



Intel Accelerator Functional Unit Simulation Environment Quick Start User Guide

Updated for Intel® Acceleration Stack for Intel® Xeon® CPU with FPGAs: **1.2.1 and 2.0.1**



Online Version



Send Feedback

UG-20165

ID: **683200**

Version: **2020.03.06**

Contents

1. About this Document.....	3
2. System Requirements.....	4
3. Setting Up the Environment.....	5
4. Simulating hello_afu in Client-Server Mode.....	7
4.1. Simulation in Client-Server Mode.....	9
4.1.1. Simulation Log Files.....	9
4.1.2. Design Declarations.....	10
4.1.3. Troubleshooting Client-Server Simulation.....	10
5. AFU Examples.....	11
6. Troubleshooting.....	13
7. ASE Quick Start User Guide Archives.....	14
8. Document Revision History for ASE Quick Start User Guide.....	15

1. About this Document

This document describes how to simulate a sample Accelerator Functional Unit (AFU) using the Intel® Accelerator Functional Unit (AFU) Simulation Environment (ASE) environment. Refer to the *Intel Accelerator Functional Unit (AFU) Simulation Environment (ASE) User Guide* for comprehensive details on ASE capabilities and internal architecture.

The Intel Accelerator Functional Unit (AFU) Simulation Environment (ASE) is a hardware and software co-simulation environment for any Intel FPGA Programmable Acceleration Card (Intel FPGA PAC). This software co-simulation environment currently supports the following Intel FPGA PACs:

- Intel Programmable Acceleration Card with Intel Arria® 10 GX FPGA
- Intel FPGA Programmable Acceleration Card D5005

The ASE provides a transactional model for the Core Cache Interface (CCI-P) protocol and a memory model for the FPGA-attached local memory.

The ASE also validates Accelerator Functional Unit (AFU) compliance to the following protocols and APIs:

- The CCI-P protocol specification
- The Avalon® Memory Mapped (Avalon-MM) Interface Specification
- The Open Programmable Acceleration Engine (OPAE)

Table 1. 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 a 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 cache of memory via Ultra Path Interconnect (UPI).

Related Information

[Intel Accelerator Functional Unit \(AFU\) Simulation Environment \(ASE\) User Guide](#)



2. System Requirements

Here are the system requirements for Intel Accelerator Functional Unit (AFU) Simulation Environment (ASE)::

- A 64-bit Linux operating system. This release validated the following operating systems:
 - For Intel FPGA PAC D5005:
 - RHEL 7.6 with Kernel 3.10.0-957
 - For Intel PAC with Intel Arria 10 GX FPGA:
 - RHEL 7.6 with Kernel 3.10.0-957
 - Ubuntu 18.04 with Kernel 4.15
- One of the following simulators:
 - 64-bit Synopsys* VCS-MX-2016.06-SP2-1 RTL Simulator
 - 64-bit Mentor Graphics* Modelsim SE Simulator (Version 10.5c)
 - 64-bit Mentor Graphics QuestaSim Simulator (Version 10.5c)
- C compiler: GCC 4.7.0 or above
- CMake: version 2.8.12 or above
- GNU C Library: version 2.17 or above
- Python: version 2.7
- Intel Quartus® Prime Pro Edition software version 19.2⁽¹⁾

⁽¹⁾ The required version is installed with the **Acceleration Stack for Development** version.

3. Setting Up the Environment

You must set up your simulation environment and install the OPAE software before running the ASE.

1. Set the following environment variables for your simulation software:

- **For VCS:**

```
$ export VCS_HOME=<path to VCS installation directory>
$ export PATH=$VCS_HOME/bin:$PATH
```

The VCS installation directory structure is as follows:

```
admin bin etc gnu include linux mmc suse32 vcfca vgcommon
amd64 doc flexlm gui install.log linux64 packages suse64 verifc vms
```

Make sure your system has a valid VCS license.

- **For Modelsim SE/Questasim:**

```
$ export MTI_HOME=<path to Modelsim installation directory>
$ export PATH=$MTI_HOME/linux_x86_64/:$MTI_HOME/bin/:$PATH
```

The Modelsim/Questasim installation directory structure is as follows:

```
avm gcc32 ieee LICENSE mpich2 perl_src sv_std uvm-1.1d vhdopt lib
bin gcc-4.3.3-linux ieee_env linux _msidata RELEASE_NOTES synopsys uvm-1.2 vhd1_src
cov_src gcc-4.3.3-linux_x86_64 ieee_pure linux_x86_64 osver RELEASE_NOTES.html tcl uvmc-2.3.1 vital1995
docs gcc-4.5.0-linux include mc2_lib osvrm RELEASE_NOTES.txt tcl fs uvm_reg-1.1 vital2000
drill_src gcc-4.5.0-linux_x86_64 infact mc2_ams ovm-2.1.1 rnm upf_lib vco vital2.2b
examples gcc-4.7.4-linux keyring modelsim.ini ovm-2.1.2 std upf_src verilog vm_src
floatfixlib gcc-4.7.4-linux_x86_64 lib modelsim lib pa lib std developerskit uvm-1.1c verilog src vovl_src
```

Make sure your system has a valid Modelsim SE/Questasim license.

- **For Intel Quartus Prime Pro Edition:**

```
$ export QUARTUS_HOME=<path to Intel Quartus Prime Pro Edition
installation directory>
```

The Intel Quartus Prime installation directory structure is as follows:

```
adm common drivers dsp_builder extlibs32 linux64 qdesigns sopc_builder
bin cusp dspba eda libraries lmf readme.txt version.txt
```

Add the environment variable to check the Modelsim license:

```
$ export MGLS_LICENSE_FILE=<Modelsim License>
```

2. Export:

```
$ export LM_LICENSE_FILE=<Quartus Prime License>
```

3. Extract the runtime archive file, and install OPAE libraries, binaries, include files, and ASE libraries as described in the section: *Installing the OPAE Software Package* in the appropriate Intel Acceleration Stack Quick Start User Guide for your Intel FPGA PAC.

Your environment must be set up correctly to configure and build an AFU. In particular, you must install the OPAE Software Development Kit (SDK) properly. OPAE SDK scripts must be on PATH and include files and libraries that must be available to

the C compiler. In addition, you must ensure that the `OPAE_PLATFORM_ROOT` environment variable is set. Refer to *Installing the OPAE Software Package* for more information.

To ensure that the OPAE SDK and ASE are properly installed, in a shell, confirm that your `PATH` includes `afu_sim_setup`. The `afu_sim_setup` should be in the `/usr/bin` directory or in `<opae install path>` if you built the OPAE from source files.

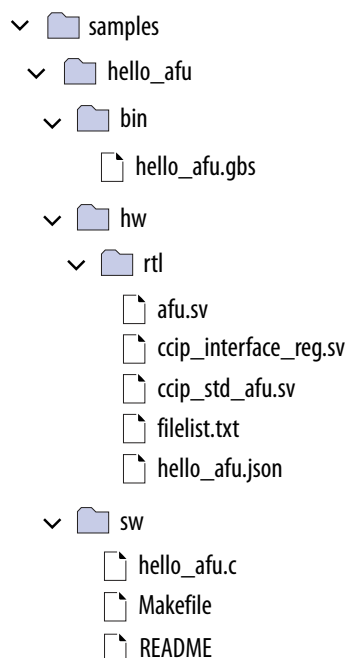
Related Information

- [Intel Accelerator Functional Unit \(AFU\) Simulation Environment \(ASE\) User Guide](#)
- [Installing the OPAE Software Package](#)
For Intel PAC with Intel Arria 10 GX FPGA.
- [Installing the OPAE Software Package](#)
For Intel FPGA PAC D5005.

4. Simulating hello_afu in Client-Server Mode

The hello_afu example is a simple AFU template that demonstrates the primary CCI-P interface. The RTL satisfies the minimum requirements of an AFU, responding to memory-mapped I/O reads to return the device feature header and the AFU's UUID.

Figure 1. hello_afu Directory Tree



Note: This document uses <AFU example> to refer to an example design directory, such as hello_afu in the figure above.

The software demonstrates the minimum requirements to attach to an FPGA using the OPAE. The RTL demonstrates the minimum requirements to satisfy the OPAE driver and the hello_afu example software.

filelist.txt specifies the files for RTL simulation and synthesis.

To successfully configure and build the AFU samples, your environment must be set up correctly, as described in *Setting Up the Environment*.

Related Information

- [Intel Accelerator Functional Unit \(AFU\) Simulation Environment \(ASE\) User Guide](#)
- [Setting Up the Environment](#) on page 5

- [Developing AFUs with the OPAE SDK](#)
In the Accelerator Functional Unit (AFU) Developer's Guide

4.1. Simulation in Client-Server Mode

The following example flow introduces the basic ASE scripts. You can simulate all examples with the ASE, except eth_e2e_e10 and eth_e2e_e40.

Simulation requires two software processes: one process for RTL simulation and a second process to run the connected software. To construct an RTL simulation environment, run the following in \$OPAE_PLATFORM_ROOT/hw/samples/hello_afu:

```
$ afu_sim_setup --source hw/rtl/filelist.txt build_sim
```

This command constructs an ASE environment in the build_sim subdirectory.

To build and run the simulator:

```
$ cd build_sim  
$ make  
$ make sim
```

The simulator prints a message that it is ready for simulation. It also prints a message prompting you to set the ASE_WORKDIR environment variable.

Open another shell for software simulation. You must ensure to set the OPAE_PLATFORM_ROOT environment variable.

To build and run the software in the new shell:

```
$ cd $OPAE_PLATFORM_ROOT  
$ export ASE_WORKDIR=$OPAE_PLATFORM_ROOT/hw/samples/hello_afu/build_sim/work  
$ cd $OPAE_PLATFORM_ROOT/hw/samples/hello_afu/sw  
$ make clean  
$ make USE_ASE=1  
$ ./hello_afu
```

Note: The specific pathname for ASE_WORKDIR may vary. Use the pathname provided by the simulator prompt.

The software and simulator run, log transactions, and exit.

4.1.1. Simulation Log Files

The simulation work directory stores the waveform, CCI-P transactions, and simulation log files.

Complete the following steps to view the waveform database:

1. Change to the directory in which you executed the make sim command.
2. Type:

```
$ make wave
```

The make wave command, invokes the waveform viewer.

4.1.2. Design Declarations

The following file and directories define the AFU simulation:

- \$OPAE_PLATFORM_ROOT/hw/samples/<AFU example>/hw/rtl/filelist.txt specifies RTL sources.
- <AFU example> is the example directory as shown in the *hello_afu Directory Tree* figure.
- filelist.txt lists SystemVerilog, VHDL, and the AFU JavaScript Object Notation (.json) file.
- The AFU .json describes the interfaces the AFU requires. It also includes a UUID to identify the AFU once downloaded to an FPGA.
- hw/rtl/hello_afu.json defines ccip_std_afu as the top-level interface by setting afu-top-interface to ccip_std_afu. ccip_std_afu is the base CCI-P interface including clocks, reset, and CCI-P TX and RX structures. More advanced examples define other interface options.
- The .json file declares the AFU UUID. An OPAE script generates the UUID. The RTL loads the UUID from afu_json_info.vh.
- sw/Makefile generates afu_json_info.h. Software loads the UUID from afu_json_info.h.

4.1.3. Troubleshooting Client-Server Simulation

If the afu_sim_setup command fails, confirm that:

- afu_sim_setup is on your PATH. afu_sim_setup should be in /usr/bin or in <opae install path> if you built OPAE from source files.
- You have Python version 2.7 or higher installed.

If you are unable to build and execute the simulator, it is likely that you did not install your RTL simulation tool properly.

When you try to build and run the software, if you see an "Error enumerating AFCs" message, you omitted setting USE_ASE=1 on the make command line. The software is searching for a physical FPGA device. To recover, repeat the steps from the make clean command.

5. AFU Examples

Table 2. AFU Examples

Each AFU example includes a detailed README file, providing an operational description and notes on how to simulate the design. For a full understanding of the simulation process, review the README file in each AFU example.

AFU	Description
hello_mem_afu	hello_mem_afu demonstrates an AFU that builds a simple state machine to access memory. The state machine is capable of several access patterns to local memory directly attached to FPGA pins, such as DDR4 DIMMs. This memory is distinct from the host memory accessed over CCI-P. The host manages the hello_mem_afu controller state machine using memory-mapped I/O (MMIO) requests to control and status registers (CSRs).
hello_intr_afu	hello_intr_afu demonstrates the application interrupt feature in the ASE.
dma_afu ⁽²⁾	dma_afu demonstrates a DMA Basic Building Block for host to FPGA, FPGA to host, and FPGA to FPGA memory transfers. When simulating this AFU, the buffer size used for DMA transfer is small to keep the simulation time reasonable. For more information, refer to the <i>DMA Accelerator Functional Unit (AFU) User Guide</i> .
n1b_mode_0	n1b_mode_0 is a CCI-P system demonstrating the memory copy test. \$OPAE_PLATFORM_ROOT/sw/opae-<release_number>/sample/hello_fpga.c includes n1b_mode_0. <pre>\$ sh regress.sh -a <afu dir> -r rtl_sim -s < vcs modelsim questa > [-i <opae install path>] -b <path to opae source dir></pre>
streaming_dma	streaming_dma demonstrates how to transfer data between host memory and an FPGA streaming port. For more information, refer to the <i>Streaming DMA Accelerator Functional Unit (AFU) User Guide</i> .
hello_afu	hello_afu is a simple AFU that demonstrates the primary CCI-P interface. The RTL satisfies the bare minimum requirements of an AFU, responding to MMIO reads to return the device feature header and the AFU's UUID.

Related Information

- [DMA Accelerator Functional Unit \(AFU\) User Guide](#)
 For information on how to compile and execute the dma_afu on your Intel PAC with Intel Arria 10 GX FPGA.
- [Streaming DMA Accelerator Functional Unit \(AFU\) User Guide](#)
 For information on how to compile and execute the streaming_dma_afu on your Intel PAC with Intel Arria 10 GX FPGA.
- [DMA Accelerator Functional Unit User Guide: Intel FPGA Programmable Acceleration Card D5005](#)
 For information on how to compile and execute the dma_afu on your Intel FPGA PAC D5005.

⁽²⁾ Available only in Client-Server mode

- [Streaming DMA Accelerator Functional Unit User Guide: Intel FPGA Programmable Acceleration Card D5005](#)
For information on how to compile and execute the dma_afu on your Intel FPGA PAC D5005.



6. Troubleshooting

If the following error appears during simulation, correct it by following the steps below.

Error Message

```
# [SIM] An ASE instance is probably still running in current directory !  
# [SIM] Check for PID 28816  
# [SIM] Simulation will exit... you may use a SIGKILL to kill the simulation  
process.  
# [SIM] Also check if .ase_ready.pid file is removed before proceeding.
```

Solution

1. Type `kill ase_simv` to kill zombie simulation processes and remove any temporary files left behind by failed simulation processes or lock ups.
2. Delete the `.ase_ready.pid` file, found in the `$ASE_WORKDIR` directory.

7. ASE Quick Start User Guide Archives

Intel Acceleration Stack Version	User Guide
2.0	Intel Accelerator Functional Unit (AFU) Simulation Environment (ASE) Quick Start User Guide
1.2	Intel Accelerator Functional Unit (AFU) Simulation Environment (ASE) Quick Start User Guide
1.1	Intel Accelerator Functional Unit (AFU) Simulation Environment (ASE) Quick Start User Guide
1.0	Intel Accelerator Functional Unit (AFU) Simulation Environment (ASE) Quick Start User Guide

8. Document Revision History for ASE Quick Start User Guide

Document Version	Intel Acceleration Stack Version	Changes
2020.03.06	1.2.1 and 2.0.1	Updated the following: • <i>System Requirements</i>
2019.08.05	2.0	• Updated the Intel Quartus Prime Pro Edition version in <i>System Requirements</i>. • Added the hello_afu in <i>AFU Examples</i>. • Removed information about simulating in regression mode. • Added a new section: <i>ASE Quick Start User Guide Archives</i>.
2018.12.04	1.2	Added Ubuntu support.
2018.08.06	1.1	Updated the system requirements, directory structure and corresponding filenames.
2018.04.10	1.0	Initial release.