MIC-SEC-2022

Philippe TANGUY

Dec 07, 2022

CONTENTS:

The evaluation of security countermeasures is essential. Experimentation on real use cases and reproducibility are also important. In the field of embedded systems security we often face a technological barrier and we have to master a multitude of software and hardware tools. Moreover, our contributions often target a specific point and therefore we are not necessarily experts of all the components of a system on chip (SoC). The technical task then usually takes a lot of time when creating an experimental test bench. We propose in this practical work to discover some tools allowing to deploy a SoC (with associated software) and to evaluate it on FPGA board.

Note: This hands-on session will try to answer the following question: **How to quickly deploy a SoC on FPGA to evaluate security solutions?**

Warning: Hands-on materials are available in the **Quick start guide** chapter. To track updates you can also clone the Git repository https://git.renater.fr/anonscm/git/mic-sec-2022/mic-sec-2022.git.

CHAPTER

ONE

INTRODUCTION

Title: How to quickly deploy a SoC on FPGA to evaluate security solutions for communicating embedded systems.

Abstract: The evaluation of security countermeasures is essential. Experimentation on real use cases and reproducibility are also important. In the field of embedded systems security we often face a technological barrier and we have to master a multitude of software and hardware tools. Moreover, our contributions often target a specific point and therefore we are not necessarily experts of all the components of a system on chip (SoC). The technical task then usually takes a lot of time when creating an experimental test bench. We propose in this practical work to discover some tools allowing to deploy a SoC (with associated software) and to evaluate it on a FPGA board.

CHAPTER

TWO

QUICK START GUIDE

This chapter explains how to launch the virtual environmment. If you want to start to build your environment from scratch you can read the *Virtual environment* chapter.

2.1 Prerequisites

We prepared a virtual environnement for the hands-on session. So if you want to use it in order to not build your environment from scratch you need the following software (Tested on Debian stable but it should work on other OS):

- Virtualbox

We also provide a Vagrant box so if you want to use our Vagrant box you need:

- Vagrant
- Virtualbox

2.2 With the Virtualbox VM

Warning: The virtual machine is available online!

Download it (size 2.5 Go) by following this link: https://filesender.renater.fr/?s=download&token=39f9a08f-8fb1-42b6-b349-d4e493a41c36

You can check that the download was ok:

```
$ sha256sum --check mic-sec-2022-litex.ova.sha256sum
```

Then you need Virtualbox

```
$ vboxmanage -v
6.1.32_Debianr149290
```

To get started, you can start Virtualbox and add the virtual machine.

```
$ vboxmanage import mic-sec-2022-litex.ova
```

Then start the virtual machine in Virtualbox. To access the VM you can use the Virtualbox GUI:

- login: vagrant

- pwd: vagrant

In the virtual environment if you want to change the key map:

```
$ sudo loadkeys fr
```

If you want to access the network you should do in the VBOX GUI for your VM

```
$ sudo ip link set enp0s3
$ sudo dhclient enp0s3
```

If you want to access the network from the VM with SSH you have to allows ssh with password first. So in the VBOX GUI edit the *sshd_config* file and allows the password authentication with *PasswordAuthentication yes*

```
$ sudo vim /etc/ssh/sshd_config
$ sudo systemctl restart sshd.service
```

Finally from your machine try to connect to the VM with SSH:

```
$ ssh -p 2200 vagrant@localhost
```

2.3 With the Vagrant box

Warning: The Vagrant box will be available online soon!

Download it (size 2.5 Go) by following this link: https://filesender.renater.fr/?s=download&token=39f9a08f-8fb1-42b6-b349-d4e493a41c36

You can check that the download was ok:

```
$ sha256sum --check mic-sec-2022-litex.ova.sha256sum
```

Then you need Vagrant

```
$ vagrant -v
Vagrant 2.2.14
$ vboxmanage -v
6.1.32_Debianr149290
```

With a terminal move to the directory

```
$ vagrant init mic-sec-2022 file://path/to/the/file
$ Vagrant up
$ Vagrant ssh
```

2.4 From the repository and Vagrant

Warning: This section is for advanced user! It is not needed for the hands-on session if you already have the Vagrant box file or The Vbox VM file.

Clone the Git repository

```
$ git clone https://git.renater.fr/anonscm/git/mic-sec-2022/mic-sec-2022.git
$ cd mic-sec-2022
$ git submodule init
$ git submodule update
```

Launch the Virtual machine

```
$ cd venv/vagrant-venv
$ vagrant up
$ vagrant ssh
```

To finalize the installation please look at the **Virtual environment** chapter.

CHAPTER

THREE

VIRTUAL ENVIRONNEMENT

Note: This chapter is not mandatory.

If you want to install directly all the tools on your system you can also follow the instructions of this chapter.

The repository contains a directory called *venv* with everything you need to set up a virtual machine for the labs.

3.1 Prerequisites

If you want to start to build your environment from scratch you need the following software (Tested on Debian stable but it should work on other OS):

- Virtualbox
- Vagrant

3.2 How to build the development environment

Note: If you want to build your environnment by yourself followed the instruction below.

In the following we explain how the virtual environnement has been build.

3.2.1 Setup the virtual machine

```
$ vagrant -v
Vagrant 2.2.14
$ vboxmanage -v
6.1.32_Debianr149290
```

Start the virtual environment with Vagrant

```
$ vagrant init ubuntu/focal64
$ Vagrant up
$ Vagrant ssh
```

3.2.2 LiteX

With Litex_setup.py

```
$ pyhton3 -m venv pyenv
$ source pyenv/bin/activate
$ cd hands-on/third_party
$ wget https://raw.githubusercontent.com/enjoy-digital/litex/master/litex_setup.py
$ chmod +x litex_setup.py
$ ./litex_setup.py --tag=2022.04 --init --install --config=full
$ cd litex
$ pip3 install meson ninja
$ wget  https://static.dev.sifive.com/dev-tools/riscv64-unknown-elf-gcc-8.3.0-2019.08.0-
↪x86_64-linux-ubuntu14.tar.gz
$ gunzip riscv64-unknown-elf-gcc-8.3.0-2019.08.0-x86_64-linux-ubuntu14.tar.gz
$ tar xvf ...
$ add to path
$ sudo apt install libevent-dev libjson-c-dev verilator
```

Without Litex_setup.py

In the following we will install LiteX without using litex_setup.py. Indeed it is usefull in a project to add all the dependencies as Git submodule. So in your Git repository you can add LiteX and some usefull elements such as in our case:

```
$ cd hands-on/third_party
$ git submodule add https://github.com/m-labs/migen.git
$ git submodule add https://github.com/enjoy-digital/litex.git
$ cd litex
$ git checkout 2022.08
$ git submodule add https://github.com/litex-hub/litex-boards.git
$ git submodule add https://github.com/enjoy-digital/litedram.git
$ git submodule add https://github.com/enjoy-digital/liteeth.git
$ git submodule add https://github.com/enjoy-digital/litescope.git
$ git submodule add https://github.com/enjoy-digital/liteiclink.git
$ git submodule add https://github.com/litex-hub/pythondata-cpu-vexriscv.git
$ git submodule add https://github.com/litex-hub/pythondata-cpu-cv32e41p.git
$ git submodule add https://github.com/litex-hub/pythondata-software-picolibc.git
$ cd pythondata-software-picolibc
$ git submodule init
$ git submodule update
$ cd ..
$ git submodule add https://github.com/litex-hub/pythondata-software-compiler_rt.git
$ git submodule add https://github.com/litex-hub/pythondata-misc-tapcfg.git
```

Setup env in Python virtual environnemnt and install dependancies:

```
$ python3 -m venv venv/pyenv-litex
$ source venv/pyenv-litex/bin/activate
$ pip3 install -r hands-on/requirements.txt
$ deactivate
```

LiteX depends on other software for simulation and/or FPGA deployment. The following sections detail some of these dependencies.

3.2.3 Verilator

Only needed for simulation

```
$ sudo apt install libevent-dev libjson-c-dev
$ sudo apt get install verilator
```

3.2.4 OpenOCD

Only needed to load the bitstream to the FPGA.

```
$ sudo apt-get install openocd
```

3.2.5 Riscv toolchain

```
$ wget  https://static.dev.sifive.com/dev-tools/riscv64-unknown-elf-gcc-8.3.0-2019.08.0-
↪x86_64-linux-ubuntu14.tar.gz
$ gunzip riscv64-unknown-elf-gcc-8.3.0-2019.08.0-x86_64-linux-ubuntu14.tar.gz
$ tar xvf ...
$ add to path
```

3.2.6 Vivado v2019.1

The installation of Vivado is not detailed here. If you want to generate your own Bitstream you will have to install Vivado.

3.3 Make environment artifacts

Create a Vagrant box:

```
$ vagrant package --output=mic-sec-2022-litex.box
$ sha256sum mic-sec-2022-litex.box.tar.xz | tee mic-sec-2022-litex.box.tar.sha256sum
$ sha256sum --check mic-sec-2022-litex.box.sha256sum
```

Create a virtual machine:

```
$ vboxmanage export mic-sec-2022 -o mic-sec-2022-litex.ova
$ sha256sum mic-sec-2022-litex.ova | tee mic-sec-2022-litex.ova.sha256sum
```

CHAPTER FOUR

LABS

Warning: The practical work has been tested on:

- **Ubuntu 20.04.4 LTS**.
- **Debian Bullseye**.

It should work on other OS.

4.1 Lab 01: Getting Started with LiteX

Objectives

1. Build a minimal System on Chip (SoC) with LiteX
2. Discover how LiteX work

4.1.1 Prerequisites

Firstly we will configure the environnement for the lab.

LiteX installation and setup has been done in a Python virtual environment, so activate it:

```
source venv/pyenv-litex/bin/activate
```

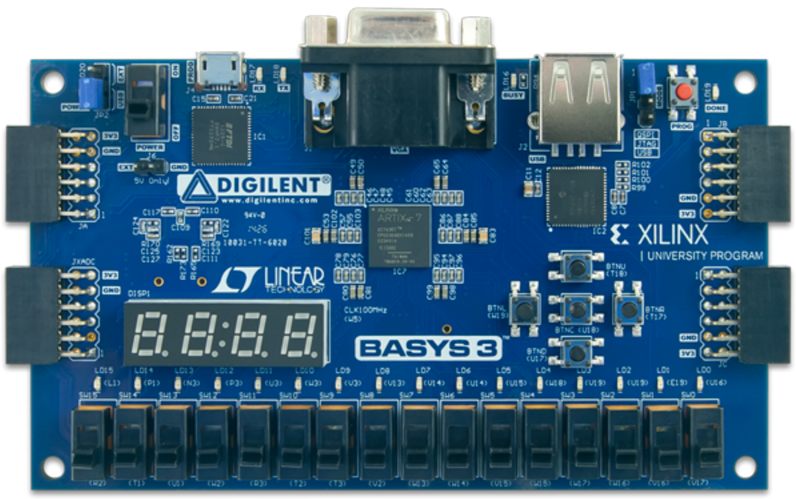
To generate the software you need a software toolchain. For LiteX to use the toolchain you have to configure it:

```
export PATH="INSTALL-PATH/riscv64-unknown-elf-gcc-8.3.0-2019.08.0-x86_64-linux-ubuntu14/
↪bin:$PATH"
```

To generate the gateware you need a FPGA toolchain. For LiteX to use the toolchain you have to configure it:

```
export LITEX_ENV_VIVADO="INSTALL-PATH/Xilinx/Vivado/2019.1"
```

The FPGA board used in this lab is a Digilent Basys3 board:



The FPGA is a Artix-7 FPGA.

All the materials for the lab are available in the following folder:

```
cd hands-on/lab01
```

4.1.2 Build your first SoC for Verilator

A makefile is available to help you execute the right commands. To look at behind the scene what happened look at inside the Makefile.

To generate the verilated model of the SoC try:

```
make soc_sim_gateware
```

To generate the software, here the demo example provided by LiteX, try:

```
make soc_sim_demo
```

Finally to test your SoC with the associated software you can try:

```
make soc_sim_demo_run
```

And you should have something like this:

```
        __   _ __      _  __
       / /  (_) /____ | |/_/
      / /__/ / __/ -_)>  <
     /____/_/\__/\__/_/|_|
   Build your hardware, easily!

 (c) Copyright 2012-2022 Enjoy-Digital
 (c) Copyright 2007-2015 M-Labs

 BIOS built on Dec  5 2022 23:35:03
 BIOS CRC passed (7c5b41e8)
```

(continues on next page)

(continued from previous page)

```
 LiteX git sha1: a4cc859d

--=============== SoC ==================--
CPU:            VexRiscv @ 1MHz
BUS:            WISHBONE 32-bit @ 4GiB
CSR:            32-bit data
ROM:            128KiB
SRAM:           8KiB
MAIN-RAM:       64KiB

--========== Initialization ============--

--=============== Boot ==================--
Booting from serial...
Press Q or ESC to abort boot completely.
sL5DdSMmkekro
Timeout
Executing booted program at 0x40000000

--============= Liftoff! ===============--

LiteX minimal demo app built Dec  5 2022 23:34:50

Available commands:
help                - Show this command
reboot              - Reboot CPU
donut               - Spinning Donut demo
helloc              - Hello C
litex-demo-app> helloc
Hello C demo...
C: Hello, world!
litex-demo-app>
```

4.1.3 Build your first SoC for FPGA

If you do not have Vivado installed in your system we already a bitstream for you. It is available in the *demo-digilent-basys* folder inside the virtual machine. If you have Vivado installed in your system you can try:

```
make soc_basys3_gateware
```

You can generate the doc of the SoC

```
make soc_basys3_doc
```

And then open it

```
open build/digilent_basys3/doc/_build/html/index.html
```

You can load the generated bitstream to the board

```
make soc_basys3_load_bitstream
```

Connect a serial terminal to the board

```
litex_term /dev/ttyUSB1
```

Then press the BTNC button to reset the board. You should have something like this

```
        __   _ __      _  __
       / /  (_) /____ | |/_/
      / /__/ / __/ -_)>  <
     /____/_/\__/\___/_/|_|
   Build your hardware, easily!

 (c) Copyright 2012-2022 Enjoy-Digital
 (c) Copyright 2007-2015 M-Labs

 BIOS built on Dec  5 2022 22:46:08
 BIOS CRC passed (729066ee)

 LiteX git sha1: a4cc859d

--================ SoC ==================--
CPU:            VexRiscv @ 75MHz
BUS:            WISHBONE 32-bit @ 4GiB
CSR:            32-bit data
ROM:            128KiB
SRAM:           8KiB
MAIN-RAM:       128KiB

--========== Initialization ============--
Memtest at 0x40000000 (128.0KiB)...
  Write: 0x40000000-0x40020000 128.0KiB
   Read: 0x40000000-0x40020000 128.0KiB
Memtest OK
Memspeed at 0x40000000 (Sequential, 128.0KiB)...
  Write speed: 123.9MiB/s
   Read speed: 63.9MiB/s

--=============== Boot ==================--
Booting from serial...
Press Q or ESC to abort boot completely.
sL5DdSMmkekro
Timeout
No boot medium found

--============= Console ================--

litex>
```

4.1.4 Load an application from the LiteX BIOS

Generate the demo example for the gateware

```
make soc_basys3_demo
```

Load the application over the serial bus

```
make soc_basys3_load_demo
```

Then press the BTNC button to reset the board and observe.

4.1.5 Take aways

- A SoC functional in few minutes!
- A lots of supported boards!
- Several FPGA toolchains supported!
- Open source community https://github.com/enjoy-digital/litex
- High level description of your Gateware.

4.2 Lab 02: Create a minimal SoC with LiTeX

Objectives

1. Create a minimal SoC from scratch
2. See how to instantiate different CPU cores

4.2.1 Prerequisites

Firstly we will configure the environnement for the lab.

LiteX installation and setup has been done in a Python virtual environment, so activate it:

```
source venv/pyenv-litex/bin/activate
```

To generate the software you need a software toolchain. For LiteX to use the toolchain you have to configure it:

```
export PATH="INSTALL-PATH/riscv64-unknown-elf-gcc-8.3.0-2019.08.0-x86_64-linux-ubuntu14/
↪bin:$PATH"
```

All the materials for the lab are available in the following folder:

```
cd hands-on/lab02
```

4.2.2 Discover a minimal SoC for simulation

The folder contains a file called *lab02-litex-sim.py*. It describes a minimal SoC which target a simulation platform.

You can test it by doing

```
make soc_sim_gateware
make soc_sim_demo
make soc_sim_demo_run
```

4.2.3 Build a SoC and change the CPU

Change the CPU used by using a *picorv32*. To use this cpu you need to install the *pythondata-cpu-picorv32* if not available in the *third_party* folder. To do that observe the Makefile and change the *cpu-type* option.

At the end of the process you should have something like that

```
        __   _ __      _  __
       / /  (_) /____ | |/_/
      / /__/ / __/ -_)>  <
     /____/_/\__/\__/_/|_|
   Build your hardware, easily!

 (c) Copyright 2012-2022 Enjoy-Digital
 (c) Copyright 2007-2015 M-Labs

 BIOS built on Dec  6 2022 01:26:43
 BIOS CRC passed (99b0f0ac)

 LiteX git sha1: a4cc859d

--=============== SoC ==================--
CPU:            PicoRV32 @ 1MHz
BUS:            WISHBONE 32-bit @ 4GiB
CSR:            32-bit data
ROM:            128KiB
SRAM:           8KiB
MAIN-RAM:       64KiB

--========== Initialization ============--

--============== Boot ==================--
Booting from serial...
```

You can observe that the CPU has changed.

4.2.4 Take aways

- Python as hardware description language allows to quickly add new features to your SoC: add new IP core, etc.
- A lot a different configuration specially to test easily different CPU
- Interesting illustration of what can be done with the LiteX embench tester:
  - https://antmicro.github.io/embench-tester/
  - https://github.com/antmicro/embench-tester
- For a same CPU type you can define CPU variants

4.3 Lab 03: Software app for a SoC with LiTeX

Objectives

1. Develop a software baremetal app
2. Deploy and test your software in a simulation environment

4.3.1 Prerequisites

Firstly we will configure the environnement for the lab.

LiteX installation and setup has been done in a Python virtual environment, so activate it:

```
source venv/pyenv-litex/bin/activate
```

To generate the software you need a software toolchain. For LiteX to use the toolchain you have to configure it:

```
export PATH="INSTALL-PATH/riscv64-unknown-elf-gcc-8.3.0-2019.08.0-x86_64-linux-ubuntu14/
↪bin:$PATH"
```

All the materials for the lab are available in the following folder:

```
cd hands-on/lab03
```

4.3.2 Baremetal app

To create a baremetal app a good starting point is to use the demo app provided in LiteX.

```
cp -r ../third_party/litex/litex/soc/software/demo myapp
```

Look at the *main.c* file and try to add your own function in the menu that will display “Awesome Winter School 2022!”.

Finally test your programm in Verilator:

```
make soc_sim_gateware
make soc_sim_myapp
make soc_sim_myapp_run
```

4.3.3 Take aways

There is existing project with Linux and RTOS (Zephyr, Tock OS):

- https://github.com/litex-hub/linux-on-litex-vexriscv
- https://github.com/litex-hub/linux-on-litex-rocket
- https://github.com/litex-hub/zephyr-on-litex-vexriscv
- https://docs.tockos.org/litex/index.html

CHAPTER FIVE

CONCLUSION

Note: We hope this hands-on session will allow you to automate your experimental test bench to evaluate your solutions and make your work reproducible.

5.1 Related work

In the hardware security community other researchers are using this kind of workflow proposed in this hands-on session. Please find below a list of related works using LiTeX, it will help you to get inspired if you want to use this kind of tools in your work:

- **Tool: rowhammer-tester**
  - https://rowhammer-tester.readthedocs.io/en/latest/
  - https://github.com/antmicro/rowhammer-tester
- **Tock OS ported to LiteX SoC Arty**
  - https://github.com/tock/tock/releases

5.2 Credits

LiteX github

5.3 Acknowledgements

- Open source community for the development of the following nice tools: LiteX, Migen, nMigen, Verilator, RiscV, OpenOCD, . . .

CHAPTER SIX

INDICES AND TABLES

- genindex
- modindex
- search