The University of Manchester
Department of Computer Science
Project Report 2024

My Little Operating System

Author: Thomas H. Jones

Supervisor: Dr James Garside

Abstract

My Little Operating System

Author: Thomas H. Jones

Test

Supervisor: Dr James Garside

Acknowledgements

Thanks page?

Contents

1 Introduction **5**
- 1.1 Aims . . . 5
- 1.2 Motivations . . . 5
- 1.3 Project Plan . . . 6
- 1.4 Report Outline . . . 6

2 Background **7**

3 Design and Implementation **8**
- 3.1 Toolchain . . . 8
- 3.2 Board Configuration . . . 8
  - 3.2.1 Clock settings . . . 8
- 3.3 Processes . . . 10
- 3.4 Scheduler . . . 10
- 3.5 Memory Management . . . 10
- 3.6 IO . . . 10

4 Outcome **11**

5 Evaluation **12**

6 Conclusions and further work **13**

List of Figures

3.1 High Frequency Clock Diagram 9
3.2 Low Frequency Clock Diagram 9

List of Tables

Chapter 1

Introduction

1.1 Aims

This project will be based upon the following goals

- A basic implementation of processes, which can run arbritrary user code
- A scheduler that schedules processes interactively
- Memory management, to ensure that user code has limited access to memory
- IO, for example a text input and output, or other visual methods of output

1.2 Motivations

This project is an exploration of concepts and methods of operating systems, with the overall goal to implement these concepts to create a simple operating system. In addition to this, the project will be done using a board that implements the open source, RISC-V processor architecture, which will require research into the operation of RISC-V, as well as any implementation specific features to the board in use. The board used will be the Sifive Hifive1 Rev B. The main goal is the implementation of an interactive scheduler to run multiple processes at once, as this is something that will not be implemented unless part of an operating system.

1.3 Project Plan

The initial part of this project will be detemining an appropriate microcontroller board and setting up the toolchain required to compile and load code onto the board. Following this, the basic components of the system will be set up, such as the interrupt handler. Simple IO will be setup, to allow for ease of debugging, which will then be reimplemented later to be consistent with other IO elements. The implementation of processes will be split into multiple parts, such as the storage of process information, the creation of processes, the scheduling of processes and the deletion of processes. As part of this, memory management will be developed at the same time, as a basic implementation is required for user code to function.

1.4 Report Outline

- Introduction: This chapter gives the basic overview of the planned project
- Background: This chapter will introduce key concepts to enable a more comprehensive understanding of the projects details
- Design: This chapter will outline the key design decisions made before and during the project
- Implementation: This chapter will give details on the practical implementation of concepts and designs mentioned in previous chapters
- Evaluation: This chapter will discuss the implementation and findings made during the implementation
- Conclusion: This chapter will give a summary of how the aims of the project were met

Chapter 2

Background

Chapter 3

Design and Implementation

3.1 Toolchain

3.2 Board Configuration

On the Hifive1, there are several important configuration options that affect general operation of the board. The most notable of these are the clock settings, as these indicate the frequency of the processor, input and output frequencies, and timer interrupts.

3.2.1 Clock settings

The Hifive1 has 3 clock regions, a high frequency clock, a low frequency clock, and a clock used to drive the JTAG connection. The JTAG driver is constant and only used for debugging through JTAG, so is not relavent here.

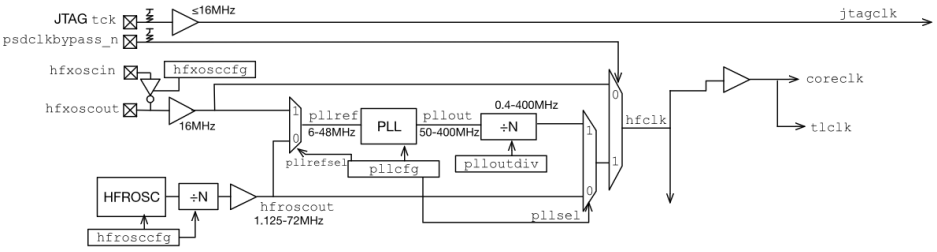

Figure 3.1: The high frequency clock generation scheme, specifying how the high frequency clock is driven and configured

The high frequency clock controls the processor frequency, and the baud rate of input and output is derived from it. The high frequency clock can be driven from two sources, an internally trimmable high frequency ring oscillator and an external high frequency crystal oscillator. The ring oscillator can produce frequencies ranging form 1 MHz to 75 MHz, whereas the crytal will produce a constant frequency of 16 MHz. Both of these clock sources may be used 'as is', or can be modified using a PLL and divider, giving a available range of 48 MHz to 384 MHz.

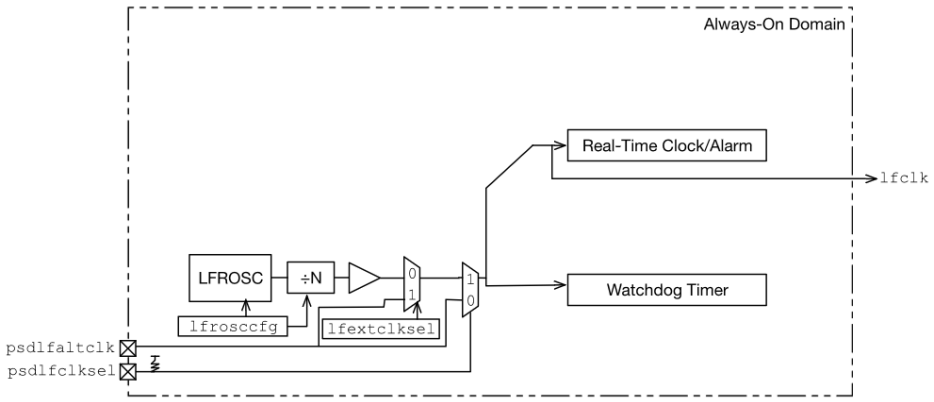

Figure 3.2: The low frequency clock generation scheme, specifying how the low frequency clock is driven and configured

The low frequency clock is part of the Hifive1 'always on block' and controls the watchdog timer, which can be used to cause a reset on malfunction, and both the realtime clock and the machine timer, both of which are used to generate timed

interrupts. Similar to the high frequency clock can be driven from a ring oscillator or from an external clock, which in the Hifive1 is a crystal oscillator. The low frequency ring oscillator functions at 1.5 KHz to 230 KHz using a frequency divider, and the implemented external clock runs at a contant 32.768 KHz, with no option to divide the frequency.
For both clock domains, the crystal oscillator was chosen. The ring oscillator gives the option to operate at a higher frequency, which would result in a higher number of operations per second. While in a practical operating system this would be desirable, since this system is not intended for practical use, a constant frequency was more desirable as it would give more predictable results, and makes IO operations more reliable. For the low frequency clock, a high frequency would be beneficial for a real time system as the higher frequency would allow for more precise timing of interrupts and other functions, however for an interactive system this precision is not required, and so similar to the high frequency domain the constant frequency of a crystal oscillator was selected.

3.3 Processes

3.4 Scheduler

3.5 Memory Management

3.6 IO

Chapter 4

Outcome

Chapter 5

Evaluation

Chapter 6

Conclusions and further work