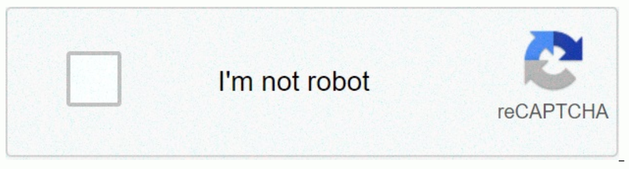

Continue

Accelerator functional unit(afu developer’ s guide

OPAE Manual Environment Acceleration Acceleration Unit Acceleration (AFU) (ASE) solves both beginner and experienced developers. To succeed, you should have knowledge and experience in the following areas: C/C++ Verilog/SystemVerilog RTL Simulator such as Synopsys VCS-MX* or Mentor Graphics ModelSim-SE* or QuestaSim* In addition, you can create a team of developers who specialize in RTL or software development. Previous FPGA (PAR) experience is not required to be successful, but parchc experience is also a useful skill. ASE provides a consistent transaction-level hardware interface and software API that allows you to develop a production quality acceleration function unit (AFU) and host software application. ASE supports both Intel Xeon® Processor with Integrated FPGA and Intel Programmable Acceleration Card (Intel FPGA PAC), and Intel FPGA Programmable Acceleration Card D5005 including a Stratix device® 10. The figures in this manual illustrate Intel PAC with Arria® 10 GX FPGA. The figures show a local memory that is only available on the PAC. To use an ASE environment you must have the source code in an rtl simulation language that can be explained. The following languages are possible: Verilog SystemVerilog VHDL Note ASE supports an AFU and one application at a time. ASE does not support multiple slot simulations. ASE is a dual process simulation. A process that runs the AFU RTL simulation. Another process connects to software that runs on the RTL AFU simulation. This unified simulation environment reduces the time it takes to develop AFU hardware and software. OPAE software distribution includes ASE. ASE offers two interfaces: Software: OPAE API deployed in programming language C. Hardware: Core Cache Interface (CCI-P) specification implemented in SystemVerilog. Use these interfaces to deploy your IP on the integrated FPGA Platform or Intel PAC with the Arria 10 GX FPGA. ASE provides a protocol checker to ensure the protocol is correct. ASE also provides methods for identifying potential problems early, before deployment in the system. ASE can help identify certain locking and configuration conditions and registry status (CSR) address mapping and cursor math errors. ASE presents a memory model for AFU memory tracking requirements as the work space accelerates. The memory model immediately flags illegal memory transactions to the location outside the required memory space. Therefore, you can repair incorrect memory access early, during the simulation phase. ASE does not guarantee that you can synthesize an AFU. After you verify the AFU RTL function in ASE, use ase and Intel Quartus® Prime Pro Edition software over and over again to create acceleration (AF) functionality. ASE provides a defined data risk checker Traffic patterns can cause write after writing (WAW), read after writing (RAW), and write after reading (war) hazards. Debugging transactions using a wave viewer or avoiding them using a factory that produces relevant memory properties (MPF) shim. ASE does not require administrator permissions. After installing all the necessary tools, you can run ASE on a linux machine with vanilla users. When using ASE in the application development cycle, consider the following limitations: ASE is a transaction-level simulation. It does not model or Intel UPI- or PCIe-specific package structure and protocol layers. ASE does not simulate caching and is not a cache simulation. It cannot reliably simulate cache collisions or capacity issues. Although ASE models some latency parameters, it cannot model real-time system-specific latency. It is also not an accurate time simulation of the design or latency and bandwidth of the real system. ASE models allow you to develop the correct acceleration function. ASE does not simulate multiple AFU or multiple configuration sockets. AFU development using ASE consists of the following four stages: Learning/Training: Learning how to use ASE and understanding interface and platform specifications. Review the sample code to understand CCI-P characteristics and OPAE API function calls. Run the template in an ASE simulation. Development stage: Use ASE to develop AFU RTL and software applications in a single working process. Develop RTL from specifications or by modifying existing sample RTL. ASE includes a behavior model of FPGA interface management (FIM) that provides immediate feedback on functionality during the development phase. ASE flags errors in CCI-P protocols, transactions and memory access. Therefore, you can fix these errors before moving on to the time-consuming bitstream creation phase. Bitstream Generation: Once AFU RTL and the software function correctly, open rtl AFU in Intel Quartus Prime Pro Edition software. Run location and route tools (ADMINISTRATIVE REFORM) for your platform. Use a general report to troubleshoot issues in RTL AFU. Then, go back to the development stage and re-validate in ASE. Bitstream generation can take hours depending on complex design, area, and so on. After creating a successful bitstream, perform a time analysis to check the corners of time, set and hold violations, close the clock, and so on. After repair failure is found during analysis, revalidate in ase environment. When the AFU is error-free, create the Acceleration Function (AF) bit stream that represents the AFU. System deployment: Af test in system hardware. Use Signal Tap to troubleshoot problems. Develop platform-specific software optimization. The RTL AFU code and opae software code you create in ASE are compatible with Intel Quartus software PAR if the following two software true: * The RTL AFU code is synthesizable. * RTL code AFU meets the time. In a simulation environment, complete the following steps to create bitstream AF and hardware programming: 1. Compile AFU RTL in VCS-MX synopsys or in the Mentor Graphics ModelSim-SE or QuestaSim simulator. 2. Compile software applications to implement a specific ASE of opae API. 3. Synthesize RTL AFU in Intel Quartus Prime Pro software to create a bitstream. 4. Program the hardware using this bitstream. Note ASE only works using AFU RTL source code. It cannot take the AF bit stream as input. Opae software releases include ASE. The current OPAE ASE release supports both Acceleration Stack for Intel Xeon Processors with built-in FPGA and Acceleration Stack for Intel PAC cards. ASE is only available on 64-bit Linux operating systems. Better ASE performance on a 64-bit RTL simulator; however, ASE also supports a 32-bit ModelSim. Consult your RTL simulation provider for Synopsys or Mentor Graphics for specific simulation requirements. The current ASE version supports the following RTL simulator: Synopsys VCS-MX (test version) * VCS-MX H2013.06-SP1 * VCS-MX J2014.12-SP3 * VCS-MX K2015.09-SP1* VCS-MX L2016.06 Mentor Graphics Modelsim-SE and Questasim (test version) * QuestaSim 10.5b * ModelSim-SE 10.5a * QuestaSim 10.4d ASE using Inter-Process Communication (IPC) structure. In most cases, these structures work without hiccups. The following Linux locations should exist and can be scored. In most Linux distributions, /dev/shm is pre-installed as the default option. Here are other ASE requirements: C-Compiler: gcc 4.8.5 or higher * Enhanced Development Library * UUID Development Library * JSON Development Library * Please see the dependency of the process of building the OPAE CMake system library: version 2.8.11 or higher GLIBC: version 2.19 or higher Python: version 2.7 or higher Intel Quartus Prime Pro Edition: ASE must find the folder $QUARTUS_HOME/eda/sim_lib/. You specify this folder in the project definition in Intel Quartus Prime Pro Edition software. ASE provides bash env_check.sh /sw/opae-x.x.x/ase/script. Run this script to verify your settings. Check RTL simulated product information for supported operating systems, installation notes, and other related information. The RTL simulator must be able to perform the following functions: * Compile the SystemVerilog Live Programming Interface (DPI) structure * Compile standard examples included in the installation * Support for SystemC Download ASE as a source or in RPM format. The source folder tree is: OPAE_BASEDIR |-- ase | |-- api | | '-- src | |-- rtl | | '-- dcp_emif_model | |-- scenario | '-- sw | |-- cmake |-- popular | |-- | | |-- opae | | |---safe_string | |-- doc |-- libopae |-- platform |-- safe_string |-- template |-- script |-- this folder tree tool displays the package structure of ASE distribution. The following folders make and run ASE simulations: ase: This is the folder that performs ase simulations. It contains the following child folders: * api/src: This folder contains the implementation of OPAE Intel ASE as a compilation library. You can link static or dynamic to this library. * rtl: This folder contains RTL components of ASE. You can compile this RTL for either platform. * dcp_emif_model: This is the local DDR memory model for the PAC card. Compile this model to simulate PAC mode. * Scenario: This folder contains some useful scenarios. Refer to the ase script section for more information. * sw: This folder contains the software components of ASE. All simulations require software components. The GNU Compiler Collection (GCC) compiles these components. Common: This folder contains OPAE library definitions. It identifies different macros to access an FPGA in an OPAE context. libopae: This library is a dedicated deployment to the Intel platform of the OPAE API. Platform: This directory contains scripts and RTL to manage the connection between a platform and the AFU top-level interface. Scenario: This folder contains experimental scripting. tools: This directory contains FPGA tools including fpgadiag and fpgaconfig tools. ASE distribution under ase/scripts includes several scenarios. Use these scripts to initially, set up, and clean an existing ASE simulation environment. Use ase/scripts/ase_setup_template.sh as a template script to set up the necessary tools. This scenario has plenty of free placeholders for site and information-specific environments. Consult your electronic design automation tool (EDA) administrator or RTL simulator manual for help setting up tools. This scenario checks the status of the distributed operating system, distro, and available system libraries. This check is incomplete. It only looks for the most important dependency, such as GCC version, GLIBC version, etc. $./ase/scripts/env_check.sh Before configuring ASE, follow the instructions to build OPAE SDK and make sure that the installed OPAE container or OPAE barrel directory is located on the PATH of your bark. To simulate an AFU, copy the ASE source tree and add a specific AFU configuration. Installing OPAE includes several scenarios to perform this task. The main script, afu_sim_setup, is in the OPAE recycle bin folder. Refer to the ASE Examples section below for definitions of sample workloads and implementation streams. The afu_sim_setup read a file that contains a list of RTL sources (<rtl_sources.txt>) and configures a simulation environment for the specified sources. </rtl_sources.txt>command to copy your base ASE environment <target dir=>to the . $afu_sim_setup --source = The only 2016 afu_sim_setup is the directory for the new <rtl_sources.txt> <target dir=>AFU environment. Here are the optional indicatorss: --help Help Indicatorss lists all the 2015 afu_sim_setup. --platform: Platformargument identifies any platform identified in the platform database, including the integrated FPGA Platform or Intel PAC. This 2015 is generally not necessary when a hardware platform release is installed. In that scenario, OPAE_PLATFORM_ROOT new environments point to hardware releases, defining the platform. afu_sim_setup is a wrapper for the following scenarios. You can also directly access both of these scripts: rtl_src_config: This script converts the list of RTL sources into simulated configuration files. generate_ase_environment.py: This script initiates your simulation platform configuration. The /platforms/scripts/rtl_src_config scenario maps a simple text file containing a list of RTL source files to an ASE configuration file for simulation or an Intel Quartus Prime Pro configuration file for a combination. rtl_src_config define pre-processing variables. The source configuration file can be decentralized, with one file including another. rtl_src_config build simulated trees based on ASE or Quartus tree building. Run rtl_src_config --help for a list of required options and command syntax. The /ase/scripts/generate_ase_environment.py the platform configuration files. afu_sim_setup call it automatically. A legacy mode in generate_ase_environment.py perform a brute-force check of the specified AFU RTL directory, attempting to identify a compilation. This brute-force mode isn't perfect and lists every file ending in .sv, .vs, .vhd, or .v, and folders separated by +. It may also fail when the compilation is in dependent order. Run generate_ase_environment.py --help for a list of indicatorss. The script requires a directory path to your AFU RTL. -t: The default for the optional -t (tool) is VCS. For the Mentor tool, use the QUESTA option. -p: The default for the optional -p (platform) intg_xeon stands for integrated FPGA Platform. Discrete use for Intel PAC. -x: Optional exclusion 2016 lists exclusions for path searches. Synopsys and Mentor Graphics RTL simulations create the following scenarios. * Summary: Create synopsys_sim.setup and vcs_run.tcl in the VCS configuration folder. * Mentor Graphics: vsim_run.tcl in the QUESTA configuration folder. Run time simultation using .tcl files. Details of the files created: * vlog_files.list: Lists all Verilog and SystemVerilog files found in the AFU folder path. * vhdl_files.list: Lists all VHDL files found in the AFU folder path. * ase_sources.mk: h&#;</target> </rtl_sources.txt> </target> </target> two files on DUT_VLOG_SRC_LIST and DUT_VHD_SRC_LIST Makefile. * ASE_PLATFORM: Set the platform type to the default type or type you specify. * Set additional VCS or QUESTA options using the SNPS_{VLOGAN, VHDLAN, VCS}_OPT or MENT_{VLOG, VCOM, VSIM}_OPT options in Makefile. Refer to the ASE Makefile variable for more information. The simulated files use absolute paths when possible. To improve mobility across users and teams, replace the environmental variables in the files created built and run the simulator. Note You must check this file for accuracy yourself before using it in simulations. Use script/ipc_clean.py ASE cleanup scripts to destroy zombie simulation processes and temporary files left behind by unsuccessful simulation processes or problems. - ./ase/scripts/ipc_clean.py ##Kịch # cleanup will try to clean up the IPC built up by the user 'user_foo ' Remove .ase_ready your ... Type 'y' to clean up all ase_simv zombie processes: y Go ahead with cleaning up the ASE process opened by $user_foo ASE AFU as a server-client simulation environment. AFU RTL is the server process. Software applications compiled and associated with OPAE ASE libraries are customer processes. Communication between the server and the client uses the pipe name. ASE summarizes most of the simulation infrastructure. You don't have to modify it. Server process: * Server process interface for 3rd party RTL Simulator packages. The server process now supports Mentor Modelsim-SE*Questasim* and Synopsys CS-MX* through the SystemVerilog-DPI library and simulation software interface. * Naming pipelines that carry out communication with customers. Pipe naming also implements control, status and session management. The server process includes a pipeline event monitoring tool. * The server process also provides a physical memory model that simulates RTL AFU access to physical addresses. The physical memory model simulates addresses that translate from virtual addresses to physical addresses. * SystemVerilog manages the CCI-P interface. All CCI-P events are recorded and time stamped. * The server also includes a CCI-P protocol inspector that checks CCI-P transactions originating from AFU. This tester identifies CCI-P protocol issues, hazards, race conditions, etc. * Calls that allocate map buffers to MEMORY-sharing POSIX (/dev/shm). The server-client process shares information about the buffer using the pipe name. The physical addresses generated in ASE are not realistic and do not replicate in the system. Client process: * The client performs the OPAE interface and library to access ASE platform functions including MMIO, buffer management, and session control. The available depending on the platform you specify at the time of construction. These functions are available using the OPAE API. * A compilation program compiled and linked to the ASE implementation of opae libraries. All OPAE calls route to ASE instead of OPAE platform drivers. Separate construction scenarios build server and client processes. Server: A makefile in the ase directory that compiles the ASE server process, containing ASE software, SystemVerilog tools, and AFU RTL logic code. Client: The main cmake scenario in the root of the distribution build deploys the OPAE library to the system and ase. Cmake script installs library in lib folder. Follow these instructions if you are building OPAE SDK from a tar source file. If not go directly to the step of building the application. Note In these instructions, the /tmp/opae/ folder is the base folder where you install the OPAE API software distribution. The premedo /tmp can point to any system or user directory. SOURCE ASE directory pointing to/tmp/opae/ase/ To build opae libraries: # Change folder to opae folder $ cd / tmp / opae / # Check content $ ls ase cmake popular libopae doc platform safe_string sample script tool CMakeLists.txt COPYING README RELEASE_NOTES # Create an out-of-build folder, and change it $ mkdir mybuild $ cd mybuild # Configure building software using cmake (see option) # $ cmake <Path to= cmakelists.txt =>&# Installation directory, in this example it is / <Path to= opae=>myinst / $ cmake .. / -DBUILD_ASE =YES -DCMAKE_INSTALL_PREFIX=/tmp/opae/myinst/ -- C compiler identification is GNU 4.8.4 -- CXX compiler identification is GNU 4.8.4 -- Check compiler C working: /usr/bin/cc -- Check compiler C working: /usr/bin/cc -- active. Done - Build files that have been written to: /tmp/opae/mybuild #Build and install libraries$ make$ make installation #Environment Variable PATH should point to/myinst/bin #Check mail institute installation paths $cd/tmp/opae/myinst/$ls doc including lib$ls lib libopae-c-ase.so libopae-c.so libopae-c.so.0 libopae-c.so.0.0... Build software applications for ASE by using libopae-c-ase.so gcc commands. You can build the software application using one of the following three methods: Use the implementation of the OPAE ASE library and compile it into the OPAE software application directly in one command using the GCC. # Compile commands $ gcc -g -o hello_fpga_ase hello_fpga.c /tmp/opae/myinst/lib/libopae-c-ase.so -I /tmp/opae/myinst/include/ -std=c99 -luuid # When running the application, Start the simulation, then set up LD_LIBRARY_PATH and ASE_WORKDIR$./hello_fpga_ase ASE using a platform distinguishing key in makefile simulation to enable different platform features and create a simulated configuration based on distinguished keys. These keys automatically set by</Path> </Path> </Path> and generate_ase_environment.py. You must note the following required construction configurations: Description of the ASE_PLATFORM This is the platform difference. It must be placed in ase_sources. mk to point to the necessary simulation features. FPGA_PLATF ORM_INTG_XEON and FPGA_PLATFOR M_DISCRETE unique legal value. FPGA_PLA TFORM_INTG _XEON SIMULATOR This must be put in ase_sources. mk to point to the correct RTL simulation. VCS and QUESTA are the only legal values. VCS DUT_VLOG_SRC_LI ST, DUT_VHDL_SRC_LI ST, and DUT_INCDIR These options point to AFU Verilog, VHDL and include the path settings required for the RTL simulator to build the simulation model correctly. Not provided — You must specify this. For more information about other switches, refer to the ASE Makefile Goal Use script/generate_ase_environment.py script to create AFU files specific to the simulation build. This scenario is not inesquestally requiring a manually checking of the configurations created to achieve accurate results. Note Adjust ase_sources.mk to add AFU RTL-specific defined macros or include paths. You can use script-generated ASE environments to create AFU files ase_sources.mk and vlog_files.list requests. # change to FOLDER ASE $ cd / tmp / opae / ase / # Use the script / generate_ase_environment.py $. / scripts / generate_ase_environment.py -t VCS -p discrete . . . # Edit files when <Path to= sample= rtl=>necessary. $ls ase_sources.mk vcs_run.tcl synopsys_sim.setup vlog_files.list... You may need to check vlog_files.list and ase_sources.mk for accuracy. Now, build the simulation with a 'make' command. $cd <opae>/ ase/$ make. . $ls work/ase_simv work/ase_simv Server-Client simulation ASE makes the server before the client. Here are the run time options: $ make sim \ [SIMULATOR =<VCS| QUESTA>] \ [ASE_CONFIG =<Path to= ase.cfg=>] \ [ASE_SCRIPT =] Use<Path to= ase_regress.sh= if= ase_mode=4>two terminal windows to start simulation. Terminal 1: Run as a sim. ASE initials and AFU issues a reset and then wait for the transaction to arrive. The software application must wait until the message Is ready for the simulation to display. Specify the variable environment ASE_WORKDIR Terminal 1. # Call simulator $ make sim SIMULATOR = VCS CC = gcc # FPGA Library # # AFU Simulation Environment (ASE) #MCP_SKYLAKE #SIM-SV: Transaction Logger Start SIM-SV: Simulator begins... SIM-C: +CONFIG /tmp/opae/ase/ase.cfg found! SIM-C: +SCRIPT /tmp/opae/ase/ase_regress.sh found! SIM-C: PID's simulation is 41819 SIM-C : Read /tmp/opae/ase/ase.cfg SIM-C configuration file: ASE was started in mode 1 (Server-Client without SIMKILL) ASE mode ... Server-Server</Path> </Path> </opae> </Path> </Path> without SIMKILL Do not work kill-switch ... DISABLE Simulated Seed Reuse... ENABLED ASE seeds ... 1234 ASE Trading view... Activate the user clock frequency ... 312.500000 MHz, T_uclk = 3200 ps Amount of physical memory ... 128 GB . . . SIM-C : ** NOTE: BEFORE running the software application ** Place env(ASE_WORKDIR) in the terminal where the application will run (copy and paste) = > $SHELL | Run: --------|---------- bash/zsh | =ASE_WORKDIR/tmp/opae/ase/work tcsh/csh | setenv ASE_WORKDIR /tmp/opae/ase/work For any $SHELL, consult your Linux SIM-C administrator: Get ready for simulation... SIM-C: Press CTRL-C to close the simulator... You can close Terminal 1 making simby release an SIGTERM to the ase_simv or by typing CTRL-C. Terminal: First set up the ASE_WORKDIR as specified in terminal 1. In this example, ASE_WORKDIR set to /tmp/opae/ase/work. Then, start the software application. # Set up environmental variables ASE_WORKDIR$ export ASE_WORKDIR = / tmp / opae / ase / work / # Run application $ export LD_LIBRARY_PATH = / tmp / opae / myinst / lib / $ LD_PRELOAD = libopae-c-ase.so ./ hello_fpga Note After the application exits, the simulation is complete. Close the simulator to allow the wave format process to complete. In Terminal 1, type the CTRL-C command. <Simulator running=> . . . SIM-C: Ready to run the next test 818602500C0Tx AlmFull toggled from 1 to 0 818602500C1Tx AlmFull toggled from 1 to 0 <CTRL-C key= hit=>^ CSIM-C : Close the message queue and unlinking ... SIM-C : Unsyming shared storage zones.... SIM-C : Sim-C removal file session: Remove the message queue and handle the buffer ... SIM-C : Cleaning session files ... SIM-C : Simulation of creating file transaction log files | $ASE_WORKDIR/ccip_transactions.tsv Workspaces info | $ASE_WORKDIR/workspace_info.log ASE Seeds | $ASE_WORKDIR/ase_seed.txt SIM-C : Test run = > 1 SIM-C : Send kill command ... SIM-SV: Simulated kill command received... Number of transactions | VA VL0 VH0 VH1 | MCL-1 MCL-2 MCL-4 = == MMIOWrReq 9 | MMIORdReq 2 | MMIORdRsp 2 | UMsgHint 0 | UMsgData 0 | RdReq 16384 | 0 0 16384 0 | 16384 0 0 RdResp 16384 | 0 0 16384 0 | WrReq 16385 | 0 0 16385 0 | 16385 0 0 WrResp 16385 | 0 0 16385 0 | 16385 0 0 WrFence 1 | 0 0 1 0 | WrFenRsp 1 | 0 0 1 0 | $finish from file /tmp/opae/ase/rtl/ccip_emulator.sv, line 2657. $finish at the time of simulation 1514962500 V C S i m u l a t o n R e p o r r t Time: 1514962500 ps CPU Time: 142,500 seconds; Data structure size: 1.1Mb Tuesday 20 13:29:13 2017 After completion, the simulation creates the following files: Wave format of the export: makes the wave wave open wavelength for the selected tool. * inter.vpd: VCS wave format file * "vsim.wlf': File format Mentor/Questa. $ASE_WORKDIR/ccip_transactions.tsv:</CTRL-C> </Simulator> </Simulator> The event log lists all the events observed on the CCI-P interface. The timesa can marker that shows the corresponding time period in the wavelength-ed VPD file. $ASE_WORKDIR/workspace_info.log: Information about the simulation buffer opens. $ASE_WORKDIR/ase_seed.txt: Information about simulated seeds. $ASE_WORKDIR/ccip_warnings_and_errors.txt: Information about CCI-P alerts and errors. The ase module ccip_logger.sv records CCI-P events for designs using the CCI-P protocol. This module allows you to compare transactions and traffic through CCI-P interfaces. Note Use a different file name for each instantiation of ccip_logger. ASE turns off the simulation gracefully, closes all mutexes, locks, POSIX structures safely before exiting through $finish statement in SystemVerilog. If your AFU design uses $error $finish to identify error conditions during simulation, consider using start_simkill_countdown() instead. start_simkill_countdown() call $finish after completing the steps for a graceful shutdown. You can simulate ASE scenarios. When the simulator is ready to use, the simulator records the key file, $ASE_WORKDIR/.ase_ready, to the work folder. # Start Simulator in the background make sim & # Wait for simulation readiness # When .ase_ready is available in $ASE_WORKDIR, ASE is ready to simulate while [! - f $ASE_WORKDIR / .ase_ready.pid] sleep 1 done # Start application cd $PATH_TO_APPLICATION #

Export ASE_WORKDIR export variables ASE_WORKDIR = {Path to simulation mentioned in GREEN PRINT ASE} ./Application makes the following changes to the default simulation settings: * Turn off wave initials: Check rtl provider suggestions for turning off wave format results. In some cases, it is possible to create a wave for a specific modular hierarchy. * Turn off display notifications by ENABLE_CL_VIEW = 0 in ase.cfg. To create a random CCI-P transaction for each run, modify the ASE, $ASE_SRCDIR/rtl/platform.vh latency model. Restriction *_LATRANGE defines a minimum/maximum pair: 'defines X_LATRANGE minutes, maximum Minimum and maximum values the minimum and maximum number of cycles for AFU return responses. Simulation assigned to random latency transactions in range (minimum, maximum). The greater the difference between (minimum, maximum), the greater the standard deviation of latency for a given type of transaction. You can assign different values (minimum, maximum) to different types of transactions. Note Latency values are specified for simulation only for testing. They may not accurately represent the true lag of your system hardware. /* * Latency model * Encrypted as minimal, maximum drag * ---------- * RDLINE_LATRANGE: ReadLine turnaround time * WRLINE_LATRANGE : WriteLine turnaround time * UMSG_LATRANGE UMsg Latency * INTR_LATRANGE: Break turnaround time * LAT_UNDEFINED: 15 MMIO_LATENCY 'RDLINE_S_LATRANGE 20,118' identified RDLINE_I_LATRANGE 20,118' WRLINE_M_LATRANGE 20118 'identified WRLINE_I_LATRANGE 20,118' identified UMSG_START2HINT_LATRANGE 39.41' UMSG_HINT2DATA_LATRANGE 41.45' 82.85' UMSG_START2DATA_LATRANGE definition INTR_LATRANGE 10.15' defines LAT_UNDEFINED 300' definition RDWR_VL_LATRANGE 20,118' RDWR_VH_LATRANGE The 140,180' definition ASE_MAX_LATENCY 300 ASE includes a CCI-P protocol inspector, $ASE_SRCDIR/rtl/ccip_checker.sv. Use this module to verify CCI-P compliance. The checker sniffs out transactions, conditions and title settings and flags warnings or errors. Identify multiple layers of problems in hw/ccip_sniffer.sv. ASE records all alerts and $ASE_WORKDIR/ccip_warnings_and_errors.txt. Memory error in transaction: Memory error of the highest severity. The simulation stops immediately. ASE records memory errors, including times stamps, $ASE_WORKDIR/ase_memory_error.log. Protocol warning: The warning does not stop the simulation. ASE records the warning in $ASE_WORKDIR/ccip_warning_and_errors.txt. ASE can also log memory hazards (RAW, WAR and WAW hazards). A guide to CCI-P systems is available in a separate Basic Building Block repository in the sample/tree guide. For the first example, 01_hello_world is afu_sim_setup flow described above. Get started with the guide's README file for configuration and implementation instructions. For example define a set of sources and go through the process of creating an ASE tree, running the simulation and connecting it to a host program. The ASE Makefile template includes multiple targets and switches. Build your own compilation script for more complex simulations. For a complete list of supported versions, see System requirements. Complete the following steps to compile ASE Simulator: Compile ase's software objects into a library file * First compile the software components located in the ase/sw/folder into a software library. Note SIM_SIDE as a compiled macro when compiling ASE simulation objects. The SIM_SIDE is for simulation software, not the OPAE ASE software library. Compile ASE SystemVerilog files that are in the ase/rtl/folder. * Compile ASE RTL platform components for integrated FPGA platform or Intel PAC into simulation database*. For PACs, compile the embedded memory interface (EMIF) memory controller* model into the ASE environment. If the simulation requires intel library FPGA Gate, compile * the model into the ASE environment. Compile AFU components into the ASE environment. * Use RTL simulation software tools to compile AFU components. Use synopsys or mentor utilities to compile VHDL or SystemVerilog components. Describe all goals default construction standards, build simulators in VCS-MX title printing version and opening environmental test sw_build build ase/sw/components into a software library. SIM_SIDE switch builds simulation software components to help Print help information vcs_build Summary VCS-MX model build flow questa_build Mentor Graphics Modelsim-SE / Qu estaSim template build flow sim Start the ASE Simulator wave Open selected RTL waveform viewer clean Clean build, simulation, and log file distclean clean distribution - remove the configuration file AFU Makefile target describes the default value VCS_HOME Synopsys VCS-MX install path Specific tool installation MTI_HOME Mentor installation path Install specific tool QUARTUS_HOME Intel Quartus Prime Pro install path specific tool installation ASE_PLATFORM Platform conversion selection : Specify either tight-couple d FPGA or FPGA programming to speed up cards using FPGA_PLATFOR M_INTG_XEON and FPGA_PLATFOR M_DISCRETE, respectively FPGA_PLATFORM_ INTG_XEON SIMULATOR Simulator Key to the VCS or QUESTA VCS ASE_SRCDIR ASE source source current source path location ASE_WORKDIR The location where THE ASE runs, typically $ASE_SRCDIR/Working Environment field-spec ific performs work path simulating library compilation work location "ASE_DISABLE_LOGGER" 'Switch to disable logger build 0"ASE_DISABLE_CHECKER " Switch to disable test build warning : Disable this checker can have side effects on the correct protocol 0 GLS_SIM Allows port simulation build 1 library GLS_VERILOG_OPT allows intel port simulation Quartus EDA simulates path library ASE_CONFIG ASE Run-time file configuration (description 'here <#ase-ru ntime-configuration ation-options> '__) $ASE_SRCDIR/as e .cfg ASE_SCRIPT ASE Receding Script Path $ASE_SRCDIR/as e_regress.sh TIMESCALE Simulator timescale 1ps/1ps ASEHW_FILE_LIST ASE RTL File list Refer to ase/Makefile ASE_MEM_FILE_LIST ASE RTL file list for programmable FPGA acceleration card mode ASE memory Reference ase/Makefile ASE_INCDIR ASE Including ase/Makefile reference directory path ASE_TOP ASE top-level (unchanged) ase_top CC_OPT ASE Software Library compiler builder build option "-g -m64 -fPIC -D SIM_SIDE = 1 -I w/ -D SIMULATOR= VCS -D $(ASE_PLA TFORM) -Wall -I $(VCS_HOME)/incl ude/" ASE_LD_SWITCHES ASE Software Linker switches "-lrt -lpthread -l SNPS_VHDLAN_OPT Synopsys VCS-MX VHDL compile options (add extra options to ase_sources. mk) Reference ase/Makefile SNPS_VLOGAN_OPT Synopsys VCS-MX SystemVerilog compile options (add additional options to ase_sources. mk) Refer to the ase/Makefile SNPS_VCS_OPT Synopsys VCS-MX option to build simulation implementation (add additional options to ase_sources. mk) Refer to ase/Makefile SNPS_SIM_OPT Synopsys VCS-MX Simulation Options (more additional options are possible to mk) Refer to ase/Makefile MENT_VCOM_OPT Mentor Modelsim-SE/Qu estasim VHDL optional compilation (more more to ase_sources. mk) Refer to ase /Makefile MENT_VLOG_OPT Mentor Modelsim-SE/Qu estasim {System}Verilo g compile options (add extra options to ase_sources. mk) Refer to ase/Makefile MENT_VSIM_OPT Mentor Modelsim-SE/Qu estasim VHDL simulation options (add extra options to ase_sources. mk) Reference ase/Makefile DUT_VLOG_SRC_LIST Point to a text file listed AFU {System}Verilo g file (usually vlog_files.l ist) User created DUT_VHDL_SRC_LIST Point to a Text files list VHDL AFU files (usually vhdl_files.l ist Users create DUT_INCDIR AFU lists consist of folders spaced by a '+' (usually in ase_sources. mk) The user creates an ASE configuration file that configures simulated behavior. An example configuration scenario is available at ase/ase.cfg Default name conversion describes ASE_MODE 1 ASE mode with the following valid values: 1: Standard Server-Clien t Mode2: Simulation stopped after clock ASE_TIMEOU T3: Software turns off simulation when client application releases session 4: Call mode receding> = 5 : Ignore (back "ASE_MODE = 1") ASE_TIMEOUT 50000 (only if ASE_MODE = 2) The watchdog timer turns off the simulation after the ASE_TIMEOU T clock of the CCI-P interface does not work. ASE_NUM_TESTS 500 (only ASE_MODE = 4) The number of tests in rehabilitation mode. If set incorrectly, the simulator may exit before maturity or delay waiting for the experiment to begin. ENABLE_REUSE_D 1 When set to 1, reuse the simulated seed, so that the CCI-P transaction is play back to the previous address. When set to 0, get a new seed. ASE_SEED 1234 (only if ENABLE_REUSE_SEED = 1) ase seeds set, activated when ENABLE_REU SE_SEED is set to 1, otherwise the simulator uses a different seed. At the end of the simulation, ASE scores the current seed $ASE_WORKD IR/ase_seed. Txt. ENABLE_CL_VIEW 1 ASE print all CCI-P transactions. When running a long simulation, setting ENABLE_CL_ VIEW to 0 can reduce simulation time. PHYS_MEMORY_AVAILABLE_ GB 32 Limit power generator to address ase specified memory range. ASE provides the following three levels to record verbosity messages. By default, these messages print into stdout: * ASE_INFO: Print the required information message needed to determine the activity. * ASE_ERR: Print the error message during operation. * ASE_MSG: Print the general message that says the checkpoint in ASE. Prevent these messages by setting the environment variable ASE_LOG to 0. The following command prevents category ASE_MSG: $ASE_LOG = 0 LD_PRELOAD = libopae-c-ase ./hello_fpga You cannot prevent warnings and errors. The following list of ase errors and warnings is not comprehensive: the problem observes the next step or all transactions are not seen or the simulation ends earlier than expected. Ase simulation inactiv activity is too short for applications using simulated cases success in ASE. If using ASE_MODE = 2 (Daemon with timeout), in ase.cfg ase.cfg increase your ASE_TIMEOUT or disable ASE_TIMEOUT. Ase simulation build errors - compiling, or linking GCC versions may not be too old. In the ase folder try the following command $ sw_build ase build correctly with GCC 4.8.5 or higher. Use ase/scripts/env_check .sh to identify problems. VCS-MX summaries dump stacks while compiling or running possible corruption of compiled objects or problems with increased compilation. Clean the ASE environment using $ cleaning If this command is unsuccessful, clean the distribution with $./distclean.sh rebuild the simulation. ERROR: Too many files open Through the process of running the ASE simulation are not clean and may have left behind open IPC instances. Use $ASE_SRCDIR/scripts/i pc_clean.py to clean IPC versions. Check that system requirements have been met. If the problem continues, increase the resource limit for your Linux distribution. $ASE_WORKDIR variable environment was not set up application could not find a valid simulation session Follow the print steps when the ASE simulation started. These instructions are in green text. .ase_timestamp can open at <DIRECTORY> The simulator may not have started yet. Note that at the start, simulated prints: Ready for $ASE_W orkdir simulations may not be set up correctly. Check the ASE_WORKDIR variables. "$ echo $ASE_WORKDIR" Wait for the simulation to print: "Ready for simulation' ase_sources.mk: No fi le or ASE directory Environment has not been created. Create an AFU RTL list (vlog_files.list and ase_sources.mk). You can use ase/scripts/generate_ ase_environment.pyt o create these files. An ASE case is probably still running in the current folder. The ASE simulation is running in the $ASE_WORKDIR. If the simulation process is uns usable or unconnectable, use ase/scripts/ipc_clean .py scripts to clean up temporary simulated files using: $ cleaning. Then rebuild the simulation. Change document version 2019.04.22 Update to list support for Intel FPGA PAC for Stratix 10 FPGA. 2018.04.16 Make the following changes: * Added etup script afu_sim_s read the file containing the list of RTL sources and built an ASE tree for compilation and simulation. * Add/platform s/script/r tl_src_conf ig script that maps a simple text file containing a list of RTL source files to an ASE configuration on. * Updated figures to show available hardware in Intel PAC with Arria® 10 GX FPGA.