

Dyplo on Alveo

Release 2020.2

Niek van Agt

May 11, 2021

CONTENTS

1 What is the Dyplo on Alveo development package? 1

2 HW / FPGA part 3
- 2.1 Static FPGA part . . . 3
- 2.2 Dynamic FPGA part (partials) . . . 3

3 SW part 5
- 3.1 Dyplo driver (kernel-module-dyplo) . . . 5
- 3.2 Dyplo unlocking app (Accelize DRM) . . . 5
- 3.3 Dyplo library (libdyplo) . . . 5
- 3.4 Dyplo user/demo application (dyplo-qt-image-demo) . . . 6

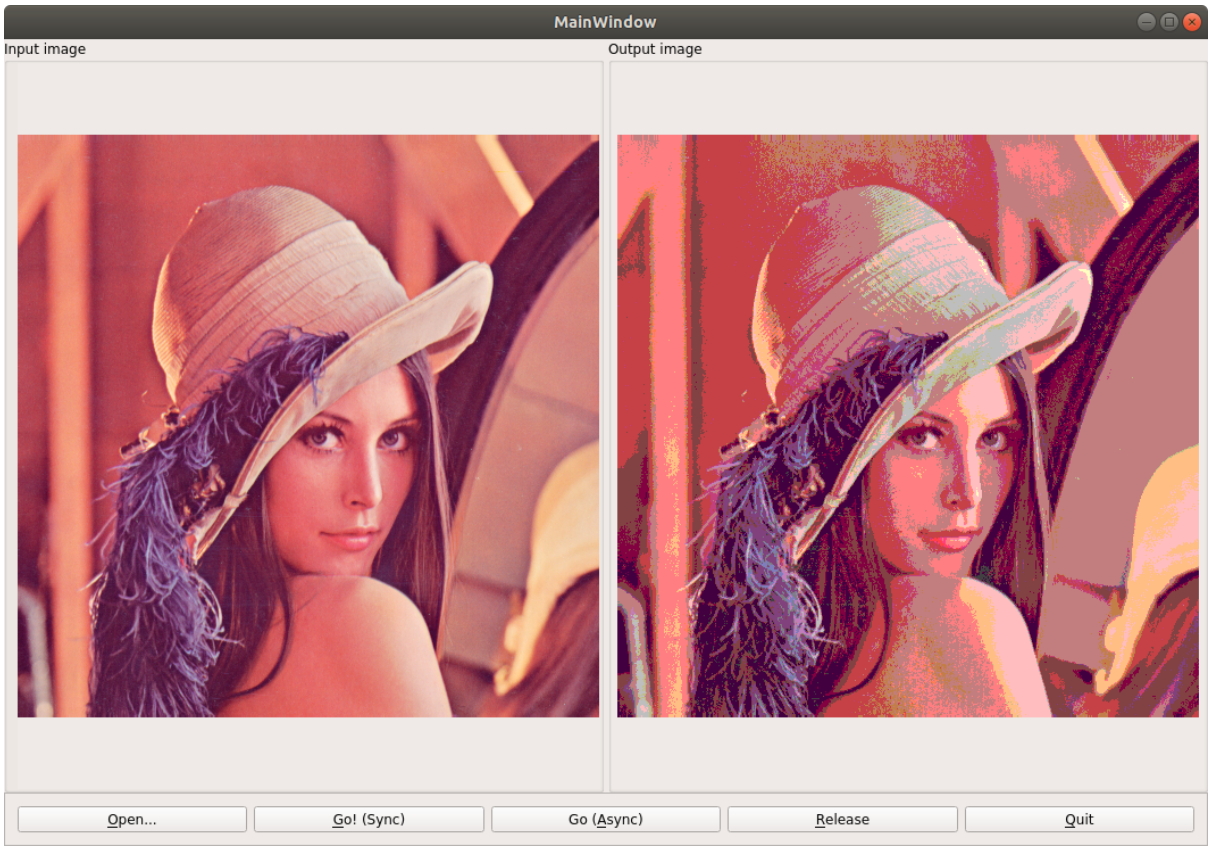
4 Debugging / Known issues 7

CHAPTER

ONE

WHAT IS THE DYPLO ON ALVEO DEVELOPMENT PACKAGE?

The Dyplo on Alveo development package is a demo package to showcase and try-out Dyplo running on an Xilinx Alveo board. A Dyplo framework running on the Alveo board is connected through PCIe to be used by the host PC. The host PC is running an Dyplo example application that streams an image file through a filter in the Dyplo framework. The 'before' and 'after' results are then showed in a GUI created with QT. See image below:



A short explanation of the functions behind the buttons

- **Open**: Opens an 'file open' dialog to select a image file from disk.
- **Go! (Sync)**: Run the image through Dyplo and the filter (in a blocking way) and show the result.
- **Go! (Async)**: Run the image through Dyplo and the filter (in a non-blocking way, with separate threads for data reading/writing) and show the result.
- **Release**: Clear the 'result' image.
- **Quit**: Exit the application.

The package consists out of 2 main parts. The software and the hardware part. Therefore this documentation is also divided into 2 sections like that.

Note: This development package is build and tested with an **Alveo U50** board and a **Ubuntu 18.04** OS. The following packages have to be installed: python, make, gcc, pkg-config and qtcreator. Run the following to install:

```
sudo apt-get install python make gcc pkg-config qtcreator
```

Also configure the system to use dash as the default command interpreter by running the command below. Choose 'No' when asked for.

```
sudo dpkg-reconfigure dash
```

CHAPTER

TWO

HW / FPGA PART

The HW/FPGA part for running Dyplo on the Alveo board has 2 parts. The static part, where the Alveo board will be programmed with the static FPGA image containing the PCIe link and the Dyplo framework. And the dynamic part which will be the creation of a HDL node filter.

Note: Use **Vivado 2020.2** for the next steps. Other versions might work but are not supported by TOPIC.

2.1 Static FPGA part

The static part consists of only 1 step which is programming the Alveo board with the static FPGA image. This can be done by following the next (sub) steps:

1. Connect the Alveo board to your host PC using the JTAG connection
2. Open the Hardware Manager inside a Vivado session
3. Add the configuration memory device to the Alveo's FPGA (for U50: **mt25qu01g-spi-x1_x2_x4**)
4. Program it with the .mcs file in `FPGA/build/dyplo_on_alveo.mcs` as delivered with the package.

2.2 Dynamic FPGA part (partials)

The dynamic FPGA part is the creation of a custom (user) filter. The default filter that is delivered (pre-built) with the pacakge is a thresholding function. This function alters the pixel values based on the range the original pixel is. If the pixel is between 0 and 32, the pixel will become 0. If the pixel is between 32 and 64, the pixel will become 32. Etc...

The demo can be run with this default threshold function. If so, most of the steps in this 'dynamic' part can be skipped. Only the last 2 need to still be done.

To create a custom filter, follow the steps below. To make running of these steps easier, set the `TRD_DIR` environment variable.

```
export TRD_DIR=<path_to_unzipped_dyplo_on_alveo_package>
```

The first step is to generate the HDL node template by running:

```
pushd ${TRD_DIR}/FPGA/HDL_node_template
vivado -mode tcl -source generate_hdl_node_template.tcl
popd
```

Note: This only has to be run once to configure/prepare the HDL node template for further use

Create a HDL node from the template by running:

```
pushd ${TRD_DIR}/FPGA/HDL_node_template
vivado -mode tcl -source hdl_node_template.tcl -tclargs my_node ../hdl_nodes/my_
↪node xcu50-fsvh2104-2-e
popd
```

Now it is time to add an actual filter to the HDL node. Anything can be added to the HDL node as long as it has in between 1 and 4, 64 bits wide, streaming inputs and outputs. The HDL node also has a memory-mapped interface to configure parameters of the filter but this is not covered in this example.

Adding a filter can be done in 2 ways, by using handcrafted HDL or by using Vitis HLS. To change via handcrafted HDL, open the HDL node project in Vivado and develop the filter. To use HLS to generate HDL code, an HLS project template is provided in the FPGA/HLS_project_template directory. To create and run this project run the following:

```
pushd ${TRD_DIR}/FPGA/HLS_project_template/
vitis_hls -f run_hls.tcl
popd
```

The next step is to synthesize the HDL node and create a checkpoint of it. Do this by running:

```
pushd ${TRD_DIR}/FPGA
vivado -mode tcl -source ./scripts/TCL/synthesize_hdl_node.tcl -tclargs hdl_nodes/
↪my_node/my_node.xpr my_node xcu50-fsvh2104-2-e
popd
```

With the checkpoint of the HDL node, we can now create the partial bitstreams for it by running:

```
pushd ${TRD_DIR}/FPGA
vivado -mode tcl -source ./scripts/TCL/impl_dynamic_logic.tcl -tclargs build/
↪checkpoints/top_impl_static.dcp my_node hdl_nodes/my_node/hdl_node_my_node_synth.
↪dcp 1?2 ./build/
popd
```

The application needs .bin files to work with so we'll have to convert the generated .bit file to .bin files. Do this by running:

```
pushd ${TRD_DIR}/FPGA
mkdir build/partials
./scripts/package_partials.sh build/bitstreams build/partials
popd
```

And the last step is to copy the partials to the host PC. The subfolders (rgb_treshold, my_node) need to be placed in `/usr/local/share/bitstreams/`.

The software part will find the partials here and use them to program filters on the Alveo board.

CHAPTER

THREE

SW PART

The software for this demo is delivered in 4 components that work together to make the demo work.

3.1 Dyplo driver (kernel-module-dyplo)

This is the kernel-module / driver for the Dyplo IP. This has to be compiled and installed on the host PC as described in its README.

Note: Don't forget to copy the dyplo.rules file (see README).

3.2 Dyplo unlocking app (Accelize DRM)

This app is required to unlock the Dyplo IP on the Alveo board. It is licensed through the Accelize DRM toolset. This app needs 2 JSON config files to be able to run. Both files need to be located in the `${TRD_DIR}/SW/dyplo-accelize-drm-app` directory.

- conf.json : File with configuration of the DRM IP (delivered with package `${TRD_DIR}/SW/dyplo-accelize-drm-app/conf.json`)
- cred.json : Credentials file, needs to be purchased/downloaded via the Xilinx App Store.

To build this app follow the instructions in its README.

Note:

This application must be run within 15 seconds after each boot to prevent the the Dyplo IP locking itself. Instructions to

enable this are available in the README of this app.

3.3 Dyplo library (libdyplo)

This is the Dyplo library / API that functions as the bridge between the driver and the user-space Dyplo application(s). Also this library needs to be compiled/installed on the host PC as described in its README.

3.4 Dyplo user/demo application (dyplo-qt-image-demo)

This is the user-space Dyplo application. This example is a QT application with a small GUI that works with images. In the application a image file (.jpg) is streamed through a Dyplo filter where a tresholding function is performed. The result is displayed as 'before -> after' result in the GUI. This application can be used by installing QT Creator on the host PC and importing the project file (as described in its README).

CHAPTER

FOUR

DEBUGGING / KNOWN ISSUES

When problems occur while trying to run the application check the following list with possible issues and solutions

---

Issue: *The application 'hangs' or crashed when I pressed one of the 'Go' buttons...*

Possible problem: The Dyplo IP is locked. This can be checked by running:

```
cat /proc/dyplo
```

If Dyplo is locked, the last line of the output will be *WARNING: License expired, logic is locked.*

Solution: Reboot the PC and make sure the dyplo-accelize-drm-app is executed within 15 seconds after boot.

Possible problem: .bit files are copied instead of .bin files. Check the contents of directory `/usr/local/share/bitstreams/` There should be only .bin files (within subfolders), no .bit files.

Solution: Copy the right .bin files from `${TRD_DIR}/FPGA/build/partials` to the host PC and remove any .bit files present.

Issue: dyplo-accelize-drm-app: *failed to mmap /dev/dyplo-drm*

Possible problem: The PCIe Alveo card is not detected by the OS. (It is a known issue that this happens sometimes at cold boot)

Solution: Reboot the PC.

Possible problem: The `/dev/dyplo-drm` file does not have the correct file permissions.

Solution: Make sure to copy the dyplo.rules file to the right location. See *kernel-module-dyplo*.

Last updated on 2021-05-11