DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF NEBRASKA—LINCOLN

# CSCE 230 Semester Project

## Team 2

**[Nathan Doher, Molly Lee, Shea Winkler]**

**11/20/2017**
**[Version 1]**

The project consists of the design of a simple processor (and additional components, as time allows) for a NIOS II Architecture using primarily VHDL.

## Revision History

[This table documents the various major changes to this document]

| Version | Description of Change(s) | Author(s) | Date |
|---|---|---|---|
| 1.0 | Phase II: Data Path Implementation and R-Type functionality<br>- First draft of Design-Document<br>-Created the title page and headers for the major sections<br>-Wrote Introduction and Scope of the Project sections | Doher, Nathan; Lee, Molly; Winkler, Shea | 3 Nov. 2017 |
| 2.0 | Phase III: Memory and Instruction Implementation<br>-Added sections on Memory and the Instruction Address Generator | Doher, Nathan; Lee, Molly; Winkler, Shea | 13 Nov. 2017 |
| 3.0 | Phase IV: Adding I/O and ARM-like Conditional Execution<br>-Changed format of authors and date<br>-Added I/O section | Doher, Nathan; Lee, Molly; Winkler, Shea | 20 Nov. 2017 |

## Contents

# Introduction

[Provide a short introduction to this document, the project and the context in which it is being developed]

This document provides a description of Team 2’s efforts to create a NIOS II processor architecture using primarily Very High Speed Integrated Circuit Hardware Description Language (VHDL) for design. Implemented will be a substantial subset of a Reduced Instruction Set Computer (RISC) instruction set architecture (ISA). The ISA to be implemented will resemble a subset of the NIOS II architecture and includes some features that are unique to Advanced RISC Machines (ARM). A goal of this project is to reduce the time needed for processes and to efficiently use hardware resources. A basic processor will result from this design and implementation, and there will be opportunities to extend its functionality (….details here coming later….).

## 1.1 Purpose of this Document

[Describe the purpose of this document; the goal(s) that its content was intended to achieve]

This document should provide a guide for explaining the design decisions Team 2 used in creating its simple processor. It will also elaborate on complicated details and explain operation of the different parts of the processor and any additional parts Team 2 includes.

## 1.2 Scope of the Project

[Describe the scope of the project, what features and functionality it covers (at a high-level). Describe the problem statement and context in which this project is being developed. Who is it for, what is it for, etc.? You may also explicitly indicate what is *not* within the scope—other potential pieces of the overall project that are not covered by this document]

This project is to be completed in order to successfully complete CSCE 230 at UNL. The processor will be able to do basic functions (simple math, Boolean logic, sequential and conditional analysis). Additional functions may be added (e.g., replace polling with interrupting) as time allows. The processor will have nowhere near the functionality nor complexity of a modern processor found on a computer chip.

In Phase I, Team 2 worked to create the control unit for the processor. The control unit is responsible for setting certain flag values that affect the way in which the processor functions. Phase II will implement the control unit to initialize a functioning processor.

In Phase II, Team 2 created the Data Path File for the project and compiled a basic processor for the first time. In this phase, it was only necessary to implement Register-type (R-type) instructions. This was the simplest version of the processor that functions. The successive phases have been built from the working product of this phase.

In Phase III, Team 2 modified the Data Path and the Control Unit to add new R-type, D-type and B-type instructions, including JR, CMP, LW, SW, ADDI, B and BAL. Team 2 also integrated the memory interface and instruction address generator into the existing processor in this phase. Phase IV will build off Phase III and implement I/O.

In Phase IV, Team 2 stared at the computer again for too many hours, making small changes that only solved minute problems. The main purpose of Phase IV was to add basic I/O and ARM-like execution [2].

## 1.3 Definitions, Acronyms, Abbreviations

### 1.3.1 Definitions

[Define any terms that require a definition—domain specific terms, non-standard terms, or terms that are used in non-standard ways]

ADD instruction – Adds the Values from Two Registers and Places Result in the Destination Register
AND instruction – Returns '1' if All Inputs are '1' and Returns '0' Otherwise
B-type instruction – a branch instruction for the processor. B-type instructions include branch (B) and branch and link (BAL) instructions
D-type instruction – a data transfer instruction for the processor which transfers data between the memory and the CPU. D-type instructions include load word (LW), store word (SW), add immediate (ADDI) and subtract immediate (SUBI)
OR instruction – Returns '1' if at least One Input is '1' and Returns '0' Otherwise
R-type instruction – an instruction for the processor that is passed to the processor from a register memory location where the instruction is stored. R-type instructions include arithmetic logic unit (ALU) instructions, shift left logical (SLL), compare (CMP) and jump register (JR)
SUB instruction – Subtracts the Values from Two Registers and Places Result in the Destination Register

### 1.3.2 Abbreviations & Acronyms

[Define all abbreviations and acronyms used in this document here. This relieves you of the need to define such things within the context of the document itself and provides an easy reference for the reader.]

ADDI – Add Immediate
ALU – Arithmetic Logic Unit
ARM – Advanced RISC Machines
B – Branch
BAL – Branch and Link
B-type – Branch Type
CMP – Compare
CPU – Central Processing Unit
CSCE – Computer Science and Computer Engineering
D-type – Data Type
I/O – Input/Output
ISA – Instruction Set Architecture
JR – Jump Register
LW – Load Word
NIOS – Netware Input/Output System
RISC – Reduced Instruction Set Computer
R-type – Register Type
SLL – Shift Left Logical

SUBI – Subtract Immediate
SW – Store Word
VHDL – Very High Speed Integrated Circuit Hardware Description Language
XOR – Exclusive Or

## 2. Overall Design Description

[Provide an overall summary/description of the project. Identify the major design components, technologies, etc.]

This project is divided into five main parts. Phase I consisted of building a 16x16-bit register file, a 16-bit ALU and a control unit. Phase II included integrating the register file, ALU, control unit and other components into a basic data path for executing R-type operations, including Add, Sub, And, Or, and Exclusive Or (XOR). Phase III included adding additional R-type, D-type, and B-type instructions, as well as implementing the memory interface and the instruction address generator. Phases IV consisted of implementing I/O and ARM-like execution. Phases V-VI have yet to be completed.

### 2.1 Alternative Design Options

[If applicable, describe and discuss alternative design options that you considered and discuss why they were not chosen. What advantages and disadvantages do the alternatives provide and what advantage/disadvantages do the chosen design elements provide. Provide some justification for why the chosen elements' advantages/disadvantages outweighed the alternatives]

## 3. Detailed Component Description

In this section, we will discuss the components used in each phase of the project.

### 3.1 Register File

The 16x16 bit register file holds general purpose registers that operate as a small fast memory block. Two registers are generally read by the processor at the same time at outputs A and B. Two fields in the IR act as select bits for the register file to determine which two registers to select, while input C and a corresponding select bit determine the register to be written to. The register file was developed in phase I.

#### 3.1.1 Component Testing Strategy

[This section will describe your approach to testing this particular component.]

### 3.2 Arithmetic Logic Unit

The ALU was designed in Phase I. This 16-bit ALU could produce the NZVC flags [2].

#### 3.2.1 Component Testing Strategy

[This section will describe your approach to testing this particular component.]

### 3.3 Control Unit

The control unit was also designed in Phase I, and updated in subsequent phases. In Phase II, most of the R-type instructions were added. In Phase III, the remainder of the R-type instructions, as well as D-type and B-type instructions were added. In Phase IV, the conditional bits of the instructions were implemented. Team 2 used an execute_enable bit to check whether the conditional bits matched the flags from the PS. If they matched, the execute_enable bit was turned on, and this execute_enable bit must be on for stages 3-5 of the processor to execute.

### 3.3.1 Component Testing Strategy

[This section will describe your approach to testing this particular component. Describe any test cases, unit tests, or other testing components or artifacts that you developed for this component. What were the outcomes of the tests? Did the outcomes affect development or force a redesign?]

## 3.4 Data Path

[This section should detail your Data Path—the components and how they relate to each other. It is highly recommended that you document these elements using tables, UML diagrams, and other visually-informative methods. Figures and tables should have proper captions and be referenced in the main text just like in **Error! Reference source not found.**. You should provide subsections to organize your p resentation as applicable.]

In Phase II, the DataPath was first implemented. This DataPath was updated in every subsequent stage of the project.

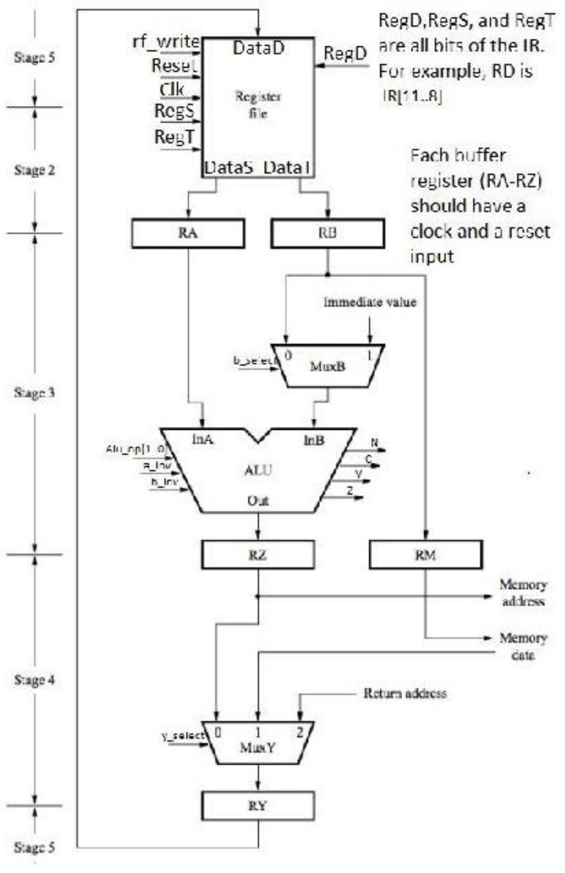

Figure 1: This is the Data Path that was implemented for Phase II of the CSCE 230 Project. [3]

### 3.4.1 Component Testing Strategy

[This section will describe your approach to testing this particular component. Describe any test cases, unit tests, or other testing components or artifacts that you developed for this component. What were the outcomes of the tests? Did the outcomes affect development or force a redesign?]

Phase I: A (ControlUPDATE.do) testing file was provided and the control unit was tested using the do file to ensure it was working properly.

Phase II: A (PhaseTwo.do) testing file was created by Team 2 to test the Data Path file (and functioning processor) to ensure that it worked properly.

Phase III: A (PhaseThree.do) testing file was created by Team 2 to test the new Data Path file after implementing the Memory Interface and Instruction Address Generator. Team 2 also created a (MemoryInitialization.mif) instruction file to provide the instructions to memory.

Phase IV: After archiving the Datapath project in Quartus, the same (PhaseThree.do) and (MemoryInitialization.mif) files were used for testing, but they were completely redesigned to be compatible with the instructions for the Phase IV checkoff [5]. The Altera DE1 board was also used for testing the green LEDs and pushbuttons. Although the program was able to run through the entirety of the 14 instructions for the checkoff in ModelSim, the program was unsuccessful at executing B-type instructions.

## 3.5 Memory Interface

[This section will be used to detail phase III where Team 2 implemented the Memory Interface.]

In Phase III, the Memory Interface was implemented into the Data Path. Many of the memory components were provided to Team 2 on Canvas, and Team 2 solely had the responsibility of downloading and integrating these components into the Data Path, as well as updating some of these components to meet the checkoff requirements. The provided components for Phase III included: Adder, Const, InstructionAddressGenerator, MainMemory, MemoryInitialization.mif, MemoryInterface, MuxINC, MuxPC, PC and PC_temp. All of these files were found in the memFilesProcessor.zip file on Canvas.

The ControlUnit was updated as well in this phase.

### 3.5.1 Component Testing Strategy

[This section will describe your approach to testing this particular component. Describe any test cases, unit tests, or other testing components or artifacts that you developed for this component. What were the outcomes of the tests? Did the outcomes affect development or force a redesign?]

A .do File and .mif File were created to test the Memory Interface. See the DataPath section for more information on these test files.

## 3.6 Instruction Address Generator

[This section documents the Instruction Address Generator.]

In Phase III, the Instruction Address Generator was implemented into the Data Path. The components were given as block diagram components. By using the component generator, the inputs and outputs were then able to be mapped to the Data Path in order for the Instruction Address Generator to be initialized. To fully connect the Instruction Address Generator and new Mux was needed named MuxMa. MuxMa selects either from the register RZ or the Instruction Address Generator.

#### 3.6.1 Component Testing Strategy

[This section will describe your approach to testing this particular component. Describe any test cases, unit tests, or other testing components or artifacts that you developed for this component. What were the outcomes of the tests? Did the outcomes affect development or force a redesign?]

### 3.7 I/O

[This section will be used to detail phase IV and the I/O that you designed—how it conformed to the requirements, how it worked, other tools or methods that you designed to assist, how it handles corner cases and the expectations or restrictions that you've placed on the user of the I/O. In earlier phases this section may be omitted or a short note indicating that details will be provided in a subsequent revision of this document.]

I/O was implemented in Phase IV. The IO_MemoryInterface component and MuxMDO were added. The select bit for MuxMDO and the first 4 bits from the MemoryAddress were added to the ControlUnit component in the DataPath. These 4-bits from the MemoryAddress were used to determine if I/O was used. If I/O was used, it was to be selected from the select bit for MuxMDO. The signal going to the IO_MemoryInterface was changed to RZ instead of going through the MuxMA. Instead of pulling 1 from MuxMA, we took the signal directly from RZ_out.

An IF statement was added to the ControlUnit to control what happens during the falling edge of the clock cycle. Within this IF statement, the execute_enable signal was added to the ControlUnit. This execute_enable signal tested the conditional statements to determine if stages 3-5 should execute or not.

#### 3.7.1 Component Testing Strategy

[This section will describe your approach to testing this particular component. Describe any test cases, unit tests, or other testing components or artifacts that you developed for this component. What were the outcomes of the tests? Did the outcomes affect development or force a redesign?]

## 4. Additional Material

[This is an optional section in which you may place other materials that do not necessarily fit within the organization of the other sections.]

In Phase VI, bonus material such as an Assembler and/or additional I/O instructions will be implemented. Coming soon…

## 5. Bibliography

[This section will provide a bibliography of any materials, texts, or other resources that were cited or referenced by the project and/or this document. You *must* consistently use a standard citation style such as APA or MLA (good reference: http://www.cws.illinois.edu/workshop/writers/citation/).]

[1] *Lab 9 – Control Unit.* (2017). Retrieved 18 October, 2017, from https://canvas.unl.edu/courses/19800/files/folder/Lab/Week%209?preview=1207084

[2] *CSCE 230 Project Overview*. (2017). Retrieved 25 October, 2017, from https://canvas.unl.edu/courses/19800/files/folder/Project/Overview%20and%20Helpful%20docs?preview=1285596

[3] *CSCE 230 Project Part II Datapath implementation*. (2017). Retrieved October 25, 2017, from https://canvas.unl.edu/courses/19800/files/folder/Project/Part%202?preview=1245925

[4] *CSCE 230 Project Part III Memory and instruction implementation*. (2017). Retrieved November 5, 2017, from https://canvas.unl.edu/courses/19800/files/folder/Project/Part%203?preview=1285586

[5] *CSCE 230 Project Part IV Adding I/O and ARM-like Conditional Execution*. (2017). Retrieved November 15, 2017, from https://canvas.unl.edu/courses/19800/files/folder/Project/Part%204?preview=1353550

(Do we need to list all the provided files that were on Canvas?- How do we cite a .zip file??)

Book Example:

[3] Eckel, B. (2006). *Thinking in Java* (4th ed.). Prentice Hall.

Table or Image Example:

**(Insert caption here)**

(Insert table here)