phoeniX RISC-V Core V0.1

Arvin Delavari – Faraz Ghoreishy

arvin7807@gmail.com – farazghoreishy@gmail.com

Summer 2023

Iran University of Science and Technology

Electronics Research Center – Digital Design Laboratory

This page was initially left blank.

Preface:

phoeniX is a 5 stage pipelined 32-bit RISC-V processor written in Verilog HDL. This project was started in summer of 2023 in Digital Design Laboratory of Electronics Research Center at Iran University of Science and Technology. This document contains technical specifications and user manual for version 0.1 of phoeniX processor. This version of the processor is able to execute RV32I (integer operations extension) instructions of official RISC-V instruction set manual [1]. Other extensions will be supported by the core in the future updates.

phoeniX core is capable of execution of assembly and C codes using iverilog simulator. Guidance is included in this document for the flow of code execution in this processor. For execution of C codes, the processor will use RISC-V GCC compiler toolchain and for assembly codes, you can use GCC compiler or Venus simulator which is a free extension for the recognized code editor, Microsoft Visual Studio Code. Further descriptions are included to relative sections of this document.

This project is an open source CPU under the GNU V3.0 license [2] and is free to use. You can find source codes and documentations in the following GitHub repository:

<https://github.com/ArvinDelavari/phoeniX-RV32>

We would like to express our sincere gratitude to the following individuals for their invaluable guidance and remarkable insights, which played a pivotal role in the completion of this project:

* Professor Sattar Mirzakuchaki PhD

Iran University of Science and Technology

* Professor Hadi Shahriar Shahhoseini PhD

Iran University of Science and Technology

* Nima Amirafshar MSc

Iran University of Science and Technology

Arvin Delavari – Faraz Ghoreishy

Iran University of Science and Technology – Summer 2023

Contents:

1. Introduction
   1. phoeniX core structure
   2. phoeniX software interface
   3. Required software
2. Building Blocks (Modules)
   1. Fetch Unit
   2. Instruction Decoder
   3. Immediate Generator
   4. Address Generator
   5. Arithmetic Logic Unit
   6. Register File
   7. Hazard Detection and Forwarding Unit
   8. Jump and Branch Unit
   9. Load Store Unit
3. Memory Interface Logic
4. Code Executant Software
   1. Code execution and simulation flow
   2. Windows (Venus Simulator)
   3. Linux (RISC-V GCC toolchain)
5. Synthesis Result
   1. Qflow toolchain
   2. TSMC 180nm PDK
   3. phoeniX ASIC result
6. References

Chapter 1

Introduction

The phoeniX project was initiated in summer of 2023 in Electronics Research Center at Iran University of Science and Technology [3]. This RISC-V processor was designed in order to be an original base processor with acceptable performance and specification which aims to lay a solid groundwork and foundation for future computer architecture research and development endeavors.

The processor uses RISC-V open source Instruction Set Architecture (ISA) with a custom designed microarchitecture. The core is written in Verilog HDL and it is synthesizable for both FPGA and ASIC targets. Building blocks of this core were all written in separate modules which would help developers to add features and test computer architecture techniques on the base core. This modular design leads to a simpler way of debugging and also extending the core.

In the following version (V0.1), phoeniX is a 5 stage pipelined RV32I processor which supports “I-Extension” of RISC-V ISA. Other extensions will be added to the core in the upcoming updates soon.

phoeniX stands out with its user-friendly and straightforward flow for simulating and executing C and assembly codes on the processor. Unlike many other open-source processors, phoeniX offers a simplified approach that is easier to follow. The repository of this project provides comprehensive documentation, offering step-by-step guidance on executing codes on the core. This resource is highly accessible and designed to facilitate a seamless development experience in computer architecture and digital design field.

* 1. phoeniX core structure

In the beginning, one of the most specialties of this processor which should be mentioned is not having a single control unit. This processor uses a “Distributed Control Logic” which leads to elimination of control unit from the building blocks. By DCL it means that after decoding the instructions, relative fields such as opcodes, funct3, funct7 and other fields are straightly directed to the linked modules, and the process of control signals generation will be taken place in the target module hardware. For example, in a basic processor decoded fields are sent to control unit and inside this module, signals will be generated to define ALU operation, ALU multiplexers (for selection of registers bus or immediate value), bypassing and forwarding multiplexers and etc. In phoeniX processor, all of these control signals are determined inside the target modules, by sending the relative decoded fields directly to the target. In fact, modules are somehow known to be self-controllable which in this case we call the general system, Distributed Control Logic.

phoeniX is a 5 stage pipelined processor which has the classic machine cycle stages: Fetch, Decode, Execute, Memory Interface and Write back. Building blocks of the processor are mostly designed in separated modules which gives developers freedom for changes and the mentionable benefit of modularity. Registers used for latching data in different stages of the core are not designed as modules are completely written inside the main module named “phoeniX”. Detailed descriptions about the hardware design of the core and its modules are included Building Blocks (Modules) section of this documents.

Memory interface of the core is completely written and described in testbench file because of the limitations of memory simulation using Hardware Description Languages. Further information about the interface logic is inside Memory Interface section of this document.

The phoeniX core also has a hazard detection unit which gives the core, the ability of performing data-forwarding and bypassing techniques in order to skip stalls in the code including dependencies and hazards.

The maximum delay time (critical path) of phoeniX processor analyzed using Yosys [4] open-source synthesis tool and Vesta [5] static time analysis software, in different stages is something about 3800-3900 picoseconds which leads us to clock cycle width of 4 nanoseconds. This means the phoeniX processor will perform in a frequency rate of nearly 250MHz which is a remarkable number in benchmarking with other processor. Most of the commercial and industrial embedded processors designed by ARM with Cortex M0, M3 and M4 [6] microarchitecture will not reach this frequency rate. These processors find extensive integration with microcontroller series such as STM32 (by ST) or LPC (by NXP), making them highly prevalent and widely employed in the field of embedded systems.

Even other open-source RISC-V embedded processors such as PicoRV32 by Claire Wolf [7] won’t reach a higher frequency than the mentioned. PicoRV32, is one of the most recognized open-source processors and even used in fabricated microcontroller chips has a frequency rate ranged in 50 to 100MHz depending to the technology. Another well-known RISC-V processor is named ibex core by ETH Zurich [8] university which also has a frequency range between 100MHz and 300MHz in highest performance.

The phoeniX processor is a reliable and user-friendly solution, offering modularity and easy extensibility for developers and computer architecture researchers. It features a simplified design that is easy to understand, while still delivering competitive performance compared to other processors.

* 1. phoeniX software interface

The phoeniX processor uses standard GCC compiler toolchain [9] which is officially verified by RISC-V organization, in order to run C codes and assembly codes on the processor. This codes are turned into a firmware file including hex format of instructions and addresses by the compiler toolchain. The generated firmware file is given to the testbench of phoeniX CPU as the instruction memory. The processor can simulate the executed code using iverilog [10] version 12 tool. There is a shell script included in the repository which helps users to install all software requirements for simulating and coding on phoeniX processor.

The simulation and execution flow is very simplified by phoeniX code executants which will be described briefly in the upcoming sections of this document. There are easy solutions to simulate assembly and C codes both Windows and Linux systems on phoeniX processor. Linux systems can use the makefile in the main directory which will do the complete process of compiling, assembling, generating firmware file and executing on the phoeniX core without any complexity and additional user interference. There is a second solution which can be used for both Windows and Linux systems, but it can only help with simulation of assembly codes on the processor. You can write and simulate your RISC-V assembly code using Venus simulator which is a Microsoft Visual Studio Code extension. This extension has an output file which includes hex instructions of the assembly code. The output will be given to a code executant python script and after that, the desired firmware will be generated and ready to execute on phoeniX processor.

The phoeniX processor streamlines the simulation and execution process, ensuring a user-friendly and intuitive experience. Its simplified flow enables effortless utilization, making it accessible and straightforward for users of all levels of expertise. Additional details regarding the integration of further descriptions can be found in the Code Executant Software section of this document, providing comprehensive insights into how the phoeniX processor incorporates advanced functionalities and features.

* 1. Required Software

As mentioned before, there is a shell script integrated with the main directory of phoeniX RV32 core repository, which helps user to install all of the required software for simulation and execution flow without any problem. In this section there are some additional information about the software in use of phoeniX core execution flow.

* Python3: Python [11] is also required to be installed on the system in order to execute the code executant and firmware generator scripts. Linux distributions such as Ubuntu has Python3 installed at the beginning by default. On Windows systems Python needs to be installed individually.
* Iverilog: it is an open-source Verilog simulation and synthesis tool used for designing and testing digital circuits. It supports the IEEE 1364-2005 Verilog standard and provides a command-line interface for compiling and simulating Verilog code. Iverilog allows users to simulate and verify the behavior of their digital designs before actual hardware implementation, making it a valuable tool for digital circuit development and verification. In phoeniX processor project we used iverilog version 12 for HDL simulation process and final execution of C and assembly codes on the processor.
* RISC-V GCC Compiler toolchain: The RISC-V GCC compiler toolchain refers to a collection of software tools that enable the compilation and development of software for RISC-V architecture-based processors. It includes a set of open-source tools, primarily based on the GCC (GNU Compiler Collection), specifically tailored for the RISC-V instruction set architecture (ISA). The RISC-V GCC compiler toolchain plays a crucial role in the development and software ecosystem surrounding the RISC-V architecture, enabling the creation of applications, firmware, and operating systems for RISC-V processors.
* Venus Simulator (Visual Studio Code): The Venus simulator is a RISC-V instruction set architecture (ISA) simulator developed by the University of Victoria. It allows users to simulate and execute RISC-V assembly language programs, providing a platform for learning, testing, and debugging RISC-V code. The Venus simulator Visual Studio Code extension is an extension specifically designed for the Visual Studio Code (VS Code) integrated development environment. This extension integrates the Venus simulator directly into the VS Code environment, offering an enhanced development experience for RISC-V programming. With the Venus simulator extension, users can write RISC-V assembly code directly in VS Code, benefit from syntax highlighting and code completion features, and seamlessly run and debug their code using the Venus simulator. The extension provides an interactive interface within VS Code, allowing users to step through their code, set breakpoints, inspect registers and memory, and observe program execution.

In References section of this document there are useful links with detailed descriptions about the tools used in this project and their installation guides. While the provided shell script smoothly automates the installation process on Linux systems, it's important to note that for Windows operating systems, the required software needs to be downloaded and installed separately.

Chapter 2

Building Blocks (Modules)

In the following chapter, we will provide a comprehensive overview of the essential components that form the foundation of the phoeniX core. We will delve into the intricate details regarding the structure and architecture of these modules, presenting them in the order of the dataflow within the pipeline.

Within this project, we have identified 9 key modules that play a crucial role in its operation. Each of these modules will be meticulously described, ensuring a thorough understanding of their individual contributions to the overall data processing pipeline. By following the logical progression of the dataflow, you will gain valuable insights into how these modules interact and collaborate to achieve the desired outcomes.

Through this comprehensive examination, you will acquire a profound knowledge of the phoeniX core's inner mechanisms, empowering you to comprehend its intricate design and appreciate the optimized performance it offers.

In the end there is a top module named phoeniX which acts as a unifying entity that brings together all the aforementioned building blocks, along with additional registers and latches, within the pipeline. By integrating these components into a cohesive unit, the phoeniX module forms the backbone of the processor, orchestrating the flow of data and executing the desired operations.

The memory interface logic is a critical component that facilitates communication between the processor and the memory subsystem. Although it is not directly included within the individual modules, its role is of utmost importance in ensuring the overall functionality of the system. The memory interface logic is implemented separately in a dedicated testbench file. In the forthcoming chapter dedicated to Memory Interface Logic, we will provide a thorough explanation of this logic, delving into its intricate workings. We will explore how it manages the flow of data between the processor and the memory subsystem, and we will highlight its significance within the broader system architecture.

Before we introduce each of this modules, there is a table included in this section which explains what is the role of each module of the core in a short and brief way. It is needed to not that in the upcoming versions of the processor, new features and new modules will be added to the phoeniX core and the mentioned modules in this chapter are the foundation of phoeniX V0.1 CPU.

Below is the list of these modules, along with concise descriptions of their critical functions and roles in phoeniX core:

|  |  |
| --- | --- |
| Modules | Description |
| Register File | Parametrized register file suitable for GP registers and CSRs |
| Arithmetic Logic Unit | ALU with support for `I\_TYPE` and `R\_TYPE` RISC-V instructions |
| Instruction Decoder | Decoding instructions and extracting `opcode`, `funct` and `imm` fields |
| Immediate Generator | Generating immediate values according to instructions type |
| Fetch Unit | Instruction Fetch logic and program counter addressing |
| Load Store Unit | Load and Store operations for aligned addresses and word size management |
| Branch Unit | Condition checking for all branch instructions |
| Address Generator | Generating address for BRANCH, JUMP and LOAD/STORE instructions |
| Hazard Forward Unit | Hazard detection and data forwarding logic in pipelined processor |

Table 1. phoeniX core modules and descriptions

* 1. Fetch Unit

Chapter 6

References

[1] RISC-V ISA

[2] GNU LICENSE

[3] IUST ERC WEBSITE

[4] Yosys

[5] Vesta

[6] ARM Cortex M

[7] PicoRV32

[8] ibex

[9] GCC compiler toolchain

[10] iverilog

[11] Python