streaming DMA AFU registers reside at offset 0x100 in the MMIO address space.

Table 5. Memory-to-Stream DMA BBB Memory Map

Byte Address	Register Name	Span in Bytes	Description
0x00	M2S DMA BBB DFH	0x40	Device feature header for the M2S DMA BBB. This DFH points to 0x100 as the next DFH offset.
0x40	M2S DMA Dispatcher CSR	0x20	Control port for the mSGDMA within the memory-to-stream DMA BBB. The driver accesses this location to control the DMA or query its status.
0x60	M2S DMA Descriptor	0x20	Descriptor port for the mSGDMA within the memory-to-stream DMA BBB. The driver writes descriptors to this location.

3.3. Stream-to-Memory DMA BBB Memory Map

The S2M DMA BBB memory map provides the address offsets of all the locations within the BBB. The following streaming DMA AFU registers reside at offset 0x200 in the MMIO address space.

Table 6. Stream-to-Memory DMA BBB Memory Map

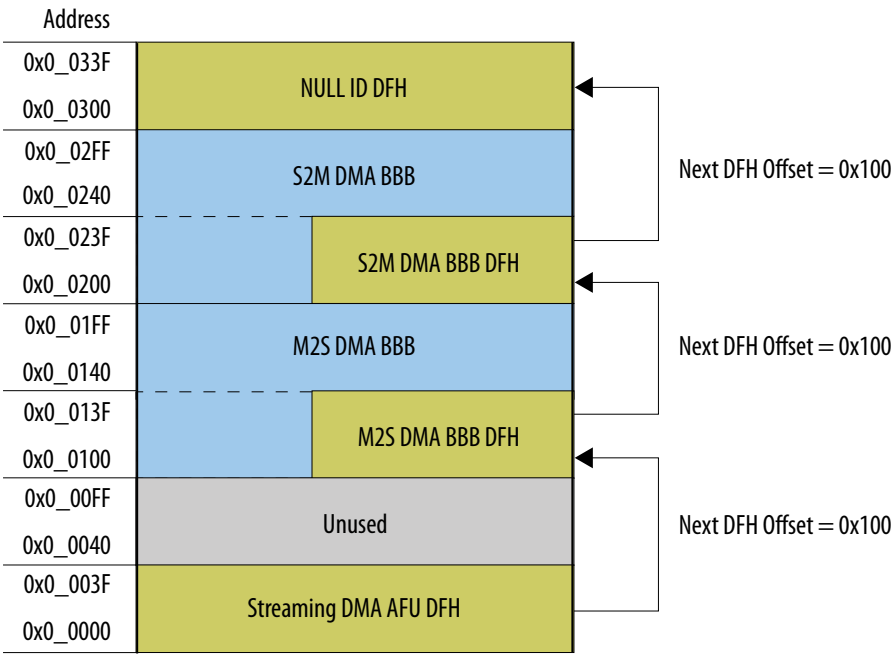
Byte Address	Register Name	Span in Bytes	Description
0x00	S2M DMA BBB DFH	0x40	Device feature header for the S2M DMA BBB. This DFH points to 0x100 as the next DFH offset.
0x40	S2M DMA Dispatcher CSR	0x20	Control port for the mSGDMA within the stream-to-memory DMA BBB. The driver accesses this location to control the DMA or query its status.
0x60	S2M DMA Descriptor	0x20	Descriptor port for the mSGDMA within the stream-to-memory DMA BBB. The driver writes descriptors to this location.
0x80	S2M DMA Response	0x8	Response port for the mSGDMA within the stream-to-memory DMA BBB. The driver reads this port to determine how much data was streamed to the memory.

3.4. Device Feature Header Linked-list

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

A NULL DFH included at the end of the list. The inclusion of the null DFH at the end of the linked list allows you to add more streaming DMA BBBs to your design. You simply need to move the NULL DFH to an address after the other BBBs. Each streaming 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 streaming DMA AFU design example.

Figure 6. Streaming DMA AFU Device Feature Header (DFH) Chaining



If you want two M2S and two S2M DMA BBBs in your design, then you can use the following address map to implement four streaming channels. The four streaming DMA BBBs can reside anywhere in the address map if they are packed together in the MMIO address space every 0x100 bytes. The DFH that follows the streaming DMA BBB must be located at offset 0x100 from the previous streaming DMA BBB channel and it can be the NULL DFH or other DFHs.

Table 7. Four-channel Streaming DMA AFU Example Configuration

Byte Address	Register Name	Span in Bytes	Description
0x000	Streaming DMA AFU DFH	0x40	Your AFU DFU. This DFH points to 0x100 as the next DFH offset.
0x100	M2S DMA BBB #1	0x100	First memory-to-stream DMA BBB. Next DFH set to 0x100.
0x200	M2S DMA BBB #2	0x100	Second memory-to-stream DMA BBB. Next DFH set to 0x100.
0x300	S2M DMA BBB #1	0x100	First stream-to-memory DMA BBB. Next DFH set to 0x100.
0x400	S2M DMA BBB #2	0x100	Second stream-to-memory DMA BBB. Next DFH set to 0x100.
0x500	NULL DFH	0x40	Null DFH at the end of the linked list.

4. Software Programming Model

The streaming DMA AFU includes a user space driver that you can use in your own host application. The streaming DMA AFU host application, including the user space driver are located at the following location:

```
$OPAE_PLATFORM_ROOT/hw/samples/streaming_dma_afu/sw
```

All the driver APIs are documented in the `fpga_dma.h` header file. The user space driver supports both blocking and non-blocking DMA transfers. While using both the streaming DMA BBBs to stream data to and from your accelerator, Intel recommends that you use non-blocking (asynchronous) transfers so that both DMAs can transfer data simultaneously. Using the blocking (synchronous) transfer API to transfer data to and from your accelerator concurrently may lead to deadlock, since each streaming DMA can only buffer approximately 8KB of data before back-pressuring.

5. Running the AFU Example

Intel recommends you refer to the Quick Start Guide for your Intel PAC to be familiar with running similar examples. Before you proceed through the following steps, verify that the OPAE_PLATFORM_ROOT environment variable is set to the OPAE SDK installation directory.

Note: Intel recommends you use the GCC (C Compiler) to compile the design example. If you compile the DMA sample application and user space driver with g++ (C++ compiler), it may result in compilation errors.

Follow these steps to download the DMA Accelerator Function (AF) bitstream, build, and run the design example:

1. `sudo sh -c "echo 20 > /sys/kernel/mm/hugepages/hugepages\ -2048kB/nr_hugepages"`

   If you have not already done so, use the above command to configure the system and allocate 20 count of 2MB hugepages for the DMA user space driver(2). This command requires root privileges.
2. `cd $OPAE_PLATFORM_ROOT/hw/samples/streaming_dma_afu/sw`
3. `make`
4. `sudo fpgaconf ../bin/streaming_dma_afu.gbs`
5. To run the software on hardware:

   `sudo ./fpga_dma_st_test 0`

Sample output from running the DMA test software:

```
Running test in HW mode
No of DMA channels = 00000002
DMA Base Addr = 00000100
DMA Base Addr = 00000200
M2S Checker:Data Verification Success!
M2S Checker:Data Verification Success!
S2M: Data Verification Success!
S2M: Data Verification Success!
Running Bandwidth Tests..
Streaming from host memory to FPGA..
M2S Checker:Data Verification Success!
Measured bandwidth = 6732.154665 Megabytes/sec
Streaming from FPGA to host memory..
Verifying buffer..
S2M: Data Verification Success!
Measured bandwidth = 5434.340969 Megabytes/sec
```

(2) If your host has multiple cards, you need 20 count of 2MB hugepages per card. For an example, a multi-channel system with four cards requires total 80 count of 2MB hugepages.

Related Information

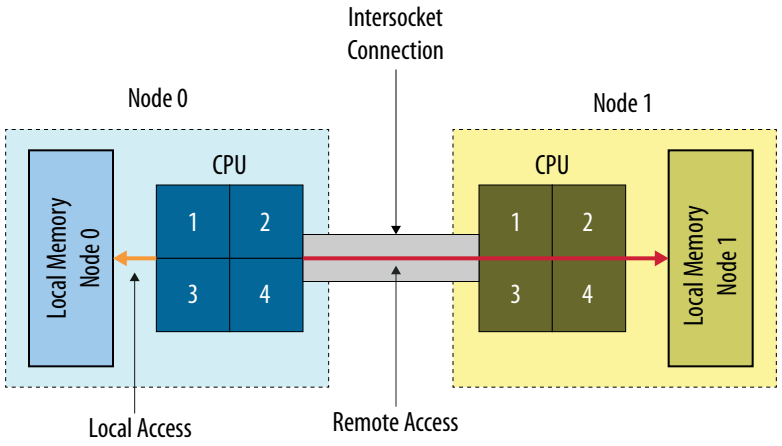
Intel FPGA Acceleration Hub: Knowledge Center
Provides more information about the related resources, collateral, and training.

5.1. Optimization for Improved DMA Performance

Implementation of NUMA (non-uniform memory access) optimization in the `fpga_dma_st_test.c` (application) allows processor to access it's own local memory. This implementation is 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.

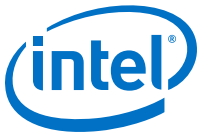
Figure 7. Typical NUMA Configuration



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");
        }
    }
```

6. Generating the Accelerator Function (AF)

To generate a synthesis build environment to generate an AF, use the `afu_synth_setup` command as following:

1. `cd $OPAE_PLATFORM_ROOT/hw/samples/streaming_dma_afu`
2. `afu_synth_setup --source hw/rtl/filelist.txt build_synth`

   From the synthesis build directory generated by `afu_synth_setup`, enter the following command from a terminal window to generate an AF for the target hardware platform:
3. `cd build_synth`
4. `$OPAE_PLATFORM_ROOT/bin/run.sh`

The `run.sh` AF generation script creates the AF image with the same base filename as the AFU's platform configuration file with a `.gbs` suffix at the location: `$OPAE_PLATFORM_ROOT/hw/samples/build_synth/streaming_dma_afu.gbs`.

7. Simulating the AFU Example

Intel recommends you refer to the Quick Start Guide for your Intel 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.

Note: Intel recommends you use the GCC (C Compiler) to compile the design example. If you compile the DMA sample application and user space driver with g++ (C++ compiler), it may result in compilation errors.

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

1. `cd $OPAE_PLATFORM_ROOT/hw/samples/streaming_dma_afu`
2. `afu_sim_setup --source hw/rtl/filelist.txt build_ase_dir`
3. `cd build_ase_dir`
4. `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=/mnt/Tools/ias/hw/samples/
streaming_dma_afu/build_ase_dir/work
[SIM]  tcsh/csh | setenv ASE_WORKDIR /mnt/Tools/ias/hw/samples/
streaming_dma_afu/build_ase_dir/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. `cd $OPAE_PLATFORM_ROOT/hw/samples/streaming_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=/mnt/Tools/ias/hw/samples/
streaming_dma_afu/build_ase_dir/work
[SIM]  tcsh/csh | setenv ASE_WORKDIR /mnt/Tools/ias/hw/samples/
streaming_dma_afu/build_ase_dir/work
```

4. `make USE_ASE=1`
5. To run the software in the simulation environment:

   `./fpga_dma_st_test 1`

Sample output from running software using simulation environment:

```
[APP]  Deallocating memory /buf15.894589435998081 ...
  [APP]  SUCCESS
  [APP]  MMIO Write     : tid = 0x07f, offset = 0x244, data = 0x00000000
  [APP]  Deinitializing simulation session
  [APP]  Closing Watcher threads
  [APP]  Deallocating UMAS
  [APP]  Deallocating memory /umas.894589435998081 ...
  [APP]  SUCCESS
  [APP]  Deallocating MMIO map
  [APP]  Deallocating memory /mmio.894589435998081 ...
  [APP]  SUCCESS
  [APP]      Took 87,877,947,778 nsec
  [APP]  Session ended
```

Related Information

Intel FPGA Acceleration Hub: Knowledge Center
Provides more information about the related resources, collateral, and training.

8. Streaming DMA AFU User Guide Archives

Intel Acceleration Stack Version	User Guide (PDF)
1.1	Streaming DMA Accelerator Functional Unit (AFU) User Guide

9. Document Revision History for Streaming DMA Accelerator Functional Unit (AFU) User Guide

Document Version	Intel Acceleration Stack Version	Changes
2018.12.04	1.2 (supported with Intel Quartus Prime Pro Edition 17.1.1)	• Updated the *Running the AFU Example* and *Simulating the AFU Example* steps. • Replaced the Write Response Bridge with the Far Reach Avalon-MM Bridge. • Added a new section *NUMA Optimization*. • Added a new chapter: *Streaming DMA AFU User Guide Archives*.
2018.08.06	1.1 (supported with Intel Quartus Prime Pro Edition 17.1.1)	Initial release.

Related Information

Intel FPGA Acceleration Hub: Knowledge Center

Provides more information about the related resources, collateral, and training.