University of Burgundy

Real-Time Imaging Project

Kimberly Grace Sevillano Colina,
Zhilling Qian, Yanyin Yao, Abdullahi Atanda

Contents

1 Introduction

1.1 Background

Field-Programmable Gate Arrays (FPGAs) are versatile and powerful tools in the field of digital design and embedded systems. They offer the flexibility to program and reprogram the hardware for a wide range of applications. Very High-Speed Integrated Circuit Hardware Description Language (VHDL) is a key language used for describing the behavior and structure of electronic systems, particularly in FPGA development. The combination of FPGAs and VHDL allows for significant experimentation and innovation in digital design.

In this project, the Nexys4 DDR FPGA board is utilized as the primary development platform. This board is widely recognized in educational and prototyping environments for its robust feature set and Xilinx Artix-7 FPGA, making it an ideal choice for implementing and testing digital designs.

1.2 Project Significance

Understanding and applying VHDL in microprocessor design is crucial for anyone entering the field of digital electronics and embedded systems. This project not only provides hands-on experience with VHDL but also offers insight into the fundamental concepts of CPU architecture and design. It bridges the gap between theoretical knowledge and practical application, using the Nexys4 DDR board as a testbed.

1.3 Objectives

The primary objectives of this project include:

- Designing a simple microprocessor using VHDL, with implementation on the Nexys4 DDR FPGA board.
- Implementing core functionalities such as an accumulator, program counter, and basic instructions (LDA, STA, ADD, etc.).
- Exploring potential extensions and applications of the processor design within the constraints of the Nexys4 DDR platform.
- Addressing the challenges encountered in FPGA-based microprocessor design and proposing viable solutions.

2 Materials and Methods

2.1 Nexys4 DDR FPGA Board

The Nexys4 DDR board is a pivotal component in this project. It's a comprehensive digital circuit development platform based on the Xilinx Artix-7™ FPGA, known for its high capacity and performance. The board's extensive external memories, variety of ports, and built-in peripherals like sensors and a speaker amplifier, make it suitable for a wide range of digital designs.

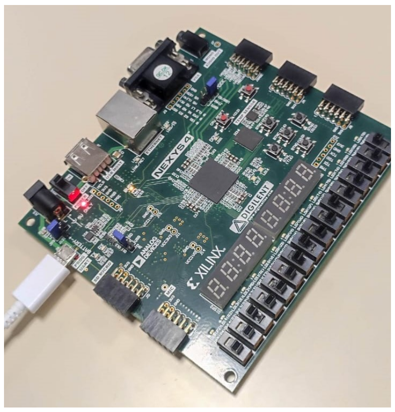
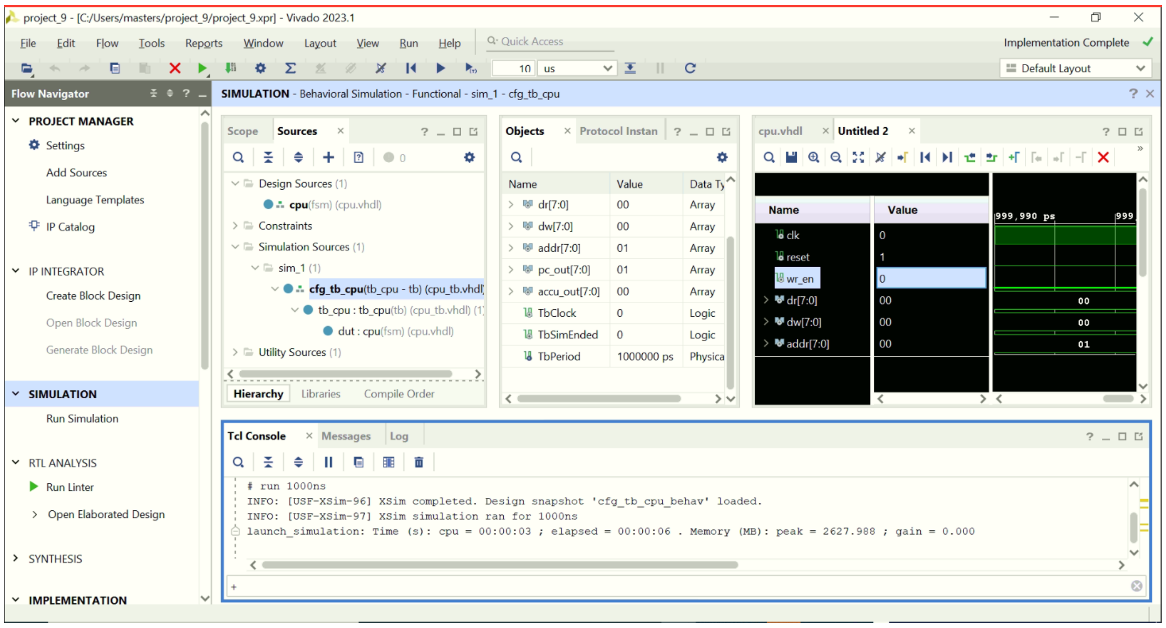


Figure 1: Nexys4 DDR board.

2.2 VHDL Development Environment

The VHDL development environment was set up to design and simulate the microprocessor. This included configuring the necessary software tools and preparing the development workflow for efficient design and testing.


project_9 - [C:/Users/masters/project_9/project_9.xpr] - Vivado 2023.1
Flow Navigator
SIMULATION - Behavioral Simulation - Functional - sim_1 - cfg_tb_cpu
Sources
Objects
Untitled 2
Tcl Console

Name	Value	Data Type
dr[7:0]	00	Array
dw[7:0]	00	Array
addr[7:0]	01	Array
pc_out[7:0]	01	Array
accu_out[7:0]	00	Array
TbClock	0	Logic
TbSimEnded	0	Logic
TbPeriod	1000000 ps	Physica

```
# run 1000ns
INFO: [USF-XSim-96] XSim completed. Design snapshot 'cfg_tb_cpu_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:03 ; elapsed = 00:00:06 . Memory (MB): peak = 2627.988 ; gain = 0.000
```

Figure 2: Integrated development and simulation environment used for the CPU design.

2.3 Initial CPU Design Approach

The initial approach to designing the CPU involved defining the basic structure and functionality in VHDL. This included setting up the entity and architecture of the CPU, focusing on the interface, op-codes, and the finite state machine (FSM) logic. The specifics of the architecture and implementation are detailed in Section 3.

3 Designing a Simple Microprocessor in VHDL

3.1 Overview of the Microprocessor Design

The design of the microprocessor aimed to create a simple yet functional CPU using VHDL. The focus was on achieving basic computational capabilities, suitable for educational and prototyping purposes. The design process involved defining the architecture, implementing the instruction set, and ensuring compatibility with the Nexys4 DDR FPGA board.

3.2 CPU Architecture

The architecture of the CPU was designed to be straightforward yet capable of performing fundamental operations. It includes an accumulator for arithmetic operations, a program counter for instruction sequencing, and a memory interface for data storage and retrieval.

3.2.1 Accumulator and Program Counter

The accumulator is an 8-bit register used for arithmetic and logic operations. It temporarily stores the results of these operations. The program counter (PC) is another crucial 8-bit register that keeps track of the memory address of the next instruction to be executed. The PC ensures the sequential execution of instructions, incrementing after each operation unless altered by specific instructions like jumps.

3.2.2 Memory Interface and Internal Registers

The CPU interfaces with memory through an 8-bit address bus and separate 8-bit input and output data buses. This design allows for efficient data transfer between the CPU and memory. Internal registers, including the accumulator and program counter, facilitate the CPU's operations, holding data and addresses relevant to the current instruction.

3.2.3 Implementation of Basic Instructions

The CPU supports a basic set of instructions, each encoded with a unique op-code:

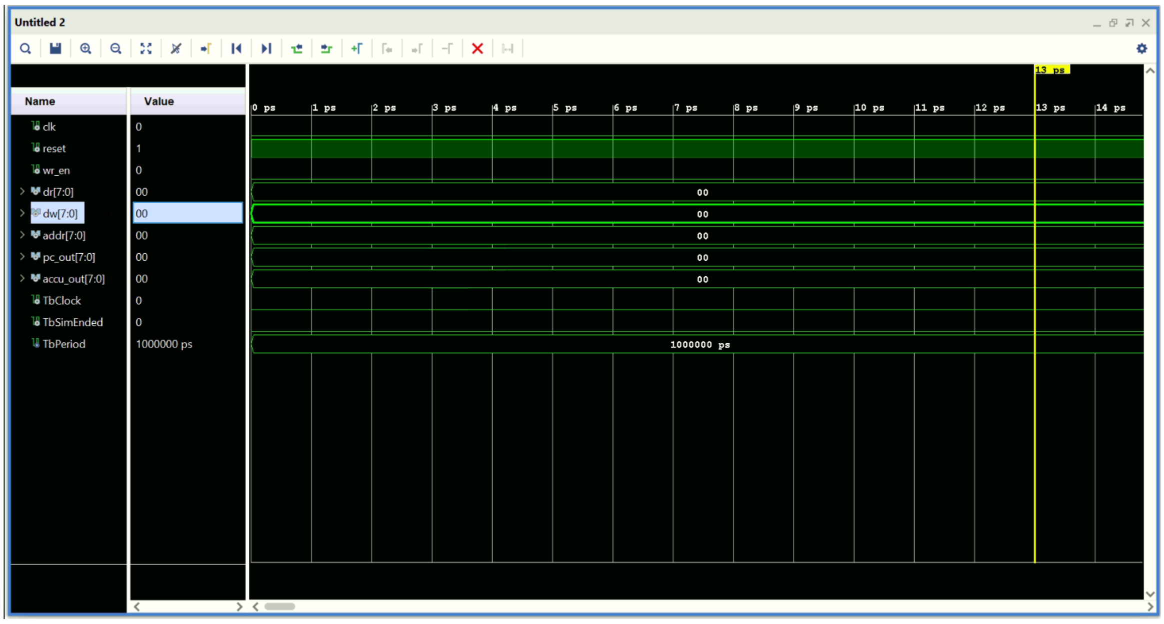
- **LDA (Load Accumulator)**: Loads data from a specified memory address into the accumulator.
- **STA (Store Accumulator)**: Stores the content of the accumulator at a specified memory address.
- **ADD**: Adds data from a specified memory address to the accumulator.
- **JNC (Jump if No Carry)**: Jumps to a specified address if the carry flag is not set.
- **JMP (Jump)**: Unconditionally jumps to a specified address.

These instructions form the basis of the CPU's functionality, enabling it to perform basic data processing tasks.

The VHDL implementation of the CPU reflects this architecture. The finite state machine (FSM) within the VHDL code controls the operation of the CPU, transitioning between states based on the current instruction and its operands. This FSM approach allows for clear and manageable CPU operation sequencing.

3.3 Testing and Validation

The CPU design was rigorously tested using simulation tools and test benches. These tests ensured that each instruction was executed correctly and that the CPU behaved as expected in various scenarios.



Name	Value
clk	0
reset	1
wr_en	0
dr[7:0]	00
dw[7:0]	00
addr[7:0]	00
pc_out[7:0]	00
accu_out[7:0]	00
TbClock	0
TbSimEnded	0
TbPeriod	1000000 ps

Figure 3: Waveform simulation of the basic CPU design showing the timing of various signals during operation.

3.4 CPU Design

3.4.1 CPU Top-Level Entity (cpu_top)

This entity serves as the top module, integrating various components and interfaces of the CPU design. Key features and ports include:

- **Ports**:
  - **clk100MHz (Input)**: Primary system clock at 100MHz for synchronizing operations.
  - **clk (Input)**: Secondary clock input, potentially used for testing or specific operations.
  - **switches (Input)**: User inputs to control or modify CPU behavior.
  - **reset (Input)**: Resets the system to an initial state.
  - **led (Output)**: Indicator LEDs for visual feedback.
  - **an (Output)**: Drives anodes of the 7-segment display.
  - **CA, CB, CC, CD, CE, CF, CG (Output)**: Controls individual segments of the 7-segment display.

  Internal Components:
  - **Clock Divider**: Generates a 50MHz clock from the primary 100MHz clock.
  - **Debounce Logic**: Ensures stable clock input by debouncing the manual clock signal.
  - **Display Logic**: Manages the output display based on the CPU state and inputs.
  - **CPU Instance**: Core computational unit processing instructions.
  - **Memory Instance (procram)**: Handles memory read/write operations and data storage.
- **Functionality**: This module is crucial for orchestrating the overall functionality of the CPU, managing inputs, processing, memory interaction, and output display.

3.5 Component Overviews

3.5.1 disp4

The **disp4** module is responsible for driving the 7-segment display. It interprets input data to control individual segments, displaying the required information:

- **Ports**:
  - **clk (Input)**: Clock input for synchronization.
  - **disp_in (Input)**: 16-bit data input for display content.
  - **an (Output)**: Controls which digit is active on the display.
  - **CA to CG (Output)**: Controls individual segments for character display.
- **Functionality**: Utilizes a procedural approach to determine which segments to illuminate based on the input data.

3.5.2 Procram

The **procram** module serves as the memory component:

- **Ports**:
  - **A (Input)**: Address bus for memory access.
  - **DI (Input)**: Data input for memory write operations.
  - **DO (Output)**: Data output for memory read operations.
  - **RESET, WR_EN, CLK (Input)**: Control signals for reset, write enable, and clock.
- **Functionality**: Manages data storage and retrieval, differentiating between RAM and ROM operations.

3.5.3 CPU Entity (cpu)

The CPU in this project is designed as a simple yet effective processing unit, built around a finite state machine (FSM). The FSM controls the CPU's operation, handling instruction execution through a series of states:

1. **Instruction Fetch**: The first step in the CPU cycle, where an opcode is fetched from memory.
2. **Instruction Decode**: The fetched opcode is decoded to determine the required operation.
3. **Execute**: The operation specified by the opcode is executed. This could involve arithmetic operations, data movement, or control flow changes.

3.5.4 Opcode Implementation

Each opcode represents a specific instruction for the CPU. The opcodes implemented in this CPU are:

- **LDA (Load Accumulator)**: Loads a value into the accumulator from memory.
- **STA (Store Accumulator)**: Stores the value in the accumulator to a specified memory location.
- **ADD**: Adds a value from memory to the accumulator.
- **JNC (Jump if No Carry)**: Jumps to a specified memory address if no carry is generated in the previous operation.
- **JMP (Jump)**: Unconditional jump to a specified memory address.

3.5.5 State Machine Operation

The state machine within the CPU is the heart of its operation. It ensures that instructions are executed in the correct order and that the data is processed correctly. The FSM transitions between different states based on the current instruction and the status of the CPU components, such as the accumulator and program counter.

3.5.6 Integration with Other Modules

The CPU interfaces with the **procram** module for memory operations and the **disp4** module for displaying output. It sends and receives data to and from these modules, coordinating the overall operation of the microprocessor system.

3.5.7 Usage in FPGA

Implementing this CPU on an FPGA allows for a flexible and efficient design, suitable for educational purposes and fundamental microprocessor applications. The programmable nature of FPGAs makes it possible to modify and extend the CPU's capabilities as required.

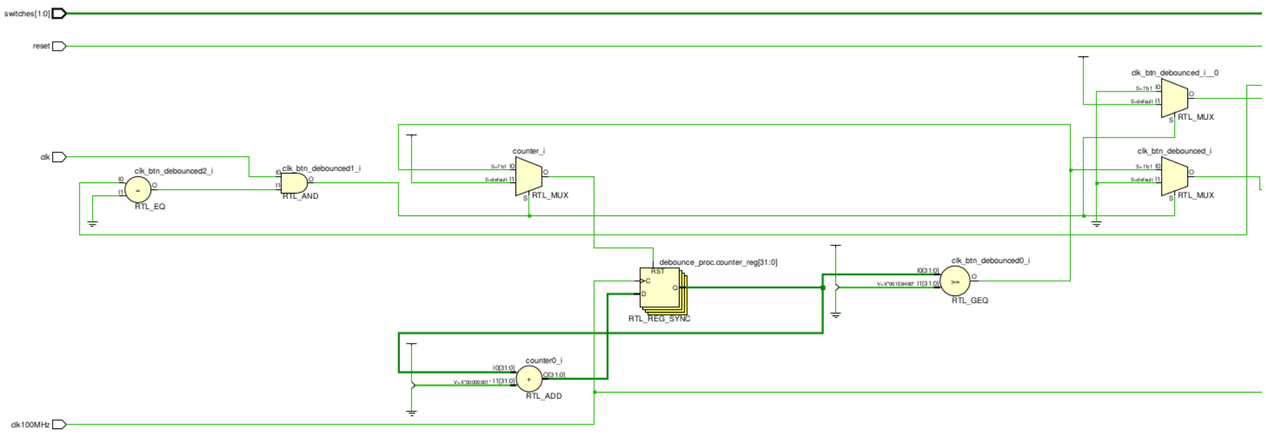


Figure 4: Schematic CPU board part 1.

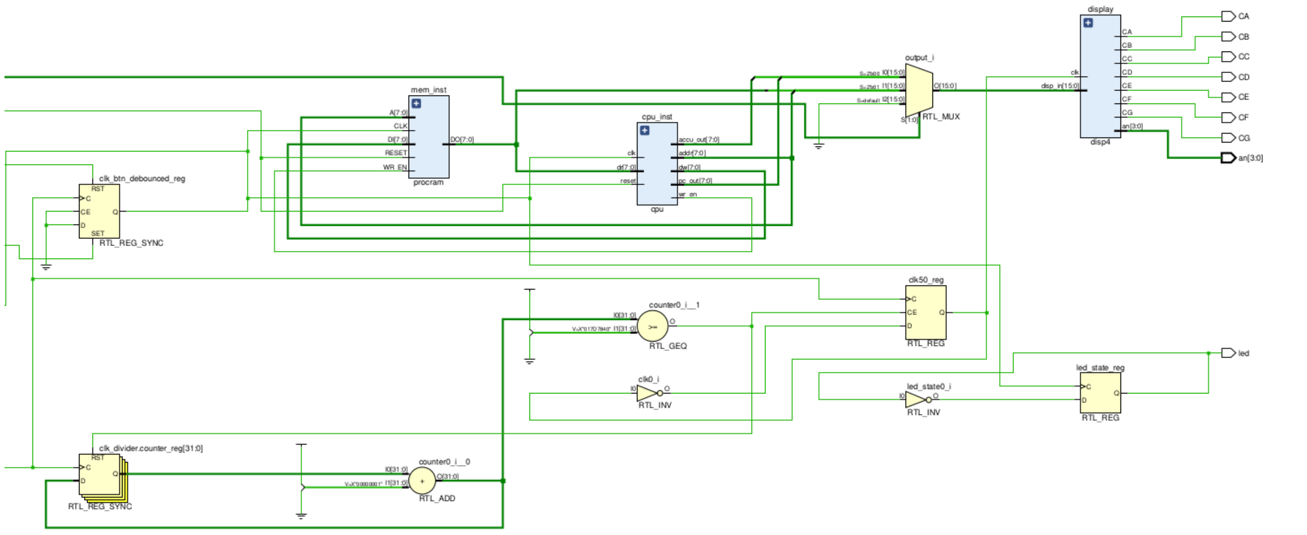


Figure 5: Schematic CPU board part 2.

4 Challenge Option 1: Emulating an Old CPU

This Challengue involves the FPGA implementation of the Intel 8080 microprocessor, presenting a softcore version of this historic CPU. The focus is on replicating the essential functionalities of the Intel-8080 using FPGA technology.

4.1 Intel 8080 Microprocessor FPGA Implementation: Core VHDL Files

The core VHDL files are crucial for the Intel 8080 microprocessor FPGA implementation. These files define the architecture, functionality, and behavior of the microprocessor within the FPGA environment. The key files include:

- **i8080_core.vhd**: The main file that defines the central processing unit (CPU) of the 8080 microprocessor.
- **i8080_alu.vhd**: Specifies the arithmetic logic unit (ALU), responsible for all arithmetic and logic operations.

- **i8080_register_file.vhd**: Manages the registers within the CPU, crucial for operations and data storage.
- **i8080_microcode.vhd**: Contains microcode instructions for the CPU, directing its operation.
- **i8080_memory_sim.vhd**: Simulates the memory component interfacing with the CPU.

The i8080_core.vhd file is a pivotal component in the FPGA implementation of the Intel 8080 microprocessor. It encapsulates the core logic and functionalities of the CPU. This file is designed using VHDL-93/02 standards and is integral to the softcore replication of the Intel 8080 microprocessor.

4.1.1 Ports and Signals

- **Ports**: Includes standard control signals like clk (clock), en (enable), and reset. It interfaces with the external environment through fin (input functions) and fout (output functions).
- **Internal Signals**: Utilizes various signals such as cs (control signals) and cpuinfo for internal control and data flow.

4.1.2 Key Components

- **ALU Interface**: Connects to the arithmetic logic unit, crucial for arithmetic and logic operations.
- **Register File**: Manages the registers, critical for data storage and operations.
- **Memory Control**: Handles memory read and write operations, interfacing with external memory components.

4.1.3 Functionality

The i8080_core.vhd file meticulously replicates the behavior of the Intel 8080 CPU. It manages instruction execution, data processing, memory interaction, and control signal management. The architecture is divided into various processes that handle specific tasks like instruction decoding, ALU operations, and register management.

5 Project Management and Author Contribution

Each team member's role was integral to the project, combining technical skills, research, strategic planning, and project management to achieve the successful design and implementation of the microprocessor.