FPGA AI NIC Readme

1. Project Description and Directories

This project is an example design of an FPGA AI NIC, described in this paper - https://arxiv.org/abs/2204.10943. FPGA AI NIC is a new smart network interface card (NIC) for distributed AI training systems using field-programmable gate arrays (FPGAs) that accelerates all-reduce operations and optimizes network bandwidth utilization via data compression. The AI smart NIC frees up the system's compute resources (e.g., CPUs, GPUs, accelerators) to perform the more compute-intensive tensor operations and increases the overall node-to-node communication efficiency. This project was developed as part of FPGA AI research conducted collaboratively by Intel PSG CTO Office and Intel Lab's Parallel Computing Lab.

The repository comprises of the following directories:

- hw: contains hardware implementation of the AI NIC, in Verilog RTL. It also includes "sim" sub directory that contains RTL testbench to simulate the AI NIC.
- sw: contains software driver for the AI NIC hardware. It also includes script to configure the network.

Possible uses of this code:

- Study AI NIC hardware RTL implementation by itself, by running a standalone RTL-simulation (Section 3).
- Deploy and run AI NIC on a CPU+FPGA system testbed with Libxsmm AI training software (Section 4), using known tested system components, including a specific FPGA card and "shell" (OPAE, IKL).
- Deploy and run AI NIC on user's own target FPGA cards, shell, and hardware testbed. In this case, the AI NIC hardware has been implemented using a well-defined interface (described in Section 2). Any FPGA can be targeted, as long as the AI NIC interface is used correctly. This may require creating FPGA shell for the card that accommodates the AI NIC interface. Or, as the code is provided here fully, user can also modify the code and interface to tailor to the target FPGA.

2. FPGA AI NIC Overview

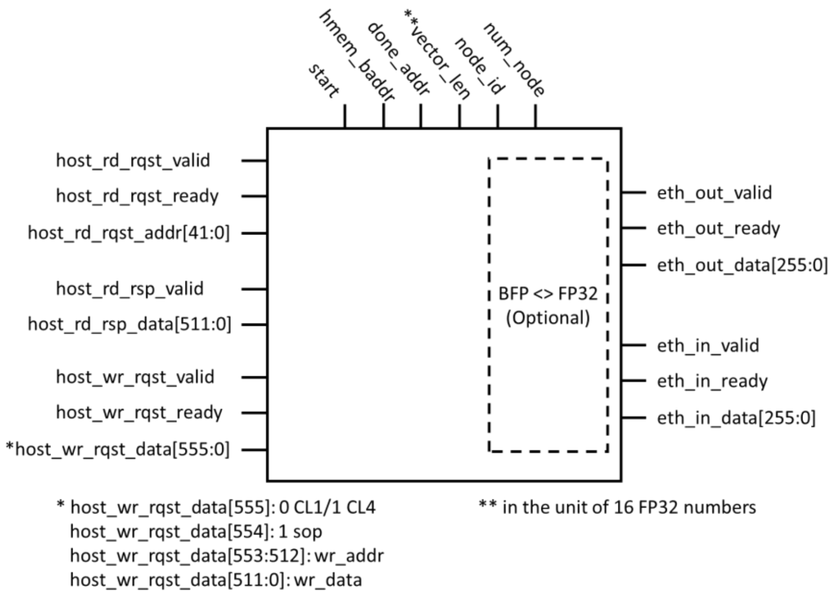


Figure 1. FPGA-AINIC interface

2.1. Interface Behaviors

Figure 1 shows the interface of FPGA-AINIC. This section explains the expected behavior of the interface so that users can build their own shell for FPGA-AINIC.

On the right side, FPGA-AINIC assumes a FIFO interface to the Ethernet. When **eth_xx_valid** is asserted, 256 bits of valid data is provided at **eth_xx_data** interface. Data is accepted by the sink when **eth_xx_ready** is asserted. Ready must be low if the sink is not ready to accept new data.

The interfaces to the CPU host are listed on the left side. First, FPGA-AINIC send memory read requests to the CPU host through **host_rd_rqst_xx** interfaces. When **host_rd_rqst_valid** is asserted, FPGA-AINIC provides a valid 42bit "cache line(64B)" address at **host_rd_rqst_addr** interface. Request is accepted by the shell if **host_rd_rqst_ready** is high. Note that FPGA-AINIC assumes **4 cache line burst mode** for memory reads, i.e., each memory request reads 4 cache lines. Secondly, the shell returns read responses through **host_rd_rsp_xx** interfaces. It must assert **host_rd_rsp_valid** signal as well as provide 512 bits of valid data at **host_rd_rsp_data** interface. Memory responses must be in the same order as memory requests. Also, for each read response, 4 cache lines must be returned in the address order. FPGA-AINIC always accept the read responses so no ready signal is required. Secondly, FPGA-AINIC sends memory write requests through **host_wr_rqst_xx** interfaces. It asserts **host_wr_rqst_valid** signal while providing write mode, start of packet(sop), address, data at **host_wr_rqst_data** interface. At CL1 mode, sop can be ignored, and FPGA-AINIC just sends 1 CL of data. At CL4 mode, FPGA-AINIC sends 4 cache lines of data. It begins with the lowest address. For the first CL, sop is high. For the subsequent CLs, sop is low, and addresses increment with the CLs. It takes 4 cycles to send a 4 CL write request. It is legal to have idle cycles in the middle of a 4 CL request, but one request will not be interleaved with another request.

On the top side listed control interfaces. **num_node** and **node_id** specify the total number of nodes in the system and the ID of the local node. **vector_len** specifies the length of the vector to be reduced, in CLs(64B). As FPGA-AINIC notifies the host when an all-reduce request is completed through writing a value to the host memory, **done_addr** specifies the base address of the memory location. To initiate an all-reduce request, **start** signal must be asserted and the base address of the vector to be reduced must be provided at **hmem_baddr** interface. Both **done_addr** and **hmem_baddr** are in CL(64B). FPGA-AINIC does not backpressure the all-reduce requests. It can buffer at most 8 requests and will process the requests in order. It automatically assigns a **done_id** to each request in round robin fashion starting from 0 to 7. Once the request is finished, it writes value **ONE** to the memory location with base address **done_addr** and offset **done_id** also in CL(64B).

2.2. System Topology Requirements

FPGA-AINIC assumes the nodes connected in a ring topology. Assuming there are N nodes, for a node with ID ***n***, ***0 <= n < N***, the upstream of node ***n*** must be node ***(n+1 mod N)***, the downstream of node ***n*** must be node ***(n-1 mod N)***.

3. FPGA AI NIC Standalone RTL Simulation

For users who do not have hardware testbed and would like to get started with this AI NIC project, we provide an RTL testbench to simulate the AI NIC Verilog RTL module using Modelsim (Questa) RTL simulator tool. This section describes the dependencies, testbench, and how to run the RTL simulation.

3.1. Dependencies

These dependencies need to be installed in the user's system in order to do FPGA AI NIC RTL simulation.

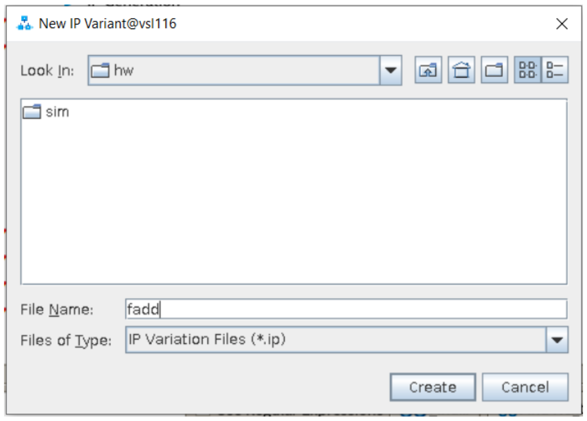
- Quartus with Arria 10 device package installed (tested with Quartus 21.4)
- Modelsim (Questa)

3.2. Testbench Description

The testbench models a simple CPU host memory, Ethernet links, and instantiates THREE instances of FPGA-AINIC modules. The CPU host memory modeled in the testbench has fixed latency, set in the parameter **HOST_MEM_DELAY**. The maximum bandwidth of the modeled host memory read plus write is 102.4 Gbps. The efficiency of bandwidth utilization can be set with the parameter **CCIP_EFFICIENCY**. The Ethernet link between every two nodes also has fixed latency, set in parameter **ETH_DELAY**. The maximum bandwidth of each modeled ethernet link is 80 Gbps. The Ethernet BW utilization can be set with **ETH_EFFICIENCY**. The size of the vector to be reduced is specified in parameter **TOTAL_LENGTH**, in CLs(64B). The testbench simulates one all-reduce transaction.

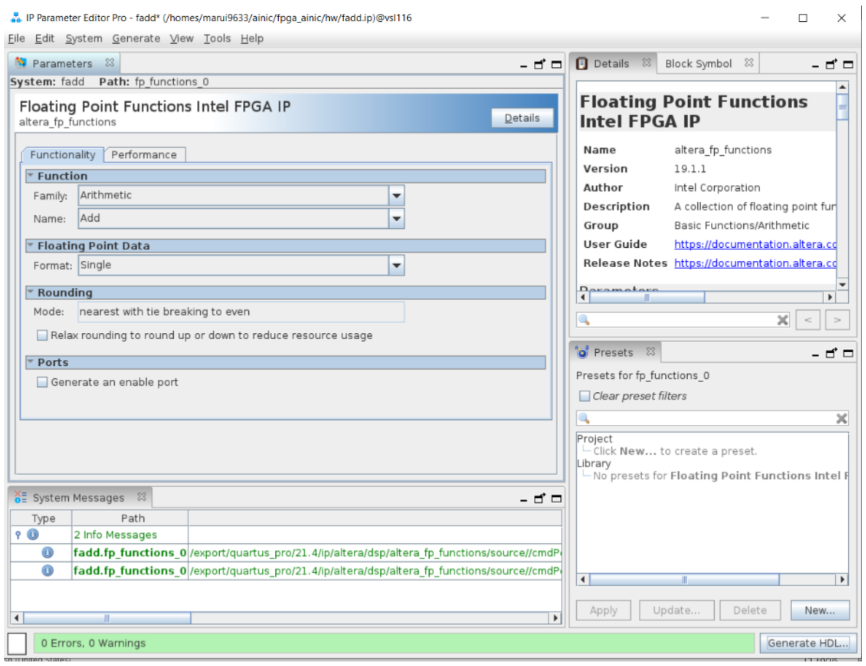
3.3. Instructions to Run Simulation

- Create a project directory
  - mkdir <project dir>
  - cd <project dir>
- Get a copy of fpga_ainic repo
- Go to simulation directory
  cd <project dir>/fpga_ainic/hw/sim
- Generate IP core and simulation script with Quartus
  - Open Quartus GUI
  - File -> Open Project
    - Select all_reduce_sim.qpf, click Open
  - Generate IP
    - Tool -> IP Catalog -> Installed IP -> Library -> Basic Functions -> Arithmetic -> Floating Point Functions Intel FPGA IP
    - Name the ip core "fadd" and save under <project dir>/fpga_ainic/hw/ directory



File Name: fadd
Files of Type: IP Variation Files (*.ip)

      Click Create
    - Set up IP core


IP Parameter Editor Pro - fadd* (/homes/marui9633/ainic/fpga_ainic/hw/fadd.ip)@vsl116

System: fadd   Path: fp_functions_0

Floating Point Functions Intel FPGA IP
altera_fp_functions

Functionality | Performance

Function
Family: Arithmetic
Name: Add

Floating Point Data
Format: Single

Rounding
Mode: nearest with tie breaking to even
☐ Relax rounding to round up or down to reduce resource usage

Ports
☐ Generate an enable port

Floating Point Functions Intel FPGA IP

Name	altera_fp_functions
Version	19.1.1
Author	Intel Corporation
Description	A collection of floating point fur
Group	Basic Functions/Arithmetic
User Guide	https://documentation.altera.co
Release Notes	https://documentation.altera.co

Presets for fp_functions_0
☐ Clear preset filters
Project
Click New... to create a preset.
Library
No presets for Floating Point Functions Intel F

System Messages

Type	Path	
	2 Info Messages	
	fadd.fp_functions_0	/export/quartus_pro/21.4/ip/altera/dsp/altera_fp_functions/source//cmdP
	fadd.fp_functions_0	/export/quartus_pro/21.4/ip/altera/dsp/altera_fp_functions/source//cmdP

0 Errors, 0 Warnings

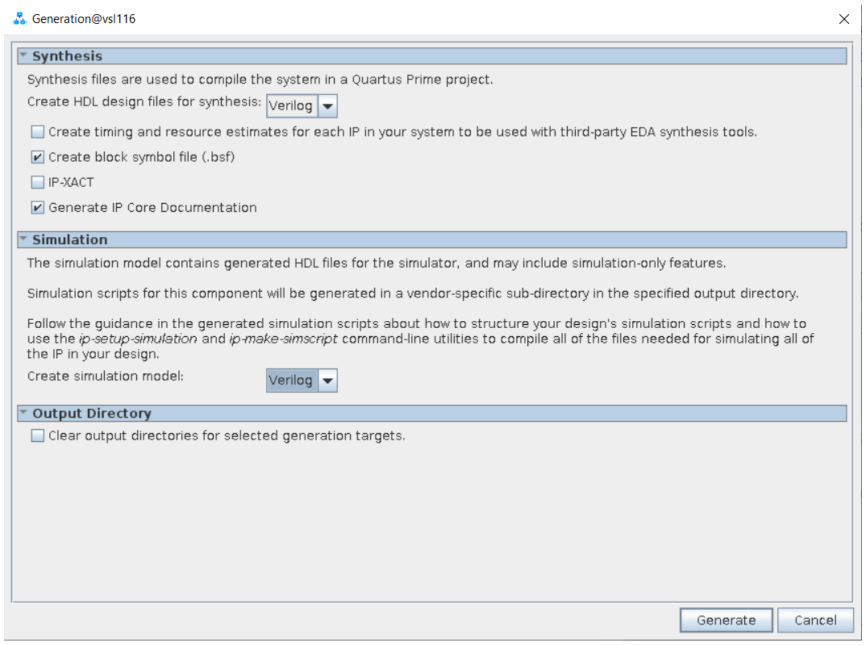
Keep default setting for the other parameters.

- Generate IP

  Close window -> Save -> Yes

  In the pull-down menu in **Simulation** section, select **Verilog**

  Check **Modelsim**


Generation@vsl116

Synthesis

Synthesis files are used to compile the system in a Quartus Prime project.

Create HDL design files for synthesis: Verilog

☐ Create timing and resource estimates for each IP in your system to be used with third-party EDA synthesis tools.

☑ Create block symbol file (.bsf)

☐ IP-XACT

☑ Generate IP Core Documentation

Simulation

The simulation model contains generated HDL files for the simulator, and may include simulation-only features.

Simulation scripts for this component will be generated in a vendor-specific sub-directory in the specified output directory.

Follow the guidance in the generated simulation scripts about how to structure your design's simulation scripts and how to use the *ip-setup-simulation* and *ip-make-simscript* command-line utilities to compile all of the files needed for simulating all of the IP in your design.

Create simulation model: Verilog

Output Directory

☐ Clear output directories for selected generation targets.

  Click Generate -> Finish

- In Compilation Dashboard, click the start bottom before “IP Generation”

- Generate simulation script
  - Tool -> Generate Simulator Setup Script for IP -> OK

- Run simulation
  - cd mentor

    - Modify all_reduce.do: line 11. Change <project dir> to the path of your project directory.
    - cd ..
    - vsim -c -do mentor/all_reduce.do
- Expected output
    - "PASSED" will be printed out in the terminal if results match the golden output. If not, "FAILED" will be printed out as well as the number of errors.
    - Note that "FAILED" will be printed out if BFP module is enabled, since BFP compression creates errors.
- Waveform will be dumped to vsim.wlf

4. Generating FPGA-AINIC Bitstream for Deploying and Testing on CPU+FPGA Testbed

4.1. Testbed Hardware Requirements

- A cluster of N nodes (tested with Xeon Platinum 8280 28-core CPU)
- Each node has an Intel Programmable Acceleration Card (PAC) with Intel Arria 10 GX FPGA attached
- PAC cards are connected to an Ethernet switch (tested with Dell EMC S6100-ON)

4.2. Dependencies

These dependencies need to be installed in the user's system in order to run FPGA AI NIC in CPU+FPGA system

- Quartus 17.1.1_pac1.2
- OPAE (https://github.com/OPAE/opae-sdk, tested with opae-install-20200101)
- IKL (https://www.intel.com/content/dam/www/programmable/us/en/others/literature/wp/wp-01305-inter-kernel-links-for-direct-inter-fpga-communication.pdf)
- Libxsmm (https://github.com/libxsmm/libxsmm)
- Intel MPI library (https://www.intel.com/content/www/us/en/develop/documentation/mpi-developer-guide-linux/top/installation-and-prerequisites/installation.html)

4.3. Generate Bitstream

- Note that there might be compatibility issue if **fadd** IP core is generated with newer Quartus. We recommend regenerating **fadd** IP core with Quartus 17.1.1_pac1.2, following the similar process described in Section 3.3
- cd <project dir>/fpga_ainic/hw/
- make

4.4. Compile Host Program

- Get libxsmm and compile the library
    - cd <project dir>/
    - git clone https://github.com/libxsmm/libxsmm.git
    - cd libxsmm
    - make
- Get intel-fpga-bbb

  - cd <project dir>/
  - git clone https://github.com/OPAE/intel-fpga-bbb.git
- Compile host program
  - cd <project dir>/fpga_ainic/sw/
  - cp ../../intel-fpga-bbb/samples/tutorial/05_virtual_addrs/base/sw/opae_svc_wrapper.* ./
  - afu_json_mgr json-info --afu-json=../hw/ikl_fpga.json --c-hdr=afu_json_info.h
  - make realclean && make AVX=3 -j 16 CC=mpiicc CXX=mpiicc

4.5. Instructions to Deploy and Run

- Program the FPGAs
  - On each node
    - cd <project dir>/hw/build/
    - fpgaconf ikl_fpga.gbs
- Go to sw directory
  - cd <project dir>/fpga_ainic/sw/
- Configure ikl_config
  - Refer to Section 8.2.3 in the IKL_v3.1e user guide
  - ssh_enabled should be set to 1
- Create hostlist file
  - Refer to this guide https://www.intel.com/content/www/us/en/developer/articles/technical/controlling-process-placement-with-the-intel-mpi-library.html
  - Note that FPGA_AINIC assumes 1 rank per node, and the rank ID will be node_id for each node.
- Run the test
  - source run.sh
  - Note that the command line in the example script run.sh assumes a 3-node cluster

5. Citations and References

If you use this FPGA AI NIC in your work/paper, please use the following paper as citation for this AI NIC work:

(1) Rui Ma, Evangelos Georganas, Alexander Heinecke, Andrew Boutros, Eriko Nurvitadhi, "FPGA-based AI Smart NICs for Scalable Distributed AI Training Systems," arXiv:2204.10943, 2022. Available at: https://arxiv.org/abs/2204.10943

The following provide further references on related technologies:

(2) [2] S. M. Balle et al., "Inter-Kernel Links for Direct Inter-FPGA Communication," Intel White Paper (WP-01305-1.0), 2020. Available at: https://www.intel.com/content/dam/www/programmable/us/en/others/literature/wp/wp-01305-inter-kernel-links-for-direct-inter-fpga-communication.pdf

(3) [3] Intel Corp., "Intel Open Programmable Acceleration Engine (OPAE)," Available at https://opae.github.io/