Vitis AI Development Flow on Avnet Ultrazed-EG board

1. Introduction

This document serves as an in-depth documentation of the Vitis-AI development flow on the Avnet UltraZed-EG. The 2020.2 version of the Xilinx Unfied Software Platform application will be used, along with v1.3.2 of the Vitis-AI stack.

1.1. Pre-requisites

The following applications and files are necessary for the Vitis-AI development flow:

Applications and packages

- Docker
- Extra Packages for Enterprise Linux (EPEL)
- Git
- Vitis AI Runtime (VART)
- Xilinx Unified Software Platform 2020.2

Files and repositories

- Avnet UltraZed-EG IOCC base Vitis Platform Project
- Xilinx Vits-AI Docker image 1.3.598
- Xilinx Vitis-AI Git repository v1.3.2
- Xilinx ZYNQMP common image

For the purposes of this documentation, we will be using an example from the following Xilinx Vitis-AI-Tutorials repository as well:

- Dogs vs Cats dataset
- Xilinx Vitis-AI-Tutorials

Docker Installation

Install Docker for your operating system from the Docker website.

Epel Package Download

Download the Extra Packages for Enterprise Linux on the host.

```
sudo yum install -y epel-release
```

Git Installation

Much of Vitis-AI's libraries and tools are hosted on GitHub. Git can be installed using the Yum package manager.

```
sudo yum install -y git
```

Vitis AI Runtime (VART) Set Up

The VART set up procedure is adapted from the official Xilinx documentation.

1. Download and execute the sdk-2020.2.0.0.sh script for setting up the Petalinux cross compilation environment.
2. When the installation is complete, run `source` on the `environment-setup-aarch64-xilinx-linux` file. This step is necessary before cross-compiling the AI models.

```
source [INSTALLATION-DIRECTORY]/environment-setup-aarch64-xilinx-linux
```

3. Download the vitis_ai_2020.2-r1.3.2.tar.gz.

```
wget https://www.xilinx.com/bin/public/openDownload?
filename=vitis_ai_2020.2-r1.3.2.tar.gz -O vitis_ai_2020.2-r1.3.2.tar.gz
```

4. Untar the contents into the `sysroots/aarch64-xilinx-linux` directory of the petalinux system. Do not delete the tar file yet, as this file will still be needed when setting up the target board.

```
tar -xzvf vitis_ai_2020.2-r1.3.2.tar.gz -C [INSTALLATION-
DIRECTORY]/sysroots/aarch64-xilinx-linux
```

Xilinx Unified Software Platform Installation

1. Download the Xilinx Unified Software Platform 2020.2 self-extracting web installer from the Xilinx download archives. Execute the binary file **with administrative/sudo rights** to launch the Xilinx GUI installation interface.

```
sudo ./Xilinx_Unified_2020.2_1118_1232_Lin64.bin
```

2. When the installation is complete, run `source` on the `environment-setup-aarch64-xilinx-linux` file.

```
source [INSTALLATION-DIRECTORY]/Vitis/2020.2/settings64.sh
source [INSTALLATION-DIRECTORY]/Vitis_HLS/2020.2/settings64.sh
source [INSTALLATION-DIRECTORY]/Vivado/2020.2/settings64.sh
```

Avnet UltraZed-EG IOCC Base Vitis Platform Project Download

1. Navigate to the `/2020.2/Vitis_Platform/uz3eg_iocc_vitis_2020_2.tar.gz` path on the Avnet sharepoint site to download the base Vitis Platform Project.

2. Extract the tar file once it has been downloaded.

```
tar -zxvf uz3eg_iocc_vitis_2020_2.tar.gz
```

Dogs vs Cats Data Download

The sample data used by this guide can be downloaded here. Unzip the zipped datasets.

Xilinx Vitis-AI Docker Image Download

Ensure that Docker has been installed on the computer. Start the Docker service and pull the Xilinx Docker image onto the local host.

```
# Enable and start Docker service.
sudo systemctl enable docker
sudo systemctl start docker

# Pull the Docker image.
docker pull xilinx/vitis-ai:1.3.598
```

Refer to the common issues section if the pull command throws a "permission denied" error.

Xilinx Vitis-AI Git Repository Download

Clone branch 1.3.2 of the Vitis-AI repository onto the local host.

```
git clone -b 1.3.2 --recurse-submodules https://github.com/Xilinx/Vitis-AI.git
```

Xilinx Vitis-AI-Tutorials Git Repository Download

Clone branch 1.3.2 of the Vitis-AI repository onto the local host.

```
git clone -b 1.3.2 --recurse-submodules https://github.com/Xilinx/Vitis-AI-Tutorials.git
```

Xilinx ZYNQMP Common Image Download

Download the Xilinx ZYNQMP common image for Embedded Vitis Platforms from the Xilinx common image archive. Ensure that version **2020.2** is selected.

Extract the tar file downloaded.

```
tar -zxvf xilinx-zynqmp-common-v2020.2.tar.gz
```

1.2. Setting up the workspace

1. Create a workspace directory containing the following items:
    - Avnet UltraZed-EG IOCC base Vitis Platform Project (extracted)
    - Xilinx Vitis-AI Git repository v1.3.2
    - Xilinx Vitis-AI-Tutorials Git repository v1.3
    - Xilinx ZYNQMP common image (extracted)

```
├── uz3eg_iocc_base
├── Vitis-AI
├── Vitis-AI-Tutorials
└── xilinx-zynqmp-common-v2020.2
```

2. Make a copy of the `dsa/DPU-TRD` directory from the Vitis-AI repository in the workspace directory. Rename it to `DPU-TRD-uz3eg_iocc`. This directory contains the DPU IP information, neccessary for setting up the DPU on the board's PL. (Run the following commands from the workspace directory we just created.)

```
cp -r ./Vitis-AI/dsa/DPU-TRD .
mv DPU-TRD DPU-TRD-uz3eg_iocc
```

3. Make a copy of the `Design_Tutorials/08-tf2_flow` directory from the Vitis-AI-Tutorials repository in the Vitis-AI directory. This will be the sample model which we will be working on in this guide. (Run the following commands from the workspace directory we just created.)

```
cp -r ./Vitis-AI-Tutorials/Design_Tutorials/08-tf2_flow ./Vitis-AI
```

4. Move the Dogs vs Cats dataset into the `08-tf2_flow/files` directory.

Your workspace should look something like this now:

```
├── DPU-TRD-uz3eg_iocc
├── uz3eg_iocc_base
├── Vitis-AI
│   ├── 08-tf2_flow          # 08-tf2_flow should be under the Vitis-AI
directory
│   │   ├── files
│   │   │   ├── dogs-vs-cats  # Dataset from Kaggle should be under the 08-
tf2_flow/files directory
│   │   │   ...
│   │   ...
│   ...
├── Vitis-AI-Tutorials
└── xilinx-zynqmp-common-v2020.2
```

1.3. Setting up the environment variables

This section will be about the set up of the environment variables. This step is crucial in ensuring that the scripts and executables provided by the Vitis-AI repository work properly. Create the following environments: * `VITIS_AI_HOME`: **Absolute** path of the Vitis-AI Git repository * `SDX_PLATFORM`: **Absolute** path of the `.xpfm` file in the Avnet UltraZed-EG IOCC Base Vitis Platform Project * `TRD_HOME`: **Absolute** path of the DPU-TRD-uz3eg_iocc directory created earlier * `EDGE_COMMON_SW`:**Absolute** path of the Xilinx Zynqmp common image

Run the following command from the workspace directory created earlier:

```
export VITIS_AI_HOME=${PWD}/Vitis-AI
export SDX_PLATFORM=${PWD}/uz3eg_iocc_base/uz3eg_iocc_base.xpfm
export TRD_HOME=${PWD}/DPU-TRD-uz3eg_iocc
export EDGE_COMMON_SW=${PWD}/xilinx-zynqmp-common-image-v2020.2
```

We are now ready to begin configuring the DPU.

2. Configuring the DPU IP

The DPU configuration to be used by the UltraZed-EG is the DPUCZDX8G. Navigate to the `DPU-TRD-uz3eg_iocc/prj/Vitis` directory:

```
cd DPU-TRD-uz3eg_iocc/prj/Vitis
```

We will be editting the following files in this directory:

```
Workspace/DPU-TRD-uz3eg_iocc
  ├── prj
```

```
    |   ├── Vitis
    |   |   ├── config_file
    |   |   |   ├── prj_config      # We will be making changes to this file
    |   |   |   ...
    |   |   ├── dpu_conf.vh         # and also this file
    |   |   ...
    |   ...
    ...
```

2.1. Changes to the `dpu_conf.vh` file:

1. **Line 26**: We change the DPU convolution architecture to `B2304`. The number within the convolution architecture's name (i.e., 2304) represents the peak operations per clock cycle. A larger number implies faster DPU performance, but also signifies a larger number of DSP slices being used in the board. For the UltraZed-EG, B2304 is the best architecture available for the UltraZed-EG, given the DSP slices available.

```
/*====== Architecture Options ======*/
// |-----------------------------------------------------|
// | Support 8 DPU size
// | It relates to model. if change, must update model
// +-----------------------------------------------------+
// | `define B512
// +-----------------------------------------------------+
// | `define B800
// +-----------------------------------------------------+
// | `define B1024
// +-----------------------------------------------------+
// | `define B1152
// +-----------------------------------------------------+
// | `define B1600
// +-----------------------------------------------------+
// | `define B2304
// +-----------------------------------------------------+
// | `define B3136
// +-----------------------------------------------------+
// | `define B4096
// |-----------------------------------------------------|

`define B2304
```

2. **Line 41 – 43**: In this step, we update the URAM configurations to suit our UltraZed-EG. Although URAM will not be used in this specific project, this step will come in handy should we decide to utilize the URAM in future projects.

```
// |-----------------------------------------------------|
// | If the FPGA has Uram. You can define URAM_EN parameter
// | if change, Don't need update model
```

```
// +-----------------------------------------------------+
// | for zcu104 : `define URAM_ENABLE
// +-----------------------------------------------------+
// | for zcu102 : `define URAM_DISABLE
// |-----------------------------------------------------|

`define URAM_DISABLE

//config URAM
`ifdef URAM_ENABLE
    `define def_UBANK_IMG_N          4
    `define def_UBANK_WGT_N          13
    `define def_UBANK_BIAS           1
`elsif URAM_DISABLE
    `define def_UBANK_IMG_N          0
    `define def_UBANK_WGT_N          0
    `define def_UBANK_BIAS           0
`endif
```

3. **Line 148**: In this step, we change the DSP usage to low, to ensure the DSP slices used is within the specifications afforded by the board.

```
// |-----------------------------------------------------|
// | DSP48 Usage Configuration
// | Use dsp replace of lut in conv operate
// | if change, Don't need update model
// +-----------------------------------------------------+
// | `define DSP48_USAGE_HIGH
// +-----------------------------------------------------+
// | `define DSP48_USAGE_LOW
// |-----------------------------------------------------|

`define DSP48_USAGE_LOW
```

Here's a script to make the above changes automatically:

```
# Choose DPU architecture for our project
sed -i '26s/.*/`define B2304 /' dpu_conf.vh

# Update URAM configurations for UltraZed-EG
sed -i '41s/.*/    `define def_UBANK_IMG_N          4/' dpu_conf.vh
sed -i '42s/.*/    `define def_UBANK_WGT_N          13/' dpu_conf.vh

# Update DSP usage configuration for UltraZed-EG
sed -i '148s/.*/`define DSP48_USAGE_LOW /' dpu_conf.vh
```

2.2. Changes to the `config_file/prj_config` file:

Next, we modify the `config_file/prj_config`. This file allows one to set the clock speed and the number of DPU cores. The UltraZed-EG only have sufficient resources for a single DPU core. Hence, the configurations under `[clock]` and `[connectivity]` are as follows:

1. We first change the `[clock]` section to modify the clock speeds. We will use `150000000`Hz for `1.aclk` and `300000000`Hz for `1.ap_clk_2`. `2.aclk` and `2.ap_clk_2` is commented out since DPUCZDX8G only has a single core. Note that `ap_clk_2` **must be twice the frequency** of `aclk`.

```
[clock]

freqHz=150000000:DPUCZDX8G_1.aclk
freqHz=300000000:DPUCZDX8G_1.ap_clk_2
# freqHz=300000000:DPUCZDX8G_2.aclk
# freqHz=600000000:DPUCZDX8G_2.ap_clk_2
```

2. We then change the `[connectivity]` section for selecting the AXI connections. Since DPUCZDX8G only has a single core, the connections of `DPUCZDX8G_2` are commented out. Finally, we change the core count to 1.

```
[connectivity]

sp=DPUCZDX8G_1.M_AXI_GP0:HPC0
sp=DPUCZDX8G_1.M_AXI_HP0:HP0
sp=DPUCZDX8G_1.M_AXI_HP2:HP1
# sp=DPUCZDX8G_2.M_AXI_GP0:HPC0
# sp=DPUCZDX8G_2.M_AXI_HP0:HP2
# sp=DPUCZDX8G_2.M_AXI_HP2:HP3

nk=DPUCZDX8G:1
```

Here's a script to make the above changes automatically:

```
# Set clock frequency for DPU
sed -i '20s/.*/freqHz=150000000:DPUCZDX8G_1.aclk/' config_file/prj_config
sed -i '21s/.*/freqHz=300000000:DPUCZDX8G_1.ap_clk_2/'
config_file/prj_config
sed -i '22s/.*/# freqHz=300000000:DPUCZDX8G_2.aclk/' config_file/prj_config
sed -i '23s/.*/# freqHz=600000000:DPUCZDX8G_2.ap_clk_2/'
config_file/prj_config

# Set AXI connections for DPU
sed -i '30s/.*/# sp=DPUCZDX8G_2.M_AXI_GP0:HPC0/' config_file/prj_config
sed -i '31s/.*/# sp=DPUCZDX8G_2.M_AXI_HP0:HP2/' config_file/prj_config
sed -i '32s/.*/# sp=DPUCZDX8G_2.M_AXI_HP2:HP3/' config_file/prj_config

# Update DPU core count
```

```
sed -i '35s/.*/nk=DPUCZDX8G:1/' config_file/prj_config
```

3. Creating The Bootable SD Card Image

While still within the `DPU-TRD-uz3eg_iocc/prj/Vitis` directory, run the following command to build the bootable SD image:

```
make KERNEL=DPU DEVICE=uz3eg_iocc
```

This should generate the necessary files for our SD card within the DPU-TRD-uz3eg_iocc/prj/Vitis/binary_container_1 directory:

```
.
├── BOOT.BIN
├── dpu.xclbin                        # This file will be needed when
running apps on target
├── dpu.xclbin.info
├── dpu.xclbin.link_summary
├── dpu.xo
├── ip_cache
├── link
├── logs
├── reports
├── sd_card
│   ├── arch.json                     # This file will be needed for cross-
compilation
│   ├── BOOT.BIN
│   ├── boot.scr
│   ├── dpu.xclbin
│   ├── Image
│   ├── image.ub
│   ├── init.sh
│   ├── platform_desc.txt
│   ├── rootfs.tar.gz
│   └── uz3eg_iocc_base.hwh
├── uz3eg_iocc_base.bif
├── uz3eg_iocc_base.img               # SD card image can be found here
├── v++_link_dpu_guidance.json
└── v++_link_dpu_guidance.pb
```

If you encounter the `/bin/Vivado: Command not found`, it is likely that the `settings64.sh` files within the Xilinx Unfied Software Platform installation directories have yet to be `source`-ed. Refer to here.

4. Running the TensorFlow2 application on target

To quantize and compile TensorFlow2 models in Vitis-AI, the pre-trained model should be a `.h5` file. For this guide, we will be using the pretrained model `f_model.h5` provided under the `08-tf2_flow/files/pretrained`.

4.1. Quantizing The Model

We will be using the simplified quantizer file named `my_quantizer.py`. This file should be placed wthin the `08-tf2_flow/files` directory:

```
Vitis-AI
├── 08-tf2_flow
│   ├── files
│   │   ├── my_quantize.py
│   │   ...
│   ...
...
```

The main lines involving the Vitis-AI quantizer are lines 1, 25 and 26:

- **Line 1**: Import `vitis_quantize`.

```
|01| from tensorflow_model_optimization.quantization.keras import
vitis_quantize
```

- **Line 25**: The `vitis_quantize.VitisQuantizer` constructor takes in a pre-trained `.h5` model and returns a `VitisQuantizer` object for the model.

```
|25| quantizer = vitis_quantize.VitisQuantizer(PRETRAINED_MODEL_PATH)
```

- **Line 26**: Use the `VitisQuantizer` object's `quantize_model()` method to pass in the calibration dataset as a `tf.data.Dataset` object and quantize the model.

```
|26| quantized_model =
quantizer.quantize_model(calib_dataset=quant_dataset)
```

Enter the Vitis-AI repository and use the `docker_run.sh` script to start the Vitis-AI Docker image downloaded previously.

```
cd ${VITIS_AI_HOME}
./docker_run.sh xilinx/vitis-ai:1.3.598
```

Activate the Tensorflow2 python virtual environment with `conda activate vitis-ai-tensorflow2`

```
conda activate vitis-ai-tensorflow2
```

Enter the `08-tf2_flow/files` directory:

```
cd 08-tf2_flow/files
```

Run `my_quantize.py`.

```
python my_quantize.py
```

This generates the quantized model as `q_model.h5`

Exit the Docker environment:

```
conda deactivate
exit
```

4.2. Cross-compiling the Model

1. Copy the arch.json file from the binary container sd_card directory to the `08-tf2_flow/files` directory. This file specifies the DPU architecture which the application should be compiled for.

```
cp ../../../DPU-TRD-
uz3eg_iocc/prj/Vitis/binary_container_1/sd_card/arch.json .
```

```
Vitis-AI
├── 08-tf2_flow
│   ├── files
│   │   ├── arch.json     <-- arch.json should end up here, alongside the
.h5 file.
│   │   ├── q_model.h5
│   │   ...
│   ...
...
```

2. Once again, return to the Vitis-AI repository and use the `docker_run.sh` script to start the Vitis-AI Docker image downloaded previously.

```
cd ${VITIS_AI_HOME}
./docker_run.sh xilinx/vitis-ai:1.3.598
```

3. Activate the Tensorflow2 python virtual environment with `conda activate vitis-ai-tensorflow2`

```
conda activate vitis-ai-tensorflow2
```

4. Compile the quantized model using the `vai_c_tensorflow2` command in the Docker image. Note the following flags and their usages
    - --model : Specify the path to the quantized `.h5` model after this flag.
    - --arch : Specify the path to the `arch.json` file after this flag. This file can be foudn in the binary container generated earlier.
    - --output_dir : Specify the output path of the compiled `.xmodel` file after this flag.
    - --net_name : Specify the compiled model's name after this flag

```
# Return to directory with quantized model
cd 08-tf2_flow/files

# Use vai_c_tensorflow2
vai_c_tensorflow2 \
    --model ./q_model.h5 \
    --arch ./arch.json \
    --output_dir ./ \
    --net_name dog-v-cat-cnn
```

This generates a `dogs-v-cats-cnn.xmodel` binary for the target board.

5. For the purpose of this guide, we will generate a small dataset for the model as well.

```
python target.py --image_dir test --target_dir target2 --num_images 1000
```

6. Exit the Docker environment:

```
conda deactivate
exit
```

4.3. Vitis AI Runtime (VART) and Vitis AI Modelzoo

Xilinx provides a set of VART APIs for programming applications to call the cross-compiled models on the target. These APIs are available in both C++ and Python.

For this guide, we will be using the VART application provided in the Vitis-AI tutorial for tensorflow:

```
Vitis-AI
├── 08-tf2_flow
│   ├── files
│   │   ├── application
│   │   │   └──  app_mt.py      # This is the VART application that will be
used to call the xmodel
│   │   ...
│   ...
...
```

Vitis AI also has a series of prebuilt AI models known as the modelzoo. Each of the `.yaml` files in the repository contains the download links for the model.

- These models are **quantized but not compiled**. `vai_c` is needed to cross-compile the models.
- This link provides a script to automatically download and cross-compile all models in the modelzoo. However, do note that the `arch.json` file should be present in the same directory.
- Most models in the modelzoo have their corresponding VART application prepared by Xilinx for calling the model on the target board. These VART applications are available in either C++ or Python. Note that C++ VART applications will have to be compiled using the `build.sh` scripts present.
- There is also a Vitis-AI library which is a set of tools prepared by Xilinx for testing the modelzoo models. They are also built on the VART API.

5. Setting Up The SD Card

1. Insert and mount the SD card. Next, use the Linux `dd` tool to burn the `.img` file in the binary container onto the SD card.

```
# Replace sd{X} with the partition which the SD card was mounted in.
sudo dd bs=4M
if=${DPU_TRD_HOME}/prj/Vitis/binary_container1/uz3eg_iocc_base.img
of=/dev/sd{X} status=progress conv=fsync
```

2. Extract the vitis_ai_2020.2-r1.3.2.tar.gz file downloaded during the VART set up earlier. Copy the `etc/*` and `usr/*` files over to `/usr` and `/etc` directories of the SD card's RootFS partition respectively.

3. Copy the `dpu.xclbin` file from `${DPU_TRD_HOME}/prj/Vitis/binary_container1/dpu.xclbin` to the `/media/Sd-mmcblk0p1` directory of the SD card's RootFS partition.

4. Remove the `xir` directory and the `runner.so` file under the `/usr/lib/python3.7/site-packages` directory.

5. Copy the following files over to the SD card as well:

- Cross-compiled `dogs-v-cats-cnn.xmodel` from `${VITIS_AI_HOME}/08-tf2_flow/files`
- VART application `app_mt.py` from `${VITIS_AI_HOME}/08-tf2_flow/files/application/app_mt.py`
- `test` directory with the test data from `${VITIS_AI_HOME}/08-tf2_flow/files/test`
- `dpu_sw_optimize.tar.gz` file from `${TRD_HOME}/app/dpu_sw_optimize.tar.gz`

6. Booting Up The Board

1. Insert the SD card prepared earlier in the board and power it up.
2. Use the Screen tool to set up a serial connection from the host to the target board.

```
sudo Screen /dev/ttyUSB1 115200
```

3. Extract the `dpu_sw_optimize.tar.gz` and execute the `dpu_sw_optimize/zynqmp/zynqmp_dpu_optimize.sh` script

```
tar -zxvf dpu_sw_optimize.tar.gz
cd dpu_sw_optimize/zynqmp
bash zynqmp_dpu_optimize.sh
cd ../..
```

4. The models can now be executed using a command of the following format:
   - C++ VART applications: `[VART APPLICATION] [XMODEL] [DATA]`
   - Python VART applications: `python [VART APPLICATION] [XMODEL] [DATA]`

For out case, this would be:

```
python app_mt.py dogs-v-cats-cnn.xmodel test
```

Common Issues

Docker Permission Denied

This section is adapted from this StackOverflow tread.

A common error encountered when running Docker commands is the `permissions denied` error:

```
$~: docker: Got permission denied while trying to connect to the Docker
daemon socket at unix:///var/run/docker.sock:
... ... ... dial unix /var/run/docker.sock: connect: permission denied. See
'docker run --help'
```

1. Create a new users group and add the user into it.

```
# Create a "docker" user group if it does not exist.
sudo groupadd docker
# Add your user to the docker group.
sudo usermod -aG docker $USER
# Switch session to the newly created group.
newgrp docker
# Restart Docker
sudo systemctl restart docker.
```

2. If the problem persists, try adding read and write rights to the Docker daemon.

```
sudo chmod 666 /var/run/docker.sock
```

`/bin/Vivado : Command not found` error when building SD card image

It is likely that the settings64.sh files in the installation directory of the Xilinx Unified Software Platform application have yet to be executed. These files have to be executed each time a new terminal is opened, in order to set up the environment.

```
source [XILINX UNIFIED SOFTWARE PLATFORM INSTALLATION
DIRECTORY]/Vitis/2020.2/settings64.sh
source [XILINX UNIFIED SOFTWARE PLATFORM INSTALLATION
DIRECTORY]/Vitis_HLS/2020.2/settings64.sh
source [XILINX UNIFIED SOFTWARE PLATFORM INSTALLATION
DIRECTORY]/Vivado/2020.2/settings64.sh
```

`AttributeError: module 'xir' has no attribute 'Graph'` error when running application on target

Remove the `xir` directory and the `runner.so` file under the `/usr/lib/python3.7/site-packages` directory of the SD card. See Github comment.