FGPU Tutorial

version 1.0

Muhammed Al Kadi

July 17, 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 the standard FGPU benchmarks in Xilinx SDK.

1 Introduction

Programming an application for FGPU goes through the following steps:

- On the hardware side:
  - The features of the FGPU core are to configured first in RTL (a *.vhd* file), e.g. # of CUs or supported floating-point instructions.
  - Then, a script implements FGPU 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 software side:
  - An OpenCL kernel is written to run on FGPU. A script compiles it and generates some C files (*.h* and *.c*) to be included in an SDK project.
  - A C/C++ host application is written to run on the ARM Cortex-A9. It should interface and control the FGPU processor.

There are 3 versions of FGPU:

1. *V1* has two clock domains. It is less scalable than the others 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 version.
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 FGPU Github 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 Software Requirements

Before you begin with this tutorial, make sure you have the following software installed on your computer:

- *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 installed as well.
- *ModelSim or QuestaSim:* only needed for RTL simulations.
- *A terminal emulator:* to communicate with the host application during execution, e.g. *minicom*.

3 Tool Setup

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

3.1 Download the FGPU Repository

If you have *git* installed, you may begin with checking out the FGPU repository from Github:

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

Otherwise, you can download and extract it manually to any local directory. Any paths given in the rest of this tutorial are relative your 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 for Xilinx SDK

Before running any application located in the directory `benchmark`, you need to:

1. Create the HW and BSP projects by executing `scripts/create_sdk_bsps.sh`. These projects will be created in hidden directories, e.g. *.FGPU_V2_bsp*.
2. Create and compile all FGPU benchmarks by executing `scripts/create_sdk_project.sh`

Moreover, the `benchmark` folder includes two non-FGPU-relevant folders:

1. `benchmark/MicroBlaze` includes an SDK project that can be used to execute any of the FGPU benchmarks on a MicroBlaze processor. The MicroBlaze runs on 180MHz and it has been optimized for best performance.
2. `benchmark/HLS` includes some of the FGPU benchmarks but programmed for Vivado HLS.
3. `benchmark/power_measurement` is an ARM C application that should run on the second ARM core. It measures the power consumption during execution.

4 HW Synthesis

- Configure your FGPU HW parameters in `RTL/FGPU_definitions.vhd`
- Set the target clock frequency in `scripts/tcl/implement_FGPU.tcl`. You may also modify the synthesis and implementation settings and the number of maximum number of threads used by Vivado in this file. FGPU can be synthesized at frequencies up to 250MHz.
- Run the implementation script `scripts/tcl/implement_FGPU.tcl`. At the beginning, it calls `scripts/tcl/create_FGPU_block_design.tcl` which creates a block diagram that connects FGPU to a clock generator and to the ARM processor[1]. It uses the four available AXI4-HP interfaces to connect FGPU to the external SDRAM memory. 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. The generated bitstream and *.hdf* files are stored in `HW/outputs`.

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

5 Write and Compile the Device Code

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

6 Write and Compile the Host Code

- Write your C/C++ application in a `benchmark/<benchmark>/src/`. It should run on the first ARM core and it has to control FGPU. Please have a look at other benchmarks, e.g. *xcorr*, and try to use the existing methods. Do not forget to copy the compiled device code from `kernels/outputs/<benchmark>.h,.c` next to your C/C++ files. It will be downloaded to FGPU by the host application.
- You may compile your code by executing `scripts/compile.sh`. Alternatively, you may execute `make` from `benchmark/<benchmark>/Release`. If you would like to use the GUI of Xilinx SDK, you should set the workspace to `benchmark`.

7 Download and Run

Unless you want to use the Xilinx SDK in GUI mode, you can run FGPU applications as follows:

- Set the bitstream you want to download in `benchmraks/program_bitstream.tcl`
- From the `benchmark` directory, open the SDK project you have 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 (Xilinx Software Command-Line Tool) [2]
  `source program_bitstream.tcl`
  This will program the desired bitstream .
- Download the *.elf* file of the corresponding SDK project by executing
  `source download_elf.tcl`
  Now you may observe the terminal output.
- If you would like to measure the power consumption during execution you have to:

[2] For further information about xsct, have a look at UG1208 (v2016.2)

– Activate the power measurement in the *main.cpp* file of your application as follows

```
const unsigned sync_power_measurement = 1;
```

This will enable the communication between the host application running on the first ARM core and the power measurement running on the second one. Please pay attention that a single power measurement takes about 1.7ms to finish. Therefore, it does not make sense to compute small problems if a power measurement is being done.

– Recompile your host code.

– If you are using the GUI, you may download and execute the power measurement program before you do the same with the host application. Otherwise, it suffices to execute:

```
source measure_power.tcl
```