SAPIENZA
UNIVERSITÀ DI ROMA

AVR Multi Motor Control

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

Corso di Laurea in Ingegneria Informatica e Automatica

Candidate

Paolo Lucchesi
ID number 1765134

Thesis Advisors

Prof. Giorgio Grisetti
Drs. Barbara Bazzana

Academic Year 2021/2022

Thesis not yet defended

Reviewer:

Prof. Silvia Bonomi

AVR Multi Motor Control

Bachelor's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Website: https://github.com/jcondor98/ammc

Author's email: lucchesi.1765134@studenti.uniroma1.it

Dedicated to my family, my granddad Pietro, my mom Anna Rosa, my dad Marco, my grandmom Pierina and my sister Valentina, which I thank everyday for everything I am.

To my dearest friends in Pitigliano, with whom I share some of the most beautiful memories I have.

To Nicola, with whom I have shared part of this path; he helped me in a dark period of my life and he is one of my dearest friends.

Abstract

Contents

Chapter 1

Introduction

Chapter 2

Client-side user interaction

A client program has been realized to manipulate the dc motors directly from the PC. It can get and set the speed of individual motors, and to apply all the previously set speeds for all of them at once.
A brief list of the client's features is given below:

- Granular handling for getting and setting motors' speed
- Modular and extensible software architecture
- Terminal User Interface, implemented as a command shell
- Support for non-interactive use (i.e. scripting)
- Communication with master controller using the serial protocol
- Compatible with POSIX-compliant environments

The client is also documented with a man page, which can be found in section 2.5.

2.1 User Interface

The end user interacts with the whole ammc ecosystem using a text-based client. It consists in a shell module, which I had written myself, offering some *internal commands* (hardcoded in the shell module itself) and is extended by *external commands* (found in a separated source code entity, and that can even be compiled in a detached transaction unit).

A particular focus was made on the software architecture: indeed, every external command can be realized standalone, and it is easy to add new commands just by altering the *client/source/shell_commands.c* source file.

2.2 Primitives offered

The commands that can be used to interface with the ammc ecosystem are the following:

connect <device-path> Connect to a master controller, given the path to the block device representing it.

get-speed <motor-id> Get the speed of a dc motor given its id. The motor id must be specified as a decimal number.

set-speed <motor-id>=<speed> Set the speed of a dc motor given its id. The motor id must be specified as a decimal number and the speed must be specified in rpm.

apply Apply the previously set speed for all the dc motors.

2.2.1 Non-interactive mode

The client shell is capable of running in non-interactive (i.e. scripting) mode with the *-s* option. If so, it will parse the input from a specified text file, or from *stdin* if not provided. A shell launched in non-interactive mode will not print shell prompts, and exit when end-of-file is encountered or on command failure.

2.3 Serial module

The client's serial module has been realized using the POSIX *termios* interface. Unlike the master controller's counterpart, all its code is reentrant, therefore multiple instances of multiple serial devices can theoretically exist at the same time.

From the client's perspective, the master controller is seen as a file descriptor, and the end user just have to specify the path of the block device file representing the serial communication channel (e.g. */dev/ttyACM0*) using the *connect* command.

2.4 Specification

2.4.1 Software modules

2.4.2 Modules dependency graph

2.5 Man page

Chapter 3

Master controller

The master controller handles all the slave controllers, dispaching arbitrary commands to them using the I2C protocol. It also communicates directly with the client application via serial port.

3.1 Hardware setup

The master controller itself is an AVR *ATMega2560* microcontroller unit. This particular MCU has some convenient features, such as:

- I2C dedicated hardware subsystem
- Serial-over-USB bridge
- Relatively powerful specifications for future feature adding
- Plenty of timers and outgoing power pins

3.2 I2C setup

3.3 Power management

3.4 Specification

3.4.1 Software modules

3.4.2 Modules dependency graph

3.4.3 Circuit schematics

3.4.4 Wiring

Chapter 4

Slave controllers

4.1 Hardware setup

4.2 I2C setup

4.3 Power management

4.4 Specification

4.4.1 Software modules

4.4.2 Modules dependency graph

4.4.3 Circuit schematics

4.4.4 Wiring

Chapter 5

Client-Master communication

Chapter 6

Master-Slave communication

Chapter 7

Conclusions