Linux-capable RISC-V CPU for IOb-SoC

Pedro Nuno de Melo Antunes
pedronmantunes@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

November 2022

Abstract

The recent appearance of the RISC-V ISA opened many exciting possibilities for building processor-based systems without the need to license the base architecture from providers like ARM. Running applications on bare metal RISC-V machines is a good starting point, but an OS is required to ease the developers' efforts for more complex applications. Linux has been around for over three decades and is a well-polished OS. The problem is that open-source SoC platforms that run Linux and simultaneously are modular and configurable do not exist. This work aims to create an SoC capable of executing a Linux OS. The author bases the work on IOb-SoC, a modular and configurable open-source SoC platform that only runs bare-metal applications. The author achieves this thesis goal by changing the IOb-SoC CPU and adding three hardware peripherals. Additionally, the author develops software that improves the IOb-SoC platform, complements the hardware components created and allows the execution of a complete OS in the new SoC. Throughout this work, the thesis might refer to the SoC developed as IOb-SoC-Linux. The IOb-SoC-Linux uses less than 10% of the FPGA resources on the supported development boards. Moreover, the Linux OS boots in five seconds in the Kintex Ultrascale and seven seconds in the Cyclone V.

Keywords: RISC-V, Linux, Systems on-Chip (SoC), Verilog, IOb-SoC

1. Introduction

The availability of fully open-source systems capable of executing an Operating System (OS) is limited. For a long time, the Linux kernel [6] and the open-source software built around it allowed developers to implement a fully open-source Linux OS on their closed-source hardware devices. However, the scarcity of open-source hardware complicated the development of fully open-source systems. With the appearance of *RISC-V* [1], open-source hardware availability started growing. Developing a *RISC-V* System on a chip (SoC) capable of executing a Linux OS would allow researchers to access a fully open-source system executing an OS. Having a Linux OS running in an SoC would allow developers to create new applications to run in that SoC without worrying about its hardware components. The Linux community is significant, and researchers are used to working with the Linux kernel. Therefore, the requirement for an SoC capable of running Linux is high.

A Linux OS allows using many features unavailable in bare-metal applications. When developers create a bare-metal application, they must be aware of the SoC hardware and are limited in terms of functionalities. Similarly, if developers were to create an application using Real-Time Operating Systems (RTOS), for example, *freeRTOS* [2], they would have access to features such as a scheduler, events, threads, semaphores and message boxes. However, a Linux OS provides those and more functionalities. A Linux OS implements memory management and protection mechanisms, allows the execution of multiple applications simultaneously, supports multiple network adapters, and can interact with the user through a terminal. A Linux OS is also more secure than bare-metal or RTOS applications since it limits the user application's access to the machine resources, preventing misuse or damage.

What most motivates the development of a *RISC-V* SoC capable of running a Linux OS is its advantages for future development. Such as creating hardware accelerators which work with a Linux OS and integrating them with the SoC the author developed to test in a real-world application.

This dissertation work aims to develop an open-source SoC and execute a minimal Linux OS on it. The author will adapt the existing *IOb-SoC* [3] to create an SoC that supports a Linux OS. *IOb-SoC* is a modular open-source *RISC-V* SoC that allows researchers to develop their own SoC. The IObundle developers use *Verilog* [5] to describe *IOb-SoC* and peripherals hardware.

The author had first to swap the current CPU used in *IOb-SoC*. The problem with the current CPU is that it cannot run an OS, only bare-metal applications. Therefore, *IOb-SoC-Linux* contains a 32-bit *RISC-V* CPU capable of running Linux. Then, since the *IOb-SoC* does not support interrupts, the author had to create and integrate into *IOb-SoC-Linux* the hardware needed to generate interrupts in a *RISC-V* SoC. Lastly, the author had to ensure the Linux OS supported the UART used in the*IOb-SoC-Linux*. Since Linux does not support the *IOb-SoC* UART, the author integrated a UART16550 in *IOb-SoC-Linux*.

Four major software components comprise a Linux OS. Those software components are the Linux kernel, the bootloader firmware, the root file system (rootfs) and a Device Tree Blob (DTB). The author had to build those software components to run a Linux OS on the *IOb-SoC-Linux*. On power-on, the *IOb-SoC-Linux* transfers the Linux OS software binary files onto the board where it runs, and the Linux OS will boot. After the OS boots, the user can run custom applications and take advantage of the Linux OS. The author also automated and documented the process of generating and deploying the Linux image to *IOb-SoC-Linux* so that, after this work, creating new images with different characteristics will be straightforward.

Finally, the system must be fully verified both on simulation and running on an FPGA board. The *IOb-SoC* needed a fast *Verilog* simulator to verify the Linux OS execution. Therefore, the author developed a simulation testbench using the free-of-charge and open-source *Verilator* [4] simulator.

2. Background

Place text here...

2.1. Sub-section...

A generic CFD design problem can be formally described as

$$\begin{aligned} \text{Minimize} \quad & Y(\alpha, \mathbf{q}(\alpha)) \\ \text{w.r.t.} \quad & \alpha\,, \\ \text{subject to} \quad & \mathcal{R}(\alpha, \mathbf{q}(\alpha)) = 0 \\ & C(\alpha, \mathbf{q}(\alpha)) = 0\,, \end{aligned} \tag{1}$$

where Y is the cost function, α is the vector of design variables and $\mathbf{q}$ is the flow solution, which is typically of function of the design variables, and $C = 0$ represents additional constraints that may or may not involve the flow solution. The flow governing equations expressed in the form $\mathcal{R} = 0$ also appear as a constraint, as the solution $\mathbf{q}$ must always obey the flow physics.

2.2. Sub-section...

More text...

3. Implementation

Place text here...

3.1. Sub-section...

More text...

3.2. Sub-section...

More text...

4. Project Results

In the following chapter, the author analyses the results obtained from the hardware and software developed in this thesis project. The candidate successfully executes the minimal Linux OS in real hardware using the developed System on a chip. All the results obtained in this thesis which communicate with the FPGA board or the SoC testbench, are executing the developed *Console* program. The hardware components comprising the SoC differ depending on the software needs.

The objective of this thesis project was to run an Operating System in the *IOb-SoC-Linux*. Table 1 presents how much time it takes to build the complete OS with the command "make build-OS". The "real" time is the time that passes since the user executes the command until it finishes. The "user" time is the time the CPU takes while executing operations in the user space. The "user" time is bigger than the "real" time because it counts the time passed in each CPU core. Part of the compilation of the RootFS and the kernel is done in parallel using two cores.

real	4m29,570s
user	8m12,039s
sys	0m56,887s

Table 1: Time it takes to build the OS.

The OS size is to big to run in the FPGA internal memory. The *OpenSBI* bootloader is 90896 Bytes. The device tree blob is 1669 Bytes. The Linux kernel is 4426152 Bytes. Lastly the root file system is 1142733 Bytes. The memory has to have at least 8 MB (2^23) to store all this software. However, the Linux kernel needs a bigger memory where it can store virtual memory pages and execute the different application processes. The device tree source describes the system had 512 MB of available memory. Consequently, the author had to implement the *IOb-SoC-Linux* on the FPGA with access to the external memory. The internal memory could never be as big as 512 MB.

In figures 1 and 2 the reader can see the start of the OS simulation with *Verilator*.

Figure 1 shows the initialization of the *Console* program. Furthermore, it shows the instantiation of the *iob-UART16550* and the *iob-PLIC*. The *iob-UART16550* and the PLIC core have an initial

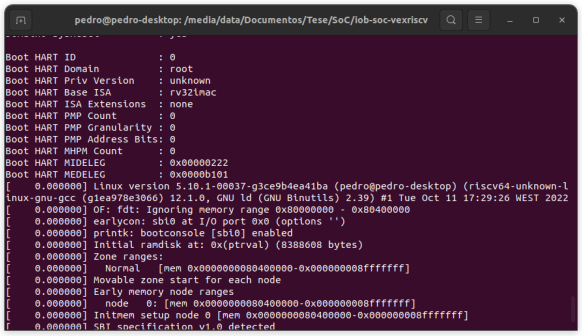

Figure 1: *iob-UART16550* and *iob-PLIC* properties.

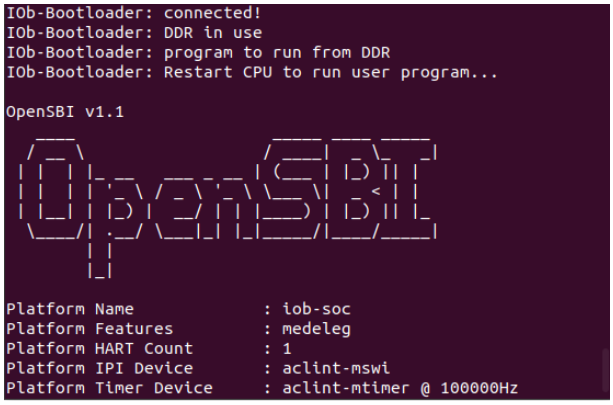

Figure 2: *IOb-SoC* bootloader and *OpenSBI* firmware.

block that prints their properties. The synthesis tools do not synthesise the initial block to real hardware, but the simulator executes it. Figure 2 shows the *iob-bootloader* and the start of the *OpenSBI* bootloader. The *iob-bootloader* in figure 2 does not transfer the software to the memory because the author executed the simulation considering that the software was already in the memory.

Figure 3 shows the end of the *OpenSBI* bootloader and the start of the Linux kernel. The first line printed by the Linux kernel indicates the author built the kernel executing, the kernel version and which toolchain he used to compile it.

While figure 3 shows the start of the Linux kernel, figure 4 shows the end of the Linux kernel booting process and the execution of the "init" script. The "init" script is the first program the OS executes after the Linux kernel mounts the RootFS and finishes booting. There exist multiple messages printed to the terminal between the output shown in figure 3 and in 4. Those messages show the progress while the Linux kernel boots. The Linux kernel boot process's last message is "Run /init as init process". After that message the SoC executes the "init" program.

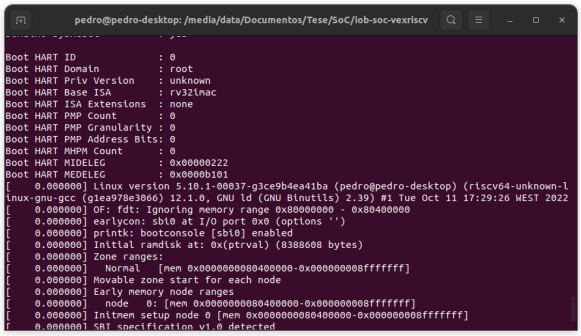

Figure 3: Start of the Linux kernel boot with *Verilator*.

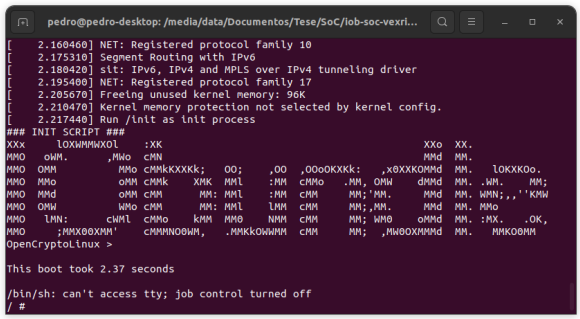

Figure 4: End of Linux kernel boot with *Verilator*.

Figure 5 shows the developed minimal OS running on an FPGA. The reader can see that the author has suppressed the shell warning. The initial part of the figure shows the final stage of the Linux kernel booting. After booting, the author tested the *ls /* command that showed the files and directories in the systems' root. Lastly the author executed the *cat init* command for the OS to print the contents of the "init" script to the terminal.

The time the Linux kernel takes to boot in real hardware, figure 5, is almost double what it takes to boot in simulation, figure 5. The time to boot is almost double because the memory module used in the simulation does not have any latency. When the L2 cache fetches data from memory in real hardware, it must wait before receiving the data burst. Using the *CYCLONE V* FPGA board the Linux kernel takes 7.01 seconds to boot. The author expected the boot to take longer since the system clock frequency used with the *CYCLONE V* is 50 MHz. The Kintex Ultrascale was able to run with a frequency of 100 MHz. The *OpenSBI* bootloader and the device tree blob had to be recompiled with the system frequency defined to 50 MHz to run in the *CYCLONE V*.

Tables 3 and 2 show the resources used by the *IOb-SoC-Linux* in the different FPGAs.

Tables 3 and 2 show that the resources utilization

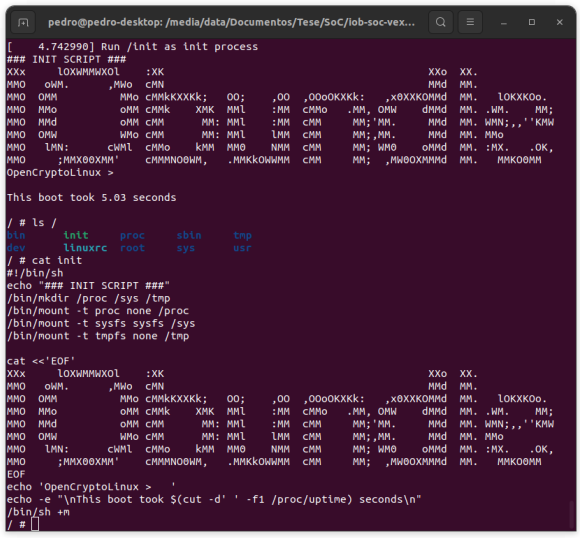

Figure 5: Linux kernel boot in the FPGA.

	Resources	FPGA usage %
ALM	11,227	10
DSP	8	3
FF	13725	2
BRAM blocks	234	19
BRAM bits	755,424	9

Table 2: Cyclone V GT

	Resources	FPGA usage %
LUTs	23126	9.54
Registers	24505	5.05
DSPs	10	0.52
BRAM	39.5	6.58

Table 3: Kintex Ultrascale

from the *IOb-SoC-Linux* is not much bigger than the *IOb-SoC*. The FPGA still has enough resources to implement hardware accelerators.

5. Conclusions

Conclusions, future work and some final remarks...

Acknowledgements

The author would like to thank his friends and professors who helped and accompanied him through his studies. Furthermore, above all, the author is thankful for his family that has been in his life since day 0, giving advice and guiding him, leading him to where he is today.

References

[1] K. Asanović and D. A. Patterson. Instruction sets should be free: The case for risc-v. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146*, 2014.

[2] R. Barry et al. Freertos. *Internet, Oct*, 2008.

[3] I. Lda. Iob-soc.

[4] W. Snyder. Verilator: Fast, free, but for me? *DVClub Presentation*, page 11, 2010.

[5] D. Thomas and P. Moorby. *The Verilog® hardware description language*. Springer Science & Business Media, 2008.

[6] L. Torvalds. Linux: a portable operating system. *Master's thesis, University of Helsinki, dept. of Computing Science*, 1997.