INSTITUTE FOR REAL-TIME COMPUTER SYSTEMS
TECHNISCHE UNIVERSITÄT MÜNCHEN
PROFESSOR SAMARJIT CHAKRABORTY

Development of a Microprocessor System (HW/SW) linking an Inverted Pendulum to an FPGA-System

Laurenz Altenmüller

Bachelor's Thesis

Development of a Microprocessor System (HW/SW) linking an Inverted Pendulum to an FPGA-System

Bachelor's Thesis

Executed at the Institute for Real-Time Computer Systems
Technische Universität München
Prof. Dr. sc. Samarjit Chakraborty

Advisor: Dipl.–Ing. Martin Geier

Author: Laurenz Altenmüller
Arcisstraße 21
80333 München

Submitted in July 2014

Acknowledgements

Vielen Dank . . .

München, im Monat Jahr

Contents

List of Figures

List of Tables

List of Symbols

RCS	Institute for Real-Time Computer Systems
I^2C	Inter-Integrated-Circuit
DAC	Digital-Analog-Converter
MCU	MicroController Unit
IC	Integrated Circuit
ISR	Interrupt Service Routine
SPI	Serial Peripheral Interface
PCB	Printed Circuit Board
GPIO	General Purpose Input / Output
FPGA	Field-Programmable Gate Array
TTL	Transistor-Transistor Logic
MIPS	Million Instructions Per Second
LED	Light Emitting Diode
SMD	Surface-Mounted Device
PWM	Pulse Width Modulation
CIE	Commission Internationale de l'Éclairage (International Commission on Illumination)
ICSP	In-Circuit Serial Programming
FLOSS	Free/Libre Open Source Software
CPLD IP	Complex Programmable Logic Device Intellectual Property

Abstract

Die Kurzfassung ...

1 Introduction

1.1 Background

This thesis is part of the DHEARTS project of the Institute for Real-Time Computer Systems at Technische Universität München. DHEARTS (Distributed Heterogeneous Embedded Architectures for Real-Time Systems) looks into implementation correctness, run-time hardware changes and communication issues in the context of interconnected heterogeneous embedded systems.

To investigate some challenges of embedded multi-control applications in run-time modifiable polymorphic cyber-physical systems, a test setup consisting of multiple inverted pendula is used. The main focus of this thesis is on the development of a control interface for these pendula.

In his bachelor's thesis, Marouane Ben Romdhane already put a microcontroller-based design forward, but left potential for future work.

1.2 Setup

The basic system setup is defined a priori. It meets the requirements of diverse concurrent and completed projects at RCS. The basic structure is visualized in Figure 1.1.

The inverted pendula are not the usual cart-and-pole model, but shelf-mounted geared DC motors that spin aluminum extrusion arms. A small weight (100g) can be attached at a distance of 16cm from the center of rotation. The motors are Harmonic Drive motors, type PMA-5A-50-01-E512ML. They are rated for 18 V, 0.85 A and feature a 1:50 step-down gearing as well as magnetic quadrature encoders with 512 quadcounts per rotation. The motors are connected via a DIN 41651 cable. There are three identical pendula, each with its own motor driver.

Maxon motor controllers, type 4-Q-DC Servoamplifier ADS 50/5 drive the motors. They are controlled via an TTL-compatible enable signal and a differential +- 10 V set voltage. Connections are made with wire clampings and a Molex 22-23-2061 cable.

A Xilinx Virtex 5 FPGA is the system controller. It is already successfully deployed in the MagLevBall project. An I^2C interface board is mounted on the FPGA board's extension header. The interface board uses PCA9600 I^2C bus buffers and RJ45 plugs for CAT5

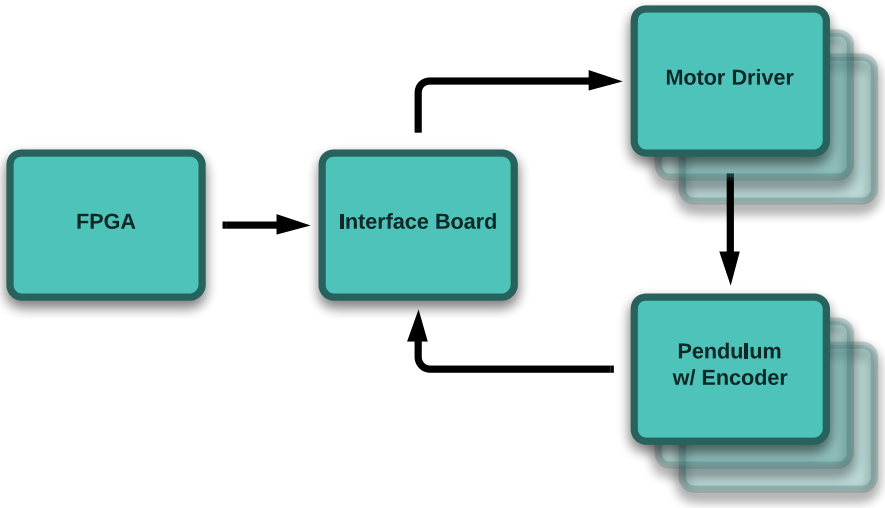

Figure 1.1: Setup overview

cable connections. This allows for multiple unidirectional I^2C busses with signal levels up to 15V, long cable connections and high clock speeds up to 1MHz.

Connected to the FPGA's I^2C busses are the so-called Plants. The MagLevBall board and the Inverted Pendula Interface board are examples for Plants. The Plants need to use the PCA9600 bus buffer to regain normal bidirectional I^2C .

2 Task Definition & Requirements Analysis

2.1 Task Definition

The goal of this thesis is to develop a microprocessor system that links an inverted pendulum to an FPGA system using the I^2C bus. Selected components need to be able to drive the pendulum motors, read the pendulum encoders and communicate driver set points and encoder feedback data with an I^2C master. Developing a PCB for the interface and implementing a controller on the FPGA is optional.

2.2 Driver Requirements

To use the motor controllers to drive the motors, the interface must generate the differential SET- and SET+ voltages and the enable signal, as specified in the motor controller datasheet. The enable signal can easily be sent by any microcontroller output pin capable of pulling the line up to 5V.

The setpoint input however requires a differential analog voltage with a common-mode voltage range of 5V. This constraint can be satisfied in two reasonable ways. Either one input is strictly tied to ground and the other one can vary between -10V and +10V, or both inputs vary between ground and +10V, where one is at ground level for positive input space and the other one is at ground level for negative input space. Both methods have drawbacks: the bipolar approach requires a negative voltage source while the unipolar approach uses twice as many digital-analog-converter (DAC) channels.

The nominal supply voltage of microcontrollers is generally not higher than 5V, and so is the maximum output voltage of any integrated DAC modules1. If such a module were to be used, an operational amplifier (OpAmp) is necessary to scale up the voltage range. While it could also be used to bias or invert the output signals, it increases circuit complexity and might have negative effects on dynamic DAC performance.

The DAC resolution and bandwidth are not of highest importance, but also not negligible.

The motor controllers use an external power supply.

2.3 Quadrature Decoding

The magnetic encoders integrated into the motors generate inverted and non-inverted quadrature-encoded output signals and index signals at TTL level.

Decoding these signals quickly and correctly is of high importance for the control feedback. The encoder resolution can be increased by a factor of 4 by analyzing edges instead of levels. The encoder has a native resolution of 512 impulses per rotation and motor is geared with a 1:50 ratio, so 102400 quadcounts imply one full pendulum rotation.

There are three ways to decode the quadrature signals into rotational position values. The first way is to use a microcontroller that interrupts on a state change of one of its inputs and uses a state machine to find out whether to increment or decrement a variable. This approach was used in Marouane Ben Romdhane's thesis. His implementation had problems at higher rotation speeds. A quick calculation shows that when the pendulum is turned at a reasonable speed of 1 rotation per second, a microcontroller clocked with 8MHz has less than 80 instructions cycles (excluding interrupt latency) between two interrupts to deal with some very big variables.

Conscious of this problem, manufacturers have integrated dedicated quadrature decoder modules in some of their higher-performance MCU families (XMC4500, STM32F4, ATxmega128A1U, PIC18F). These tend to have very capable ARM cores (or similar) that would increase development efforts. Also, most of them have just two channels, or activating a third one is very complicated and leaves it with a low resolution2

The most promising method is to use an quadrature decoding IC outside the MCU. There are several options, compared in table 2.1.

HCTL-2032	iC-MD	LS7366R	PE12024G
2 channel	1-3 channel	1 channel	1 channel
32bit counter	48bit counter	32bit counter	24bit counter
Parallel interface	SPI	SPI	Parallel interface
33MHz	40MHz	40MHz	20MHz
Easy availability	Differential input possible	Needs external clock	Also available as CPLD IP core

Table 2.1: Comparison of several quadrature decoder ICs

2.4 Microcontroller Requirements

The microcontroller reads the encoders, operates the motor controllers and communicates on the I^2C bus. Optionally it uses some LEDs, has general-purpose out- and inputs and tactile / DIP switch inputs. To keep development time short, it should be a smaller or easy to use microcontroller. The toolchain should be easy to handle and optimally

FLOSS. Performance is important, particularly I^2C slave speed: So far, all existing I^2C nodes are capable of 1000kHz I^2C .

Atmel AVRs are easy-to-use, have a very big online community and very good open source tool coverage. They can usually perform at up to 20MIPS. The maximum I^2C clock speed supported by hardware I^2C slave modules is 400kHz.

Microchip PICs are also not very complicated and have some online community support. The toolchain is not open source, but there is a free edition of their compiler which lacks optimization support. PICs can do 16MIPs at 64MHz. The maximum I^2C slave clock speed is 1000kHz.

Marouane Ben Romdhane's chose a PIC16F886. It has only one communication module and not very many free GPIOs for chip select pins or leds/switches.

A PIC18F46K22 features two communication modules, so one can be used for I^2C communication and one for SPI communication with DACs and dcoder ICs. There is plenty of free GPIOs for extra features.

2.5 Connectivity And Other Requirements

The FPGA master is to be connected to the Inverted Pendula Interface board via a special dual-unidirectional I^2C bus that requires bus buffer ICs and RJ45 plugs. The implementation should optimally be capable of the current maximum bus speed of 1000kHz.

The microcontroller must be (re-)programmable via a programming cable. Optionally it should be possible to have headers for future extensions like light barrier switches for absolute position detecting.

Some kind of user input and output not only help in debugging. They can also be of good use in the finished application, where selecting a mode and getting visible control feedback or error messages will be useful.

Last but not least the board has to have a power supply that satisfies the requirements of all parts integrated into the interface board. Some kind of safety against overvoltage, overcurrent or wrong polarity would be appropriate.

3 Hardware Implementation

3.1 Choice Of Main Components

After thorough requirement analysis, the choice fell one these main components: Three iC-MD quadrature decoder ICs, one AD5734R DAC and a PIC18F46K22 MCU. The hardware organization is depicted in Figure 3.1.

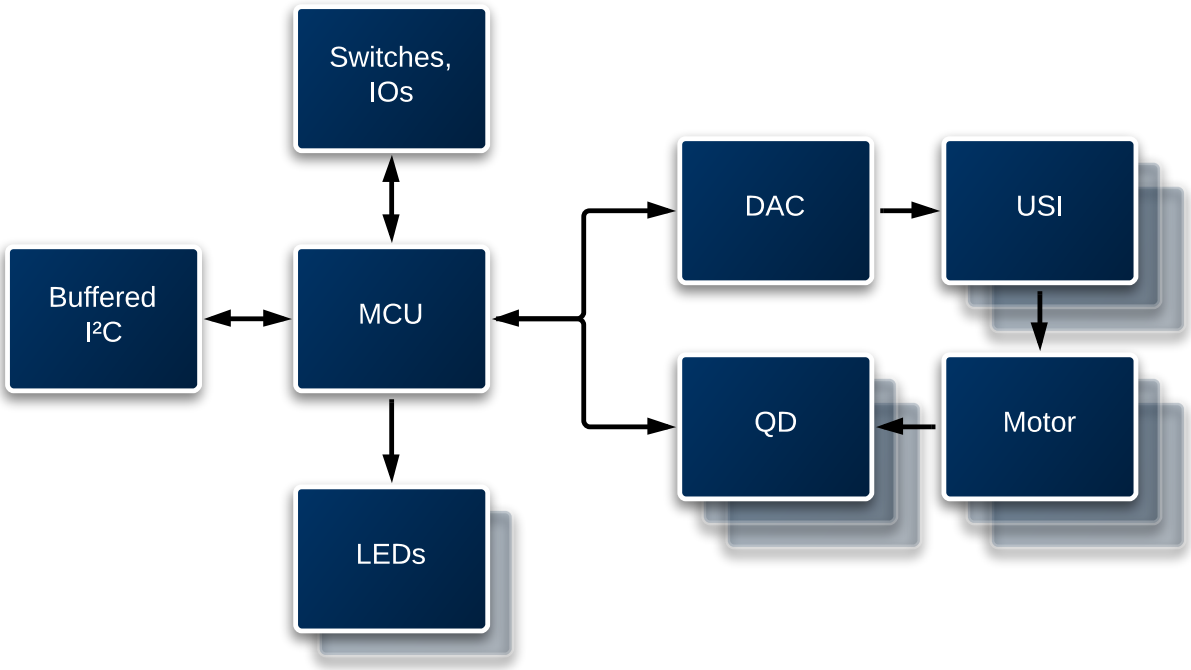

Figure 3.1: Organization of hardware systems

The iC-MD decoders promises good signal quality with its differential signal inputs. The SPI interface is very fast and multiple devices can share a bus. While the chip availability was worse than expected, manufacturer iC-Haus GmbH kindly provided a box of samples.

The PIC MCU has a 1000kHz I^2C compatibility, which allows for fast communication with the FPGA master. It has a second communication module which can be configured as SPI interface. This is used to interface the decoders and the DAC. With 64KB Flash and approx 4K RAM, lack of compiler size-optimizations should not become a problem. 16MIPS performance help to execute code fast in between interrupts.

The Analog Devices AD5754R DAC was the first choice, but turned out to be not yet available at the time of writing. However, the AD5734R offers the exact same features and pinout, only with lower resolution of 14bit instead of 16bit. It features 4 bipolar DAC channels with a range of +-10V. Its internal voltage reference simplifies the layout.

The communication with the FPGA master over the dual-unidirectional I^2C bus has to be buffered and converted to MCU compatible I^2C first. For this, the circuit on the FPGA I^2C interface board and the MagLevBall board was used. It uses a PCA9600 dual bidirectional bus buffer. The connection with the FPGA system uses RJ45 plugs.

To have some visual feedback, not only a green LED directly connected to a microcontroller pin is used. Three times ten yellow leds are individually 12bit-dimmable via two PCA9685PW LED driver ICs. They are controlled via bitbanged I^2C . In the microcontroller software, a 8bit PWM to CIE luminance lookup table is used to compensate human logarithmic light intensity sensitivity and linearize lightness.

Power is connected via a 2.1mm DC barrel jack. A transient voltage surpressor diode and a fuse supply overvoltage and wrong polarity and overcurrent protection, respectively. A 7805 linear regulator regulates the 5V power supply for all ICs. Since the board can consume up to 3-4W when all LEDs are switched on, it is recommended that the input voltage to the board is 7-7.5V. The red LED near the DC jack signals power good. A TMR0522 DC/DC module is used to generate +12V and -12V for the DAC from the 5V supply. Figure 3.2 shows the voltage supply structure.

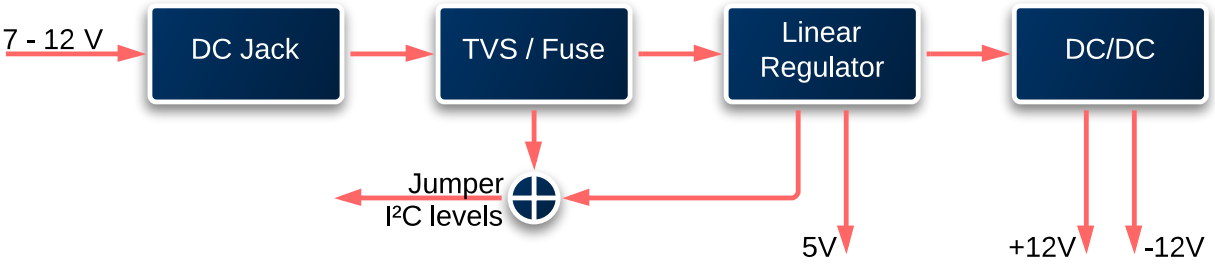

Figure 3.2: Supply structure

3.2 PCB Layout

Using a PCB instead of a breadboard prototyping area makes the use of a big range of SMD ICs possible. By having everything on a single PCB, the Inverted Pendula Interface is a nicely handleable piece of hardware, where nothing will fall off, various plugs can be used easily and even some documentation can be printed in the silkscreen layers.

Several iterations with different IC combinations were made until the perfect match of component choice, placing and routing was reached. Small pitch SMD ICs (TSSOP2x, 0805) save board space. At the right side of the PCB, motor drivers and encoders are

connected. Each pendulum has a dedicated tactile switch and 10 leds of the 30 led bargraph display. On the lower side, the pin headers for general-purpose I/Os are located together with the TTL I^2C and ICSP programming headers. On the left side are the plugs for power and I^2C as well as some voltage configuration jumpers and the 6-position DIL switch. Next to the green ACT LED is the fourth switch and on the upper side of the PCB is the fuse holder.

All bus signals have test pads for easier debugging. Traces are kept short and have mostly orthogonal crossings on the other side. They vary in width according to their potential power throughput. Analog signals are isolated from the digital signals with ground traces in between.

The layout design was done in EAGLE 5. PCB manufacturing was done by IteadStudio in China. Manual assembly was greatly sped up by use of the RCS chair's reflow soldering equipment.

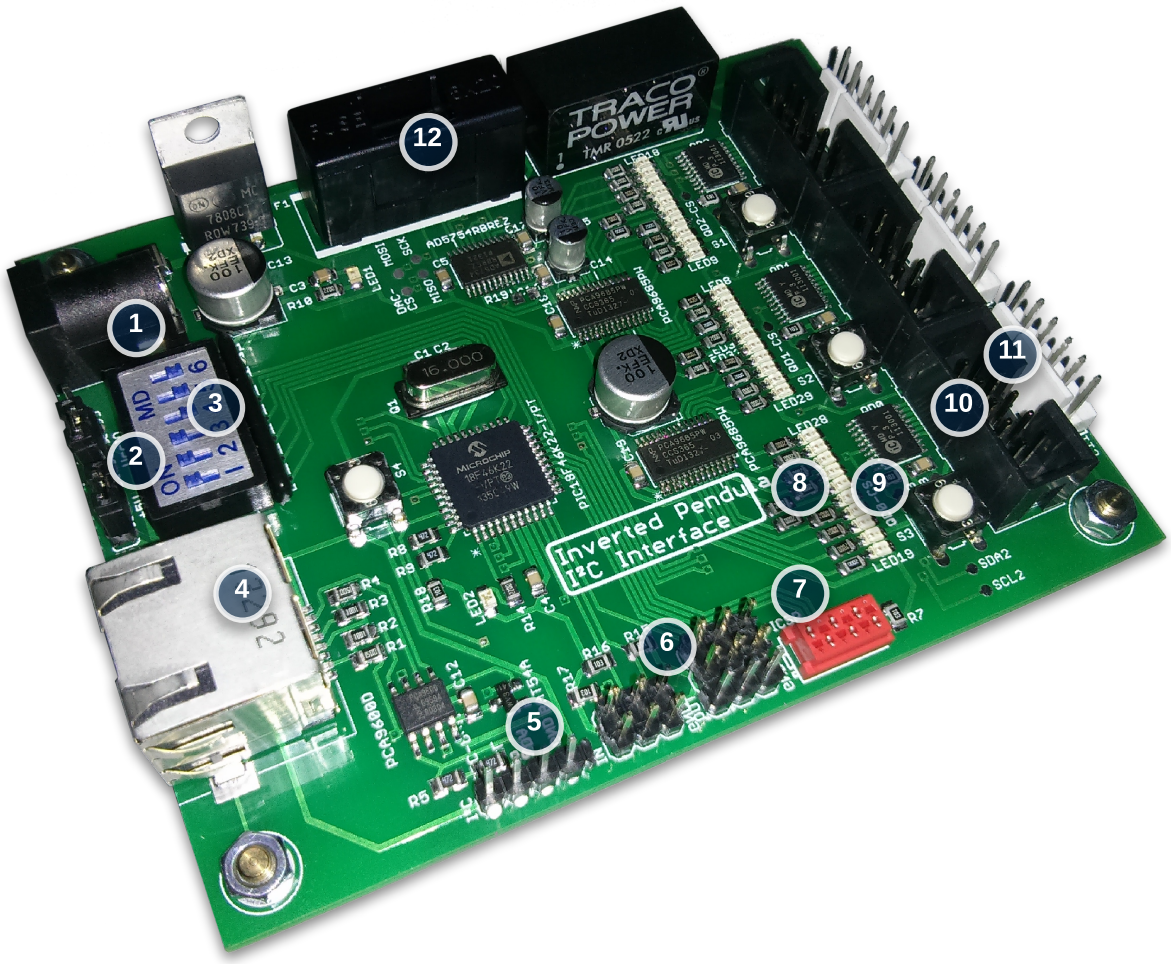

Figure 3.3: Assembled PCB

4 Software Implementation

4.1 Inverted Pendula Interface

4.1.1 Code Structure And Toolchain Overview

The Inverted Pendulum Interface board firmware source code is organized in functional groups. In total there are 24 files, as described in table XX. All source code is in C.

File name		Description
`configuration_bits.c`		List of `#pragma` config statements that define microcontroller configuration like clock sources, peripheral modes and other settings
main.h		Includes all necessary project, library and hardware definition headers
main.c		Contains main function with main loop
init.c	init.h	Initialization functions for microcontroller modules and some external peripherals
interrupts.c	interrupts.h	Contains ISRs. Timers are handled here.
dac.c	dac.h	DAC usage functions
decoderc.c	decoderc.h	Quadrature decoder usage and helper functions
i2c.c	i2c.h	I^2C ISR logic and helper functions
io.c	io.h	Functions for switches and the like
leds.c	leds.h	Functions for animating and updating LEDs and talking with LED driver ICs
pid.c	pid.h	PID functions for internal closed loop control
swi2c.c	swi2c.h	Modified Microchip PLib Software I^2C bitbang driver
interface.c	interface.h	Interface communication helper functions and interface registers

Table 4.1: Files in the project

Microchip's XC8 compiler was used from within Microchip's Netbeans-based MPLAB X IDE. To program the interface, a Microchip PICkit3 was acquired. The PICkit3 is integrates far better into the IDE than its predecessor, the PICkit2, which was used in earlier efforts on the Inverted Pendula topic. It allows breakpoints and step by step

debugging, but breakpoints can only be set by reprogramming the device. Programming the device needs about 18 seconds.

The project is compiled by the built-in XC8 compiler using auto-generated makefiles. 18% (11.8 kB) of available program space and 12.6% (0.49 kB) of % available data space are used. No optimizations are used because the compiler % is run in in Free mode. A compiler warning message states that in Pro mode, % the code could be 60% smaller and 400% faster.

4.1.2 Peripheral Initialization And Communication

When the MCU starts, it begins to set up a fallback internal oscillator. Both the internal and external oscillator run at 16MHz and are multiplied internally with a 4x PLL, so the system clock is 64MHz. Next, IO directions and output values are set (all pins default to tristate, so no connections will be shorted). Software logic and buffers are initialized. Timer 0 is configured to interrupt periodically with 1kHz. Communication modules are configured for external I^2C , SPI and internal (board-local) software I^2C . Lastly, the interrupt controller is set up to detect falling edges at the general-purpose input pins and interrupts are globally enabled.

Now, external peripherals are initialized. While the DAC chip select line is active (low), four 24bit words are sent to the DAC via SPI, segmented by 1 us long chip select high pulses to latch in the data (see figure XX). The SPI bus is in mode 0 configuration (clock polarity 0, clock phase 0) with at 16MHz (oscillator clock prescaler of 4). The first word sets the output range of all four DACs to +- 10V. The second word enables the DAC's thermal shutdown and current limit clamp safety features. The third word finally powers up the internal reference voltage supply and DACs A, B and C.

All three quadrature decoders are on the same SPI bus as the DAC. Even though they need identical configuration, they are initialized separately to avoid bus collisions on the MISO line. When their chip select line is active, the decoders expect an register address byte as input. Bit 7 of this byte marks a following read (0) or write (1) operation. There is no limit on the number of bytes that can be clocked in or out.

4.1.3 Control Flow

After initialization, the main loop periodically checks inputs like switches and decoder ICs, does necessary calculations and sets outputs correctly. Each pendulum can be individually and independently controlled. Reading from and writing values to the Inverted Pendula Interface board is done using I^2C communication. I^2C logic is handled partly by hardware and partly by software state machines in ISRs. For details, see Figure XX.

Writing a parameter is done in 3 steps. First, the master has to issue a start condition bit and send the chip address of the interface (0x50) together with the R/!W bit cleared.

After the interface has acknowledged its address the master must clock out the correct register address pointer byte which the slave will acknowledge. Only now the actual data can be sent to the interface. The number of data bytes is not limited – the slave will acknowledge all bytes. The register address pointer is incremented after every successful byte transaction, so multiple parameters can be written in one access. It is OK to access addresses not associated with word boundaries (Example: `REG_0_PID_K` is at 0x12 and 2 bytes wide. The user can write two bytes at address 0x13. This would be the MSB of `REG_0_PID_K` and the LSB of `REG_0_SET`). However, the address pointer will not overflow, so bytes received after address 0xFF will be acknowledged but discarded. The master is allowed to omit any data bytes in order to adjust the register address pointer for later read access. The register address pointer is not reset or changed under any circumstances except a reset triggered by the user. The transmission is finished when the master issues a stop condition.

Reading a parameter requires the user to set the register address pointer first, as described with write access. Similar to writing, after the master issues a start condition, the chip address is sent – this time with the R/!W bit set. The interface acknowledges its address and begins to transmit bytes from the register buffer. Bytes are sent until the master sends NACK, even when there are no more bytes available (in this case, 0x00 is sent instead).

A full list of valid register addresses is compiled in Table XX. Details can be found in the following paragraphs. Since three identical pendulums are used, the register names follow the scheme REG_x_function, where x = 0,1,2. All Parameters are little endian.

Pendulum Modes

Every pendulum has its own mode parameter at address REG_x_MODE, which can be set to MODE_OFF, MODE_CAL_SW, MODE_CAL_MV, MODE_SET and MODE_PID. MODE_OFF sets the pendulum's SET+ value to 0V and disables the enable signal. It is the default value after power up and reset. MODE_CAL_SW and MODE_CAL_MV are for calibration of the pendulum's rotational zero position. An external light barrier or endstop switch can be used to detect the zero position via the interrupts on the general-purpose inputs (SW). Alternatively the pendulum can be driven with a constant value that is just enough to lift it against gravitation over a full round. Rotational position is tracked and the difference between values with known timestamps is used to calculate rotational velocity. The point of maximum velocity is assumed to be the lowest point in the setup and thus zero position. Both methods are not implemented yet, but left for potential future work on the topic. MODE_SET is the most simple, yet most important pendulum control mode. It enables the motor driver and uses the DAC to assign a fixed SET+ voltage. This mode is recommended for closed-loop control with external master controllers. MODE_PID tries to reach a set rotational position by regulating the SET+ voltage with values from a PID controller. All PID parameters can be tuned to allow for various control behaviors. This mode is recommended for open-loop control with and

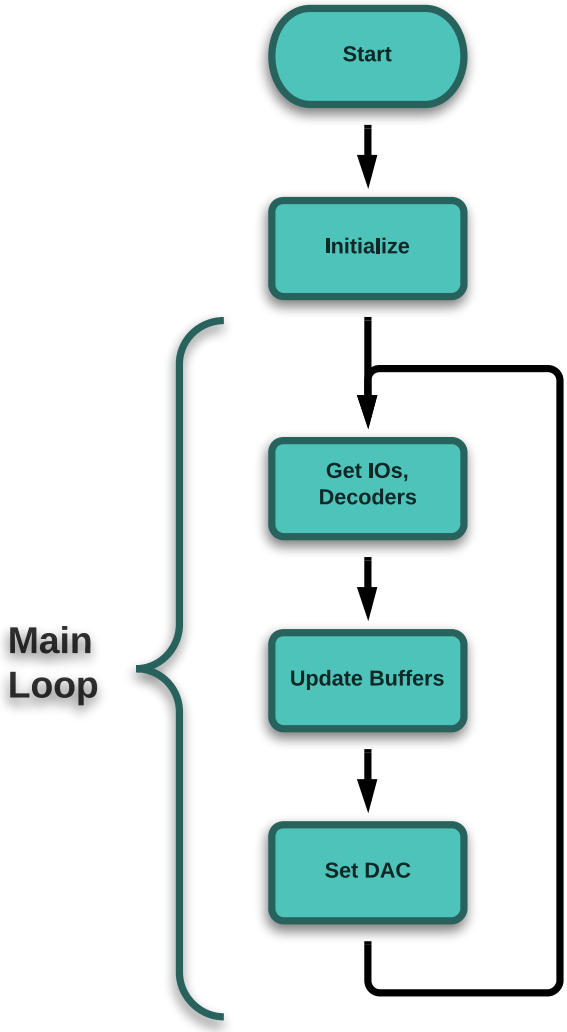

Figure 4.1: Main loop flow chart

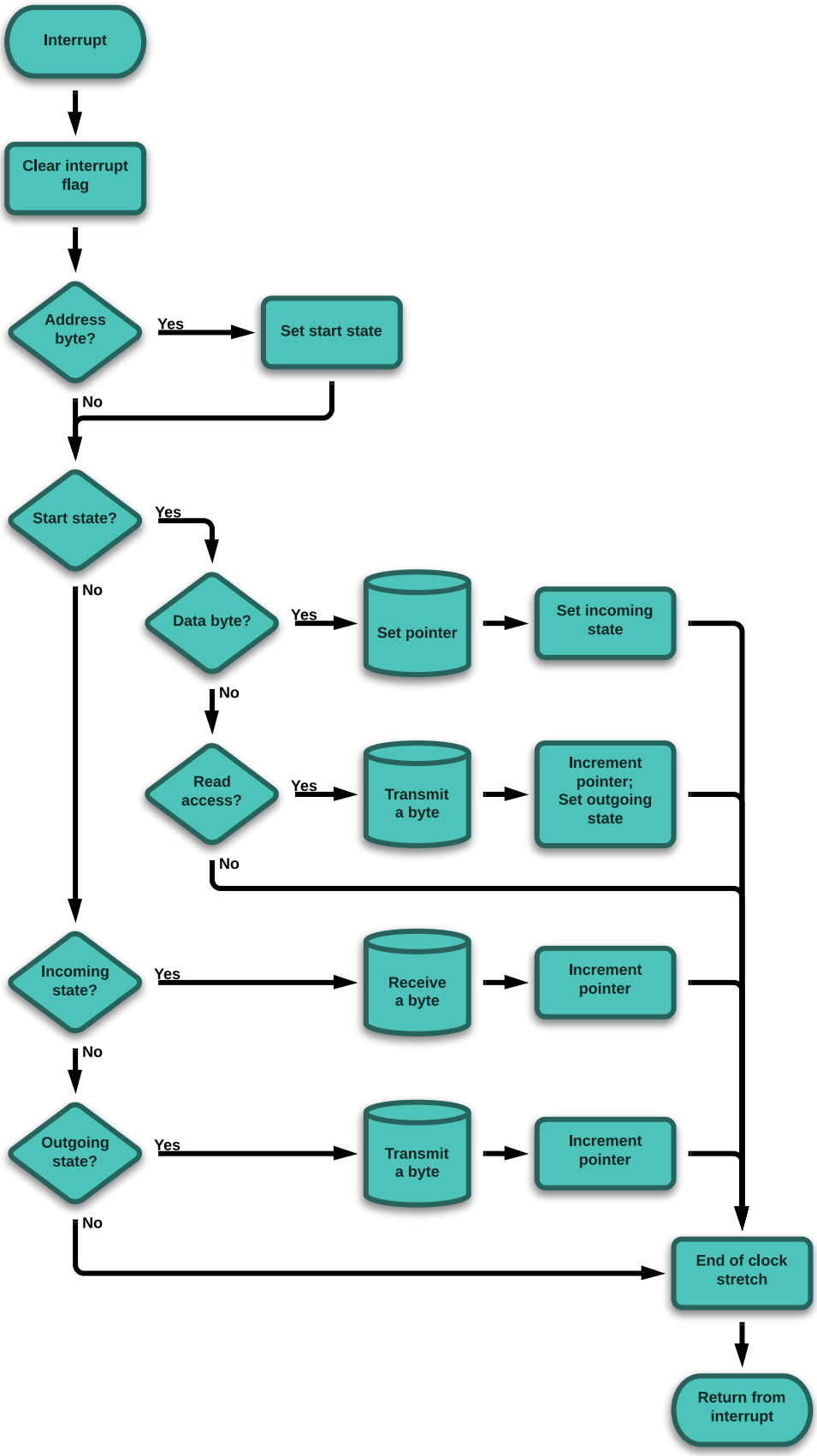

Figure 4.2: I^{2}C interrupt service routine flow chart

without external master controllers.

LED modes

Not only the pendulum but also the 10 LEDs next to its connector can be controlled in more than one way. Valid modes are MODE_LED_OFF, MODE_LED_A, MODE_LED_B, MODE_LED_C and MODE_LED_SCAN. MODE_LED_OFF switches off all 10 LEDs. This is the default state after power up and reset. MODE_LED_A maps rotational position to the LED bargraph. The setpoint is displayed as single LED with full brightness while the actual position is visualized by filling up the bargraph from the bottom with low brightness. The animation wraps at the overflow / underflow to zero position. MODE_LED_B and MODE_LED_C are not implemented and placeholders for potentially very nice looking effects. They could be used in connection with the calibration modes. MODE_LED_SCAN does a Larsson Scan on the LED bargraph to show idle state.

Pendulum Parameters

Other than the REG_x_MODE and REG_x_LED registers, the PID control parameters REG_x_PID_P, REG_x_PID_I, REG_x_PID_D, REG_x_PID_K, the setpoint REG_x_SET and the actual value REG_x_GET can be read and written (NB: writing the REG_x_GET register has no effect, the value will be overwritten at the next update). The PID parameters are the proportional, integral, derivative and scaling factor, respectively. A scaling factor is necessary to allow for fixed point arithmetic with integers. Each of them is two bytes wide and interpreted as signed 16 bit integer. The output value of the PID controller is computed as (P * error + I*sum[0,t](err) + D*dError) / K. The setpoint and actual value are both 4 byte wide signed integers. Their valid range is -51200 to 51199. Values are constrained to this range with the algorithm

```
int32\_t constrain(int32\_t wert) {
wert = (wert+51200) % 102400;
if(wert < 0) wert+=102400;
return wert-51200;
}
```

Interface Control

Five registers are used to control interface logic and GPIOs. They are REG_RESET, REG_IO_SWD, REG_IO_SWT, REG_IO_IN, REG_IO_OUT. REG_RESET can be used to reset the Inverted Pendula Interface board. Writing any value other than zero to it causes the microcontroller to initiate a hard reset. This restores default values in all state logic and buffers. In the following software initialization routines, all peripherals are reset to default states as well. REG_IO_OUT is the general-purpose output register. The values of the lower 3 bits in the value written to this register address are represented as HIGH

and LOW levels on the output pin header. Reading the register returns the value of the shadow copy register for the pins, not their actual values. REG_IO_IN, REG_IO_SWD and REG_IO_SWT reflect the states of the respective MCU inputs, namely the general-purpose input header pins, the DIL mode switch and the tactile switches. Every register holds a byte in which the lower bits describe the input states. For example, the DIL switch has 6 positions, so the upper two bits in the REG_IO_SWD are don't care, while the lower 6 bits represent the state of the switch.

4.2 Master Controller

4.2.1 Raspberry Implementation

...

4.2.2 FPGA Implementation

...

4.3 Performance Analysis

...

4.4 Conclusion

...

A Software Sources

B Hardware Sources

Bibliography