NIOS II Emulator

Software Design Document

Team Name
Jake Ediger
Alex Michael
Alex Czarnick
Avinash Nooka
Raksharth Choudhary

Introduction

Purpose

Current environment software for the NIOS II processor has become outdated and received little maintenance throughout its life. Unreliable software accompanied by high costs of boards and chips requires a new solution to be created.

Creating an emulation of the chip and creating an environment for development brings life back to an aging system. Our system is designed to be a one and done solution for testing assembly code for the NIOS II processor without needing a separate board or micro controller. The associated GUI will provide a more efficient and user-friendly interaction.

Design Goals

The NIOS II Emulator simply has to replicate the results of the NIOS II system, not the way things are calculated. This allows for the design of the system to remain simple, relative to the actual implementation of the original NIOS II FPGA. Along with simplicity, the other design goals revolve around replicating the behavior of the NIOS II without actually using the same methods.

List of Design Goals

1. Simplicity
    a. The implementation of the system architecture will be far simpler simpler than the original NIOS II, using advances made in the field and the use of high-level programming languages.
    b. Simplicity will be gauged by the ease in which instructions are implemented, as well as how memory is managed throughout execution.
    c. The implementation language, javascript, comes with more packages than most high-level languages. Although packages can make things more complex in some situations, the use of

packages for the emulator will make many operations (e.g. parsing) much more readable. It will also allow for less complex functions and fewer lines of code.

2. Maintainability
    a. Maintaining the code may seem unnecessary due to the fact the NIOS II has been around and unchanging for some time, but it is very possible additional features will be added later on.
    b. Future features would likely consist of additional capabilities that the NIOS II does not have, such as more advanced debugging or additional instructions not included in the original set.
3. User-Friendly
    a. This NIOS II should have an appealing user interface that is easy to interact with and intuitive to use.
    b. By updating the software user interface to something more comparable to what most people use on a daily basis today, the system will be much quicker to learn and use.

Design Trade-Offs

1. Simplification of software could make it seem less sophisticated.
2. Possible race condition if functions are not synchronized.

Definitions, Acronyms, and Abbreviations

Assembly Code	low-level programming language designed for specific processor implementation
Bit	a unit of information expressed as either a 0 or 1 in binary notation
Debugging	using troubleshooting steps to find errors in programs
Environment	conditions in which the system operates
Processor	a computer chip capable of doing operations of 1s and 0s very quickly
Register	an information holder inside a processor
G.U.I.	Graphical User Interface

References

1) https://www.intel.com/content/www/us/en/programmable/documentation/iga1420498949526.html
2) https://www.draw.io/
3) http://robotics.ee.uwa.edu.au/courses/design/examples/example_design.pdf

Overview

The current software design, namely the architecture, is the tool that will be used to guide development once coding has started. There will undoubtedly be changes to the software design when problems not previously thought of are encountered, which may require a redesign of a subsystem.

Current Software Architecture

No current system architecture exists, this is the initial design document.

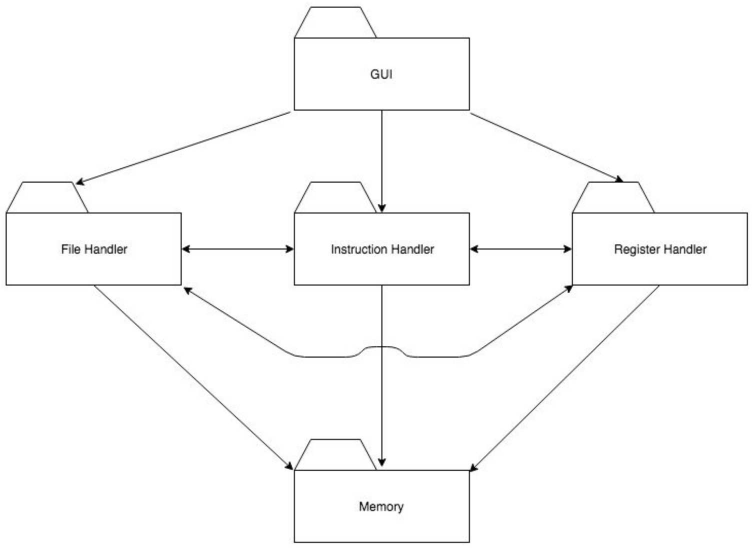
Proposed Software Architecture

Overview

The architectural style used in the NIOS II Emulator is a 3 layer webpage. The presentation layer is the first layer, consisting of the GUI its related components. The application layer is the second layer, where most of the logic and handling happens. It consists of the File Reader, Instruction Handler, and Register Handler. The third and final layer is the Database. It consists solely of the Memory subsystem.

Subsystem decomposition

1. GUI
    a. Presentation Layer
    b. The GUI gets register values for displaying to the user
    c. Give user access to restart, start, and pause functionality
    d. Initialized first with the rest of the presentation layer
2. File Reader
    a. Application Layer
    b. The File reader subsystem handles the access to the user's file system, uploading the file, and parses instructions.
    c. Validates file type, instructions, and legality of memory allocation.
3. Instruction Handler
    a. Application layer
    b. Determining what instruction is being run, and using current register values retrieved from the register handler
4. Register Handler
    a. Application layer
    b. Responsible for updating register values
    c. Getting values for instruction handler, used to perform operations for given instructions
5. Memory
    a. Database Layer
    b. Holds input file
    c. Holds instruction information
    d. Holds register values



Hardware/software mapping

NONE

Persistent data management

The persistent data will consist of hard coded memory. Memory will be represented in an array. The inputted file, instructions, registers, etc, will be stored as indices in an array. The system will do basic CRUD operation to the array values. This will inturn emulate the NIOS II memory.

Access control and security

No access control or security is necessary for the current software architecture of the NIOS II emulator.

Global software control

Synchronization will be implemented sequentially by the code. Other means of synchronization and concurrency are unnecessary as long as the code is executed in a correct sequential order.

Boundary conditions

1. File verification- before any file is run, each line of the file (as well as the file itself) will be validated to check for the following:
    a. Make sure file is a .txt
    b. Make sure all instructions are valid NIOS II instructions
    c. Make sure file does not use numbers that are out of range of the NIOS II's memory
2. Instruction Error Handling
    a. In the event of an instruction failing due to any reason, execution of the file will stop at the line of failure

3. Shutdown Behavior
    a. Shutdown behavior is minimal, the system will simply discard the file.
4. Startup Behavior
    a. The system will initialize in a webpage, and await user input

Subsystem Services

1. GUI
    a. Provides user with control of the system (Play, Pause, Restart), displays the register contents, and the pointer value.
2. File Reader
    a. Validates file and stores instructions in memory.
3. Instruction Handler
    a. Performs all instructions on the specified registers and returns the updated values.
4. Register Handler
    a. Obtains the current value of each register from memory after each instruction is performed.
5. Memory
    a. Contains the register values in a 65,536 byte array memory management system.

Packages

It is currently unknown what packages will be added once coding begins. This section will be updated with all packages used, as well as their dependencies, once it becomes clear what needs to be added. If n

Class Interfaces

TBD

Detailed Design

Class Diagrams

Glossary

Race condition: A race condition is the behavior of an electronics, software, or other system where the output is dependent on the sequence or timing of other uncontrollable events. It becomes a bug when events do not happen in the order the programmer intended.