Tai Lung RV32I Project

Wrapper document

Authors:

Christian Shakkour, chrisshakkour@gmail.com

Shahar Dror, shdrth1@gmail.com

Git repository:

https://github.com/ChrisShakkour/RV32I-MAF-project

Release Information

DATE	AUTHOR	DESCRIPTION
02-FEB-2022	Shahar	Created v1.0
06-FEB-2022	Chris	Table of contents created
04-MAR-2022	Chris	Adds RISC-V basic content
05-APR-2022	Shahar	Adds Hyperlinks

Acknowledgements

Special thanks to our supervisor Amichai Ben David for contributing knowledge and experience throughout the development of this project. Many thanks to the VLSI Lab at the Technion - Israel Institute of technology for supporting and providing tools and resources that were an integral part of the development process.

About this document

In this document you will find information about the project in high level, look at the different documents attached to dive deeper into the contents, this document will mainly give a brief introduction to RISC-V and the different tools available for developing your own RISC-V Core, we will also discuss our target expectations when we first started the project compared to what we achieved.

Table of Contents

Introduction

This project hosts the design, implementation, verification, and testing of a single core RISC-V CPU called Tai-Lung in System Verilog, the project documentation is divided into 5 separate documents, four of which are content documents including the HAS, MAS, Validation plan, and TFM (Tools Flows and methodologies), and this document that wraps them all and adds some basic content of RISC-V so the reader could educate himself about RISC-V before deep diving into the low level design features.

About RISC-V

RISC-V is an open standard instruction set architecture (ISA) that began in 2010 in the university of Berkeley, The RISC-V as the name suggests follows the RISC (reduced instruction set code) ISA computer principles compared to CISC (complex instruction set code) where RISC features simple, short, and fewer instructions compared to CISC where the instructions are long, and complex in the manner that each command can be decoded to multiple elementary operation executed separately. RISC-V is not just another RISC ISA, but it is very unique because it's simple, supports multiple extensions for many different applications, and most importantly OPEN SOURCE! This architecture is drawing huge interest upon many leading companies like, Western digital, Intel, NVIDIA, and many more. This in fact is one of the reasons we choose to develop our Core in RISC-V and not anything else from the market today like the ARMv7-M Thumb instruction set which is widely used in the CORTEX-M4 ARM processors

Base Instruction Set

RISC-V ISA strings begin with either RV32I, RV32E, RV64I, or RV128I indicating the supported address space size in bits for the base integer ISA, RV32I is featured with an integer instruction set with 32-bit memory address space.

Base	Version	Frozen?
RV32I	2.0	Y
RV32E	1.9	N
RV64I	2.0	Y
RV128I	1.7	N
Extension	Version	Frozen?
M	2.0	Y
A	2.0	Y
F	2.0	Y
D	2.0	Y
Q	2.0	Y
L	0.0	N
C	2.0	Y
B	0.0	N
J	0.0	N
T	0.0	N
P	0.1	N
V	0.2	N
N	1.1	N

Extensions

Standard ISA extensions are given a name consisting of a single letter. For example, the first four standard extensions to the integer bases are: “M” for integer multiplication and division, “A” for atomic memory instructions, “F” for single-precision floating-point instructions, and “D” for double-precision floating-point instructions. Any RISC-V instruction set variant can be succinctly described by concatenating the base integer prefix with the names of the included extensions. For example, “RV64IMAFD”

Open-source tools

Open source is powerful, look at the Linux OS today! The same shall be expected with RISC-V. here we will introduce a few open-source development tools currently available, links will be present in the references section. to compile your C or C++ code you can use the risc-v gnu gcc compiler, to validate that your Core is actually RISC-V compatible one can use the RISC-V compatibility test generator that creates a series of tests with expected output to test each command produces what is expected according to the ISA SPECS.

High-level Arch spec

High_Level_Arch_Spec_document

Microarchitecture

MicroArchitecture_document

Validating the Core

Validation_Plan_document

Tools Flows & Methodologies

Tools_Flows_Methodologies_document

What's Next?

To take this project even further, one can expand the project to the following ideas.

(1) Improving the IPC shown in the HAS document by adding a branch prediction unit.
(2) To add AHB/APB/AXI support to control the CPU from an external agent.
(3) To add CSR support and attempt to run the Zephyr RTOS.
(4) To take the core into an FPGA platform.
(5) Synthesize the Core and implement RTL changes to converge on challenging power, Area, and Frequency targets.
(6) Integrate the CPU for a multi core Architecture.
(7) Add ISA Extensions, M, A, F…

Of course, feel free to explore and suggest different options.

References

(1) RISC-V official website: https://riscv.org/
(2) RISC-V spec: https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf
(3) RISC-V gnu gcc toolchain: https://github.com/pulp-platform/pulp-riscv-gnu-toolchain
(4) RISC-V compatibility: https://riscv-ctg.readthedocs.io/en/stable/
(5) CORTEX-M4: https://developer.arm.com/documentation/ddi0439/b/CHDDIGAC
(6) Zephyr RTOS: https://zephyrproject.org/