FGPU Tutorial
version 1.0

Muhammed Al Kadi

July 16, 2017

This tutorial helps new FGPU users to get used to its tool flow. It offers a quick guide to configure, synthesize and program an FGPU core. In addition, it describes how to install the required tools, e.g. the FGPU compiler and how to run standard benchmarks in Xilinx SDK.

1 Introduction

Development with FGPU goes through the following steps:

- On the hardware side:
  - The FGPU features are defined in a *.vhd* file, e.g. # of CUs and # of AXI data interfaces.
  - Then, a script implements the corresponding FGPU core with Vivado and generates:
    - A hardware description file (*.hdf*) that is needed to generate an SDK project.
    - A bitstream to program the FPGA.
- On the hardware side:
  - An OpenCL kernel is written and compiled with the FGPU compiler to run on FGPU. A script generates *.h* and *.c* files to be included in an SDK project.
  - A C/C++ application is written to run on the ARM Cortex-A9. This program interfaces and controls FGPU.

3 versions of FGPU have been developed:

1. *V1* has two clock domains. It is less scalable and has no thread-divergence support. The development of FGPU-V1 is discontinued.
2. *V2* works on a single clock domain. It is the most recent FGPU core.
3. *V3* is a branch of V2 and it supports partial configuration of floating-point logic.

At the time of writing this tutorial, only V2 is added to the repository.

Table 1: Required packages to setup an FGPU tool flow

OS	Required Packages
Ubuntu 16.04 - 64bit	ctags, cmake, g++, gawk, gcc-multilib, zlib1g-dev:i386

2 Requirements

Make sure you have a computer with the following software:

- *Operating System:* you should have a Linux OS. The FGPU tool flow has been verified on Ubuntu 16.04.2 LTS 64bit and Fedora Workstation 22-25 64bit. It will definitely not work on Windows.
- *Vivado Design Suite:* FGPU has been synthesized with the versions 2016.1-3 and 2017.2. Only Zynq-SoC device support is required and the Xilinx SDK should be included.
- *ModelSim or QuestaSim:* only needed for RTL simulations.
- *A terminal emulator:* to communicate with the HW during execution, e.g. minicom.

3 Setup

Please ignore this section if you are using the FGPU-VM.

3.1 Download the FGPU Repository

If you have *git* installed, you may checkout the FGPU repository from github with:

```
git clone https://github.com/malkadi/FGPU
```

Otherwise, you may download and extract it manually to any local directory. Any paths given in the rest of this tutorial are relative to to the chosen installation directory.

3.2 Install the Required OS Packages

For Ubuntu 16.04.2 64bit, you have to install the packages listed in Table 1. This can be achieved by executing:

```
sudo apt install $(cat scripts/packages)
```

Otherwise, you have to install the equivalent packages for your OS

3.3 Configure the Vivado Installation Path

Please edit `script/set_paths.sh` to source the `settings64.sh` script which is included in your Vivado installation.

3.4 Download and Compile LLVM

First, the LLVM source code v3.7.1 with the clang compiler should be downloaded and extracted. Then, the FGPU backend files in `llvm-3.7.1.src.fgpu` can be copied to the LLVM source directory. Finally, the whole LLVM code can be compiled to generate the compiler executables. All these steps are automated in `scripts/download_and_compile_llvm.sh`.

3.5 Compile FGPU Benchmarks in Xilinx SDK

Before running the FGPU benchmarks located in `benhcmarks`, you need to:

1. Create the HW and BSP projects by executing `scripts/create_sdk_bsps.sh`
2. Create and compile the FGPU benchmarks by executing `scripts/create_sdk_project.sh`

4 HW Synthesis

- Configure your FGPU HW parameters in `RTL/FGPU_definitions.vhd`
- Set the target clock frequency in `scripts/tcl/implement_FGPU.tcl`. In addition, you may modify the synthesis and implementation settings and the number of maximum number of threads used by Vivado. FGPU may be synthesized at frequencies up to 250MHz.
- Run the implementation script `scripts/tcl/implement_FGPU.tcl`. It calls `scripts/tcl/create_FGPU_block_design.tcl` which creates a block diagram that connects FGPU to the ARM processor[1]. It uses the four available AXI4-HP interfaces. However, only the number of interfaces configured in `RTL/FGPU_definitions.vhd` will be used. All synthesis and implementation files can be found in `HW/synth` and `HW/implement`, respectively. Al outputs, e.g. bitstream and *.hdf* files, are stored in `HW/outputs`.

5 Write and Compile the Device Code

- Write your device code in OpenCL in a *<benchmark>.cl* file in `kernels` directory.
- Compile it with `compile.sh` or `compile_and_log.sh`. The compilation outputs can be found in `kernels/outputs`. If hard floating-point support is desired, compile your code with the `-hard-float` feature.

[1] The block diagram can be found in `HW/.srcs/sources_1/bd/bd_design`

6 Write and Compile the Host Code

- Write your C/C++ in a `benchmarks/<benchmark>/src/`. You may need to copy the compiled device code from `kernels/outputs/<benchmark>.h,.c` next to your C/C++ files.
- It is highly recommended to have a look at other given benchmarks, e.g. xcorr.
- You may compile your code by executing `scripts/compile.sh`. Alternatively, you may execute `make` from `benchmarks/<benchmark>/Release`.

7 Download and Execute

- Set the bitstream you want to download in `benchmraks/program_bitstream.tcl`
- From the `benchmarks` directory, you may open the SDK project you compiled in the last step

  ```
  ./open_sdk.sh <benchmark>
  ```

  This will open an xsct terminal.
- Make sure you have opened the terminal emulator and a connection to the board is established.
- Source the following scripts in the xsct terminal

  ```
  source program_bitstream.tcl; source download_elf.tcl
  ```

  This will program the desired bitstream and download the *.elf* file of the corresponding SDK project.
- You may begin the host code execution by executing the following command in the xsct terminal

  ```
  con
  ```

  Now you may observe the terminal output.