TÉCNICO LISBOA

Linux capable RISC-V CPU for IOb-SoC

Pedro Nuno de Melo Antunes

@undefined

Electrical and Computer Engineering

@undefined: Prof. José João Henriques Teixeira de Sousa

@undefined

@undefined: Prof. José João Henriques Teixeira de Sousa
@undefined: Prof. Horácio Cláudio De Campos Neto

13th September 2022

Abstract

Blablabla

Honor Declaration

‘Declaro por minha honra de que ’

Ort, Datum Unterschrift

Contents

List of Figures

List of Tables

Listing

1 Introduction

1.1 Motivation

1.2 Objectives

2 Must Have Concepts

During this chapter, we're going to discuss topics that help understand the technology developments made along the thesis project.

2.1 The IOb-SoC Template

2.1.1 Internal Buses

2.1.2 How peripherals work

2.2 Open Source Verification tools

For testing purposes, it is important to have a good hardware simulation environment. For that, we take advantage of already existing and well-developed tools. There exist a number of simulation tools, most of them are proprietary, as for example *'xcelium'* from *'Candence'*. Its utilization can increase the cost of a project significantly. In this Thesis we will make an effort of using open-source, free to use, verification tools. In specific, we will take advantage of *'Icarus Verilog'* and *'Verilator'*. Although both tools are for verification, they serve different purposes, due to their characteristics.

- ***'Icarus Verilog'*** is a Logic Simulator that uses verilog or system-verilog testbenchs to test the UUT (Unit Under Test). Unfortunately, its support for system-verilog is limited and some designs might not run in this simulator. *'Icarus Verilog'* is also known as *'IVerilog'*.

  After compiling the hardware design, *'IVerilog'* outputs a file which can be run line by line to simulate designed logic.

- ***'Verilator'***

The biggest differences are: *'Verilator'* only represents logic signal as 1's or 0's, contrary to *'IVerilog'* which also represents unknown values as X's; Since *'Verilator'* ends up being a C++ program it is much faster to run the simulation than with *'IVerilog'*; On another perspective *'Verilator'* is slower than *'IVerilog'* to interpret the hardware logic design. As such, it is easier to use *'IVerilog'* to detect errors on the design, but it is better to use *'Verilator'* for more complexed simulations.

2.3 RISC-V architecture

The RISC-V **CLINT** is described The RISC-V **PLIC** was first described in the privilege instructions documentation, but since version 1.10 it was moved to its own document.

2.4 What is the UART16550?

2.5 The Linux Boot Flow

2.5.1 Bootloader firmware

2.5.2 What is a device tree?

3 Existing Technologies

There already exists embedded microcontrollers capable of running Linux. Big companies as for example ARM, Qualcom, Mediatek, Intel and AMD have created microcontroller capable of running Linux. But the processor architecture of those microcontrollers is not open-source, much less the microcontroller itself. As an example, the Raspberry Pi is a very capable board where a developer can test and implement new software. But if someone wanted to use the Raspberry as a base for his costume hardware design, that would not be possible. And thus appears the need for open-source hardware that allows creating something new without having to start from zero. This led to the appearance of RISC-V the open-source CPU architecture.

3.1 Close source RISC-V Microcontrollers

Since then, a few companies using RISC-V have appeared. For example, *Western Digital* now uses RISC-V in its external storage disks. *Microchip* as launched the first RISC-V-Based System-on-Chip (SoC) FPGA, *PolarFire*.

These companies have helped pave the way for the Linux kernel to be compatible with the RISC-V architecture. But the Andes Tecnology Sifive

3.2 Open Source Solutions

Built upon the RISC-V open-source CPU architecture, various CPU designs have emerged. RISC-V CPUs are most popular in ... . Some well known CPU are ... . Those will not be discussed here sice they do not meet the requirements to run the Linux Kernel. To run a Linux based Operating System, for example, the ... would have to ... .

3.2.1 CVA6/OpenPiton

3.2.2 RocketChip

3.2.3 NaxRiscv

3.2.4 VexRiscv

Talk about litex-vexriscv!

4 Hardware Developed

4.1 Simulation Unit Under Test Wrapper

4.2 VexRiscv Wrapper

4.3 UART16550 Wrapper

4.4 CLINT Unit

4.5 PLIC Unit Wrapper

5 Software Developed

5.1 Python Console

5.2 Verilator Testbench

5.3 Barebones Interrupt Routine

5.4 IOb-SoC Linux Stage 0 Bootloader

5.5 IOb-SoC on OpenSBI

5.6 IOb-SoC Device Tree

5.7 IOb-SoC Linux *'rootfs'*

5.8 Makefiles

6 Products of the expedition, AKA Project Results

6.1 FPGA Resources Consumption

6.2 Run/Boot Linux Performance

7 Contributed Repositories

- **iob-soc**
- **iob-soc-vexriscv**
- **iob-lib**
- **iob-vexriscv**
- **iob-uart16550**
- **iob-clint**
- **iob-plic**

8 Conclusions

8.1 Achievements

8.2 Future Work

A Anhang

A.1 Anhang

Anhang 1

A.2 Anhang

Anhang 1