DPR Utility User Guide

Abbreviations:

DPR	Dynamic Partial Reconfiguration
RM	Reconfigurable Module
RP	Reconfigurable Partition
MM	Main Module

Introduction:

The DPR Utility allows the developers to convert their static design into a reconfigurable one. The DPR Utility has a simple GUI interface to facilitate the users work to apply the DPR technique.

System requirements:

1. Xilinx SDx v.2017.2, v.2017.3, v.2017.4, v.2018.1, v.2018.2 and v.2018.3
2. TCL/Tk version 8.5, Python version 3.6.5 (with tkinker library)
3. Operating Systems: Linux Ubuntu 16.04 LTS and 18.04 LTS, Windows 10

Prerequisite:

1. SDx is installed in the default directory
    a. Linux: /opt/Xilinx/SDx
    b. Windows: C:\Xilinx\SDx
2. Set your environmental variables in Linux or Windows to the Xilinx installation directory
3. In the SDx project, software modules are selected as hardware accelerated functions on the FPGA [1,2].
4. The design is built by the previous stated versions of SDx [1,2].

Supported Platforms:

1. Tulipp platform Z7030
2. Tulipp platform Ultrascale+ ZU3EG
3. Zedboard

<u>How to use:</u>

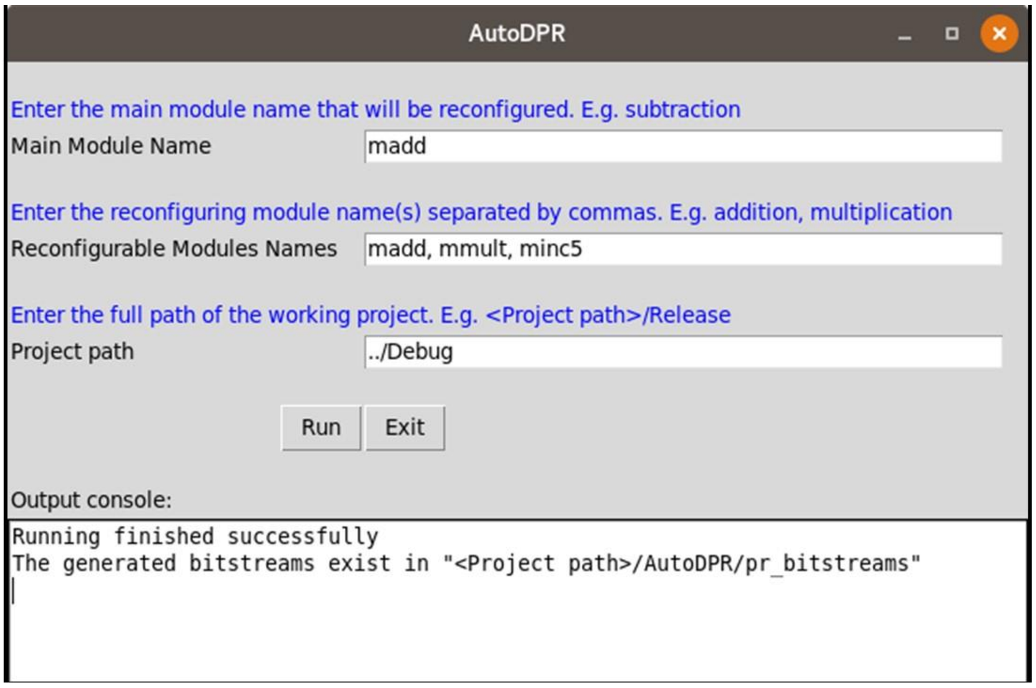
1. First put the AutoDPR in any folder. Developer can place it in the same working directory - beside the Debug or Release folder - to organize their work if they have different examples.
2. Using the executables (Linux / Windows OS)
   a. Open the terminal.
   b. In the terminal write "vivado -version" to check that the SDx tools are set correctly in the environmental variables
   c. $cd workDirectory/autodpr
   d. $./autodpr

   The terminal will show the progress of the tool

3. Using src files
   a. Open the terminal
   b. In the terminal write "vivado -version" to check that the SDx tools are set correctly in the environmental variables
   c. $cd workDirectory/autodpr
   d. $python3 autodpr.py

   The terminal will show the progress of the tool

4. The GUI has fields to be filled by the user as shown in Figure 1.



Enter the main module name that will be reconfigured. E.g. subtraction
Main Module Name: madd

Enter the reconfiguring module name(s) separated by commas. E.g. addition, multiplication
Reconfigurable Modules Names: madd, mmult, minc5

Enter the full path of the working project. E.g. <Project path>/Release
Project path: ../Debug

Output console:

```
Running finished successfully
The generated bitstreams exist in "<Project path>/AutoDPR/pr_bitstreams"
```

Figure 1 DPR tool user interface

The first field (Main Module Name) (MM):

Insert the main module name that is required to be dynamically reconfigured. This function name must be the same as the one used in the C/C++ code within the SDSoC tool.

The second field (Reconfigurable Module Names) (RM):

In this field, insert the other functions that will substitute the main module during the lifetime execution of the application.

The inserted functions must have the same names as used in the C/C++ code. If there is more than one function, they must be separated by a comma ",". For example, "Sepia, Amaro, Lark".

The third field (Project path):

In this field, the user inserts the full path of his/her SDSoC build project. If the developer builds his/her project targeting Debug mode, then he/she must point the project path to the Debug folder. If the developer uses the Release as a build target, then the project path will have the Release folder. For example, "/home/Workspace/tulipp_demo_3/Debug" or "/home/Workspace/tulipp_demo_3/Release".

The developer is free to make a custom build and name it. In this case, the project path will contain the new name, for example "/home/Workspace/tulipp_demo_3/<USER_DEFINED_BUILD>".
Defining a custom build is not within the scope of this document.

<u>The output directory:</u>

The generated bit-stream files will be located in the <build_directory>/AutoDPR/pr_bitstream directory. The directory is structured as follows:

1. Main configuration is named as follows

MM.bit: full design bit-stream including the main module as a default configuration.

MM_pblock_MM_partial.bit: MM partial bit file.

MM_MM.bin: partial bin file that can be loaded from the SD card with the PCAP.

2. RM configurations are named as follows (It applies to all RMs in the second field):

RM.bit: full design bit-stream with reconfigurable module as a default configuration.

RM_pblock_MM_partial.bit: RM partial bit file.

RM_MM.bin: partial bin file that can be loaded from the SD card with the PCAP.

3. Blank design files are named as follows:

blank.bit: full design bit-stream with blank as a default configuration.

blank_pblock_MM_partial.bit: blank partial bit file

blank_MM.bin: partial bin file that can be loaded from the SD card with the PCAP.

<u>Important notes:</u>

1. The generated bit files are used with a baremetal system on Zynq devices stated previously.
2. After the bin file generation copy the files to the SD card.
3. Rename the partial bin files according to the names that are used in the code.
4. The signatures (i.e., arguments and return values) of all functions selected as reconfigurable modules must be identical.
5. In the Zynq Z700: the PCW_GPIO_EMIO_GPIO_IO is assigned to the decoupler, which is used to decouple the outputs of the MM.
   In the Zynq Ultrascale+: the PSU__GPIO_EMIO is assigned to the decoupler.
   Accordingly, it must be set in the software project in the SDSoC and called in the proper location, for the example shown in Figure 2:
   a. Set decouple signal to 1 to decouple the outputs.
   b. Reconfigure the RM.
   c. Set decouple signal to 0 to allow the normal flow of the design.

```
XGpioPs_WritePin(&psGpioInstancePtr,iPinNumberEMIO_START, 1);
XDcfg_TransferBitfile(DcfgInstPtr, PartialCfg, PartialAddress,
PARTIAL_Config_One_LEN);
XGpioPs_WritePin(&psGpioInstancePtr,iPinNumberEMIO_START, 0);
```

Figure 2 Decouple code

Limitations:

1. Working with Xilinx tools only.
2. The tool has default floorplanning for the stated Xilinx chips that are used in the Tulipp project. If another chip is used, developers can manually edit the autodpr.tcl file and add a custom floorplanning (where they need the design to be implemented on the Chip). As shown in Figure 3.

```
if {$chip == "xc7z020clg484-1"} {
   puts "Floorplanning for Zedboard          :$chip              "
   set floorplan "SLICE_X0Y0:SLICE_X113Y49 DSP48_X0Y0:DSP48_X4Y19
   RAMB18_X0Y0:RAMB18_X5Y19 RAMB36_X0Y0:RAMB36_X5Y9"}
```

Figure 3 add custom floorplanning

References:

1. SDSoC Environment User Guide (UG1027)
2. SDSoC Environment Tutorial: Introduction (UG1028)