EDRICO - Educational DHBW RISC-V Core

Semester Paper

from the Course of Studies Electrical Engineering

at the Cooperative State University Baden-Württemberg Ravensburg

by

Levi-Pascal Bohnacker, Noah Wölki

June 2021

Time of Project	12 Weeks
Student ID, Course	6818486, 5040009, TEN18
Reviewer	Prof. Dr. Ralf Gessler

Author's declaration

Hereby we solemnly declare:

1. that this Semester Paper , titled *EDRICO - Educational DHBW RISC-V Core* is entirely the product of our own scholarly work, unless otherwise indicated in the text or references, or acknowledged below;

2. we have indicated the thoughts adopted directly or indirectly from other sources at the appropriate places within the document;

3. this Semester Paper has not been submitted either in whole or part, for a degree at this or any other university or institution;

4. we have not published this Semester Paper in the past;

5. the printed version is equivalent to the submitted electronic one.

We are aware that a dishonest declaration will entail legal consequences.

Friedrichshafen, June 2021

———————————————————

Levi-Pascal Bohnacker, Noah Wölki

Contents

Acronyms

CU	Control Unit
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
IP	Intellectual Property
ISA	Instruction Set Architecture
PMP	Physical Memory Protection
PMA	Physical Memory Attributes
RISC	Reduced Instruction Set Computer
SISD	Single Instruction Single Data
VHDL	Very High Speed Integrated Circuit Hardware Description Language

List of Figures

List of Tables

Listings

1 Introduction

These days one of the key benchmarks for technology is processing speed and calculation power. To realize mathematical operations and execute programs, different platforms can be utilized. The most commonly used unit is the standard processor consisting of transistors realized on silicium and other materials. Another crucial technology that is gaining more attention is the so-called Field Programmable Gate Array (FPGA). The FPGA consists of logical units that can be wired and configured individually for the required use-case. The advantage of FPGA is that the speed of applications can be drastically increased since the hardware will be very optimized for the specific application. This project aims to develop a Intellectual Property (IP)-core based on the Open Source Instruction Set RISC-V. The goal is to build a reusable unit of logic that can interpret compiled C-Code. The IP core is realized in the Very High Speed Integrated Circuit Hardware Description Language (VHDL) language and will be deployed on a FPGA. IP Cores are used in every computer, phone and electronic device that requires to execute some computational function. The developers of these IP Cores are big companies like Intel, ARM or AMD. These IP Cores and Instruction Sets are strictly licensed and not available for everyone. For the development of an own IP Core the Instruction Set is the main source of information and therefore the RISC-V open-source Instruction Set is used for this project.

2 Motivation

RISC-V was first proposed at Berkeley University in 2010. The architecture is therefore relatively new in comparison to others like x86, ARM or SPARC. Even though its young age is already very promising, every year new breakthroughs are achieved in the field of RISC-V based cores. MicroMagic for example announced in 2020 a chip with a total CoreMark score of 13000 and an incredible 110000 Coremark/Watt. This poses a significant development and is approximately 10 times better than any CISC, RISC or MIPS implementation in terms of Performance per Watt. Many other companies like Alibaba, Nvidia and SiFive are currently increasing research on RISC-V based cores. The Motivation behind this project was to gain experience in processor and FPGA design and verification. Furthermore it poses an interesting opportunity for students to work on a new and upcoming processor architecture.

3 Project Planning

In order to control the flow of the project, the V-Model approach was taken. The project is therefore divided into Requirements, System Design, Architecture Design, Module Design and Implementation. After Implementation the corresponding verification phases are ready to be executed, starting from the lowest level (Unit Verification) to Integration Verification, System Verification and last but not least Acceptance Verification.
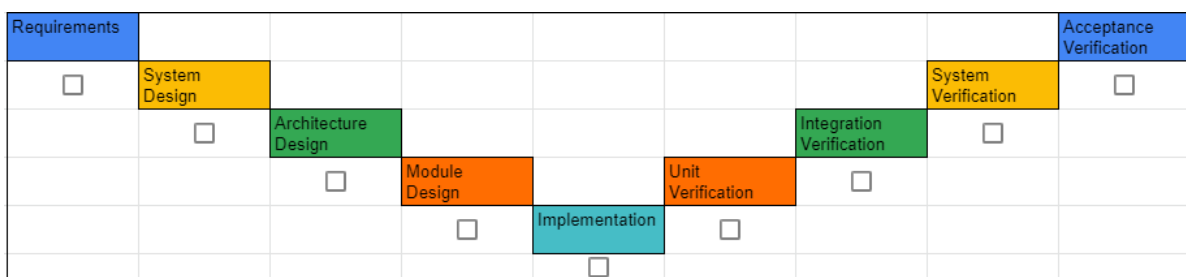

Figure 3.1: V-Model

At the beginning of every project phase, workloads were defined e.g. the definition of the Control Unit Entity. The target of Requirements Engineering was to define everything that is expected from the core and gather information about RISC-V. The data path as well as the entities of the Control Unit, Arithmetic Unit, Register Files, Exception Control, PMP & PMA Checker and AXI4-Lite Interfaces as well as a short summary of their function were defined during System Design. The next step, Architecture Design, aimed to further specify the entities mentioned above and sub-divide them into several entities. Module Design will be executed to define every single architecture, after that implementation and testing may start.

4 State of the Art

4.1 Basics of Processing Units

4.2 RISC vs CISC

4.3 RISC-V

RISC-V is an open standard Instruction Set Architecture (ISA) developed by the University of California, Berkely. The ISA is based on reduced instruction set computer (RISC) principles. The ISA supports 32, 64 and 128 bit architectures and includes different extensions like Multiplication, Atomic, Floating Point and more. The ISA is open source and therefore can be used by everyone without licensing issues and high fee requirements. Due to the open source nature of the RISC-V project, many companies like Alibaba and NVIDIA have started to develop hardware based on this ISA. RISC-V opens the opportunity to optimize and configure computer hardware to a level that would not be realizable with licensed ISA like ARM or x86. As a result of this possibility there are many projects and companies working on hardware and software that are beating common CPU in terms of performance and power usage by a lot.

4.4 Benchmarks

4.4.1 Coremark

4.4.2 SPECint

4.5 Memory Management

4.5.1 Memory Hierarchy

4.5.2 Communication Interfaces

4.6 FPGA

To verify a digital circuit software simulations as well as implementing the design on a prototype are common practice. For prototyping and even implementing a finished product, FPGA are widely used. FPGAs are special fine granularity Programmable Logic Devices. The digital logic can be described using hardware description languages such as Verilog or VHDL. These designs are then synthesized, placed and routed in order to generate a hardware configuration file, also called bitstream. The bitstream can then be loaded onto the FPGA via a programming interface e.g. JTAG. Many different vendors produce FPGAs, the most famous ones are Xilinx, Altera/Intel and Microchip. Some smaller vendors like NanoXplore produce FPGAs targeting rare use cases like space applications. Despite the many differences in design of an FPGA, the basic architecture always remains the same. An array of logic cells and building blocks of different features 9 like BRAM and DSP slices are connected to each other through configurable routing channels. Figure 4.1 shows the basic architecture of a Xilinx FPGA:

4.7 Hardware Description Languages

The CLBs in this architecture are comprised of LUTs and Flip-Flops, in order to implement boolean functions and allow the design of synchronous circuits. FPGAs produced by Xilinx are mostly SRAM based, other approaches are flash or anti-fuse based architectures.
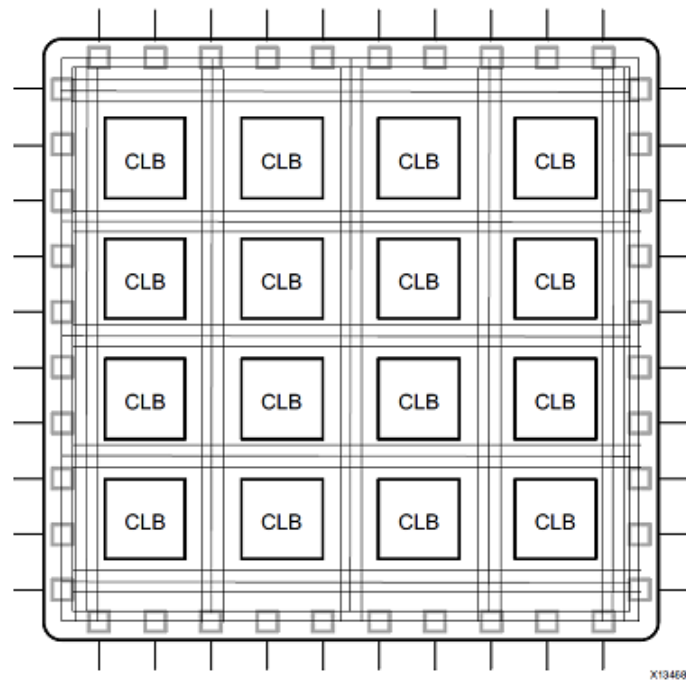
Figure 4.1: Xilinx FPGA [Xil17]

5 EDRICO (Educational DHBW RISC-V Core)

The Proposed Processor design named EDRICO implements a basic RV32I instruction set architecture. Besides the mandatory "Zicsr" extension no other instruction set extensions are implemented. To keep the implementation simple and straight-forward only one privilege mode (Machine-mode) is implemented. This mode allows full access to the processor and peripherals. Future Versions could be extended to implement S-Mode and U-Mode. The core is a simple Single Instruction Single Data (SISD) processor without any pipeline or even cache. The basic instruction cycle of fetch, decode, execute, store is performed for every instruction one at a time.

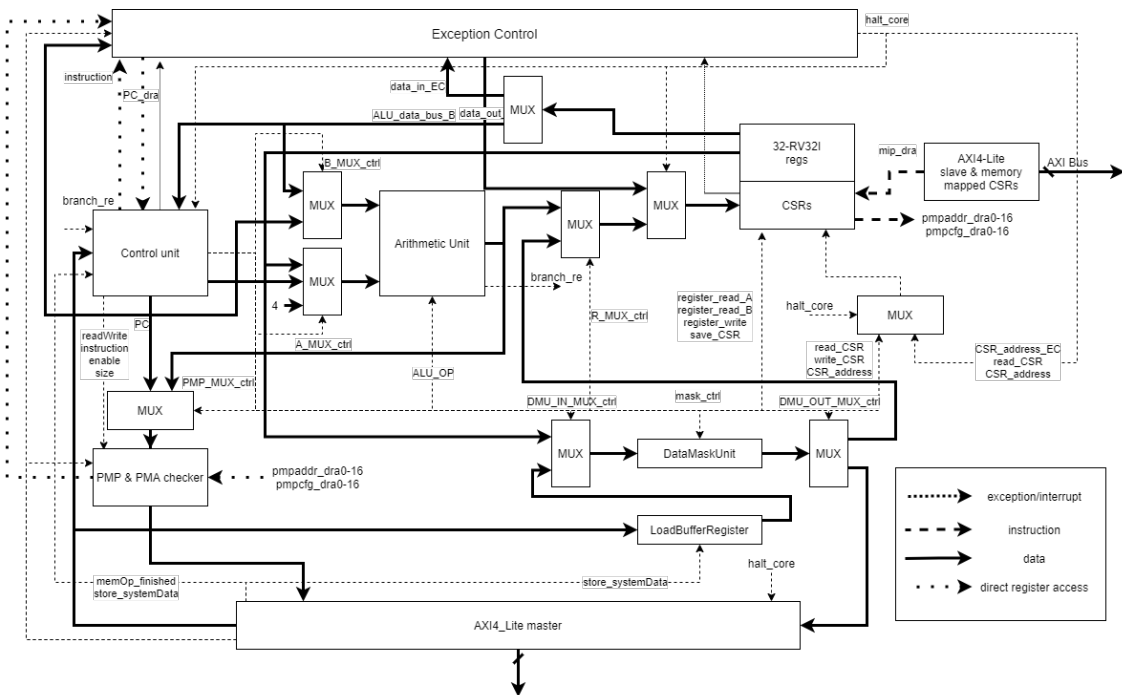Figure 5.1 shows the full overview of the processor design:

Figure 5.1: EDRICO Overview

Its main components are the Exception Control, Control Unit, Arithmetic Unit, Register Files, PMP & PMA checker and the AXI4 Interfaces. Each one of the components will be described in more detail in the following section.

5.1 Control Unit

The Control Unit (CU) is the heart of the processor and controls the other parts of the processor depending on the input instruction. The CU is responsible for fetching instructions from the instruction memory, decode the bitstream and set the respective control signals for the other processor components. Due to the complexity of the CU, there are several sub-modules which together form the overall CU. A general overview of the CU architecture is displayed in Figure 5.2.
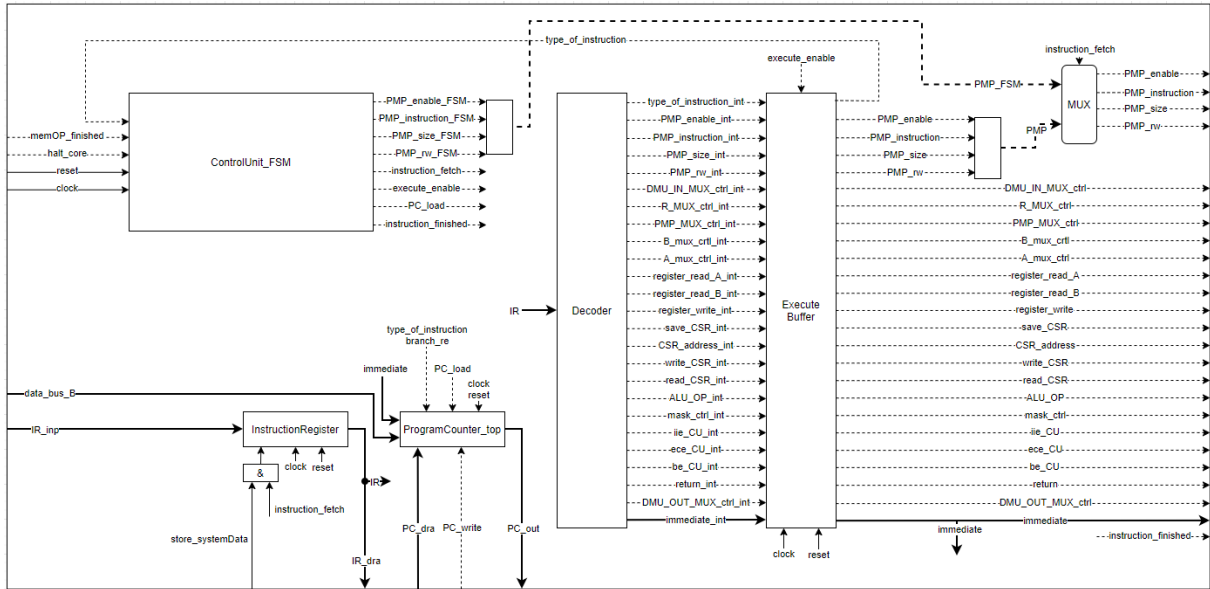

Figure 5.2: Control Unit Architecture

To describe the functionality of the CU in more detail, every sub-module will be described closely.

Since the Control Unit is responsible for the whole processor, it is important to have a persistent and stable procedure for every instruction that shall be executed. The Control Unit Finite State Machine (FSM) is responsible for the correct clock timings which is important due to memory operations and the execution time of the other processor parts. The states and conditions of the FSM are displayed in Figure 5.3.
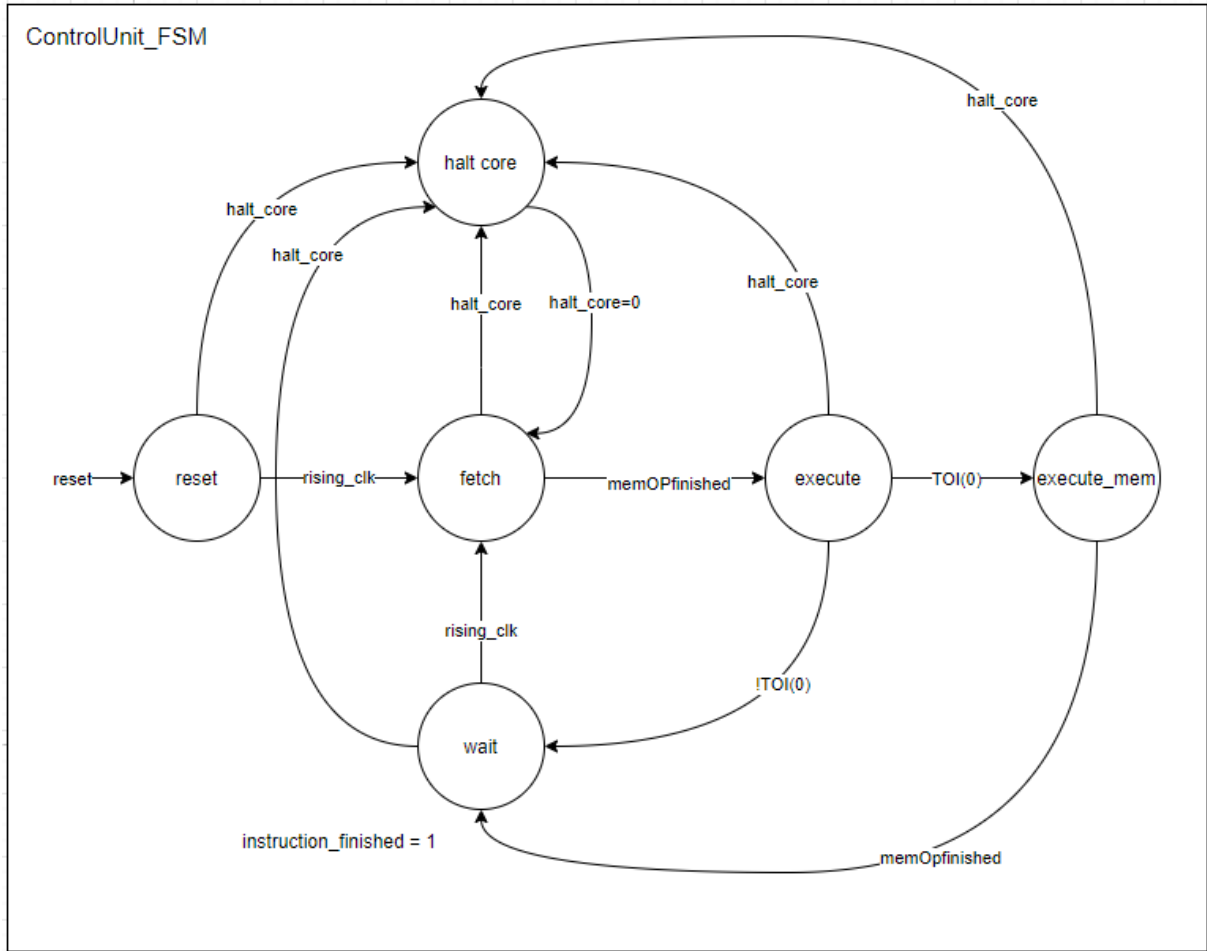
Figure 5.3: Control Unit FSM overview

Table 5.1 shows a more detailed overview of the clock cycles and the corresponding actions and states:

ClockCycle	Edge	Action	Signal
1	rising	pass the PC and enable PMP & PMA checker with respective information	
	falling	N/A	
4	rising	data is ready in instruction register - switch to execute state	*memOPfinished* & *store_systemData* is high
5	rising	execution is started - if memory operation wait for another *memOPfinished* flag, otherwise wait	*execute_enable*
x	rising	during memory operation: data loaded to buffer \store transfer finished → wait state	*memOPfinished* & *store_systemData* is high
	falling	if load: store data form buffer to specified location	
6 / x+1	rising	go to *fetch_state*	

Table 5.1: Timing of FSM

During an execution cycle, the FSM controls the rest of the CU consisting of memory, decoding unit, PC control and the different multiplexers. To understand what the purpose of the different signals are, the other components of the Control Unit are described in the following sections.

5.2 ALU

5.2.1 Architecture & Design

5.2.2 Implementation

5.3 Register Files

5.3.1 Architecture & Design

5.3.2 Implementation

5.4 PMP & PMA Checker

5.4.1 Architecture & Design

5.4.2 Implementation

5.5 Exception Control

5.5.1 Architecture & Design

5.5.2 Implementation

5.6 AXI4-Lite Master

5.6.1 Architecture & Design

5.6.2 Implementation

5.7 AXI4-Lite Slave

5.7.1 Architecture & Design

5.7.2 Implementation

6 Test & Verification

6.1 Unit & Integration Verification

6.2 System Verification

6.3 Acceptance Verification

6.4 Benchmarks

7 Future Work

8 Conclusion

Bibliography

[Xil17]   Xilinx. *Understanding FPGA Architecture*. 2017. URL: https://www.xilinx.
com/html_docs/xilinx2017_2/sdaccel_doc/topics/devices/con-fpga-
architecture.html.

Appendix