How to program a FPGA, different approaches

1. Introduction

This guide introduces two practical methods to program an FPGA, with the goal of teaching readers different workflows. On the **Genesys2** board, a **CVA6 RV32** core will be programmed using **WSL and command-line tools**, while on the **Arty A7-100T** board, a **NEORV32** core will be programmed through the **Xilinx Vivado GUI**.

By following these steps, readers will learn how to configure FPGAs using both a script-based and a graphical approach, gaining hands-on experience with open-source RISC-V cores.

2. Prerequisites

Hardware:

- Genesys2 board
- Arty A7 board
- JTAG/USB wire

Software:

- Xilinx Vivado webpackage
- Windows Subsystem for Linux (WSL)
- OpenFPGALoader on WSL
- Toolchain GNU RISC-V
- Putty

3. Flash genesys2 board

First, add the next rule to your Ubuntu based system. In the file /etc/udev/rules.d/99-ftdi.rules. It file probably doesn´t exist, you have to create it.

SUBSYSTEM=="usb", ACTION=="add", ATTRS{idProduct}=="6010", ATTRS{idVendor}=="0403", MODE="664", GROUP="plugdev"

You must have openfpgaloader(apt install openfpgaloader)

Then connect your FPGA to your PC through JTAG.

-- If you are using WSL, you have to share and attach the USB device(your FPGA) to WSL system, remember that if you want to see the output of your test you have to connect a USB in UART port and do the same with the UART port, you can check on WSL if the device is attached with command $lsusb. On PowerShell: $usbipd list; $usbipd bind --busid <busid>; $usbipd attach --wsl --busid <busid> --

With the following command you can flash the flash memory and/or load code on SRAM memory.

openFPGALoader -b genesys2 bitstream.bit # Loading in SRAM

openFPGALoader -b genesys2 -f bitstream.bit # Writing in flash

- Run FPGA:

Install openocd in your system:

You'll also need:

- make
- libtool
- pkg-config >= 0.23 or pkgconf
- libjim >= 0.79

$ sudo apt update

$ sudo apt install -y make libtool pkg-config libjim-dev

git clone https://github.com/openocd-org/openocd.git

cd openocd

mkdir build

./bootstrap

./configure --enable-ftdi --prefix=build --exec-prefix=build

make -j$(nproc)

sudo make install

export PATH=$PATH:<direction>/build/bin -- Change the direction for your correct direction --

To initialize the FPGA in a terminal run:

openocd -f corev_apu/fpga/ariane.cfg

- Debug and UART connection

Debug is possible to do by

$ riscv32-unknown-elf-gdb program.elf.

Once you are into gdb terminal you must connect to the remote target

target remote localhost:3333 -- the port 3333 was setted like that on ariane.cfg file –

Then, load the program and run it

load

c – continue --

UART output is possible to do by

$ sudo apt-get install minicom

$ minicom -D /dev/<your-port> -b <baudrate>

If you need to install the toolchain GNU RISC-V, there are two ways to do it.

First, simple but less configurable:

sudo apt update

sudo apt install -y gcc-riscv32-unknown-elf gdb-multiarch

Second, from original repository:

sudo apt update

sudo apt install -y autoconf automake autotools-dev curl python3 \

libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison flex \

texinfo gperf libtool patchutils bc zlib1g-dev git libexpat1-dev

git clone https://github.com/riscv/riscv-gnu-toolchain

cd riscv-gnu-toolchain

./configure --prefix=/opt/riscv --with-arch=rv32imac --with-abi=ilp32 (arch and abi are customizable fitting your preferences or needs)

make newlib -j$(nproc)

Set the following global variable to make your system find the toolchain

export RISCV=/opt/riscv (path to the riscv tool{32,64})

export PATH=$RISCV/bin:$PATH

export CROSS_COMPILE=riscv{32,64}-unknown-elf-

export CC=${CROSS_COMPILE}gcc

export OBJCOPY=${CROSS_COMPILE}objcopy

4. Flash Arty A7-100T board

1. Prepare your environment

- Install board files
- Make sure **Xilinx Vivado** is installed on your system.
- Connect the **Arty A7-100T board** to your PC via **USB-JTAG**.

Board files installation

Download the most recent Master Branch ZIP Archive.

Unzip the file and add all the folder what are into new/board_files into your vivado path, usually:
C:/Xilinx/Vivado/<version>/data/boards/board_files

If board_files doesn't exist, create it and copy there all the folder you copy before from new/board_files.

2. Open the Project and Prepare the FPGA

- Launch **Vivado** and open the existing project (.xpr) containing the **NEORV32 RV32** design.
- Make sure the project has been **synthesized and implemented**.
  - If the bitstream has not been generated yet, click **Generate Bitstream** in the Flow Navigator.
  - This will compile the design and produce the .bit file needed to program the FPGA.
- Program the FPGA
  - After generating the bitstream, open **Hardware Manager → Open Target → Auto Connect**.
  - Vivado should detect the **Arty A7-100T** connected via USB-JTAG.
  - Select **Program Device** and choose the corresponding **bitstream** (.bit).
  - Click **Program**. Vivado will load the design into the FPGA and display a progress bar.

- Once programming is complete, verify that the **NEORV32** core is running using UART output.

3. UART Output

Once the FPGA is programmed, you can monitor the NEORV32 output via UART. PuTTY is a common terminal emulator for this purpose.

1. Install PuTTY

- Download PuTTY from the official website: link
- Install it on your Windows PC.
- Launch Putty.

2. Configure PuTTY for UART

- Note the **COM port** assigned by Windows (check in **Device Manager → Ports (COM & LPT)**).
- In the **Session** category:
  - Select **Serial**.
  - Enter the **COM port** (e.g., COM3).
  - Set the **baud rate** to match your design (19200 on this project).