Application Note

Akademie věd České republiky
Ústav teorie informace a automatizace AV ČR, v.v.i.

TE0950 Versal™ to Artix™ Communication

Lukas Kohout
kohoutl@utia.cas.cz

Revision history

Rev.	Date	Author	Description
0	19.02.2025	L.K.	Draft
1	24.02.2025	L.K.	Acknowledgement, DTSI overlay note
2			

Content

Acknowledgement

Under grant 101096884, Listen2Future is co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or Chips Joint Undertaking. Neither the European Union nor the granting authority can be held responsible for them. The project is supported by the CHIPS JU and its members (including top-up funding by Austria, Belgium, Czech Republic, Germany, Netherlands, Norway and Spain).

National Funding

This project has also received national funding from the Ministry of Education, Youth and Sports of the Czech Republic (MEYS) under grant agreement No 9A22004.

1 Introduction

The TE0950 board [1] from Trenz Electronic has two FPGA devices, AMD Versal™ AI Edge XCVE2302 in role of the main FPGA of the system and AMD Artix™ XC7A35T FPGA as configurable levelshifter/MUX for FMC and other 3.3 V IOs. These two FPGAs have dedicated up to 14 differential pairs to communicate with. This application note describes how to program the FPGAs and how to transfer AXI4 transaction between Versal™ and Artix™.The procedure described in this text is for a reference design from Trenz Electronic for 2023.2.2 tools.

2 Description

To establish a communication between two FPGA devices there is used the Chip2Chip IP core [2] which is included in standard AMD Vivado installation (tested in 2023.2.2 version). A simplified block diagram of the system design is shown in Figure 1. The core is configured to transfer full AXI4 transactions including bursts. In case of AXI-Lite transactions, they are mapped into the full AXI4 transactions. In both FPGAs the Chip2Chip core has to be configured almost identically, the only difference is in the *Chip2Chip mode* choice. Figure 2

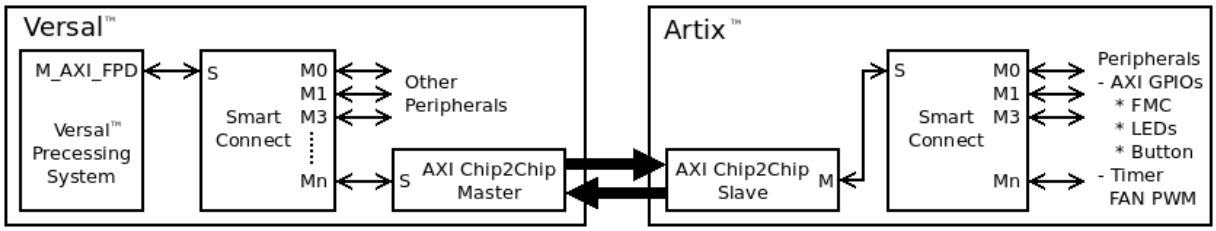

Figure 1: Chip to chip communication.

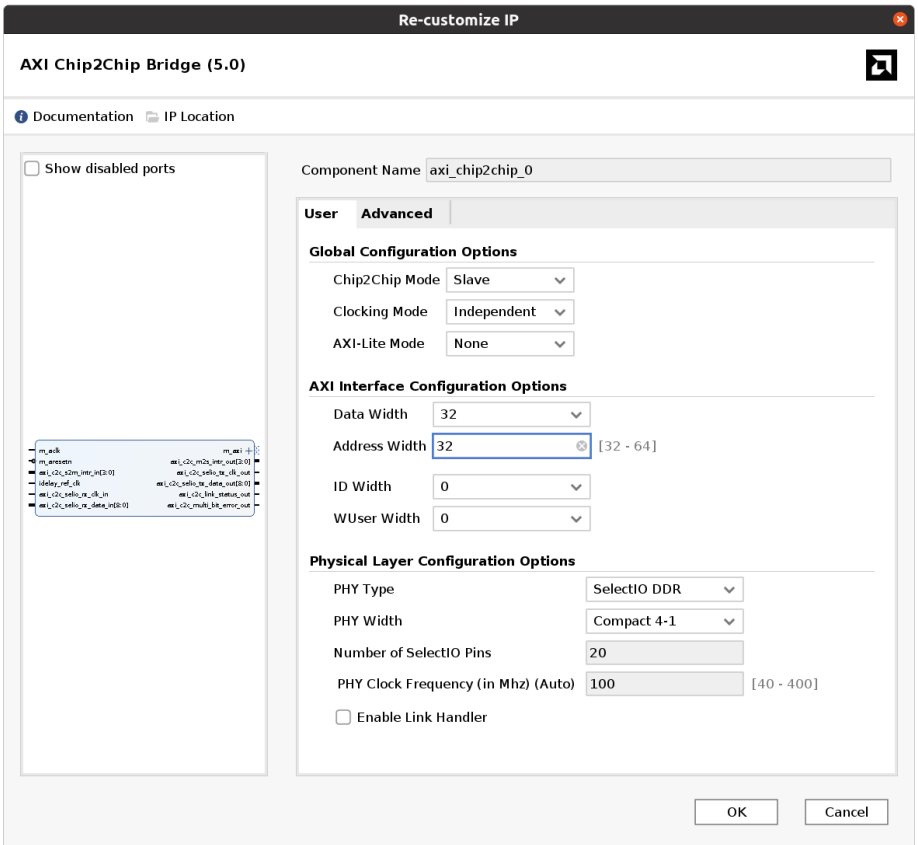

Figure 2: AXI Chip2Chip IP core configuration for Artix™ (slave).

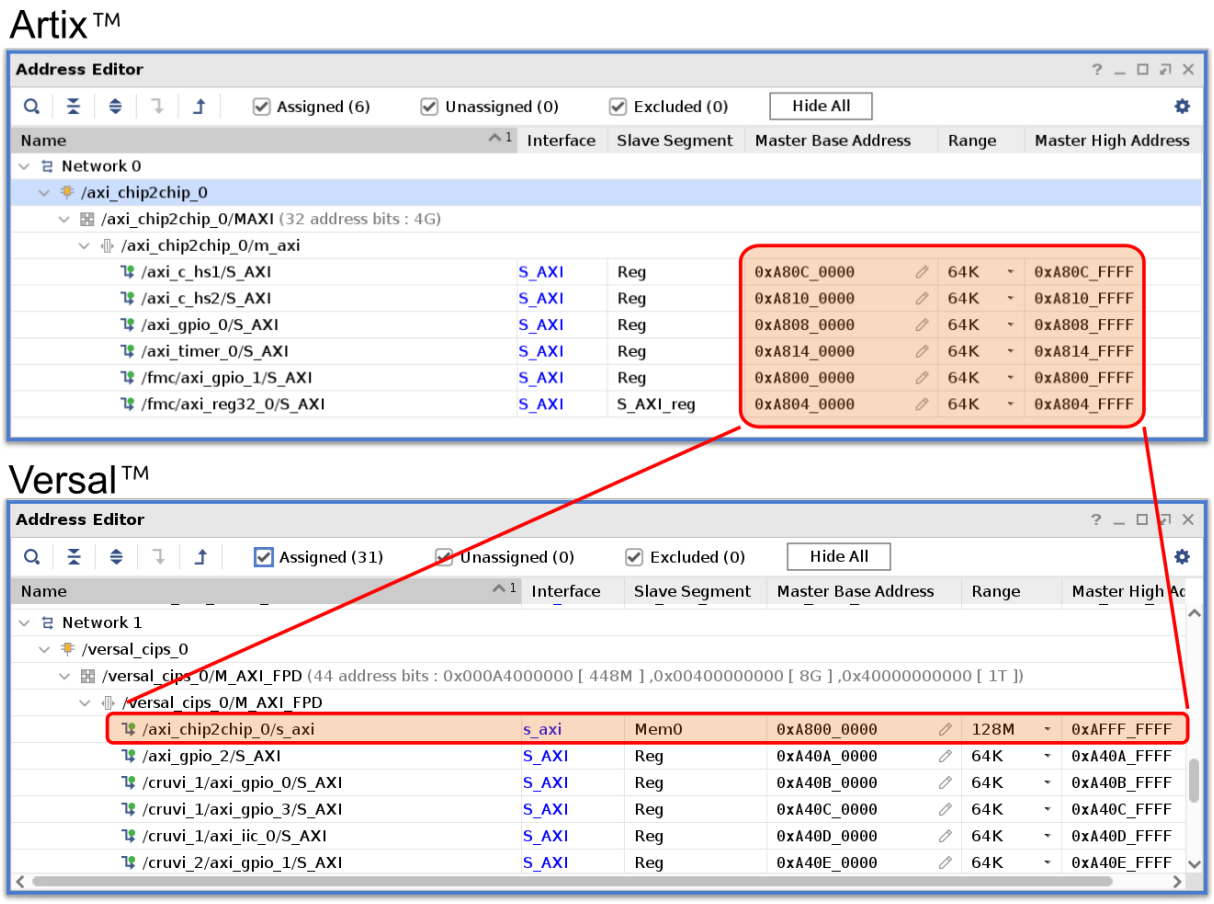

Figure 3: Versal™ and Artix™ address spaces mapping.

shows the Chip2Chip IP core configuration used in Artix™ device. This particular configuration allows a maximum theoretical throughput of 1525 Mb/s in one direction. Realistically, it is necessary to calculate from 75% of this value, i.e. 1144 Mb/s. This value is perfectly sufficient for AXI-Lite transactions, but for larger data streams, e.g. video, it can be limiting.

It is also necessary to have the address spaces set correctly. This means that on the Versal™ side, the Chip2Chip core must have an address range set such that it includes all the addresses of all the IP cores in Artix™ that we want to communicate with. The example is shown in Figure 3.

3 Quick Startup

The procedure was tested in AMD tools in version 2023.2.2 running on native Ubuntu 20.04.6 LTS Linux distribution. It is assumed that the AMD Vivado, Vitis and PetaLinux tools are already installed. The example described here builds on the reference design for the TE0950 board. It is based on the reference design, which contains two "independent" designs, one for Versal™ and another one for Artix™. The reference design packages are available for download at TE0950 web page:

- Artix™: https://shop.trenz-electronic.de/trenzdownloads/Trenz_Electronic/Development_Boards/TE0950/Reference_Design/2023.2/test_board/TE0950-test_board_artix-vivado_2023.2-build_4_20240531084104.zip

- Versal™:
  https://shop.trenz-electronic.de/trenzdownloads/Trenz_Electronic/Development_Boards/TE0950/Reference_Design/2023.2/test_board/TE0950-test_board-vivado_2023.2-build_4_20240531092954.zip

All the steps described here are reported and tested for the board TE0950-03-EGBE21A equipped with the original Trenz Electronic cooler, which has a four pin connector allowing PWM regulation of the FAN [3].

3.1 Artix™ HW Compilation

1. Unpack downloaded package:
   TE0950-test_board_artix-vivado_2023.2-build_4_20240531084104.zip
   It creates the *test_board_artix* folder.
2. Go to the *test_board_artix* folder and set all necessary scripts executable. From the command line execute:
   ```
   cd test_board_artix
   chmod +x _create_linux_setup.sh console/base_sh/*.sh
   ```
3. Create Vivado design, from the command line execute:
   ```
   ./_create_linux_setup.sh
   ```
   Select: **Module selection guide, project creation...**
   ```
   0
   ```
   Select: **TE0950-03-EGBE21A**
   ```
   5
   ```
   Confirm the selected board
   ```
   y
   ```
   Select: **Create vivado project**
   ```
   1
   ```
   It creates a Vivado project and open it.
4. Compile the design, from the Vivado TCL console execute:
   ```
   TE::hw_build_design -export_prebuilt
   ```
   When the compilation is finished, Vivado opens the *Bitstram Generation Completed* dialog. Select **Open implemented Design** and click **OK**. The resultant bitstream is in folder *test_board_artix/prebuilt/hardware/35_2c_03*.
5. Convert the bitstream to a format suitable for QSPI Flash memory from which the Artix™ FPGA boots automatically on power on. From the Vivado TCL console execute:
   ```
   TE::EXT::generate_app_bit_mcs
   ```
   It creates file *test_board_artix/prebuilt/boot_images/35_2c_03/fpga/fpga.mcs*.
6. Connect the JTAG/UART on the TE0950 board (micro-USB connector J2) to the PC.
7. Switch the board to use JTAG, set switch S2.1 = 0, S2.2 = 0 and S2.3 = 0.
8. Set the JTAG chain to include Artix™ (its QSPI Flash memory). Set switch S4.1 = 1, S4.2 = 0 and S4.3 = 1.
9. Power the board ON.

10. Program the Artix™ QSPI Flash memory, from the Vivado TCL console execute:

```
TE::pr_program_flash -swapp fpga
```

11. Close Vivado.
12. Power the board OFF.

3.2 Versal™ HW Compilation

1. Unpack downloaded package:

   TE0950-test_board-vivado_2023.2-build_4_20240531092954.zip

   It creates the *test_board* folder.

2. Go to the *test_board* folder and set all necessary scripts executable. From the command line execute:

```
cd test_board
chmod +x _create_linux_setup.sh console/base_sh/*.sh
```

3. Create Vivado design, from the command line execute:

```
./_create_linux_setup.sh
```

   Select: **Module selection guide, project creation...**

```
0
```

   Select: **TE0950-03-EGBE21A**

```
3
```

   Confirm the selected board

```
y
```

   Select: **Create vivado project**

```
1
```

   It creates a Vivado project and open it.

4. Compile the design, from the Vivado TCL console execute:

```
TE::hw_build_design -export_prebuilt
```

   When the compilation is finished, Vivado opens the *Bitstram Generation Completed* dialog. Select **Open implemented Design** and click **OK**.

5. Create the *BOOT.bin* file, from the Vivado TCL console execute:

```
TE::sw_run_vitis -all
```

   It compiles all necessary components and wraps them together into one booting file *test_board/prebuilt/boot_images/23_1lse_8gb/u-boot/BOOT.bin* and open Vitis. Close Vitis.

6. Close Vivado.

3.3 Petalinux Compilation

1. Extend a device tree for Versal™ with Artix™ peripheral definition.
   a. Modify DTSI file with Artix™ extensions, edit file:

      test_board/os/petalinux/project-spec/meta-user/recipes-bsp/device-tree/files/artix-overlay.dtsi

- Remove first two lines defining the DTSI file as an overlay plugin, remove these lines:

```
/dts-v1/;
/plugin/;
```

NOTE: This petalinux project has another "overlay" DTSI files, they are needed for different cameras connected via MIPI SCI2 interface on the TE0950 board. For the purposes of this application note, they are not required and we will leave them as they are without touching them.

- Add *artix_usr_gpio* blob into the *amba_pl* section

```
artix_usr_gpio: gpio@a8080000 {
  #gpio-cells = <2>;
  clock-names = "s_axi_aclk";
  clocks = <&artix_clk 0>;
  compatible = "xlnx,axi-gpio-2.0", "xlnx,xps-gpio-1.00.a";
  gpio-controller ;
  reg = <0x0 0xa8080000 0x0 0x10000>;
  xlnx,all-inputs = <0x0>;
  xlnx,all-inputs-2 = <0x0>;
  xlnx,all-outputs = <0x0>;
  xlnx,all-outputs-2 = <0x0>;
  xlnx,dout-default = <0x00000000>;
  xlnx,dout-default-2 = <0x00000000>;
  xlnx,gpio-width = <0x3>;
  xlnx,gpio2-width = <0x20>;
  xlnx,interrupt-present = <0x0>;
  xlnx,is-dual = <0x0>;
  xlnx,tri-default = <0xFFFFFFFF>;
  xlnx,tri-default-2 = <0xFFFFFFFF>;
};
```

- Add *artix_hs1_gpio* blob into the *amba_pl* section

```
artix_hs1_gpio: gpio@a80c0000 {
  #gpio-cells = <2>;
  clock-names = "s_axi_aclk";
  clocks = <&artix_clk 0>;
  compatible = "xlnx,axi-gpio-2.0", "xlnx,xps-gpio-1.00.a";
  gpio-controller ;
  reg = <0x0 0xa80c0000 0x0 0x10000>;
  xlnx,all-inputs = <0x0>;
  xlnx,all-inputs-2 = <0x0>;
  xlnx,all-outputs = <0x0>;
  xlnx,all-outputs-2 = <0x0>;
  xlnx,dout-default = <0x00000000>;
  xlnx,dout-default-2 = <0x00000000>;
  xlnx,gpio-width = <0x2>;
  xlnx,gpio2-width = <0x20>;
  xlnx,interrupt-present = <0x0>;
  xlnx,is-dual = <0x0>;
  xlnx,tri-default = <0xFFFFFFFF>;
  xlnx,tri-default-2 = <0xFFFFFFFF>;
};
```

- Add *artix_hs2_gpio* blob into the *amba_pl* section

```
artix_hs2_gpio: gpio@a8100000 {
  #gpio-cells = <2>;
  clock-names = "s_axi_aclk";
  clocks = <&artix_clk 0>;
```

```
  compatible = "xlnx,axi-gpio-2.0", "xlnx,xps-gpio-1.00.a";
  gpio-controller ;
  reg = <0x0 0xa8100000 0x0 0x10000>;
  xlnx,all-inputs = <0x0>;
  xlnx,all-inputs-2 = <0x0>;
  xlnx,all-outputs = <0x0>;
  xlnx,all-outputs-2 = <0x0>;
  xlnx,dout-default = <0x00000000>;
  xlnx,dout-default-2 = <0x00000000>;
  xlnx,gpio-width = <0x2>;
  xlnx,gpio2-width = <0x20>;
  xlnx,interrupt-present = <0x0>;
  xlnx,is-dual = <0x0>;
  xlnx,tri-default = <0xFFFFFFFF>;
  xlnx,tri-default-2 = <0xFFFFFFFF>;
};
```

- Define *artix_usr_gpio* pin names, add *artix_usr_gpio* blob into the *root* section of the DTSI file (outside the *pl_amba* section).

```
&artix_usr_gpio {
  gpio-line-names = "A_USR_LED3", "A_USR_LED2", "A_USR_DIPSW_S5_4";
};
```

Already modified *artix-overlay.dtsi* file can be found in the attached package.

b. Include the *artix-overlay.dtsi* file into the main custom DTSI file, edit:

test_board/os/petalinux/project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dsti

Insert *include* section on the second line:

```
/include/ "artix-overlay.dtsi"
```

Already modified *system-user.dtsi* file can be found in the attached package.

c. Include the *artix-overlay.dtsi* file in the compilation procedure, edit:

test_board/os/petalinux/project-spec/meta-user/recipes-bsp/device-tree/device-tree.bbappend

Extend the *SRC_URI* string with path of the *artix-overlay.dtsi* file:

```
SRC_URI:append = "file://system-user.dtsi file://mipi_csi2.dtsi
file://artix-overlay.dtsi"
```

Already modified *device-tree.bbappend* file can be found in the attached package.

2. Set the system path to see *petalinux* tools, from the command line execute (modify the path according your working system environment):

```
source /opt/petalinux/2023.2/settings.sh
```

3. Go to the *petalinux* project directory and update the project with current hardware specification (XSA file), from the command line execute:

```
cd test_board/os/petalinux
petalinux-config --get-hw-description=../../vivado
```

There is no need to change anything, close the *menu config* window, choose *Exit*.

4. Build *petalinux*, from the command line execute:

```
petalinux-build
```

3.4 Prepare SD Card and Start the Board

1. Take an empty micro SD card formatted to FAT32 and copy these files on it:
   - *test_board/prebuilt/boot_images/23_1lse_8gb/u-boot/BOOT.bin*
   - *test_board/os/petalinux/images/linux/bl31.elf*
   - *test_board/os/petalinux/images/linux/boot.scr*
   - *test_board/os/petalinux/images/linux/image.ub*
   - *test_board/os/petalinux/images/linux/system.dtb*
   - *test_board/os/petalinux/images/linux/u-boot-dtb.elf*

   NOTE: In case you want to use MIPI CSI2 interface to connect camera, copy complete test_board/os/petalinux/images/linux/dtbos folder to the micro SD card, too. This folder contains compiled overlay device tree files that correspond to currentrly supported cameras. This is optional and not needed for this application note.
2. On the micro SD card rename file *u-boot-dtb.elf* to *u-boot.elf*.
3. Insert the micro SD card into the TE0950 card reader slot J4.
4. Switch the board to boot from micro SD card, set switch S2.1 = 0, S2.2 = 1 and S2.3 = 0.
5. Power the board ON.
6. Start serial terminal, *putty* for instance. The terminal settings are:
   - Baud rate: 115200
   - Data bits: 8
   - Stop bit: 1
   - Parity: none
   - Flow control: none
7. Work with serial terminal.

4 Examples

This section describes two examples that show how to interact with peripherals implemented in the Artix™ FPGA with the Versal™ processing system. The first example shows how to use an AXI Timer to control the FAN speed with PWM. The second one, it describes controlling user LEDs via AXI GPIO IP core.

4.1 FAN Control

If the TE0950 board is equipped with the cooler that has a FAN with four pin connector allowing PWM regulation, the FAN speed can be controlled with the timer which is in the Artix™ FPGA. The PWM value can be controlled by reading/writing of the system file */sys/class/hwmon/hwmon0/pwm1*. This value is in the range 0 to 255. When the board starts the PWM value is set to its maximum 255. To read the current value, execute from the serial terminal this command:

```
cat /sys/class/hwmon/hwmon0/pwm1
```

To set a new value of the PWM, 100 for example, execute from the serial terminal:

```
echo 100 > /sys/class/hwmon/hwmon0/pwm1
```

To determine the current system temperature for appropriate fan speed control, read file */sys/bus/iio/devices/iio:device0/in_temp160_temp_input*, from the serial terminal execute:

```
cat /sys/bus/iio/devices/iio:device0/in_temp160_temp_input
```

In the attached package there is an example that shows how to control the FAN speed according to current system temperature automatically. It is a simple python script which

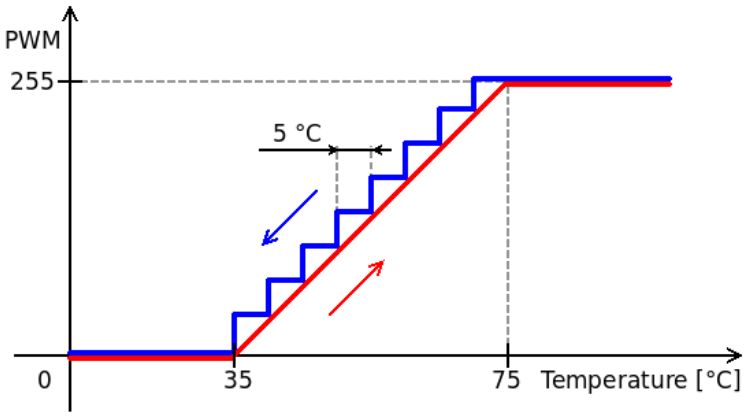

Figure 4: FAN speed control according to current system temperature.

reads the current system temperature every second and then sets the FAN PWM value accordingly. The minimal PWM value 0 corresponds to 35 °C, the maximum PWM value 255 is reached at a temperature of 75 °C. The PWM value increases linearly with increasing temperature. As the temperature decreases, the PWM value decreases by a hysteresis that corresponds to 5 °C steps. This behavior is illustrated in Figure 4. To see the details, you can examine the script located in the attached package *fancontrol/fancontrol.py*. To start this example automatically on each board startup, copy these files from the package folder *fancontrol*:

- *fancontrol.py*
- *init.sh*

directly to the root of the micro SD card.

4.2 GPIO LEDs and Switch

In the whole system there are many GPIO peripherals. Some of these are found in the Versal™ processing system part, others are located in the Versal™ programmable logic, and still others are in the Artix™ programmable logic. To identify all GPIO peripherals use *gpiodetect* command from the serial terminal:

```
gpiodetect

gpiochip0 [a8000000.gpio] (34 lines)
gpiochip1 [a8080000.gpio] (3 lines)
gpiochip10 [a4120000.gpio] (5 lines)
gpiochip11 [versal_gpio] (58 lines)
gpiochip12 [pmc_gpio] (116 lines)
gpiochip2 [a80c0000.gpio] (2 lines)
gpiochip3 [a8100000.gpio] (2 lines)
gpiochip4 [a40a0000.gpio] (2 lines)
gpiochip5 [a40b0000.gpio] (8 lines)
gpiochip6 [a40c0000.gpio] (12 lines)
gpiochip7 [a40e0000.gpio] (8 lines)
```

```
gpiochip8 [a40f0000.gpio] (8 lines)
gpiochip9 [a4110000.gpio] (2 lines)
```

Those GPIOs whose address begins with *"a8"* are located in the Artix™ FPGA. The GPIO peripheral whose address is 0xA80800000 is the one that controls two user LEDs and one user DIP switch. From the *gpiodetect* report, it can be seen that it is named as *gpiochip1*. The *gpiochip1* details can be obtained by executing *gpioinfo* command from the serial terminal:

```
gpioinfo gpiochip1

gpiochip1 - 3 lines:
        line   0: "A_USR_LED3"           unused   input  active-high
        line   1: "A_USR_LED2"           unused   input  active-high
        line   2: "A_USR_DIPSW_S5_4"     unused   input  active-high
```

It is seen that the GPIO lines have not been set yet. The connected LEDs and switch are active low. To light up the LED connected to the line 0 of the *gpiochip1*, use command *gpioset* as follows:

```
gpioset gpiochip1 0=0
```

Observe the TE0950 board, the user LED3 should start to light up. To turn the LED3 OFF use *gpioset* command again but with different argument:

```
gpioset gpiochip1 0=1
```

The user LED2 can be controlled similarly. To get the current value of the user DIP switch S5.4, use command *gpioget*:

```
gpioget gpiochip1 2
```

Toggle the switch and repeat the *gpioget* command to see the different result.

5 Package content

```
.
├── fancontrol
│   ├── fancontrol.py
│   └── init.sh
├── TE0950-Versal-Artix-communication-appnote-v1.pdf
└── test_board
    └── os
        └── petalinux
            └── project-spec
                └── meta-user
                    └── recipes-bsp
                        └── device-tree
                            ├── device-tree.bbappend
                            └── files
                                ├── artix-overlay.dtsi
                                └── system-user.dtsi
```

6 References

[1] Trenz Electronic, „AMD Versal AI Edge Evalboard with VE2302 device, 8 GB DDR4 SDRAM, 15 x12 cm,“ [Online]. Available: https://shop.trenz-electronic.de/en/TE0950-03-EGBE21C-AMD-Versal-AI-Edge-Evalboard-with-VE2302-device-8-GB-DDR4-SDRAM-15-x12-cm. [2025].

[2] AMD, „AXI Chip2Chip v5.0 LogiCORE IP Product Guide,“ [Online]. Available: https://docs.amd.com/r/en-US/pg067-axi-chip2chip/AXI-Chip2Chip-v5.0-LogiCORE-IP-Product-Guide. [2025].

[3] Trenz Electronic, „CoolJag BUF-A4 Fansink for Trenz Electronic Evalboard TE0950,“ [Online]. Available: https://shop.trenz-electronic.de/en/34097-CoolJag-BUF-A4-Fansink-for-Trenz-Electronic-Evalboard-TE0950. [2025].