Virtual Platform On AWS FPGA

1. Overview

Virtual Platform (1. Overview (vp.html#overview)) can run on Amazon Web Services (AWS) FPGA platform. Fig. 90 below shows the top level diagram of the NVDLA virtual simulator on AWS FPGA platform.

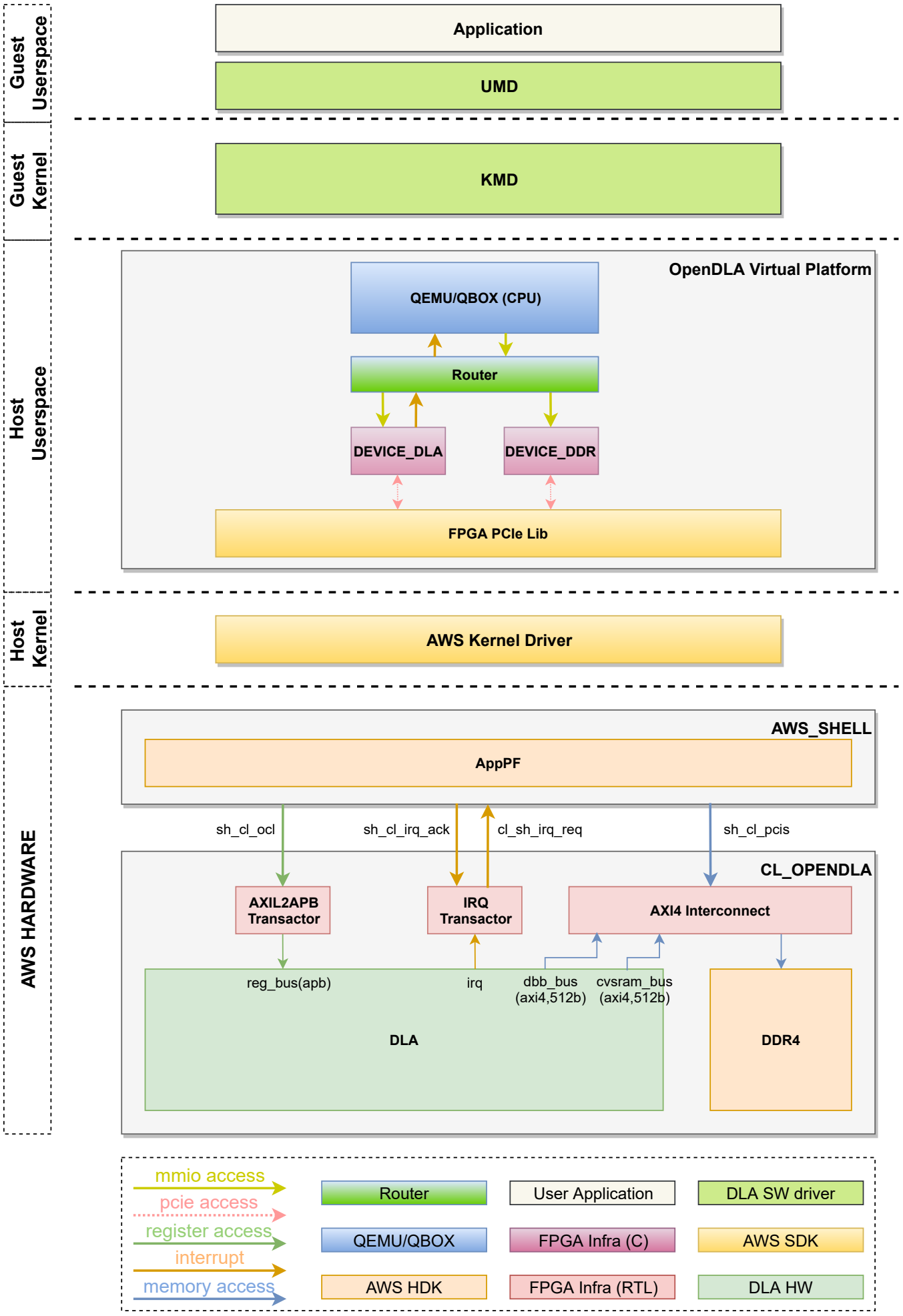

Fig. 90 - NVDLA Virtual Platform on AWS FPGA platform.

To launch an instance which is a virtual server in the cloud, Amazon Machine Image (AMI) is required. Virtual simulator can run on the instance with FPGA board connected. The following sections describe how virtual simulator runs on different AMIs.

- 2. Using the Virtual Simulator on NVIDIA AMI: NVIDIA provides NVDLA AMI (ubuntu14.04 based) included all necessary system requirements which allow users to run virtual simulator without any additional operation.
- 3. Using the Virtual Simulator on AWS AMI: Virtual simulator can run on basic ubuntu/centos AMI delivered by AWS. This section shows how to run virtual simulator on basic AMI step by step.
- 4. Generating the AFI on AWS FPGA AMI: Amazon FPGA Image (AFI) is necessary for AWS FPGA platform. Before running virtual simulator on AWS FPGA platform, AFI needs to be registered and loaded to AWS. NVIDIA provides one demo AFI which has been verified. Users also can develop their own FPGA design, when design completes, users can generate AFI with FPGA AWS AMI pre-built included FPGA development and run-time tools. This section shows how to generate AFI on AWS FPGA AMI step by step.

2. Using the Virtual Simulator on NVIDIA AMI

2.1 Setup AWS EC2 instance machine

2.1.1 Set up and log into your AWS account

Log into the AWS Management Console (https://console.aws.amazon.com/) and set up your root account.

Note
Please set your aws region to US East (N.Virginia) since each AWS region is independent.

2.1.2 Launch an Amazon EC2 instance

In the Amazon EC2 Dashboard (https://us-west-2.console.aws.amazon.com/ec2/v2/home?region=us-west-2), choose “Launch Instance” to create and configure your virtual machine.

2.1.3 Configure your instance

In this tutorial, you have the option to configure your instance features. Below are some guidelines on setting up your first instance.

- Choose an Amazon Machine Image (AMI): we recommend the *“nvdla_vp_fpga_ami_ubuntu”*
- Choose an instance type: we recommend the *“f1.2xlarge”*.
- Launch instance: review your instance configuration and choose “Launch”.
- Create a key pair: Select “Create a new key pair” and assign a name. The key pair file (.pem) will download automatically - save this in a safe place as we will later use this file to log in to the instance. Finally, choose “Launch Instances” to complete the set up.

Note
It may take a few minutes to initialize your instance.

2.1.4 Connect to your instance

After you launch your instance, you can connect to it and use it the way that you’d use a computer sitting in front of you. To connect from the console, follow the steps below:

- Select the EC2 instance you created and choose “Connect”.
- Select “A Java SSH client directly from my browser”. Ensure Java is installed and enabled.
- Enter the Private key path (example: C:KeyPairsmy-key-pair.pem). Choose “Launch SSH Client”.

Note

You can also connect via SSH or PuTTY, click here (http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstances.html) to learn more.

2.2 Setup AWS SDK

```
$ clone https://github.com/aws/aws-fpga.git                # Pull AWS SDK
$ git checkout v1.4.0                                      # Use tag v1.4.0 version of AWS SDK
$ cd [aws-fpga prefix] && source sdk_setup.sh              # Setup AWS SDK
$ lsmod | grep edma                                        # Check if the edma driver is intall
$ cd $SDK_DIR/linux_kernel_drivers/edma                    # If nothing shows up, following the
$ make
$ echo 'edma' | sudo tee --append /lib/modules-load.d/edma.conf
$ sudo cp edma-drv.ko /lib/modules/`uname -r`/
$ sudo depmod
$ sudo modprobe edma-drv
```

2.3 Load AWS FPGA image

```
$ sudo fpga-clear-local-image -S 0
$ sudo fpga-load-local-image -S 0 -I agfi-09c2a21805a8b9257     # Load MSI-X interrupt fixed AF
$ sudo fpga-clear-local-image -S 0
$ sudo fpga-describe-local-image -S 0 -H
$ sudo fpga-load-local-image -S 0 -I <your-image-afi-global-id> # Load the image
$ sudo fpga-describe-local-image -S 0 -R -H
$ sudo rmmod edma-drv                                           # Only needed if edma driver ha
$ sudo insmod $SDK_DIR/linux_kernel_drivers/edma/edma-drv.ko    # Re-install the edma driver to
```

AFI *agfi-09c2a21805a8b9257* is necessary for MSI-X interrupts issue.

You can generate your own AWS FPGA Image (AFI) by 4. Generating the AFI on AWS FPGA AMI Or can use NVIDIA Sample AFI (refer to VP AWS FPGA README (https://github.com/nvdla/vp_awsfpga/blob/master/README.md)) if just want to run tests on AWS FPGA platform.

More details please refer to AWS Getting Started (https://aws.amazon.com/ec2/getting-started/).

2.4 Running Virtual Simulator

```
$ cd /usr/local/nvdla
$ sudo ./aarch64_toplevel -c aarch64_nvdla.lua --fpga
Login the kernel with account 'root' and password 'nvdla'
```

You should be able to find the string *"Initialize AWS FPGA with slot_id=0, pci_vendor_id=0x1d0f, pci_device_id=0xf001"* from output if virtual simulator is running on FPGA platform.

2.5 Running software sanity test

After login the kenerl system, you can run one software sanity test for NVDLA small configure.

```
# mount -t 9p -o trans=virtio r /mnt
# cd /mnt/sw/prebuilt/linux/
# insmod drm.ko
# insmod opendla_small.ko
# ./nvdla_runtime --loadable kmd/CDP/CDP_L0_0_small_fbuf
```

You should be able to see 'Test pass' printed in the screen at the end of test. You are now ready to try out the NVDLA software in the virtual simulator on FPGA! Please refer to Software Manual (sw/contents.html) for details.

If you want to exit the virtual simulator, press 'ctrl+a x'.

3. Using the Virtual Simulator on AWS AMI

3.1 Setup AWS EC2 instance machine on AWS AMI

Please refer to 2.1 Setup AWS EC2 instance machine and the AMI we recommend to choose *"ami-38708b45"* (Ubuntu) or *"FPGA Developer AMI"* (CentOS)

3.2 Download the Virtual Simulator

3.2.1 Download Virtual Simulator

Please refer to 2.2 Download the Virtual Simulator (vp.html#download-the-virtual-simulator)

3.2.2 Download NVDLA AWS FPGA Custom Logic (CL)

```
$ git clone https://github.com/nvdla/vp_awsfpga.git
```

3.3 Install Dependencies

3.3.1 Install required tools and libraries

For CentOS:

```
$ sudo yum install cmake swig glib2-devel git pixman-devel boost-devel libattr-devel libpcap-de
```

For Ubuntu:

```
$ sudo apt-get update
$ sudo apt-get install g++ cmake libboost-dev git lua5.2 python-dev libglib2.0-dev libpixman-1-
```

3.3.2 Download and install SystemC 2.3.0

Please be noted that SystemC 2.3.1/2.3.2 is currently not supported currently

```
$ wget -O systemc-2.3.0a.tar.gz http://www.accellera.org/images/downloads/standards/systemc/sys
$ tar -xzvf systemc-2.3.0a.tar.gz
$ cd systemc-2.3.0a
$ sudo mkdir -p /usr/local/systemc-2.3.0/
$ mkdir objdir
$ cd objdir
$ ../configure --prefix=/usr/local/systemc-2.3.0
$ make
$ sudo make install
```

3.3.3 Download and install Lua 5.3.2 (For CentOS)

```
$ curl -R -O http://www.lua.org/ftp/lua-5.3.2.tar.gz
$ tar zxf lua-5.3.2.tar.gz
$ cd lua-5.3.2
$ make linux CFLAGS="-fPIC -DLUA_USE_LINUX" test
$ sudo make install
```

3.3.4 Download and install perl package required

We need to install perl package YAML.pm and Tee.pm to build NVDLA CMOD.

```
$ wget -O YAML-1.24.tar.gz http://search.cpan.org/CPAN/authors/id/T/TI/TINITA/YAML-1.24.tar.gz
$ tar -xzvf YAML-1.24.tar.gz
$ cd YAML-1.24
$ perl Makefile.PL
$ make
$ sudo make install
$ wget -O IO-Tee-0.65.tar.gz http://search.cpan.org/CPAN/authors/id/N/NE/NEILB/IO-Tee-0.65.tar.
$ tar -xzvf IO-Tee-0.65.tar.gz
$ cd IO-Tee-0.65
$ perl Makefile.PL
$ make
$ sudo make install
```

3.3.5 Download and build NVDLA CMOD and VMOD

Please refer to Tree Build (hw/v1/integration_guide.html#tree-build) for details on building the NVDLA hardware tree, and make sure the required tools listed in Environment Setup (hw/v1/integration_guide.html#env-setup) are installed first.

```
$ git clone https://github.com/nvdla/hw.git
$ cd hw
$ git reset --hard <HW verion index>    # HW versison must be matched with virtual simulator, r
$ make
$ tools/bin/tmake -build cmod_top -build vmod
```

The header files and library will be generated in *hw/outdir/<project>/cmod/release* and *hw/outdir/<project>/vmod/release*.

<HW verion index> must be matched with virtual simulator, refer to VP README (https://github.com/nvdla/vp/blob/master/README.md) for details.

If you need to run the random HW regression tests on FPGA, please run the below commands to build tests for random HW regression.

```
$ ./tools/bin/tmake -build verif_trace_generator
$ ./verif/tools/run_plan.py -P nv_small -tp nv_small -otag L10 L11 -l_num=4 -r_num=10 -timeout
```

After build finish, there will be nv_small_XXXX folder in hw tree and trace tests are generated in nv_small_XXXX/nvdla_utb. You can also generate random HW regression tests for other configurations like nv_large by command "./verif/tools/run_plan.py -P nv_large -tp nv_large -otag L10 L11 -l_num=4 -r_num=10 -timeout 500 -monitor –dump_trace_only".

3.4 Build and Install the Virtual Simulator with NVDLA FPGA

3.4.1 Download AWS EC2 FPGA Hardware and Software Development Kit

```
$ git clone https://github.com/aws/aws-fpga.git
$ git checkout v1.4.0
```

Note

Always sync latest version for AWS EC2 FPGA Hardware and Software Development Kits. Please sync aws repository to specified version (refer to VP AWS FPGA README (https://github.com/nvdla/vp_awsfpga/blob/master/README.md)) which was verified with any issue.

3.4.2 Setup AWS SDK and edma driver

```
$ cd [aws-fpga prefix] && source sdk_setup.sh              # Setup AWS SDK
$ lsmod | grep edma                                        # Check if the edma driver is intall
$ cd $SDK_DIR/linux_kernel_drivers/edma                    # If nothing shows up, following the
$ make
$ echo 'edma' | sudo tee --append /etc/modules-load.d/edma.conf
$ sudo cp edma-drv.ko /lib/modules/`uname -r`/
$ sudo depmod
$ sudo modprobe edma-drv
```

aws-fpga prefix is the local aws repository.

3.4.3 Load AWS FPGA image

Please refer to 2.3 Load AWS FPGA image.

3.4.4 Cmake build under the vp repository directory

For CentOS:

```
$ cmake -DCMAKE_INSTALL_PREFIX=[install dir] -DSYSTEMC_PREFIX=[systemc prefix] -DNVDLA_HW_PREFI
```

For Ubuntu:

```
$ cmake -DCMAKE_INSTALL_PREFIX=[install dir] -DSYSTEMC_PREFIX=[systemc prefix] -DNVDLA_HW_PREFI
```

install dir is where you would like to install the virtual simulator, *systemc prefix* is the SystemC installation directory, *nvdla_hw prefix* is the local NVDLA HW repository, *nvdla_hw project name* is the NVDLA HW project name and *aws sdk prefix* is the AWS sdk directory

Example:

For CentOS:

```
$ cmake -DCMAKE_INSTALL_PREFIX=build -DSYSTEMC_PREFIX=/usr/local/systemc-2.3.0/ -DNVDLA_HW_PREF
```

For Ubuntu:

```
$ cmake -DCMAKE_INSTALL_PREFIX=build -DSYSTEMC_PREFIX=/usr/local/systemc-2.3.0/ -DNVDLA_HW_PREF
```

3.4.5 Compile and install:

```
$ make
$ make install
```

3.5 Running HW regression tests on FPGA

3.5.1 Run NVDLA L0/1/2 tests

```
$ cd [vp_awsfpga prefix]/cl_nvdla/verif/regression
$ make AWS_FPGA=1 NVDLA_HW_ROOT=[nvdla_hw prefix]
$ make check    # Check last regression status
```

nvdla_hw prefix is the local NVDLA HW repository, *vp_awsfpga prefix* is the local nvdla aws fpga CL repository.

3.5.2 Run NVDLA random regression tests

You can run NVDLA random regression tests which has HW full coverage with below commands.

```
$ cd [vp_awsfpga prefix]/cl_nvdla/verif/regression
$ make AWS_FPGA=1 NVDLA_HW_ROOT=[nvdla_hw prefix] NVDLA_HW_TRACE_LIST=nv_small_random NVDLA_HW_
$ make check NVDLA_HW_TRACE_LIST=nv_small_random # Check last regression status
```

nvdla_hw prefix is the local NVDLA HW repository, *vp prefix* is the local nvdla aws fpga CL repository.

3.6 Running the Virtual Simulator

3.6.1 Prepare Kernel Image

A demo linux kernel image is provided in the github release. You can run this image in the virtual simulator, and run the NVDLA KMD/UMD inside it.

If you would like to build a linux kernel on your own, please refer to 3. Building Linux Kernel for NVDLA Virtual Simulator (vp.html#building-linux-kernel).

After the image is ready, modify the *conf/aarch64_nvdla.lua* for the image and rootfs file location.

3.6.2 Standard QEMU Arguments

The configuration of the virtual simulator is defined in *conf/aarch64_nvdla.lua*. You can change the standard QEMU arguments in *extra_arguments* inside the lua file.

3.6.3 Running Kernel Image In the Virtual Simulator

Start the virtual simulator:

```
$ sudo ./build/bin/aarch64_toplevel -c conf/aarch64_nvdla.lua --fpga
Login the kernel. The demo image uses account 'root' and password 'nvdla'.
```

Some demo tests are provided in the *tests* directory, you can run them after login as root:

```
# mount -t 9p -o trans=virtio r /mnt
# cd /mnt/tests/hello
# ./aarch64_hello
```

You should be able to see 'Hello World!' printed in the screen. You are now ready to try out the NVDLA software in the virtual simulator! Please refer to Software Manual (sw/contents.html) for details.

If you want to exit the virtual simulator, press 'ctrl+a x'.

3.7 Debugging the Virtual Simulator

Refer to 2.7 Debugging the Virtual Simulator (vp.html#debugging-the-virtual-simulator) to debug virtual simulator

4. Generating the AFI on AWS FPGA AMI

4.1 Setup AWS EC2 instance machine on FPGA AMI

- 2.1 Setup AWS EC2 instance machine: we recommend to choose *"FPGA Developer AMI"* (CentOS)
- 3.3 Install Dependencies: Follow the CentOS steps.

4.2 Download source code

- Download NVDLA AWS FPGA Custom Logic (CL)
- Download and build NVDLA
- AWS EC2 FPGA Hardware and Software Development Kits

4.3 Build NVDLA RTL

```
$ cd [nvdla_hw prefix]
$ make
$ ./tools/bin/tmake -build vmod
```

Please refer to Tree Build (hw/v1/integration_guide.html#tree-build) for details on building the NVDLA hardware tree, and make sure the required tools listed in Environment Setup (hw/v1/integration_guide.html#env-setup) are installed first.

4.4 Generate Vivado IP

Before generate NVDLA AFI, users need to generate some necessary Xilinx Vivado IP in AWS EC2 instance. The generated IP is not distributed with the NVDLA source distribution because of licensing restrictions.

Note

- You need to access the AWS EC2 instance from GUI machine to use Xilinx tool.
- Vivado veresion is /opt/Xilinx/SDx/2017.1.op/Vivado/bin/vivado
- For IP location, we recommend to use [vp_awsfpga prefix]/common/design/xilinx_ip/

4.4.1 Start Xilinx tool in AWS EC2 instance

```
$ vivado
```

4.4.2 Configure IP setting

- Click “Manage IP”
- Click “New IP Location”
- Click “Next”
- Configure the Manage IP Settings page, set “part” to “xcvu9p-flgb2104-2-i”
- Click “Finish”

4.4.3 Generate IP axi2apb

- In the IP catalog, search axi_apb
- Double click “AXI APB Bridge”
- Set “Component Name” to “axi_apb_bridge_0”
- Set “Number Of Slaves” to “1”
- Click “OK”
- Click “Generate”
- Click “OK” and wait the task in “Design Runs” panel to finish

4.4.4 Generate IP axi_interconnect_nvdla_64b

- In the IP catalog, expand “AXI_Infrastructure”, double click “AXI Interconnect RTL”
- Set “Component Name” to “axi_interconnect_nvdla_64b”
- Click the tab “Global”
- Set “Number of Slave Interface” to “2”
- Set “Slave Interface Thread ID Width” to “8”
- Set “Address Width” to “64”
- Set “Interconnect Internal Data Width” to “512”
- Click the tab “Interfaces”
- Set “Master Interface Data Width” to “512”
- Set “Slave Interface 0 data width” to “512”
- Set “Slave Interface 1 data width” to “64”
- Click the tab “Read Write Channels”
- Set all the “Acceptance” to “32”
- Set all the “FIFO Depth” to “512”
- Click “OK”
- Click “Generate”
- Click “OK” and wait the task in “Design Runs” panel to finish

4.4.5 Generate IP axi_interconnect_nvdla_512b

- In the IP catalog, expand “AXI_Infrastructure”, double click “AXI Interconnect RTL”
- Set “Component Name” to “axi_interconnect_nvdla_512b”
- Click the tab “Global”
- Set “Number of Slave Interface” to “3”
- Set “Slave Interface Thread ID Width” to “8”
- Set “Address Width” to “64”
- Set “Interconnect Internal Data Width” to “512”
- Click the tab “Interfaces”

- Set all the “Data Width” to “512”
- Click the tab “Read Write Channels”
- Set all the “Acceptance” to “32”
- Set all the “FIFO Depth” to “512”
- Click “OK”
- Click “Generate”
- Click “OK” and wait the task in “Design Runs” panel to finish

4.4.6 Generate IP axi_interconnect_nvdla_256b

- In the IP catalog, expand “AXI_Infrastructure”, double click “AXI Interconnect RTL”
- Set “Component Name” to “axi_interconnect_nvdla_256b”
- Click the tab “Global”
- Set “Number of Slave Interface” to “3”
- Set “Slave Interface Thread ID Width” to “8”
- Set “Address Width” to “64”
- Set “Interconnect Internal Data Width” to “512”
- Click the tab “Interfaces”
- Set all the “Data Width” to “256”
- Click the tab “Read Write Channels”
- Set all the “Acceptance” to “32”
- Set all the “FIFO Depth” to “512”
- Click “OK”
- Click “Generate”
- Click “OK” and wait the task in “Design Runs” panel to finish

4.4.7 Generate IP axi_protocol_converter_axi_to_axil

- In the IP catalog, expand “AXI_Infrastructure”, double click “AXI Protocol Converter”
- Set “Component Name” to “axi_protocol_converter_axi_to_axil”
- Set “Address Width” to “64”
- Set “Data Width” to “64”
- Click “OK”
- Click “Generate”
- Click “OK” and wait the task in “Design Runs” panel to finish

4.4.8 Generate IP axi_dwidth_converter_512b_to_64b

- In the IP catalog, expand “AXI_Infrastructure”, double click “AXI Data Width Converter”
- Set “Component Name” to “axi_dwidth_converter_512b_to_64b”
- Set “Address Width” to “64”
- Set “SI Data Width” to “512”
- Set “SI ID Width” to “16”
- Click “OK”
- Click “Generate”
- Click “OK” and wait the task in “Design Runs” panel to finish

4.5 Install AWS CLI

```
$ aws configure        # to set your credentials (found in your console.aws.amazon.com page) a
```

Note

You need to setup access keys for your AWS account, please refer to Managing Access Keys for Your AWS Account (https://docs.aws.amazon.com/general/latest/gr/managing-aws-access-keys.html)

4.6 Generate design checkpoint (DCP)

```
$ cd [aws fpga prefix] && source hdk_setup.sh
$ export CL_DIR=[vp_awsfpga prefix]/cl_nvdla
$ export NV_HW_ROOT=[nvdla_hw prefix]
$ cd $CL_DIR/build/scripts
$ ./filelist.sh
$ $HDK_DIR/common/shell_stable/build/scripts/aws_build_dcp_from_cl.sh -foreground -clock_recipe
```

The DCP generation process could take hours to finish, you should not stop the EC2 instance during this process. After the DCP is generated successfully, a tarball file should be generated under [vp_awsfpga prefix]/cl_nvdla/build/checkpoints/to_aws.

4.7 Generate AFI

```
$ aws s3 mb s3://<your-bucket-name> --region <region>        # Create an S3 bucket (choose a un
$ aws s3 mb s3://<your-bucket-name>/<your-dcp-folder-name>   # Create folder for your tarball f
$ aws s3 cp $CL_DIR/build/checkpoints/to_aws/<your-dcp-tallball> s3://<your-bucket-name>/<your-
$ aws s3 mb s3://<your-bucket-name>/<your-logs-folder-name>  # Create a folder to keep your log
$ touch LOGS_FILES_GO_HERE.txt                               # Create a temp file
$ aws s3 cp LOGS_FILES_GO_HERE.txt s3://<your-bucket-name>/<your-logs-folder-name>/  # Which cr
$ aws ec2 create-fpga-image --name <your-afi-name> --description <your-afi-description> --input

NOTE: The trailing '/' is required after <dcp-folder-name>
```

You will get a unique AFI ID and global AFI ID for your fpga image. You do not need to keep the EC2 instance running during this process. You can check the status using:

```
$ aws ec2 describe-fpga-images --fpga-image-ids <your-image-afi-id>
```

More details please refer to How to submit checkpoint to aws (https://github.com/aws/aws-fpga/blob/master/hdk/cl/examples/README.md#3-submit-the-design-checkpoint-to-aws-to-create-the-afi)