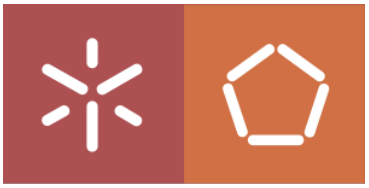

Universidade do Minho
Escola de Engenharia

ESRG

EMBEDDED
SYSTEMS
RESEARCH GROUP

Master's in Industrial Electronics and Computers Engineering

University of Minho

8051 on FPGA

Embedded Systems Master's Degree

Authors: Embedded Systems 24/25 Class

Professor: Adriano Tavares

2024/2025

Contents

List of Figures

List of Tables

Acronyms

UART Universal asynchronous receiver/transmitter

Chapter 1

Introduction

1.1 Problem Statement

This project involves the design and implementation of an 8051-compatible microcontroller using Verilog HDL on a ZyboZ7 FPGA development board. The 8051 architecture is well known by the class and serves as a practical foundation for applying digital design concepts. The project is distributed across the entire class, with themes rotating every two weeks. In addition to the core components—peripherals, single-cycle CPU, and pipelined CPU—a final theme will involve the development of a simple C compiler targeting the designed microcontroller.

The project is divided into two main versions of the microcontroller architecture:

1. **Single-Cycle Version**
2. **Pipelined Version**

Each version must be capable of executing a defined subset of the 8051 instruction set and interfacing with a set of peripherals also developed by the class. The final implementation should demonstrate correct execution of programs and proper interaction with these peripherals through memory-mapped I/O or equivalent interfacing strategies.

Both architectures must include:

- Instruction fetch, decode, execution, and write-back stages (as applicable)
- A register file, ALU, and control logic
- Memory interface for instruction and data access
- Access to a shared peripheral interface/bus

This project will be completed in phases, with key milestones to be defined by the Professor and class schedule. Documentation, simulation, synthesis, and on-board testing are required deliverables.

The C-compiler for this 8051 will be presented in a separate documentation.

This project follows a **waterfall development approach**, where each stage of the system's creation—analysis, design, implementation, testing, and deployment—is.addressed in sequence. The purpose of this chapter is to analyze the functional and structural requirements of the 8051-compatible microcontroller and its subsystems, which include the single-cycle CPU, pipelined CPU, peripheral interfaces, and C compiler backend.

1.2 Requirements

1.3 Constraints

1.4 System Overview

Chapter 2

Analysis

Design Implementation

Given that the 8051 architecture is well known by the class, the analysis focuses on translating its key functional elements into hardware modules suitable for FPGA implementation. Emphasis is placed on the modular division of tasks across the class, the interfaces between components, and the timing and control flow requirements that will shape later design decisions.

Each component is considered in terms of:

- Instruction set requirements and execution flow
- Memory access patterns and latency handling
- Peripheral interaction and I/O protocols
- Control logic and state transitions

This foundational understanding guides the design and verification steps in the subsequent phases of the waterfall model.

2.1 FSM

Design Implementation

In the context of the 8051 microcontroller project, a **Finite State Machine (FSM)** is a key design structure used to manage the control flow of sequential logic elements, particularly in the CPU and peripheral modules. FSMs are essential for defining how a system transitions between various operational states based on inputs and internal conditions.

1. CPU Control Unit

In both the single-cycle and pipelined implementations of the 8051 CPU, FSMs are employed in the control unit to govern the sequence of operations. The instruction execution path is typically broken into the following stages:

1. **Fetch (IF)** – Instruction is fetched from instruction memory.
2. **Decode (ID)** – Instruction is decoded, and control signals are prepared.
3. **Memory Access (MEM)** – Any required memory read or write is performed.
4. **Execute (EX)** – The ALU performs computation or address resolution.
5. **Write-back (WB)** – Results are written back to the register file.

The FSM transitions between these states on each clock cycle, generating appropriate control signals to activate the required components. This staged breakdown allows the design to support more complex operations and prepares the structure for pipelining.

2. Peripherals

Each peripheral device (e.g., UART, timer, GPIO) includes its own FSM to manage internal behavior such as data transfer, mode switching, and status updates. For example, a UART FSM might transition through states such as `Idle`, `Start`, `Data Transfer`, `Stop`, and `Ready`.

3. Compiler Integration and Memory Loader

FSMs can also be used in modules that handle:

- Instruction or data memory loading prior to program execution.
- Interaction with a C compiler backend or debugger interface.

These FSMs typically control the sequencing of memory writes, flag signaling, or program counter initialization.

Summary

FSMs offer a reliable, modular, and synthesize approach to managing sequential behavior in both CPU and peripheral subsystems. Their deterministic nature ensures predictable transitions and clear operational flow, which is essential for the complexity of a multistage microcontroller like the 8051.

2.2 Program Counter

Design Implementation

2.3 Fetch

Design Implementation

2.4 Decode

Design Implementation

2.5 Memory Access

Design Implementation

2.6 Execute

Design Implementation

2.7 Writeback

Design Implementation

2.8 Peripherals

Design Implementation

2.8.1 Interrupt Controler

Design Implementation

2.8.2 GPIO

Design Implementation

2.8.3 PS2

Design Implementation

2.8.4 Timer

Design Implementation

2.8.5 UART

Design Implementation

2.8.6 VGA

Design Implementation

2.9 Pipeline

Design Implementation

2.9.1 Cache

Design Implementation

2.9.2 Hazard Control

Design Implementation

Chapter 3

Design

Analysis Implementation

3.1 FSM

Analysis Implementation

3.2 Program Counter

Analysis Implementation

3.3 Fetch

Analysis Implementation

3.4 Decode

Analysis Implementation

3.5 Memory Access

Analysis Implementation

3.6 Execute

Analysis Implementation

3.7 Writeback

Analysis Implementation

3.8 Peripherals

Analysis Implementation

3.8.1 Interrupt Controler

Analysis Implementation

3.8.2 GPIO

Analysis Implementation

3.8.3 PS2

Analysis Implementation

3.8.4 Timer

Analysis Implementation

3.8.5 UART

3.8.6 VGA

Analysis Implementation

3.9 Pipeline

Analysis Implementation

3.9.1 Cache

Analysis Implementation

3.9.2 Hazard Control

Analysis Implementation

Chapter 4

Implementation

Analysis Design

4.1 FSM

Analysis Design

4.2 Program Counter

Analysis Design

4.3 Fetch

Analysis Design

4.4 Decode

Analysis Design

4.5 Memory Acess

Analysis Design

4.6 Execute

Analysis Design

4.7 Writeback

Analysis Design

4.8 Peripherals

Analysis Design

4.8.1 Interrupt Controler

Analysis Design

4.8.2 GPIO

Analysis Design

4.8.3 PS2

Analysis Design

4.8.4 Timer

Analysis Design

4.8.5 UART

Analysis Design

4.8.6 VGA

Analysis Design

4.9 Pipeline

Analysis Design

4.9.1 Cache

Analysis Design

4.9.2 Hazard Control

Analysis Design

Chapter 5

Conclusion

5.1 References

livro do Adriano
net
chapgpt

a santa trindate