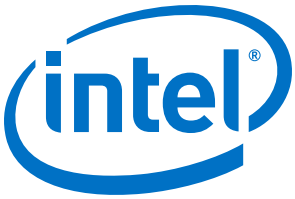


Accelerator Functional Unit (AFU) Developer's Guide

Updated for Intel® Acceleration Stack: **1.0 Production**

Subscribe
Send Feedback

UG-20114 | 2018.04.11
Latest document on the web: **PDF** | **HTML**

Contents

1. About this Document

This document serves as a hardware developers guide for developing Accelerator Functional Units (AFUs) for the Intel Acceleration Stack for Intel Xeon® CPU with FPGAs product, hereafter referred to as the Acceleration Stack.

1.1. Intended Audience

The intended audience consists of FPGA RTL designers developing AFUs for the Acceleration Stack on the Intel Programmable Acceleration Card with Intel Arria® 10 GX FPGA (referred to as Intel PAC with Intel Arria 10 GX FPGA throughout this document) hardware platform.

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. Related Documentation

Table 2. Item Description

Item	Description
Intel Acceleration Stack Quick Start Guide for Intel Programmable Acceleration Card with Intel Arria 10 GX FPGA	This document describes the Acceleration Stack and provides instruction for hardware and software installation and setup required for development with the stack.
Acceleration Stack for Intel Xeon CPU with FPGAs Core Cache Interface (CCI-P) Reference Manual	This document describes the CCI-P protocol and requirements placed on AFUs.

continued...

Item	Description
Intel Accelerator Functional Unit (AFU) Simulation Environment (ASE) User Guide	This document provides instructions on how to use the Intel Accelerator Functional Unit (AFU) Simulation Environment (ASE).
Open Programmable Acceleration Engine (OPAE) Tools Guide	This user guide documents the utilities provided in the Open Programmable Acceleration Engine (OPAE) software component of the Acceleration Stack.
Design Debugging with the Signal Tap Logic Analyzer	This documentation describes Signal Tap and its use for general FPGA debug and provides a baseline reference for remote Signal Tap debug of AFUs.

1.4. Acronym List for Accelerator Functional Unit Developer's Guide

Table 3. Acronyms

Acronyms	Expansion	Description
AFU	Accelerator Functional Unit	Hardware Accelerator implemented in FPGA logic which offloads a computational operation for an application from the CPU to improve performance.
AF	Accelerator Function	Compiled Hardware Accelerator image implemented in FPGA logic that accelerates an application. An AFU and associated AFs may also be referred to as GBS (Green-Bits, Green BitStream) in the Acceleration Stack installation directory tree and in source code comments.
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.
CCI-P	Core Cache Interface	CCI-P is the standard interface AFUs use to communicate with the host.
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.
FIM	FPGA Interface Manager	The FPGA hardware containing the FPGA Interface Unit (FIU) and external interfaces for memory, networking, etc. The FIM may also be referred to as BBS (Blue-Bits, Blue BitStream) in the Acceleration Stack installation directory tree and in source code comments. The Accelerator Function (AF) interfaces with the FIM at run time.
NLB	Native Loopback	The NLB performs reads and writes to the CCI-P link to test connectivity and throughput.

continued...

Acronyms	Expansion	Description
OPAE	Open Programmable Acceleration Engine	The OPAE is a software framework for managing and accessing AFs.
PR	Partial Reconfiguration	The ability to dynamically reconfigure a portion of an FPGA while the remaining FPGA design continues to function.
TCP	Transmission Control Protocol	TCP is a standard internet protocol that defines how to establish and maintain a network conversation through which application programs can exchange data.

1.5. Acceleration Glossary

Table 4. 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).

2. Introduction

This chapter outlines the prerequisites for AFU development.

2.1. Getting Started with the Acceleration Stack

Before using this guide, refer to the *Intel Acceleration Stack Quick Start Guide for Intel Programmable Acceleration Card with Intel Arria 10 GX FPGA*, referred to as *Quick Start Guide* throughout this document. The *Quick Start Guide* provides an overview of the Acceleration Stack and provides instruction for installation and setup of hardware and software components of the stack. It is essential to familiarize yourself with the concepts developed for the Acceleration Stack and to complete the installation and setup procedures covered in the *Quick Start Guide*.

This guide for AFU development builds on the concepts and environment setup established in the *Quick Start Guide*.

Related Information

Intel Acceleration Stack Quick Start Guide for Intel Programmable Acceleration Card with Intel Arria 10 GX FPGA

2.1.1. Development Environment References

Throughout this guide, the following two references taken from the *Quick Start Guide* are used to refer to the Acceleration Stack installation:

- $DCP_LOC—This reference points to the directory where the Acceleration Stack release tarball was unarchived as instructed in the *Quick Start Guide*.
- $OPAE_LOC—This reference points to the directory where the OPAE software source code included in the Acceleration Stack release tarball was unarchived as instructed in the *Quick Start Guide*.

All code and command line references and examples in this guide assume your PATH environment variable includes the following locations to executables in the Acceleration Stack installation:

- `$DCP_LOC/bin`—This location contains the utilities and helper scripts included in the Acceleration Stack release tarball:
  - `run.sh`
  - `clean.sh`

2.1.2. FPGA Tools and IP Requirements

Generating Accelerator Function (AF) for OPAE requires the following software and IP:

- Intel Quartus® Prime Pro Edition software version 17.0.0 (only version supported)
- Intel FPGA PCI Express SR-IOV Block IP license
- `python2-jsonschema` package from the **epel** repository

For requirements when using ASE for AFU functional verification, refer to the *Intel Accelerator Functional Unit (AFU) Simulation Environment (ASE) User Guide*.

Related Information

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

2.2. Base Knowledge and Skills Prerequisites

The Acceleration Stack is an advanced application of FPGA technology. Most of the platform-level complexity has been abstracted away for the AFU developer by the FPGA Interface Manager (FIM) in the FPGA static region. This guide assumes the following FPGA logic design-related knowledge and skills:

- Familiarity with PR compilation flows, including the Intel Quartus Prime Pro Edition PR flow, concepts of physical and logical partitioning in the FPGA, module boundary best practices, and resource restrictions.

  The physical and logical partitioning of the FIM static region and the PR regions for AFUs has already been done. User AFUs conveniently plug-in to the structure defined by the Acceleration Stack with a well-structured set of standard interface signals to the FIM. This level of abstraction allows you to concentrate on your area of expertise in end application space by minimizing time and effort on the PR flow itself. The Acceleration Stack provides helper scripts to automate the PR flow during compilation of the AFU RTL for generating an AF for use by OPAE. The Acceleration Stack has already laid out the structure, and familiarity with PR flows is a plus for design of an AFU within this predetermined structure.
- Knowledge and skills in static timing closure, including familiarity and skill with the TimeQuest Timing Analyzer tool in Intel Quartus Prime Pro Edition, applying timing constraints, Synopsys* Design Constraints (`.sdc`) language and Tcl scripting, and design methods to close on timing critical paths.
- Knowledge and skills with industry standard RTL simulation tools supported by the Acceleration Stack. For more information, refer to the *Intel Accelerator Functional Unit (AFU) Simulation Environment (ASE) Quick Start Guide*.
- Knowledge and skill with the Signal Tap Logic Analyzer tool in the Intel Quartus Prime Pro Edition software.

Related Information

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

3. Getting Started with AFU Development

This chapter guides you through the process to generate an AF for the `nlb_mode_0` example AFU provided in the Acceleration Stack installation. Successful completion of the steps in this chapter quickly verifies your AFU development environment using a known-good design.

Build the `nlb_mode_0` example AFU by invoking the `run.sh` script from a terminal window as shown in Example 1.

Note: This step takes about 45 minutes to complete.

Example 1. Compile `nlb_mode_0` Example AFU

```
$ cd $DCP_LOC/hw/samples/nlb_mode_0
$ $DCP_LOC/bin/run.sh
```

When the shell script completes, it indicates successful generation of the AF: `$DCP_LOC/hw/samples/nlb_mode_0/nlb_400.gbs`.

You can optionally repeat the steps in the *Quick Start Guide* to run the `hello_fpga` host application with the newly generated AF.

You can optionally restore the fresh state of the `nlb_mode_0` example AFU design by invoking the `clean.sh` script from a terminal window as shown in Example 2.

Example 2. Restore the nlb_mode_0 Example AFU Design

```
$ cd $DCP_LOC/hw/samples/nlb_mode_0
$ $DCP_LOC/bin/clean.sh
```

Successfully compiling the `nlb_mode_0` example AFU verifies that your environment is setup and ready to begin developing your own custom AFUs.

4. Custom AFU Development

4.1. AFU Integration within the Acceleration Stack

To facilitate dynamically loading AFUs, the Acceleration Stack utilizes a partial reconfiguration (PR) scheme. The FIM contains one or more PR regions for loading AFUs and a static region that provides services and resources to loaded AFUs.

4.1.1. The FPGA Interface Manager (FIM)

The FIM includes the static region and one or more PR region partitions for loading AFUs from OPAE. The static region provides services to AFUs loaded in PR regions that include a host connection via CCI-P protocol over PCIe SR-IOV, a local pool of SDRAM memory, and clock and reset resources. The FIM static region also provides services to OPAE for dynamically loading AFUs and performing system management tasks (for example, version identification).

The FIM is part of the Intel PAC with Intel Arria 10 GX FPGA hardware platform and is not modifiable.

The PR regions in the FIM are undefined AFUs preconfigured upon power up – host applications must use OPAE to load AFUs into the PR regions.

Note: The 1.0 Production release of the FIM supports one PR region.

The FIM bitstream is included in the Acceleration Stack installation and initially configures the FPGA at power up from configuration flash residing on the Intel PAC with Intel Arria 10 GX FPGA.

For instructions on flashing the on-board configuration flash with the FIM bitstream, refer to the *Quick Start Guide*.

Related Information

Intel Acceleration Stack Quick Start Guide for Intel Programmable Acceleration Card with Intel Arria 10 GX FPGA

4.1.2. The AFU PR Regions

Host software uses OPAE utilities and APIs to load an AF into a PR region in the FIM . An AF is the combination of an AFU PR bitstream and associated AFU metadata. The AFU PR bitstream is the output from Intel Quartus Prime Pro Edition PR compilation of your AFU RTL design with the FIM design database provided in the Acceleration Stack installation. The AFU metadata is used to provide OPAE information on AFU characteristics and operational parameters and is defined in a separate JSON file. The **Packager** utility included in the Acceleration Stack installation generates the AF from

the AFU PR bitstream and AFU metadata. It is possible to have several AF variations for a given AFU revision by combining its PR bitstream with unique metadata using the **Packager** utility.

The 1.0 Production release supports dynamically swapping multiple AFUs within a single PR region for each Intel PAC with Intel Arria 10 GX FPGA installed in a system.

The rest of this chapter describes how to design an AFU within the platform and services provided by the FIM static region, compile an AFU PR bitstream compatible with the FIM, and generate an AF for use by OPAE.

For usage information on the **Packager** utility and JSON file metadata format, supported keyword parameters, and minimum metadata requirements, refer to the *OPAE Tools Guide*.

Related Information

Open Programmable Acceleration Engine (OPAE) Tools Guide

4.2. Intel Quartus Prime Pro Edition Flow

This release of the Acceleration Stack is designed to work with Intel Quartus Prime Pro Edition, version 17.0.0.

The Intel Quartus Prime Pro Edition partial reconfiguration (PR) compilation flow is used to compile AFUs in combination with the FIM design database to generate AFU PR bitstreams. The PR compilation flow is supported only at the command line using the scripts provided with the Acceleration Stack – you cannot use the Intel Quartus Prime Pro Edition GUI to generate a PR bitstream for the AFU.

The shell script, `$DCP_LOC/bin/run.sh`, implements the necessary command line steps.

You can use the GUI point tools in Intel Quartus Prime Pro Edition for tasks such as TimeQuest timing analysis, Chip Planner view, adding debug instances and nodes, and viewing compilation reports.

4.3. Designing and Compiling AFUs

To generate an AFU PR bitstream, AFU developers should perform synthesis, place and route, and timing closure on their AFU while importing the FIM design database from the library as part of the PR compilation performed by the `run.sh` script.

The Acceleration Stack installation includes example AFU designs in the `$DCP_LOC/hw/samples` directory.

The overall flow to generate an AFU PR bitstream is as follows:

- The Acceleration Stack installation provides the compiled database for the FIM and an Intel Quartus Prime PR build project structure to support integrating your AFU within the framework provided by the FIM static region.
- The FIM design database and the Intel Quartus Prime PR build project structure reside in `$DCP_LOC/hw/lib`.

  Note: Do not modify these library files.
- The Intel Quartus Prime PR build project contains two revisions:
  - **afu_synth**
  - **afu_fit**
- The revision **afu_synth** is used to synthesize the user AFU.
- The revision **afu_fit** is used to generate an AFU PR bitstream by importing the **qdb** of the FIM from the library and the synthesized snapshot of your AFU from the **afu_synth** revision.
- AFU developers should close timing on their AFU on the **afu_fit** revision.
- Only script-based steps are supported for your AFU synthesis, place and route (PAR), and bitstream generation. GUI-based steps can be used for timing analysis, adding debug instances, and viewing compilation results.
- To run synthesis of multiple AFUs or PAR jobs in parallel, create multiple copies of the AFU's project directory.

4.3.1. Available Resources in the PR Region

The PR region in the FIM has the following FPGA resources available to the AFU design:

- ALMs: 382,273
- M20Ks: 2468
- DSP Blocks: 1402

4.3.2. AFU Project Structure

An AFU project is a design directory that contains the following required components:

- A subdirectory named "hw".
- A Quartus settings file (`.qsf`) located in the "hw" subdirectory, named "`afu.qsf`".

For example, the AFU project directory for the `hello_afu` example included in the Acceleration Stack installation is located at `$DCP_LOC/hw/samples/hello_afu`. The required settings file is located at `$DCP_LOC/hw/samples/hello_afu/hw/afu.qsf`.

Place all Intel Quartus Prime settings for compiling your AFU design in the `afu.qsf` file.

At minimum, the `afu.qsf` settings file must point to all AFU design flles, including RTL source, Qsys subsystems (`.qsys`), IP variations (`.ip`), timing constraint files (`.sdc`), Signal Tap files (`.stp`), and Tcl scripts (`.tcl`). AFU design files can be located

anywhere in the filesystem that can be resolved at Intel Quartus Prime PR compile time from the path references in the `afu.qsf` file. However, the following additional project structure facilitates the use of the OPAE tools and the ASE:

- Place RTL source and `.json` files in `<afu_project_dir>/hw/rtl`
- Place simulation scripts for ASE in `<afu_project_dir>/hw/sim`
- Place the host OPAE application source and build scripts in `<afu_project_dir>/sw`

Relative path references to AFU design files in afu.qsf must have the following format:

```
../<path-to-design-file-relative-to-afu-proj-dir>
```

For example, the AFU design files for the hello_afu example are located in `$DCP_LOC/hw/samples/hello_afu/hw/rtl`. The `hello_afu` example's `afu.qsf` file points to the top level AFU RTL source file, `afu.sv`, with the following setting:

```
set_global_assignment -name SYSTEMVERILOG_FILE "../hw/rtl/afu.sv"
```

Where for this example `<path-to-design-file-relative-to-afu-proj-dir>` is `hw/rtl/afu.sv`.

4.3.3. AFU Design Structure

The top-level wrapper for all AFU designs is provided by the `ccip_std_afu` module, which is defined in the `ccip_std_afu.sv` file included in the Acceleration Stack installation. All AFU top-level logic and submodules must be instanced within the `ccip_std_afu` top-level wrapper module. Your AFU is restricted to the module port list defined by `ccip_std_afu`, which includes all FIM clock and reset resources, and the FIU and DDR4 bank interfaces. There are no restrictions to the design hierarchy beneath this top-level wrapper.

Your AFU project must include the `ccip_std_afu.sv` source file by reference in the `afu.qsf` file. This file is included in the design file sets of the example AFUs included in the Acceleration Stack installation. You can either directly refer to `ccip_std_afu.sv` or copy it over to your AFU design file set from the `hello_afu` example's design file directory: `$DCP_LOC/hw/samples/hello_afu/hw/rtl`.

The `hello_afu` example provides a simple example of how to implement an AFU design in the `ccip_std_afu.sv` top-level wrapper file.

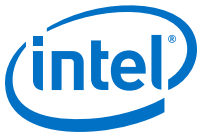
The `nlb_mode_0` example AFU provides an example for a more involved AFU that includes multiple RTL source files and Synopsys Design Contraints (`.sdc`) files for the AFU.

4.3.4. Utilizing Intel FPGA Basic Building Blocks (BBBs)

Intel FPGA Basic Building Blocks (BBBs) are reference designs of common functions that can be used in AFU designs. These references are provided as-is. They are not validated by Intel. The available BBBs, including documentation, are maintained at the GitHub site.

Related Information

Basic Building Blocks (BBB) for OPAE-managed Intel FPGAs



4.3.5. Partial Reconfiguration Design Guidelines

- The bitstreams used for Partial reconfiguration should be generated using the script-method provided by the `$DCP_LOC/bin/run.sh` script.
- Partial reconfiguration switches the PR region from one AFU to another AFU. Any software application exercising an AFU in the PR region should be terminated before initiating PR with OPAE to switch in a new AFU. This includes the remote debug feature.
- LAB/MLAB with initial content is not supported either in RAM mode or in ROM mode within the AFU. There are no such restrictions on **M20K** block usage.
- When using **M20K** or MLAB on-chip memory blocks with initialized contents, implement clock enable logic in the AFU to avoid spurious writes into the memories upon exit of PR.
- Logic in the AFU should not depend on initial values or states coded through initial statements.
- After partial reconfiguration, the registers in the PR region (AFU) come up in an indeterminate state. To restore initial condition, a reset pulse is generated at the CCI-P interface after PR. AFUs must use this reset to restore all initial conditions.
- The PR region must contain only core resources like LABs, RAMs and DSPs. PLLs and Clock control blocks cannot be instantiated in the PR region.
- If PR compilation fails due to M20K memory block overutilization, add the following `quartus.ini` setting to enable a more aggressive conversion to available MLABs during compilation:

  ```
  fit_restrict_meab_usage=2394
  ```

  If a `quartus.ini` file does not already exist in the AFU project directory's `build` subdirectory (*<afu_project_dir>*`/build`, created after invoking the `run.sh` script), then use your preferred text editor to create it with the above setting added on a single line.
- If PR compilation results in timing violations in the FIM static region, retry PR compilation with a different fitter seed value.

4.3.6. AFU Design Guidelines

Follow these guidelines when designing a custom AFU:

4.3.6.1. General Guidelines

- The AFU build flow supports the following RTL language standards:
  - SystemVerilog 2005
  - VHDL 1993
- Reset and initialize all output registers.
- Generate an AFU ID for new AFUs using the third-party tool, UUID Generator.

Related Information

- Intel Quartus Prime Pro Edition Handbook Volume 1 Design and Compilation
  For more information on RTL language standards.

- Acceleration Stack for Intel Xeon CPU with FPGAs Core Cache Interface (CCI-P) Reference Manual
  For more information on generating AFU IDs.

4.3.6.2. Utilizing Clock Resources

The FIM provides several clock resources for use by AFUs. One set of clock resources is the user clock group, which includes `uClk_usr` and `uClk_usrDiv2`. Unlike `pClk` and its derivatives whose frequencies are fixed by the *CCI-P Specification*, the user clocks can be programmed for a range of frequencies supported by the AFU.

User clocks get provisioned by OPAE when an AF is loaded by the `fpgaconf` utility. When the `fpgaconf` utility loads an AF, it will configure the PLL in the FIM that sources the user clocks with the frequency specified by a `key:value` pair found in the AF metadata generated by the packager utility. The desired user clock frequency `key:value` pair can be specified in a `.json` file or can be specified with a command line option (overrides entry in the `.json` file) to the packager utility. You can use the packager to generate AFs with unique metadata user clock frequency values for a single AFU PR bitstream.

The FIM reset resource, `pck_cp2af_softReset`, is not released until all clock resources are stable and locked, including the user clocks.

The AFU design must close timing on the user clocks at the maximum frequency to be supported by the AFU. Place associated clock timing constraints in a `.sdc` file, and refer to the `.sdc` file in the `afu.qsf` file.

For usage information on the **Packager** utility and `.json` file metadata format, supported keyword parameters, and minimum metadata requirements, refer to the *OPAE Tools User Guide*.

Related Information

Open Programmable Acceleration Engine (OPAE) Tools Guide

4.3.6.3. Interfacing with the FPGA Interface Unit (FIU)

The *The Intel Acceleration Stack for Intel Xeon CPU with FPGAs Core Cache Interface (CCI-P) Reference Manual* documents all the requirements on an AFU interfacing with the FIU in the FIM over the CCI-P protocol as well as requirements for CSR and address mapping. An AFU design must meet all the requirements specified in the following sections of the CCI-P reference manual:

- *CCI-P Interface*
- *AFU Requirements*
- *Device Feature List*

The above sections in the CCI-P reference manual include requirements unique to the Intel Xeon Processor with Integrated FPGA (referred to as Integrated FPGA Platform throughout this document) hardware platform, but most of the information applies to the Intel PAC with Arria 10 platform. The notable differences between the two platforms are that the PAC does not have a UPI channel or second PCIe link, and no accelerator cache is implemented in the FIM.

The `hello_afu` example AFU included with the Acceleration Stack provides an example implementation of a simple Device Feature List that meets the requirements for an AFU as specified by the CCI-P reference manual. The `nlb_mode_0` and `dma_afu` example AFUs provide example implementations of more featured Device Feature Lists.

For more information about the Avalon®-MM interface, refer to the *Avalon Memory-Mapped Interfaces*.

Related Information

- CCI-P Interface
- AFU Requirements
- Device Feature List
- Avalon Memory-Mapped interfaces

4.3.6.4. Accessing Local DDR4 SDRAM

The AFU accesses local memory on the PAC through Avalon Memory-Mapped (Avalon-MM) slave interfaces provided by the FIM. Each bank has its own 512-bit wide Avalon-MM slave interface and operates at 267 MHz. Each bank interface is synchronous to its own 267 MHz clock source provided by the FIM, which AFUs must use to synchronize accesses to DDR4. The DDR4 Avalon-MM slave interfaces in the FIM support single bursts. There is no support for response status or posted writes.

4.3.7. Using the Scripts to Generate AFs

The Acceleration Stack installation includes two scripts to facilitate Intel Quartus Prime Pro Edition PR compilation of AFUs. These scripts are located in the `$DCP_LOC/bin` directory.

4.3.7.1. run.sh

This script performs AFU synthesis on the **afu_synth** revision, fits the synthesis snapshot of the AFU and the final snapshot of the FIM design database, and invokes the **Packager** to generate the AF file (.gbs).

For example, the command sequence shown in Example 3 compiles the `hello_afu` example AFU and generates an AF.

Example 3. Compile hello_afu Example AFU

```
$ cd $DCP_LOC/hw/samples/hello_afu
$ run.sh
```

By default, the `run.sh` script will generate the loadable AFU image using the first matching `.json` metadata file found relative to the AFU project directory: `hw/rtl/*.json, hw/*.json, *.json`

You can explicitly specify a particular `.json` file to use for generating the AF by passing a positional argument as shown in Example 4.

Example 4. Pass an Argument to a Specific `.json` File

```
$ cd $DCP_LOC/hw/samples/hello_afu
$ run.sh <my-alternate-json-location>/<filename>.json
```

If the `.json` filename starts with the "-" character, pass the argument with a preceding "--".

The `run.sh` script supports the following options:

```
--packager|-p <path-to-the-packager-binary>/packager
```

By default, `run.sh` relies on your PATH environment variable to point to the **Packager** binary located in `$DCP_LOC/bin`. If PATH has not been setup in this way, use the `–packager|-p` option to explicitly point to the location of the **Packager** binary for generating the AF.

```
--bbs-lib|-l <path-to-the-bbs-database>/lib
```

By default, `run.sh` finds the FIM design database using its relative path in the Acceleration Stack installation. As installed, the FIM design database is located at `$DCP_LOC/hw/lib`, but if this path or the `run.sh` script's relative location to it has been altered, use the `–-bbs-lib|-l` option to point to the FIM design database.

4.3.7.2. clean.sh

When run from the AFU's project directory, the `clean.sh` script will restore the AFU design by deleting all Intel Quartus Prime Pro Edition compilation output from an invocation of the `run.sh` script. The `clean.sh` script takes no arguments or options. Example 5 shows an example of using the `clean.sh` script.

Example 5. Clean Up the AFU Design Directory

```
$ cd $DCP_LOC/hw/samples/hello_afu
$ clean.sh
```

4.4. Using the Packager to Update Metadata

The `run.sh` script invokes the **Packager** after compiling an AFU to generate an AF. For situations where you want to either update the metadata in an existing AF or create an additional AF with unique metadata without recompiling the AFU, run the **Packager** standalone.

The **Packager** utility is included in the OPAE installation.

For more information, refer to the *OPAE Tools Guide*.

Related Information

Open Programmable Acceleration Engine (OPAE) Tools Guide

5. AFU Functional Verification

The AFU supports functional verification of AFU RTL code using host application C code developed for the OPAE API without the need for Intel PAC with Intel Arria 10 GX FPGA hardware. The ASE virtualizes the AFU's physical link with the host, models certain aspects of the OPAE host memory model, and supports communication between the OPAE host application and supported RTL simulation tools used to emulate the AFU running on actual Intel Programmable Acceleration Card (PAC) hardware.

ASE is useful for verifying your AFU's interoperability with the rest of the Acceleration Stack using a quick, iterative functional debug environment to minimize time spent in subsequent portions of the AFU development flow that involve more time-intensive steps (for example, PAR, timing closure). ASE also enables a more cost-efficient development environment by removing the dependency on PAC hardware for early functional debug of AFU interoperability within the Acceleration Stack.

Related Information

- Intel Accelerator Functional Unit (AFU) Simulation Environment (ASE) User Guide
  For more information about ASE.
- Intel Accelerator Functional Unit (AFU) Simulation Environment (ASE) Quick Start User Guide

6. AFU In-System Debug

The Acceleration Stack provides a remote Signal Tap facility. Use remote Signal Tap to debug an AFU in-system. The Signal Tap II Logic Analyzer, included in the Intel Quartus Prime Pro Edition, allows you to trigger on AFU signal events and capture traces of signals in your AFU design. The remote capability allows for control of trigger conditions and upload of captured signal traces from a networked workstation running the Signal Tap GUI.

Signal Tap is an in-system logic analyzer that you can use to debug FPGA logic. Conventional (non-remote) Signal Tap uses the physical FPGA JTAG interface and a Intel FPGA Download Cable II to bridge the Intel Quartus Prime Signal Tap application running on a host system with the Signal Tap controller instances embedded in the FPGA logic. With Remote Signal Tap, you can achieve the same result without physically connecting to JTAG, which enables signal-level, in-system debug of AFUs deployed in servers where physical access is limited.

In addition to Signal Tap, the remote debug facility in OPAE supports the following in-system debug tools included with the Intel Quartus Prime Pro Edition:

- In-system sources and probes
- In-system Memory Content Editor
- Signal Probe
- System Console

This section describes how to generate an AF with remote Signal Tap enabled. This section then describes how to debug a user AFU using OPAE's **mmlink** utility, the System Console utility, and Intel Quartus Prime Pro Edition.

The `nlb_mode_0_stp` variation of the `nlb_400` example AFU is used to illustrate how to enable and use remote Signal Tap.

Related Information

Design Debugging with the Signal Tap Logic Analyzer
For more information about Signal Tap.

6.1. Remote Signal Tap Setup and Use

6.1.1. Instrumenting the AFU Design for Signal Tap

To add Signal Tap trigger and data nodes from signals in your AFU, follow the method documented in the related information for Signal Tap.

For working within the PR compilation flow for compiling the Signal Tap-enabled AFU, follow these flow steps:

1. If the run.sh script has been run on the AFU project, skip to step 2, otherwise copy the Intel Quartus Prime PR build project from the Acceleration Stack installation to your AFU project directory:

   ```
   $ cp -rLf $DCP_LOC/hw/lib/build <path-to-afu-proj-dir>
   ```

2. Open the Intel Quartus Prime PR build project from the Intel Quartus Prime Pro Edition GUI:

   ```
   $<path-to-afu-proj-dir>/build/dcp.qpf
   ```

   Select the afu_synth revision.

   If you have already run the run.sh script or otherwise ran your AFU through the synthesis step, skip to step 4.

3. From the Quartus GUI, perform an Analysis & Elaboration on your AFU RTL to generate a netlist from which to add debug nodes with the Signal Tap tool.
4. Invoke the Signal Tap tool from the Intel Quartus Prime Pro Edition GUI and add your AFU signals for trigger/data debug nodes as usual.
5. When done adding debug nodes, save the .stp file and optionally choose to add the .stp file to the Intel Quartus Prime Pro Edition project and enable Signal Tap for the project.
6. Exit Signal Tap.
7. Exit the Intel Quartus Prime Pro Edition GUI.

In the `nlb_mode_0_stp` example, `<path-to-afu-proj-dir>` is `$DCP_LOC/hw/samples/nlb_mode_0_stp`.

The `nlb_mode_0_stp` example already has a `.stp` file: `$DCP_LOC/hw/samples/nlb_mode_0_stp/hw/par/stp_basic.stp`.

6.1.2. Enable Remote Debug and Signal Tap

Signal Tap must be enabled in the AFU `afu.qsf` file. You must add the following settings to the `afu.qsf` file even if you enabled Signal Tap when saving the `.stp` file and exiting the Signal Tap GUI.

The following shows the required Intel Quartus Prime settings in `afu.qsf`:

Quartus Settings for Enabling

```
set_global_assignment -name VERILOG_MACRO INCLUDE_REMOTE_STP

set_global_assignment -name SIGNALTAP_FILE \
../<path-relative-to-afu-proj-dir>/<stp-filename>.stp

set_global_assignment -name ENABLE_SIGNALTAP ON

set_global_assignment -name USE_SIGNALTAP_FILE \
../<path-relative-to-afu-proj-dir>/<stp-filename>.st
```

The `nlb_mode_0_stp` example already has the above settings added to its `afu.qsf` file located in `$DCP_LOC/hw/samples/nlb_mode_0_stp/hw`.

6.1.3. Generate the Remote Debug Enabled AF

After adding the above settings to the AFU's `afu.qsf` file, generate the remote debug enabled AF:

```
$ cd <afu-proj-dir>
$ run.sh
```

The `nlb_mode_0_stp` example already has a remote debug enabled AF : `$DCP_LOC/hw/samples/nlb_mode_0_stp/bin/nlb_mode_0_stp.gbs`.

6.1.4. Prepare the Remote Debug Host

Copy the following files from the Acceleration Stack installation over to a convenient working directory on the remote debug host:

- The Signal Tap `.stp` file compiled with your AFU. In the case of the `nlb_mode_0_stp` example AFU, the `.stp` file is located in the Acceleration Stack installation as `$DCP_LOC/hw/samples/nlb_mode_0_stp/hw/par/stp_basic.stp`.
- The following two files support establishing a connection on the remote debug host to the AFU Signal Tap instances on the Intel PAC with Intel Arria 10 GX FPGA. These files are part of the Acceleration Stack release – do not modify them.

```
$DCP_LOC/hw/remote_debug/mmlink_setup_profiled.tcl$DCP_LOC/hw/remote_debug/
remote_debug.sof
```

6.1.5. Running a Remote Debug Session

6.1.5.1. Connect to the AFU Target

Follow these steps on the debug target host with the PAC installed:

1. If not already done, load the Signal Tap-enabled AFU.

```
$ sudo fpgaconf $DCP_LOC/hw/samples/nlb_mode_0_stp/bin/nlb_mode_0_stp.gbs
```

2. Open a TCP port to accept incoming connection requests from remote debug hosts.

```
$ sudo mmlink -P 3333
```

Follow these steps on the remote debug host:

1. Use **System Console** to connect to the debug target host's TCP port for Signal Tap debug connection on the target AFU. If the remote debug host is a Windows platform, open a command shell to run the below commands.

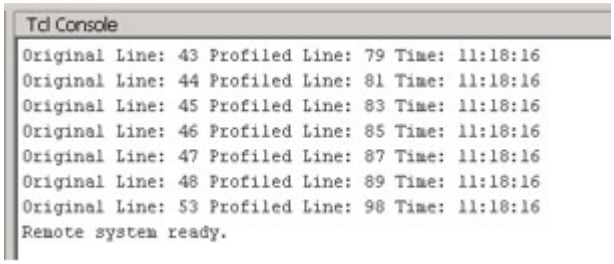
```
$ cd <path-to-debug-working-directory>
$ system-console --rc_script=mmlink_setup_profiled.tcl
remote_debug.sof <IP-address-of-debug-target-host> 3333
```

The above command assumes your PATH environment variable on the remote debug host is setup to point to the following location in the Intel Quartus Prime Pro Edition installation:

```
<installation-path>/<q-edition>/sopc_builder/bin
```

where <q-edition> is "quartus" for Intel Quartus Prime Pro Edition or Intel Quartus Prime Standard Edition. For an Intel Quartus Prime Programmer Edition installation, <q-edition> is qprogrammer.

2. After issuing the above commands, the System Console window appears. Wait for the "Remote system ready" message in the Tcl Console pane.



6.1.5.2. Using Signal Tap with a Remote Target Connection

Perform these steps on the remote debug host:

1. Invoke the Signal Tap GUI.
2. From **File ➤ Menu**, navigate to and open the `.stp` file you copied over from the "Prepare the Remote Debug Host" section when you were preparing the remote debug host for debugging the AFU.
3. Complete connecting to the Signal Tap controller instances in the target AFU by selecting **"System Console on ... Sld Hub Controller System"** from the Hardware drop-down option box in the **JTAG Chain Configuration** pane.
4. Wait for the **"JTAG ready"** response.

At this point, you are ready to perform in-system debug with the Signal Tap GUI in the same manner as with the conventional target connection method.

Related Information

- Prepare the Remote Debug Host on page 20
- Design Debugging with the Signal Tap Logic Analyzer
  For more information about Signal Tap.

6.1.5.3. Stimulating the Target AFU for In-System Debug

Use host application C code software designed for the OPAE API to stimulate the AFU and verify proper operation within the Acceleration Stack. Leave the `mmlink` tool running in a separate terminal window on the debug target host while the remote debug host is connected. The `mmlink` process will continuously output status to the terminal window. Invoke OPAE host application or test software from their own terminal windows on the debug target host.

6.1.5.3.1. Accessing the AFU in Shared Mode

When using OPAE application/test code running on the debug target host to stimulate the AFU for the purposes of in-system debug, both the `mmlink` tool and your host application/test code must have simultaneous access to the AFU. For this to happen, any user space code calls to the `fpgaOpen()` OPAE API function must pass the FPGA_OPEN_SHARED flag. The Acceleration Stack installation uses the

FPGA_OPEN_SHARED flag with calls to `fpgaOpen()` in the source code for the `mmlink` tool and the `hello_fpga` sample application, which enables remote debug as delivered in the installation for the `nlb_mode_0_stp` example AFU stimulated by the `hello_fpga` sample application without modification.

Here is an example call to `fpgaOpen()` for shared access to the AFU:

```
fpgaOpen(afc_token, &afc_handle, FPGA_OPEN_SHARED);
```

Refer to the following sources in the Acceleration Stack installation for examples of using the FPGA_OPEN_SHARED flag:

```
$OPAE_LOC/tools/mmlink/main.cpp,$OPAE_LOC/samples/hello_fpga.c.
```

Any other sample applications included in the Acceleration Stack installation or host code of your own design must use the shared flag when used to stimulate the AFU during in-system remote debug where `mmlink` is required to run simultaneously.

6.1.5.4. Disconnect from the AFU Target

When you are finished debugging, follow these steps to gracefully end the debug connection:

First, on the remote debug host…

1. Save trace captures and exit the Signal Tap GUI.
2. From the **System Console** File menu, click exit to disconnect from the target AFU.

On the debug target host…

You can either keep the `mmlink` instance active and host debug sessions from other remote debug hosts, or you can terminate `mmlink` with the `<Ctl-C>` key sequence from its terminal window. If you choose to keep `mmlink` active, you can only debug the currently loaded AFU. If you want to debug another AFU, you must first terminate the active `mmlink` process. Before loading another AFU, make sure to terminate any OPAE host application code accessing the current AFU.

6.1.6. Remote Debug Guidelines

- Signal Tap debug feature becomes non-functional when `mmlink` or **System Console** applications are closed.
- When performing PR, the AFU is non-existent and cannot be debugged. Therefore, **System Console** and `mmlink` applications should be terminated before attempting a partial reconfiguration of the AFU. Failing to do so might cause both PR and Signal Tap utilities to fail, taking the system into an unknown state. The system might have to be rebooted to restore the initial condition.
- The time to upload Signal Tap trace captures increases exponentially with sample depth. It is recommended to use sample depths less than "2K" for better Signal Tap user experience. Remote debug would still be functional even for larger depths but the time to upload the captured samples is significantly higher.
- **System Console** must be started after launching the `mmlink` application. If **System Console** returns an error, close the `mmlink` application, re-invoke `mmlink`, and launch **System Console** again.

6.1.7. Troubleshooting Remote Debug Connections

If you get a **Failed to connect** message after invoking **System Console**, consider adding port tunneling. Do this when the debug target host is behind a firewall with respect to your remote debug host is not.

On the debug target host, run **mmlink** as before. Note that **mmlink** provides an option to specify a port number. Port 3333 is the default.

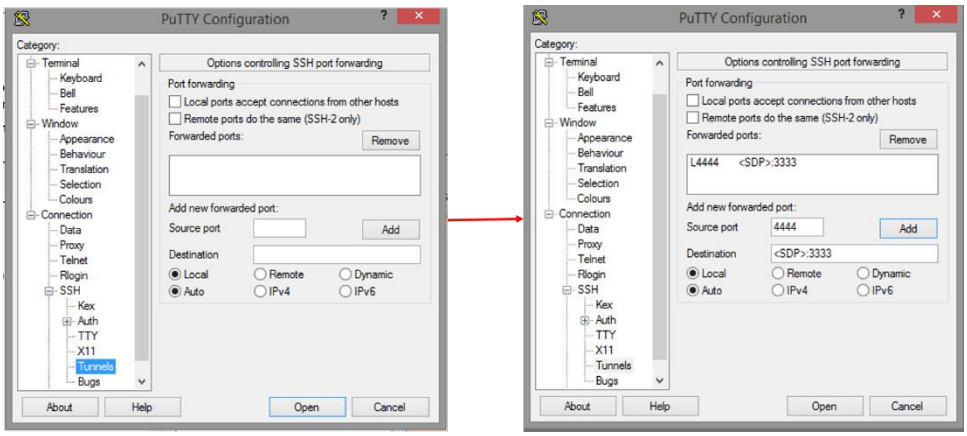
Refer to the following:

```
$ mmlink --port=3333
```

Setup port tunneling on the remote debug host. This example shows how to do so on a Windows remote debug host using PuTTY.

Use a PuTTY configuration screen as shown in the *SSH Tunneling with PuTTY* figure. For <*SDP*>, enter the name of the debug target host. This forwards the local port on your Windows host 4444 to port 3333 on the debug target host.

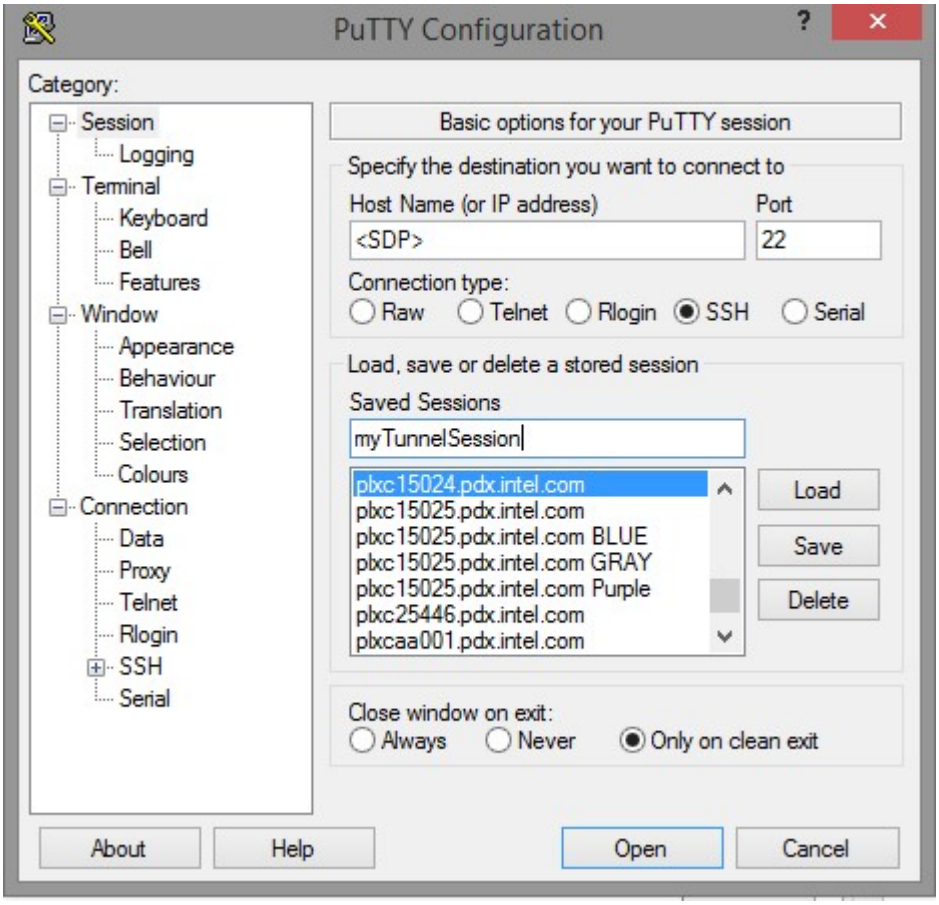
Figure 1. SSH Tunneling with PuTTY



Then, Click **Session**, specify the name of the debug target host, click **Save**, and then **Open**. Login to the debug target host. This is your tunneling session.

Figure 2. Save and Open the Tunneling Session

This figure specifies **localhost** and the port **4444**.



Once the tunneling session is setup this forwarding is complete. Open a Windows Command Window and issue the **system-console** command as shown in the "Save and Open the Tunneling Session" figure.

Run the "System Console with Port Forwarding" command:

```
$ system-console --rc_script=mmlink_setup_profiled.tcl remote_debug.sof
localhost 4444
```

As before, the Quartus System Console comes up. Wait for the **Remote system ready** message on the tcl console of the System Console.

7. Document Revision History for Accelerator Functional Unit (AFU) Developer's Guide

Document Version	Intel Acceleration Stack Version	Changes
2018.04.11	1.0 Production (supported with Intel Quartus Prime Pro Edition 17.0.0)	• Replaced mention of *AFU Loadable Image* with *Accelerator Function (AF)* across the document. • Removed `packager` utility from `$DCP_LOC/bin` location in the the Acceleration Stack release tarball in *Development Environment References* section. • Updated FPGA resources numbers in *Available Resources in the PR Region* section. • Updated information in: — *Acronym List for Accelerator Functional Unit Developer's Guide* — *AFU Project Structure* — *AFU Design Structure* — *Partial Reconfiguration Design Guidelines* • Added following new topics in *AFU Design Guidelines* section: — *General Guidelines* — *Utilizing Clock Resources* — *Interfacing with the FIU* — *Accessing Local DDR4 SDRAM* • Added links to the related documentation.
2017.12.22	1.0 Beta (supported with Intel Quartus Prime Pro Edition 17.0.0)	First release of comprehensive developer's guide to replace the Accelerator Functional Unit (AFU) Information Brief.
2017.10.02	1.0 Alpha (supported with Intel Quartus Prime Pro Edition 17.0)	Initial Release named "Accelerator Functional Unit (AFU) Information Brief".