NucleusRV

Release 0.1

Usman Zain

Jan 16, 2023

CONTENTS

Note: This project is under active development.

NucleusRV is a 32-bit 5 stage pipelined RISC-V core written in Chisel. It implements I base ISA, M multiply and divide, and C compressed instructions (RV32IMC). NucleusRV has been taped out in Google's sponsered OpenMPW-6 shuttle on SKY130nm process node.

The documentation is split into 3 sections.

The *Overview* section explores the features of NucleusRV from bird's eye view.

The *User Guide* section provides information necessary to setup and run NucleusRV. It is aimed at software developers writing software for NucleusRV.

The *Developer Guide* section gives detailed explanation of source code and different design decisions. It highlights contribution guidelines and will be helpful for people making changes to NucleusRV

CHAPTER

ONE

OVERVIEW OF NUCLEUSRV

NucluesRV is an embedded 32 bit RISC-V core. It is written in Chisel and it implements base ISA I, compressed instructions *C*, multipy and divsion instructions *M* and floating point unit *F*.

CHAPTER

TWO

NUCLEUSRV USER GUIDE

This user guide provides information necessary to setup and run NucluesRV. It is aimed at software developers writing software for NucleusRV and hardware developers integrating NucleusRV into a design

CHAPTER

THREE

NUCLEUSRV DEVELOPER GUIDE

This section gives detailed explanation of the source code and different design decision. It describes contribution guidelines and will be helpful for people making changes to NucleusRV code base.

3.1 Instruction Fetch

Fig. 1: Instruction Fetch (IF) stage

The Instruction Fetch (IF) stage fetches one instruction from the memory, increments the PC and supplys the instruction to Instruction Decode (ID) stage. It serves one instruction per cycle.

The core supports misaligned instruction address by allowing the program counter to increment by 2. This happens when a compressed instruction is encountered. The `is_comp` signal denotes that whehter we have received compressed instruction.

We cannot pass misaligned address that we may have ended up with after incrementing the program counter by 2 and thats where the Realigner module comes in. This module takes as input the instruction address (PC) and corresponding instruction and make sures that instruction address (PC) is word aligned. The module operates as follows: If the address is aligned, it is passed to instruction memory to fetch the instruction as is. If it is misaligned, the state machine performs the following actions:

1. Store the upper half-word of current instruction, halt the PC for one cycle, and send the address to the next instruction. Meanwhile, NOP will be fed to the core.
2. After one cycle, when the instruction arrives, the lower half-word of this instruction is concatenated with the previously stored upper-half word. And this new instruction will be fed to the core.

The instruction is then passed to Compressed Decoder which decodes the 16 bit instruction instruction into its equivalent 32 bit instruction and also sets the `is_comp` flag. The next pc address is calculated on the basis of this flag, that is, it increments by 2 if it is true and by 4 if false.

The calculations for jump and branch addresses are done in Instruction Decode stage and then passed to Instruction fetch when the branch is taken.