Development of a sensory feedback system in hand prostheses

Alexander Jordansson and Nima Haji Sheykhi

2017

LUND
UNIVERSITY

Master's thesis in
Electrical Measurements

Faculty of Engineering LTH
Department of Biomedical Engineering

Supervisor: Christian Antfolk

"Nothing is impossible, the word itself says I’m possible!"
-Audrey Hepburn

Abstract

This master thesis is about improvement, optimization and further development of the sensory feedback in a prosthetic hand previously investigated at the department of Biomedical Engineering in Lund. The first approach will be to investigate and further see into the need of a sensory feedback in hand prostheses. There were several reports that showed that the lack of sensory feedback in hand prostheses increased the rejection factor for the user.

The aim will be to investigate necessary functions needed in a sensory feedback system in a hand prosthesis and look into appropriate hardware that fulfills these requirements.

The core hardware is built on two Arduino Nano circuits. The force sensing sensors, electromyography sensors and the sensory feedback vibrators are the main components for the setup. To fully utilize the capacity of the micro controllers and to fulfill the requirements in the aim components such as a real time clock, data logger, Bluetooth and a calibration mode will be applied in the setup.

Subsequently a printed circuit board will be developed while investigating the robustness, rapidity and stability of the Arduino setup.

The result of this project is ready to test EMG controlled hand prosthesis with sensory feedback. The response time is around 6 ms. In addition to this, several features have been added to the setup such as an calibration mode with Bluetooth interface, a real time clock and the opportunity to store information on an SD card. An implemented sleep mode that makes the setup as power saving as possible is also implemented.

Preface

This master thesis was completed at the department of Biomedical Engineering at Lund University in conclusion of the master science program in Electrical Engineering. The work was achieved under supervision of Christian Antfolk as an extension of his doctoral thesis "On Sensory Feedback in Hand Prostheses"[1].

The master thesis was begun on the 20:th of September and was finalized on the 2:nd of February, covering 30 ECTS credits.

Acknowledgments

First and foremost we would like to express our greatest gratitude to our supervisor Christian Antfolk who has guided us through this challenging time. We still do not understand how he could think so many steps ahead, warning about difficulties that would occur some weeks later. An explicit thank you is extended to our examiner Hans W Persson for reaching out to Christian and mentioning us as candidates for this master thesis.

We would also like to express a huge thanks to our colleagues in the department of Electrical and Biomedical Engineering for making us feel as part of the department. There are some colleagues that have made an extra impact on us so a special thank you also goes to Axel Tojo for his input on our SD card problem and helping us with the development of the PCB, Nebojša Malešević for insightful comments on Eagle, the project as a whole and asking if we were finished yet everyday of the week, Ingrid Svensson for spreading joy and always managing to turn a bad day into a good one and Désirée Jarebrant for solving all the formalities.

Lastly we would like to thank all others who have supported us, not only through this master thesis but during the whole education. You know who you are, thank you!

Alexander Jordansson
Nima Haji sheykhi

Contents

Chapter 1

Introduction

Research has shown that straight forward and uncomplicated hand prostheses that fulfill their purpose of improving the quality of life for a person in need of aid - and also provide full sensory feedback - simply are not commercially available [2, 3, 4]. In one survey it was shown that amputees requested more of the sensory feedback [5]. The prostheses need to be simple, accessible and withstand the everyday use of the common man. There are over 100 000 people in the USA that have had a loss of the upper limb [6].

Christian Antfolk previous work explored a new concept for providing sensory feedback in hand prostheses. In one of the papers published a test including giving sensory feedback with servomotors on the upper arm on healthy subjects. The result was that the subject were able to discriminate between three stimulation at a mean (SD) of 97% (5.3%) of the trials.[7] Research also showed that such a sensory feedback system could coexist with myoelectric control system [8] and a pilot study with amputees was completed on how they can experience a robot-like advanced hand prosthesis as part of their own body. The experimental design was to see if brushing a visible prosthesis and at the same time brush the corresponding skin on the amputees stump which was not visible would give a phantom feeling of ownership of the prosthesis. The result was that the participants expressed a feeling of ownership of the hand prosthesis [9]. A non-invasive simple sensory feedback system was also presented [10] in one of his papers. The system consist of pads in the prosthesis

and pads on the patients stump. The pads are driven by air so it is a direct sensory feedback. Approximately 96 % of the test subject managed to locate the correct touch in the trial. Sensory feedback was also investigated where vibrotactile feedback was compared with mechanotactile. Result showed that the vibrotactile feedback was harder to discriminate than mechanotactile [11] but Vibrotactile grasping force feedback improves grasping performance [12].

A further development of the work developed at the department of Biomedical Engineering in Lund seemed to be an obvious choice as a degree project, for the development of future hand prostheses. Even though the hand prosthesis will have its original main task such as grabbing, the remake of the prosthesis will give the impression of a closer and more sensitive device. The prosthesis will give enough feedback to enable the amputee a feeling if better harmonizing with their own body.

1.1 Aim

The aim of this thesis is to further develop previous investigated concepts for a sensory feedback system in a hand prosthesis.
Key issues:

- Investigate the overall need and usability for a sensory feedback system.
- Design and implement an sensory feedback system for the Ottobock hand prosthesis using Arduino circuit boards.
- Implement different add-ons on the Arduino boards such as a real time clock, data logger and Bluetooth interfaces.
- Build a custom made circuit meaning a PCB (printed circuit board).
- Make the circuit as low power consuming as possible.

1.2 Outline

The outline of this master thesis follows. Chapter 2 gives a short introduction of the prosthetic hand used in the thesis, as well as explains different types of prosthetic hands available. Chapter 3 discusses the basics of the Arduino circuit board and its different features. It also covers all the different sensors used, such as the force sensors, the sensory feedback vibrators, step motors and the electromyography (EMG) sensors which detects muscle movement in order to control the prosthesis. Chapter 4 is the experimental chapter which contains all the different approaches and experiments conducted before presenting the results. Chapter 5 presents the discussion where different problems and approaches are presented. At last chapter 6 states the conclusion and further work.

Chapter 2

Background

Amputation has been around for thousands of years. In the early days the amputated limb was even buried and then reburied after the person's death. The reason was that people believed that even in the afterlife the person would be without the limb if not buried together. Dating back to the fifteenth century BC a mummy was discovered in Egypt without its right big toe. The amputated toe was replaced with a prosthesis made of leather and wood.[13]

2.1 Amputation

The removal of a limb from the body is called amputation. The word *amputation* is borrowed from the Latin *amputare* which means, to cut off or cut away [14]. The reasons of an amputation are many, such as trauma, diseases or other medical events affecting the limb. The effects of an amputation is a loss of sensory and motor functionality which affects the physical and mental health of the individual[15].

2.2 Different prostheses

Prostheses have been developed for the purpose to regain functionality, for cosmetic reasons or for the mental wholeness for the amputee. Prosthesis can be divided in in two different categories: active and passive. The passive prostheses can either be task specific or for cosmetic appear-

ance. The active prostheses are either body powered or electrically powered. Body powered prostheses are constructed with a cable that is attached to a suitable body part to control the hand. [16]

The benefits with body powered prostheses includes that they are more robust and it gives feedback in stiffness of the wire [17]. The Electrical powered prostheses (see Fig. 2.1) operate with an electrical motor. Depending on the complexity of the hand prosthesis it could contain one or several electrical motors. To control an active electrical prosthetic hand EMG sensors are used. The sensors are placed on muscles to register activity. EMG sensors are explained more in section 3.4.3. Overall, the only feedback on the electrically power prostheses available commercially today is the visual feedback. [16, 17, 18]

Figure 2.1: An active electrical hand prosthesis from Ottobock.

Chapter 3

Theory

This chapter covers the theory of the chosen hardware and software applied in this project. Firstly the Arduino board is explained, with all its features. Further, the different software alternatives for communications are described. This is followed by a brief explanation of how the interrupt function works in Arduino, a short introduction of how pulse width modulation (PWM) operates and how the internal memory functions for stability purposes. Thereafter all the different breakout boards are explained. Finally the different sensors are discussed before ending the chapter with some comments about the printed circuit board (PCB).

3.1 Arduino Nano

The Arduino boards are open source boards based on different microcontrollers, with a variation of different layouts. These boards are very popular to experiment with since they are open source and there is a large community working with them.

The Arduino Nano board Rev.3.1 (see Fig. 3.1), used in this project, is based on the ATmega328-AU microcontroller. The PCB dimensions are 45x18 mm and weighs 7g. It has a 5V operating voltage but it can handle 6-20 V. The ATmega328-AU has 32 KB of flash memory of which 2 KB used by the bootloader, 2KB of SRAM and 1KB of EEPROM. The clock speed is 16MHz. The board has 8 analog input pins and 14 digital I/O pins which 6 provides the

PWM feature.

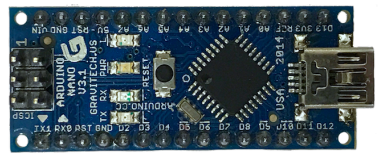


Figure 3.1: The Arduino Nano Rev3.1 board.

The analog input pins have a resolution of 10 bytes, meaning it can handle 1024 different values. They measure from ground to 5 volts. The analog input pin A4 (SDA) and the analog pin input A5 (SCL) are specifically used for the I^2C (see section 3.1.6) transfer. The 14 digital ports available can be used as inputs and/or outputs, hence calling them I/O ports earlier.

The most important digital pins are:

- The serial transfer digital pins 0 (Rx-Receive) and 1 (Tx-transmit). These pins transmit and receive TTL serial data and are connected to the corresponding pins of the FTDI USB-to-TTL Serial chip so that data can be transferred through USB.
- Digital pin 2 and 3 which are the external interrupts pins. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value on how the digital pin 2 is used as a external interrupt pin.
- The digital PWM ports; 3, 5, 6, 9, 10, and 11.
- The pins who support SPI communication, port 10 (SS-slave select), 11 (MOSI-MasterOutSlaveIn), 12 (MISO-MasterInSlaveOut), 13 (SCK-Clock).[19]

3.1.1 Baud rate

Signal rate or baud rate as referred to Arduino context, is the number of signal elements sent in 1 second [20]. The unit is the baud and possible baud rates set directly in the Arduino interface are 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200 Bd [21].

3.1.2 Serial monitor

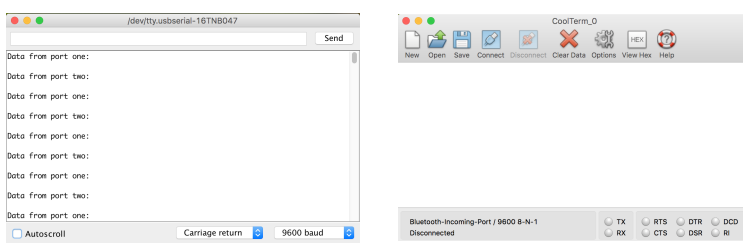
Two main monitors are used to read and evaluate the serial data. The built-in monitor in the Arduino program and a freeware software called CoolTerm.

Arduinos monitor

The Arduino software includes a serial monitor which allows data to be sent to and from the Arduino board (see Fig. 3.2a). The Arduino serial monitor can only have one serial monitor open at once and the baud rate must be the same as set in the program code in order to function properly.

CoolTerm

The CoolTerm (see Fig. 3.2b) program is a hobby developed program by Roger Meier who lets the user open several different monitors and have more options available to setup the serial monitor [22].



(a) The Arduino monitor. (b) The CoolTerm monitor.

Figure 3.2: Different serial monitors.

3.1.3 Serial interface

The built-in serial interface is asynchronous so timing is unimportant[23]. It sends 8 bits of data with two extra bits, start and stop. The receiver needs to recognize the pattern[24]. This is performed by two ports, a transmit port(Tx) and a receive port (Rx). To send data between two devices the two ports need to be connected crosswise, Tx to Rx (see Fig. 3.3).

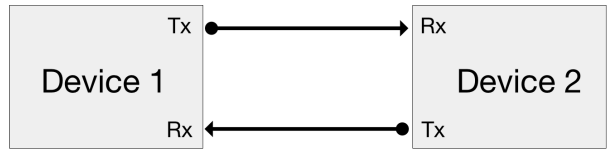


Figure 3.3: The serial interface.

3.1.4 SPI

Microcontrollers can use the Serial Peripheral Interface (SPI) for communicating with one or several devices. SPI offers a very high data rate (up to 10MHz). Before a connection is established, the master device sends demands to other devices, the slaves, to perform tasks. There are in general three main connection lines between the devices and one line that chooses which device to communicate with (see Fig. 3.4). Each line describes the related communications between the master and slave units. [25]

- MISO (Master In Slave Out) - The Slave line for sending data to the master.
- MOSI (Master Out Slave In) - The Master line for sending data to the peripherals.
- SCK (Serial Clock) - The clock pulses which synchronize data transmission generated by the master.
- SS (Slave Select) - The pin on each device that the master can use to enable and disable specific devices.

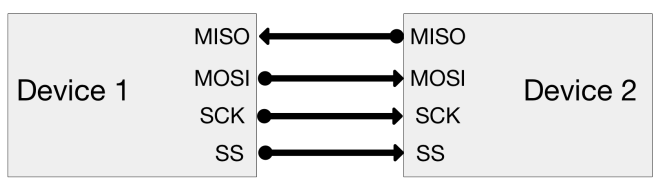


Figure 3.4: The SPI interface.

When the slave select pin is low for a certain device, it communicates with the master device otherwise it ignores the master. This allows the user to have multiple SPI devices sharing the same MISO, MOSI and CLK line [26].

3.1.5 SoftwareSerial

The Arduino Nano hardware includes a built-in chip that is called UART which supports native serial support. This means that the Atmega328P microcontroller is able to receive serial communication when working on other task. The only limit is the 64 byte serial buffer.[27] The built-in support for the serial communications is build-in on digital pins 0(Rx) and 1(Tx). With the SoftwareSerial library it is possible to allow serial communication on other digital pins than 0 and 1. This creates a virtual Tx and Rx which can be observed in a new monitor.

3.1.6 I^2C

The inter-integrated circuit known as I^2C also requires two lines to communicate just as the serial interface. It is slower than SPI with its 100 kHz or 400 kHz but the 400 kHz data transfer is more than enough for the purpose of this thesis. For every 8 data bits, it also sends an acknowledge (ACK) or an not acknowledge (NACK) bit. The I^2C bus consist of two lines, SCL which is the clock signal and SDA which is the data line. The clock is generated by the current master controlling the line. The lines are always open (HIGH/open drain), which is done with pull-up resistors, but this also means that the line can only be drawn to low, making sure that if the bus is busy (low) no other device can control it to avoid collisions (see Fig. 3.5).[28]

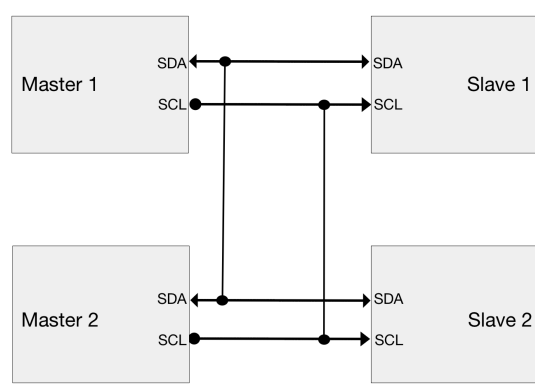


Figure 3.5: The I^2C interface.

Transfer

The 8-bit data protocol is used to transfer information between two devices. There are two main lines, SCL and SDA. The data packages contains an 7 address bits and 8 bits of data.

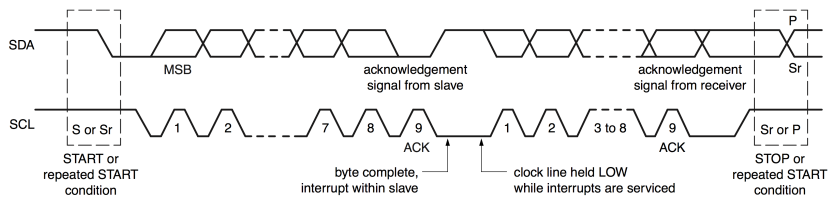


Figure 3.6: The I^2C communication protocol courtesy of Nxp UM10204 I2C-bus specification and user manual[28].

Start condition

When the master initiates a transfer it will pull down SDA to low before SCL to let all the slaves know that a transfer is about to begin, and to start listening. If there are more than one master that wants to send data at the same time, the device that pulls SDA down to low first gains the control over the transfer.

Address Frame

The address frame contains 7-bits of data with the most significant bit first, read (1) or write (0) bit as instructions. The last bit is the NACK/ACK bit. This is used to know whether the data has been received or not. If the receiving device do not turn the SDA line to low on the ACK-bit then the sender knows that the data was not received.

Data frames

The data frame contains 8-bits of data with the most significant bit first and NACK/ACK-bit last.

Stop condition

When all data is transferred a stop condition is generated. The stop condition is defined as the SDA goes from low to high when SCL is high. Normally the SDA do not change when SCL is high, which is specific for the stop condition (see Fig. 3.6).[28, 29]

3.1.7 Sleep Modes

In order to save power, different sleep modes can be implemented for the Atmega chip. There are several ways to put the Arduino into sleep and a few ways to wake it up. The four different ways to wake it up which are of interest for this project are:

- via an external interrupt.
- via the UART (USB serial interface).
- via an internal timer.
- via I^2C interface.

The five main ways of sleep for the Atmega328P are:

- SLEEP MODE IDLE
- SLEEP MODE ADC
- SLEEP MODE PWR SAVE
- SLEEP MODE STANDBY
- SLEEP MODE PWR DOWN

The power down sleep mode gives the most power saving but also limits the different ways of waking the Arduino from sleep [30].

3.1.8 Interrupt

There are both internal and external interrupts to implement for the Atmega328 [31]. In this project only the external interrupts are used.

External interrupts

External interrupts are reserved on pin 2 and pin 3 on the Arduino. When the pin change states, either from rising or falling, high to low or low to high the interrupt is triggered [32].

3.1.9 PWM

Pulse Width Modulation (PWM) is a method to convert an analog signal to a pulsing voltage signal. A square signal is created from the analog signal with two different states, on and off states. The signal variate from 0V (off), 5V (on) and variate in time for how long a signal is set to be on and off, which is the pulse width. The Arduino have values on an interval from 0-255 where 0 is 0% and 255 is 100% of the duty cycle, figure 3.7.[33, 34]

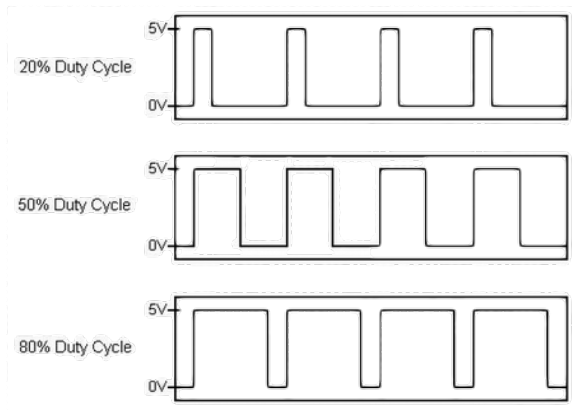


Figure 3.7: 20%, 50% and 80% duty cycles (Image courtesy of NI [33]).

3.1.10 Memory

It is necessary to understand the essential building blocks of memory in the Arduino before building a stable system. There are mainly three types of memory. While building a more complex system this also means more load in the memory system and thus a higher risk for instability in the long run. Upcoming sections explains the different memory types.

Arduino Memory Limitations

Arduino is build up with three main memories of following types:

- Program memory or flash memory.

- SRAM - Static Random Access Memory.
- EEPROM - Electrical Erasable Programmable Read-Only Memory. [35]

Program/Flash memory

Flash memory is the more common word for program memory and is used to store data. Flash memory is a non-volatile memory and is regular in everyday USB-flash drives or SD cards. The data will remain even when the power is turned off. The information on the flash memory can not be modified. Before the data can be modified it must be copied to the SRAM, processed and then copied back to the flash memory. [36, 37]

SRAM - Static Random Access Memory

The executing program can read, write and modify data in the SRAM. The SRAM have three main tasks.

- **Static Data**: All the global variables and the executing program are stored here. All data that is static, unchangeable during execution remains here as well. The program start executing from this point.
- **Heap**: The heap change in size and it is dynamic. This is caused by changeable data that is allocated here and therefore the heap memory of the Arduino is dynamic.
- **Stack**: The stack stores local variables, interrupts and functional calls. The stack is dynamic in size and is depending on the program code.[36, 37]

EEPROM

EEPROM stands for Electrically Erasable Programmable Read-Only Memory. EEPROM is a non-volatile memory and like the flash memory the only difference is that EEPROM read and write data byte by byte. Therefore the EEPROM is a slower option then SRAM. [36, 38]

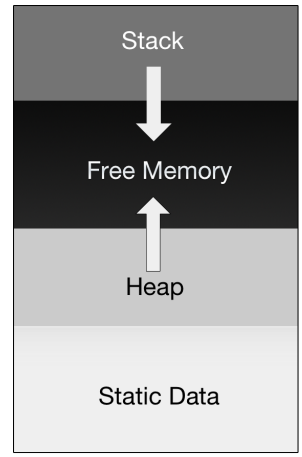


Figure 3.8: SRAM: The heap and the stack memory is dynamic. If there is no more free memory the heap and the stack can no longer change in size and system errors would occur.

Memory Optimization

To avoid collision between the heap and the stack is it necessary to optimize the memory and the program code for the Arduino such as:

- Remove dead code such as unused libraries, variables and functions.
- Rewrite repeated code as a functions.
- Since strings are static it can be moved from SRAM to the flash memory by the command *printf()* instead of the ordinary print() function.
- Use suitable data type for variables. [36]

3.2 Breakout boards

Breakout boards are electrical components integrated on a circuit, also called integrated circuit (IC) [39]. The breakout boards basic concept is to add on a functionality, property and features to a circuit. Breakout boards usually have pins so they can be connected to the main board with just

a few cables. The amount of pins can change depending on the board and functionality but the most common pins that exits on a board is for the power supply, ground and some sort of transfer pins. There are many breakout boards available on the market and the ones used for this project are presented below.

3.2.1 MicroSD card breakout board

A data logger's main purpose is to read and write data to a storage place. The most common and space efficient way to store data is by using an SD card. The data logger in this project is an IC created by Adafruit and the SD card used is a class 10 Samsung microSD (see Fig. 3.10). The MicroSD card breakout board (see Fig. 3.9) uses a level shifting chip and not resistors which means it has faster read-/write access. The board supports both FAT16 or FAT32 for the filesystem. [40]

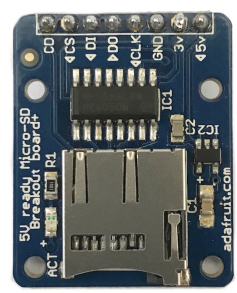


Figure 3.9: Data logger.

3.2.2 SD card

SD cards were first introduced in 1999 and announced by a collaboration between Toshiba, SanDisk and Panasonic [41]. Secure digital card or mostly know as SD card is an non-volatile memory[42]. SD cards store data and is most common in portable devices. The SD card used is a Samsung 64GB, microSD UHS-1 enabled class 10 card.

Figure 3.10: Samsung micro SD card.

3.2.3 Real time clock

A real time clock (RTC) is a circuit that keeps track of the current time. The RTC used is based on the DS3231N chip which is a extremely accurate (1 min/year deviation), low cost and temperature compensated RTC [43]. It is used in most electronic devices that needs to track time. RTC:s have a high accuracy and is often equipped with a back up power source to be independent and keep up the time counting while the main device is turned off. This RTC:s use a crystal oscillator. By counting the oscillation of the crystal the time can be tracked [44].

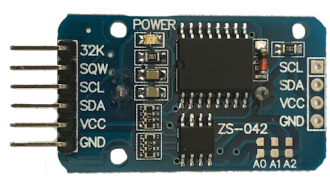


Figure 3.11: This RTC is build on DS3231N and communicates through I^2C.

3.2.4 Bluetooth module

The Bluetooth module (see Fig. 3.12) from Sparkfun is based on the RN-41 Bluetooth module (see section A.8). The Bluetooth mate gold is a class 1 Bluetooth module with long range connectivity meaning 100 meters of transmission distance and its frequency is based around 2.402 2.480 GHz. It uses a encrypted connection, operates with 3.3v-6v and can use as high as 115200bps with its serial connection [45].

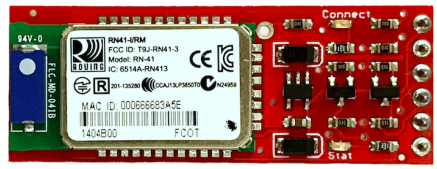


Figure 3.12: Bluetooth breakout board for Arduino

3.3 MyoBock hand prosthesis

The hand prosthesis used in this master thesis is the MyoBock from Ottobock (see Fig. 2.1). It is an active electrical hand prosthesis. The hand has a three finger grip with the possibility to disguise it with a cosmetic rubber glove in form of a real hand. To control the electrical motor there are eight connection pins on the bottom of the hand.

The pins (see Fig. 3.13) are numbered as seen in the list below and are presented with the following functionality.

1. +7V
2. GND
3. +7V
4. GND
5. EMG signal - Closes the hand
6. +7V
7. GND
8. EMG signal - Opens the hand

3.4 Sensors

The particular sensors used in this master thesis are explained in this section.

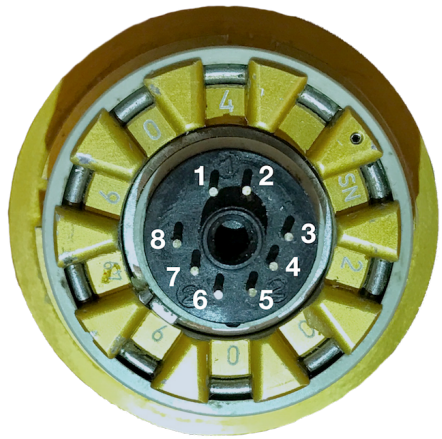


Figure 3.13: The pins of the Ottobock hand.

3.4.1 Force sensing resistor

The force sensing resistor (FSR) Interlink Electronics FSRTM 400 has a diameter of 3 inch (7.62mm) and it is constructed with a polymer thick film (PTF) (see Fig. 3.14).

Figure 3.14: The FSR 400.

When no force is applied on the sensor it delivers a resistance of approximately 10MΩ. The resistance decreases as in figure 3.15 when force is applied. The output voltage approximately follows the equation (see equation 3.1).

$$V_{out} = \frac{R_m * V+}{(R_m + R_{FSR})} \tag{3.1}$$

The voltage output follows the graph (see Fig. 3.16) which has an exponential decay curve. The R_m can be chosen according to the figure for desired response. The FSR has a rise time of less than 3 microseconds (see sppendix A.9).

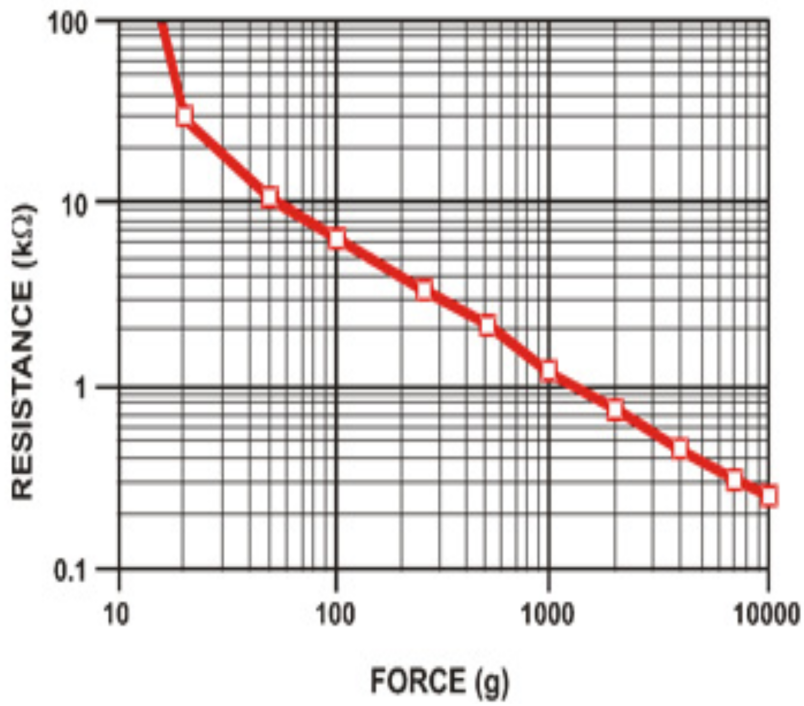


Figure 3.15: Resistance decay when applied force.

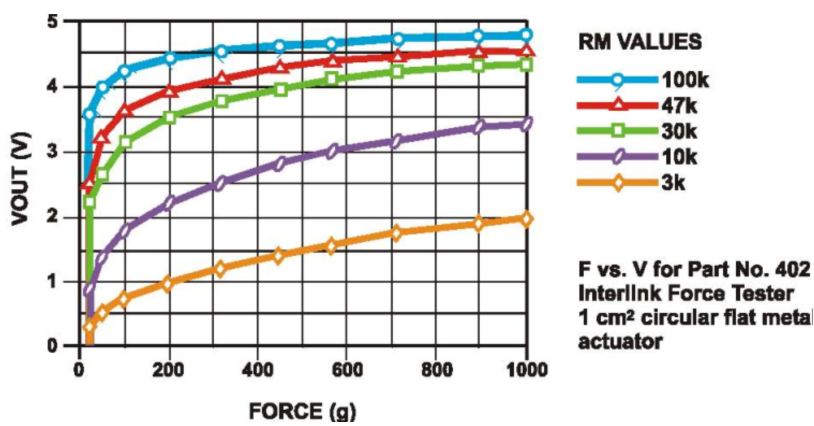


Figure 3.16: Voltage output when applied force.

3.4.2 Vibrators

The Precision Microdrives (310-101) vibrators (see Fig. 3.17) are 3.4 mm in body length, 10mm in body diameter, weigh 1.2g and are easy to implement. They have a voltage range of 2.5-3.8 V, a rated speed of 12000 rpm and vibration amplitude of 0.8G. [46]

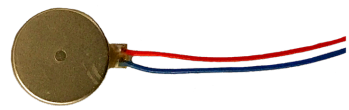


Figure 3.17: The 310-101 vibrator.

3.4.3 Myoelectric control

Myoelectric control as used in the EMG sensors for this project, uses the electrical activity of a contracting muscle as a control signal. The electrodes are placed over two muscle sites, flexor and extensor. The output from the flexor could be used for opening the hand and as a contrary the extensor signal is used to close the hand. When a contraction in the muscle occurs and with values over a certain threshold the EMG sensor sends the data on the analog output.[16]

3.4.4 EMG hardware

The Ottobock electromagnetic sensors (see Fig. 3.18) have an operating voltage of 4.8-7.2V. The electrodes are made

(a) The EMG sensor.

(b) The back of EMG sensor.

Figure 3.18: The EMG sensor for registration activities in the muscles.

of titanium, which makes them suitable for most people, they weigh not more than 4.5g. The EMG senors has built in amplifiers which are easily adjusted to the user (see Fig. 3.18b). They have two inputs and one analog output. The pin connection for the EMG sensor (see Fig. 3.19) have following pins:

1. Signal output
2. GND
3. 4.8V-7.2V

Figure 3.19: The pin connection of the EMG sensors (see appendix A.6).

3.5 PCB

A printed circuit board (PCB) is a board that connects electrical components via conductive tracks and pads on an isolated surface. Depending on how many sides that are electrically conductive on the PCB it can be single sided, double sided or multi-layered whereas the different layers are connected through vias. The conductive surface is usually a copper plate. The design of the PCB is constructed in a suitable program, such as EagleCAD (see section 3.5.1) and then manufactured accordingly.[47]

3.5.1 EAGLE

Eagle is a free schematic editor, compatible with Mac OS, Windows and Linux. The free version has all the necessities for simpler circuits but has a limit of two layers. The program has a schematic view and a circuit board layout. In the schematic view all the components are connected as

intended and they are imported through different libraries which are available for most manufactures online. In the PCB layout view all the components need to be dragged on the circuit, switched around and placed in a sensible matter. The rats nest button refines some of the connections drawn. Eagle also has an auto-router that routes the tracks accordingly to user settings such as cost factors, sizes and vias. The program compiles the circuit, tries to find errors and gives a percentage of completed circuit, where 100% is eligible. The program also has a rip-up and total rip-up option that removes all the tracks if the schematic needs an overview and a redo.

When the circuit is fully compiled, the necessary files needs to be generated that are suitable for the CAM machine to produce the circuit. The schematic layout is presented under appendix D.[48]

Chapter 4

Experimental work and result

Before implementing the various breakout boards and different approaches, a stand-alone Arduino device was used. The Arduino Uno was used as a test device before transferring functional code to the actual setup. This approach meant gradually implementing functions and methods in the stand-alone device before implementing them in the actual setup. The experimental work is to, step by step, use the theory explained in previous chapter and add features and modifications until a functional feedback system is presented.

The following chapter will cover the setup performed with LabVIEW, how the appropriate choice of transfer functions, how the basic reading of the packages were analyzed. The whole setup will be thoroughly explained and the code will be commented. The total setup is shown in figure 4.1. This chapter continues by explaining the slave setup and its connected hardware meaning how the force sensors are connected to the slave Arduino and how the output signal for the hand prosthesis is generated. The chapter proceeds to explain how the master Arduino function and how the breakout boards are connected and implemented.

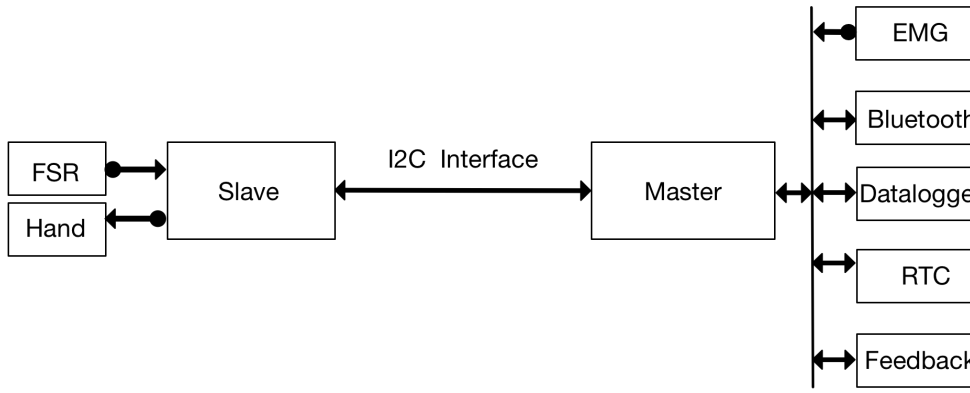


Figure 4.1: The total setup.

4.1 LabVIEW

LabVIEW is a graphical programming tool used to communicate and process data from hardware[49]. The goal is to communicate with the Arduino and make a rough setup, collect the data from FSR and send processed data from LabVIEW back to the feedback system.

4.1.1 Arduino setup for LabVIEW

The FSRs are used and represented as five fingers (see Fig. 3.14). When force is applied the resistance will drop and the voltage will rise (see Fig. 3.16).

The overall setup is presented as following (see Fig. 4.2).

1. The Arduino will register the data from the FSRs.
2. Sends the data though serial to LabVIEW.
3. LabVIEW will process the data and present it in a graph.
4. Send values back to the Arduino.
5. The data is processed in the Arduino and adapted to the feedback system.

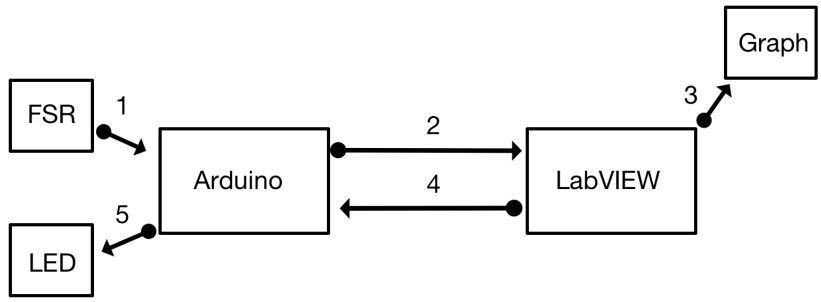


Figure 4.2: Sketch of the setup between LabVIEW and Arduino.

The received data from LabVIEW is sent to the Arduino and sent as PWM to the LED:s which represent the feedback system. The LED:s will emit light in different intensities depending on the force applied on the force sensors (see code LabVIEW and Arduino in appendix B).

4.1.2 LabVIEW setup

The goal with LabVIEW is to get a clear overview of the force sensor values, separate the data from the Arduino and address the corresponding value to the fingers.

LabVIEW code

The connection begins by using an auto detect program in LabVIEW for detecting the Arduino board (see appendix B.1)[50, 51]. The data from Arduino is processed in the while loop in LabVIEW. The received data arrives as the following format:

Finger1+Finger2+Finger3+Finger4+Finger5

In the while loop the data is collected and sorted out with the LabVIEW code (see appendix B.1). Each value is presented in a graph to the corresponding finger (see Fig. 4.3).[51]

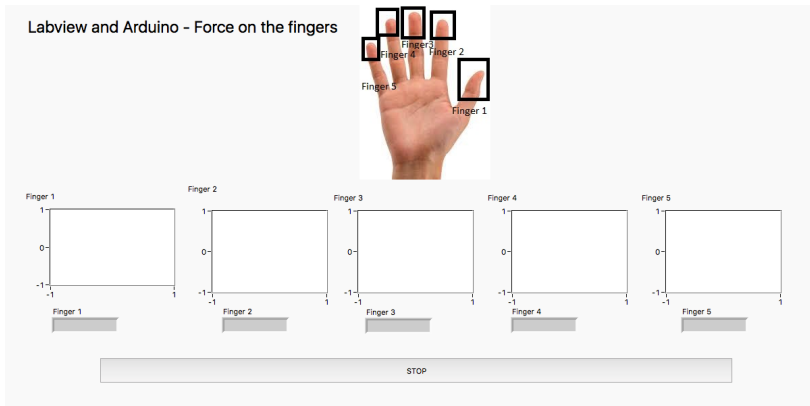


Figure 4.3: Graphical template presenting the force over time for each finger, no data in figure.

4.2 The data transfer between the two Arduinos

The hand prosthesis and the socket are connected via a coaxial cable, which only contains four cables. The only sensible connection between the slave and master is software serial or I^2C (called two wire interface in Arduino [52]). For reasons explained in discussion under section 5 only the I^2C transfer will be explained here. The setup with a slave and a master is chosen because there need to be one microcontroller in charge, being the one controlling the different transfers so no collisions occur and so the setup is rapid, robust and functional.

4.2.1 Reading packages

In order to choose the appropriate transfer method a investigation of the transferred data through serial was conducted. This was done by sending the letter "F" and reading it on the oscilloscope (see Fig. 4.4). The figure shows that the data package starts with Tx as high to notify that nothing is on the line. Before a transfer begins the signal starts with a start bit followed by *01100010* and a stop bit. The data package is decoded to the letter "F" in hexadecimal.

They whole package, 10 bits was transferred in *1.0445ms*

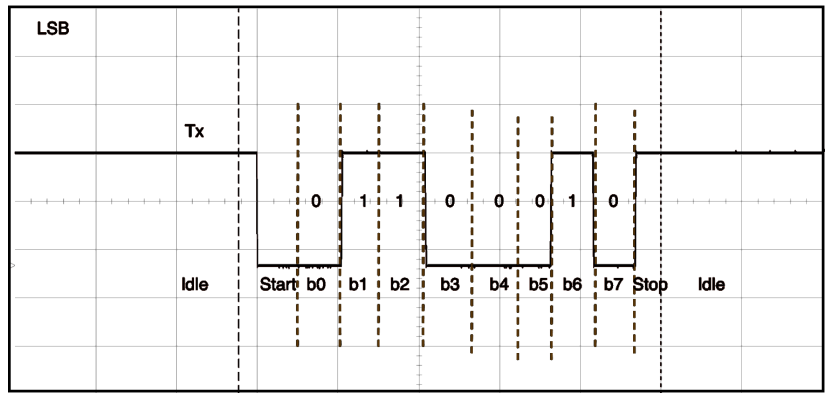


Figure 4.4: Reading data package and translate it.

and with this parameter the baud rate is calculated.

$$\frac{1}{0.00010445} \approx 9574 \tag{4.1}$$

The baud rate was set to 9600 Bd in the serial window and in the program code. Higher baud rate will increase the transfer speed but if the set baud rate is set to a totally different setting than for the serial window only nonsense will appear in the print. The chosen baud rate for this project is 115200 Bd from now on.

4.2.2 I^2C

The connections between the slave and master Arduino are: ground, +5 or +7 V, SDA (Serial Data Pin), and SCL (Serial Clock Pin). On the Arduino Nano card the analog input port A4 is used as SDA and the A5 port is the SCL (see the Arduino schematic in appendix A.1).

The wire transfer begins (see the code in appendix C) when one of the devices claims the transfer bus and that the device would like to begin sending information. The transmission is a request event sent to one of the devices, interrupting it to stop its process and start transmitting (see code in appendix C.2.3). The receiving device has a receive event function that handles the transmitted data [52].

4.3 The master Arduino

The master Arduino will be placed closest to the socket, receiving input data from the EMG sensors and start the sensory feedback, which in this case are vibrators with the information received from the slave via I^2C. The master microcontroller has the most added breakout boards and controls the whole setup (see Fig. 4.5). In the following section the main setup will be described along with all the interfaces added to the microcontroller.

4.3.1 The master setup

For the master Arduino, libraries needed explicit for the different breakout boards used are added in the setup (see Arduino code appendix C.1.1). All the imported libraries are commented in the program code (see code in appendix C.1.1). Most of the global declarations in the master are connected with the Bluetooth interface (see code in appendix C.1.3) and the calibration (see code in appendix C.1.4), these are commented in the Arduino code as well.

The wire ID for the master is set to 0, which is the default for a master device. Some functions such as the clock, reading the text files for different settings and running the calibration program will run when rebooting the Arduino and these functions will be explained in the sections 4.2.1, 4.3.5 and 4.3.7.

4.3.2 The loop

The received FSR values are calibrated with the constants stored in text files. Before transmitting the force sensor signal to the vibrators, the values are written to the SD card along with a time stamp. In the masters loop the EMG values are also read and mapped accordingly to the desired values stored in the text files. In order to start the I^2C transfer there is a threshold value that claims if the EMG values are above a certain value (30) the transfer should begin, otherwise it should check if any values occur on the FSRs and if not, go to sleep (see sleep mode section C.1.9).

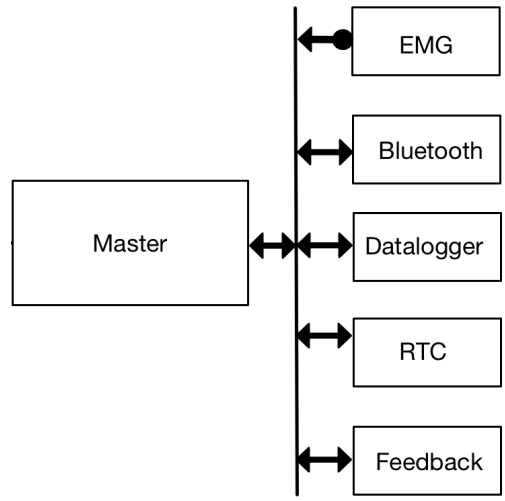


Figure 4.5: The hardware connected to the master Arduino.

4.3.3 EMG sensors

The EMG sensors are connected to the analog input A0 and A1 on the master. These values are linear calibrated with the values stored in the text files for the EMG sensors. The EMG values are then mapped so the appropriate voltage will be delivered to the hand prosthesis in the slave (see calibration section 4.3.5).

4.3.4 The digital output

The vibrator output needs to be a PWM driven output, since there needs to be a change in intensity. The available PWM on the Arduino Nano are D3, D5, D6, D9, D10, and D11 (see the schematic in appendix A.1). These have different output frequencies which is connected to the different built-in timers in the Arduino Nano. D3, D9, D10 and D11 are running a PWM signal of 490Hz while D5 and D6 are running on 980Hz (see Fig. 4.6) [53]. The upper graph is the signal with a higher frequency, meaning the D10 port with its 980Hz, and the lower graph is the D6 pin with 490Hz, which is 0.5 ms longer in its duration. There is a option to change the frequency for the ports but not so it will match up with the exact same frequency. This will reset the built-in timers making functions such as de-

lay() malfunction. The vibrators will therefore oscillate at different frequencies but have the same voltage which results in the same force in the vibration. The vibrators are connected on the digital outputs D3, D5, D6, D9 and D10.

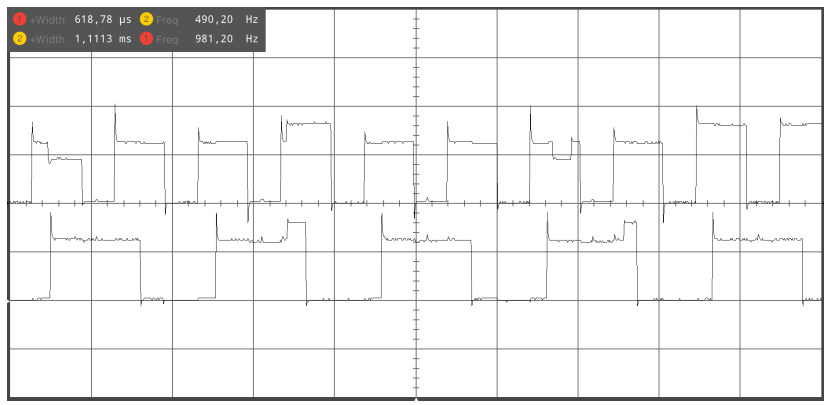


Figure 4.6: The different frequencies of digital output D6 (490Hz) and D10 (979Hz).

4.3.5 Calibration

To make it possible to use alternative sensors on the input and the output an calibration mode to calibrate according to the hardware connected was developed. When the master Arduino boots the user has 5 seconds to enter the calibration mode via Bluetooth by typing "c" in the serial while connected to the Bluetooth device. All calibrations are stored in text files saved on the SD card.

Bluetooth interface

There are three main calibration states. The Bluetooth dongle is connected to the master Arduino and the main purpose with it is to calibrate and check the setup. The Bluetooth listens on the serial input if value 0, 1, 2 or q is being typed, once entered calibration mode. The Bluetooth port is connected via software serial port 7 and 8.

State 0 is the time setting, state 1 is the EMG calibration and the force sensor calibration, state 2 is a control setting which checks RTC, time settings, the datalog.txt file, cal.txt file, calE.txt file, what the txt files contains and a check of

the SD card. To quit the calibration mode the Bluetooth dongle listens for the character q.

State 0

If the time needs to be set, for example if the RTC loses its power from the back-up battery, state 0 sets the time. This is done exactly as inputting the calibration values in state 1 (see state 1 section below) but the time is stored directly on the RTC instead of on the SD card. The time is adjusted and set with the RTC built-in feature. How the values are stored and parsed is shown and commented in the code (see Bluetooth code section C.1.3).

State 1

In state 1 new k-values, c-values, ke-values, ce-values and a voltage value are implemented and written to the SD card in three different text files (cal.txt for k and c, calE.txt for ke and ce and v.txt for voltage). The k and c value calibrates the force sensors with a linear calibration ($y = kx+c$). The ke and ce also calibrates the EMG sensors with the same linear calibration ($y = kex+ce$). The voltage value is the desirable voltage output from the digital ports connected to the hand.

The potential old values are erased when new values are given, in reality the text files are erased and replaced with new ones. Observe that the user is asked to type one character at a time to give the Bluetooth enough time to process the incoming character. For every typed character a print to the serial window (connected via Bluetooth) is made to see the progress of the incoming characters. When * is typed the user is sent to a test-mode where the values are tested if they seem to be in range. To quit this mode the user should press q and is then asked: *k, c, ke, ce, v Are these ok? Y/N* and is asked to type *y* or *Y* for yes and *n* or *N* for no. If no, the user is sent back in the loop to type new values and if yes the program continuous. A shorter print of the serial window is shown below.
1,0,1,0,
1,0,1,0,1,
1,0,1,0,1,1.2

1,0,1,0,1,1.25
Try the sensor, press enter and blankspace for new values, q for quit
FS 0 Expected value (0-255)
0
FS 1 Expected value (0-255)
0
FS 2 Expected value (0-255)
0
FS 3 Expected value (0-255)
0
FS 4 Expected value (0-255)
144
EMG 0 Expected value (0-255)
10
EMG 1 Expected value (0-255)
7
Expected voltage out
1.25
k, c, ke, ce, v Are these ok? Y/N
1,0,1,0,1.25
k, c, ke, ce, v values are set!
Press 0 for timesettings, 1 for sensor calibration, 2 for controlling stuff

State 2

State 2 is the control setting. In this mode the RTC, time setting, datalog.txt, cal.txt, calE.txt, v.txt and Sd-card are checked. The different text files are controlled by trying to opening the files and writing *FileName.txt check ok* if the file exists or writing *FileName.txt not available* to the Bluetooth serial window if the text file is not available. The calibration files also writes the different calibration values stored in the different files in the Bluetooth serial. The SD card and the RTC is controlled by trying to start them with the rtc.begin() command and the SD.begin() command. A message on the Bluetooth serial will be received if the initialization was successful or not.

Mapping the voltage value

The voltage value set in the calibration (see Bluetooth section 4.3.5) is the voltage the user needs to have as an output to the hand. The Ottobock used in this master thesis operates fairly well when the output is around 590 mV - 1.25V. In order to obtain the desirable voltage output the values are mapped to get as close as possible to the input voltage value in calibration. The mapped value is written as:

$$dataEmg[1] = map(dataEmg[1], 0, 255, 0, 65.5 * v)$$

Meaning it will map the EMG values from 0-255 to 0-65.5 and multiplied with the desirable voltage output. The 65.5 value comes from investigating different voltage output at the end of the RC-filter (see low pass filter section 4.4.3) and dividing that value with the corresponding EMG value. The result is shown in table 4.1.

PWM	LP-filter output	Byte-value	Division
0%	0.1V	0	0
25 %	1.25V	$255 * 0.25 = 64$	$64/1.25 = 52$
50 %	2.1V	$255 * 0.5 = 127$	$127/2.1 = 60.5$
75 %	2.9V	$255 * 0.75 = 191.25$	$191.25/2.9 = 66$
100 %	3.4V	$255 * 1 = 255$	$255/3.4 = 75$

Table 4.1: Table of the calculated mapped value.

The mean value of the mapped values (see equation 4.2).

$$\frac{52 + 60.5 + 66 + 75}{4} = 64 \tag{4.2}$$

Experiments were conducted and the result is shown in table 4.2 and the best result was obtained with a mapped value of 65.5.

Voltage input value	Actual voltage output	Differs
0V	0V	0V
0.5V	0.44V	50mV
0.8V	1V	200mV
1V	1.18V	180mV
1.25V	1.25V	0mV
1.5V	1.46V	0.04mV
1.8V	1.9V	100mV
2V	2,1V	100mV
2.25V	2.3V	0.05mV
2.5V	2.55V	0.05mV
2.8V	2.78V	0.02mV
3V	2.92V	0.08mV

Table 4.2: Table of the voltage input in the calibration versus the actual output voltage when using mapped value 65.5.

4.3.6 Adding values to the SD card

For debugging purposes and future work the FSR values, EMG values, a time stamp along with all the text files are stored on a SD card. The sensor values are stored in a string separated with ";" and stored in a text file called datalog.txt on the SD card. The datalog.txt contains the following information:

2017/1/25 10:4:40-0;0;0;0;0;0;0;
2;2;0;0;0;0;0;
2;1;0;0;0;0;0;
...

The lines consist of: year/month/day hour:minutes:seconds-EMG1;EMG2;FSR1;FSR2;FSR3;FSR4;FSR5;

4.3.7 Clock

The RTC is implemented with the RTClib.h library [54] which contains necessary functions and features to parse and store the date and time. The if statement in the code is just an extra control so the RTC started properly. If the

year, month or day has wrong values, the function will try to start the RTC again (see Arduino code appendix C.1.5). The program appends everything in a string called time and separates the values with "/", ":" and "-". This is the string appended to the datalog.txt to store the time in the text file.

4.3.8 Reading calibration text files

In the Readtxt() function the first text file CalE.txt is opened, the calibration values *ke* and *ce* are parsed out and stored globally and the file is closed. The text file Cal.txt is then opened, the calibration values *k* and *c* are parsed out and stored and the file is closed. Lastly the text file v.txt is opened, the value is parsed and stored and the file is closed. The code can be seen in Arduino code in appendix C.1.6.

4.3.9 Receive event

When the slave Arduino requests to send data, the receive event function (see Arduino code appendix C.1.7) is ready to receive bytes of data on the I^2C. The function holds the wire open and available until the expected amount of bytes has been received and then closes the line.

4.3.10 Sleep mode

In order to save power, the processor will go to sleep if the threshold values are below its limits. Inside of the main program there is a sleep counter called "sleep" which increases for each loop if the values are below threshold, meaning that the processor will run the program 11 times before entering sleep mode, 11 times gave the perfect amount of time for the setup to settle down. The sleep mode set in this particular setup is set for the most aggressive sleep (see code in appendix C.1.9). In sleep mode power down, only an external reset, a watchdog system reset, a watchdog interrupt, a Brown-out reset, a 2-wire serial interface address match, an external level interrupt on INT0 or INT1, or a pin change interrupt can wake up the MCU [30]. Only the external interrupt is used in this project.

Besides this only a low pin interrupt can be used with the power down sleep mode meaning, the pullup-pin (digital pin D2) is set on high in the code from the beginning and the attached interrupt will only go active once the D2 pin gets low. The different sleep modes available are set or commented (see sleep mode appendix C.1.9) as the first step in implementing the sleep mode. The most aggressive sleep mode, *SLEEP_MODE_PWR_DOWN* is set. Sleep is enabled and if an interrupt occurs it is disabled (see interrupt section 4.3.11). To enter the sleep mode in the master, the EMG values needs to be below the threshold (30) set in the code (see Arduino code appendix C.1.9). Before the microcontroller goes into sleep all the outgoing values are set to 0 so no floating values occur. The current drawn from the circuit before implementing the sleep mode was about 180 mA-110 mA depending on if the EMG sensors were toggled or if the FSRs were active. With the sleep mode enabled the current dropped to approximately 35 mA which is a current reduction of 145 mA.

4.3.11 External interrupt

One of the benefits with external interrupts is that the device can be in a deeper sleep which draws the least power. An external interrupt means that activity externally such as activity from the FSR or EMG sensors will awaken the whole system. The external interrupts pin (D2) is set be low in order to wake up the Arduino from its deep sleep. A circuit was designed with properties so the output signal goes from high to low when the circuit is detecting any activity on the in signal.

The EMGs and the FSRs are connected to two Toshiba TC4071BP quad 2 input OR gate (see appendix A.4) and then inverted in two MC14001BCP quad 2-input NOR gates (see appendix A.2) in order to get a pure digital low signal (see Fig. 4.8). Both data sheets for the integrated circuits can be found under appendix section respectively. The threshold for the MC14001BCP is when the values are above half of 5V meaning 2.5V will push the output to be set to 0.

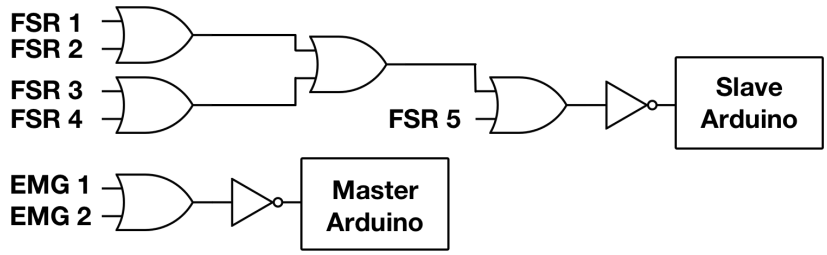


Figure 4.7: Interrupt IC schematics

4.4 The slave Arduino

The slave Arduino is connected closest to the hand prosthesis and is thus the micro-controller that receives the FSR and delivering the signal to the motor in the hand. The electromyography signal arrives from the master via I^2C.

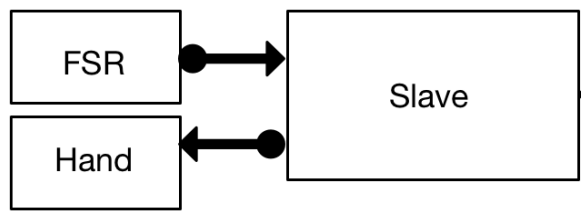


Figure 4.8: The hardware connected to the slave Arduino.

4.4.1 The slave setup

After including necessary libraries and declaring global declarations the setup is being run in the Arduino slave code. The force sensors are connected to the slave Arduino Nano card on the analog inputs A0, A1, A2, A3 and A6. The analog input values from the force sensors are divided by four since the input value varies from 0-1024, so instead of transferring several bytes, only one byte consisting of a value between 0-255 will be transferred via the I^2C interface (see Arduino code in appendix C.2.1). The slaves wire ID is set to 8. There is a threshold set before the slave is allowed to begin its I^2C transfer and the threshold states that both FSRs needs to be above 5 and the EMG:s needs to be below 30.

4.4.2 Force sensors

The FSR 400 can be connected with resistors varying from 3kΩ - 100kΩ depending on which voltage output is desired A.9. The response follows figure 3.16. In order for the interrupt to be able to go off and wake up the circuit and for proper response on the vibrators, an 47kΩ resistor is used.

4.4.3 Low pass filter

Since the Ottobock hand prosthesis requires a continuous voltage level a low pass filter (see Fig. 4.9) is needed between the analog output of the Arduino and the input of the hand. The low pass filter attenuates all of the signals with higher frequencies and let the lower frequencies pass.

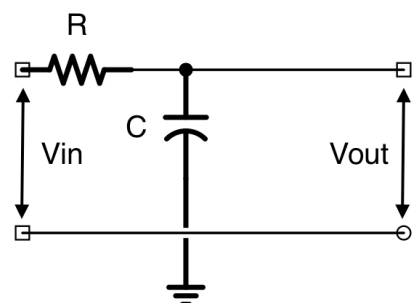


Figure 4.9: Schematic of the low pass filter.

When the frequencies increase to the cutoff frequency the capacitor will behave as a shortcut in the circuit.

The low pass filter should be designed with low ripples and so the desired voltage in relation to the amplitude and duty cycle follows equation 4.3.

$$V_{DAC} = A * (DutyCycle) \tag{4.3}$$

V_{DAC} represent the converted voltage, A is the amplitude of the signal (5V for Arduino Nano) and the duty cycle is presented in percentage of the PWM signal. This conversion from a digital signal to an analog signal is vital in order to control the motor in the hand. [55]

The values on the resistor and the capacitor were chosen to have a fast rise time and as little ripple as possible.

The low pass filter was created with a resistor of 15kΩ and a capacitor of 1μF. [56]

The PWM and the corresponding voltage output is presented (see Fig. 4.10-4.19).

4.4.4 Signal measurements

Before applying PWM on the circuit the signal from Arduino was investigated to confirm the theory and to better understand PWM for future debugging. A signal from Arduino was generated through the digital ports which contained the PWM option. The signals have small high peaks which indicate where a new period of the signal starts. Changing the value from the Arduino will change the output signal and the pulse width will be wider. The values are changed to match up and confirm with the theory (see Fig. 3.7) meaning PWM from 0% duty cycle until 100% (see Fig. 4.10-4.19).

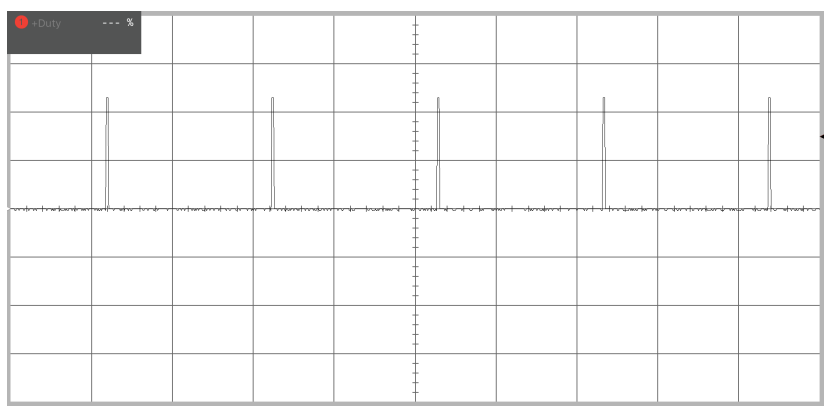


Figure 4.10: PWM output before LP-filter with duty cycle 0%.

4.4.5 The digital output

In the slave, the two digital outputs used to trigger the motor in the hand are the digital output 5 and 6 since they are PWM outputs as explained earlier. Before entering sleep mode (see sleep mode slave section 4.4.6) the digital outputs are set to 0 so no floating values occurs. The code is

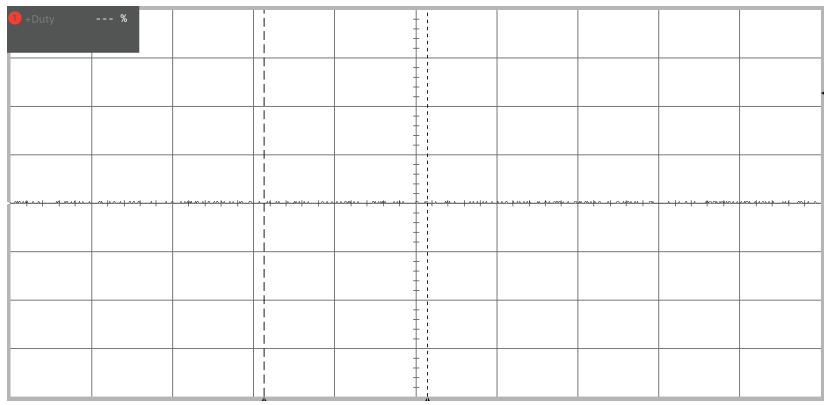


Figure 4.11: 0V output after LP-filter with duty cycle 0%.

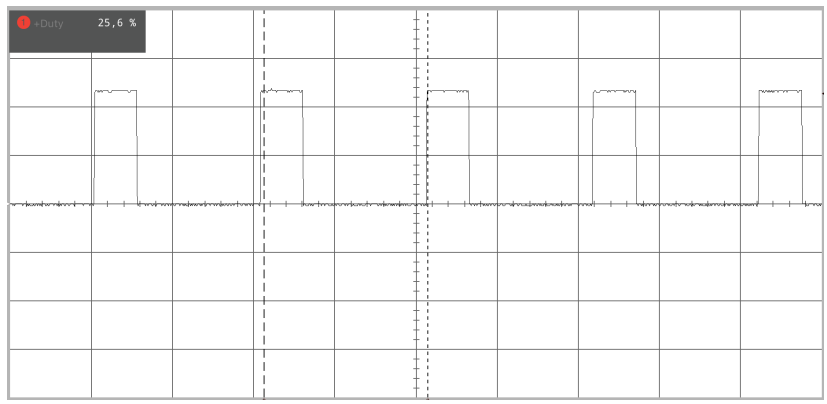


Figure 4.12: PWM output before LP-filter with duty cycle 25%.

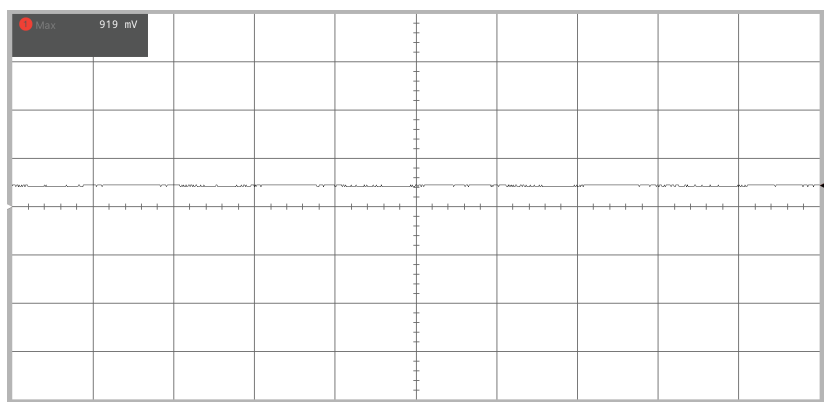


Figure 4.13: Approximately 0.9V output after LP-filter with duty cycle 25%.

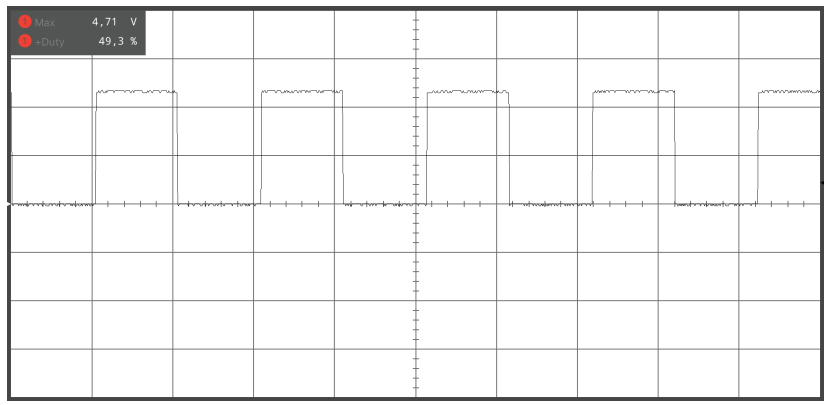


Figure 4.14: PWM output before LP-filter with duty cycle 50%

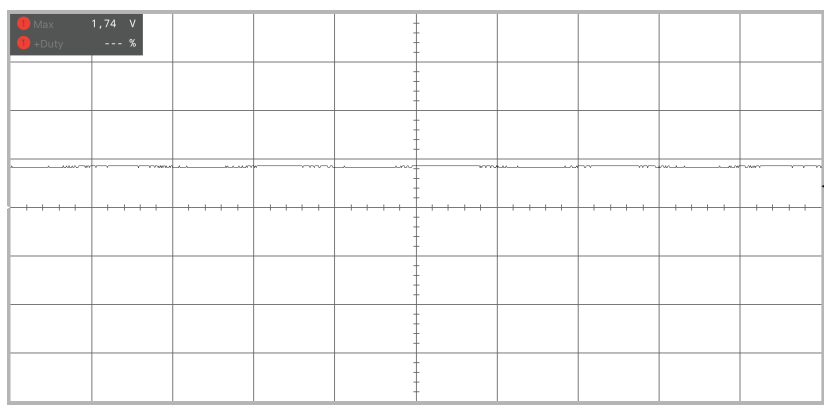


Figure 4.15: Approximately 1.7V output after LP-filter with duty cycle 50%

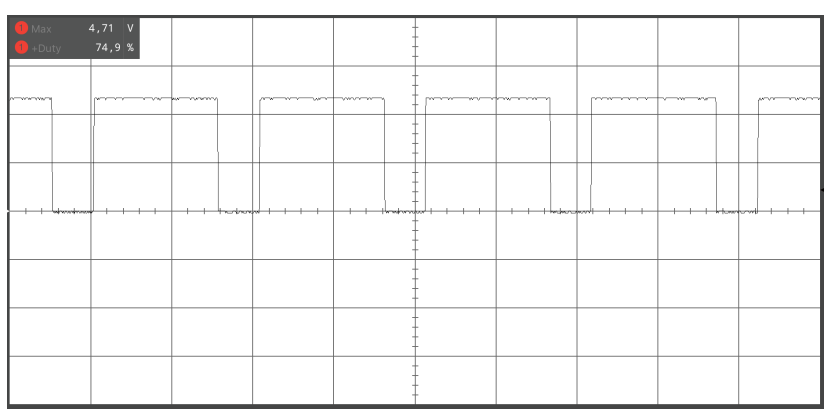


Figure 4.16: PWM output before LP-filter with duty cycle 75%

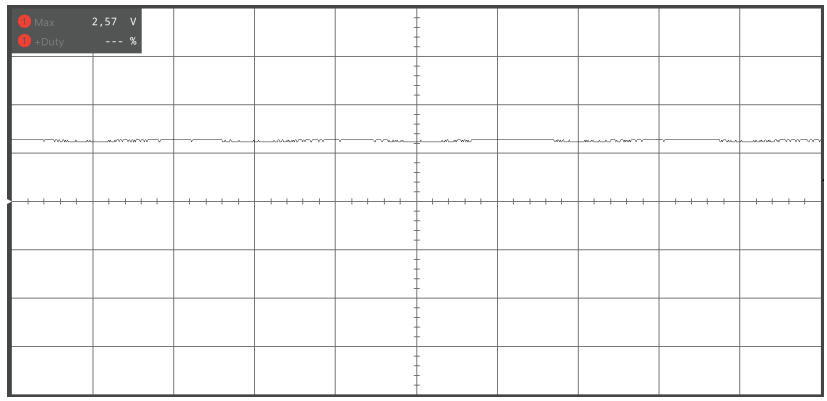


Figure 4.17: Approximately 2.6V output after LP-filter with duty cycle 75%

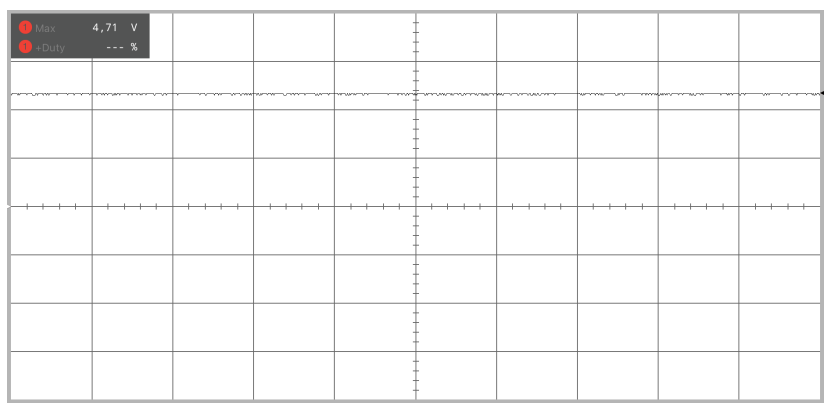


Figure 4.18: PWM output before LP-filter with duty cycle 100%

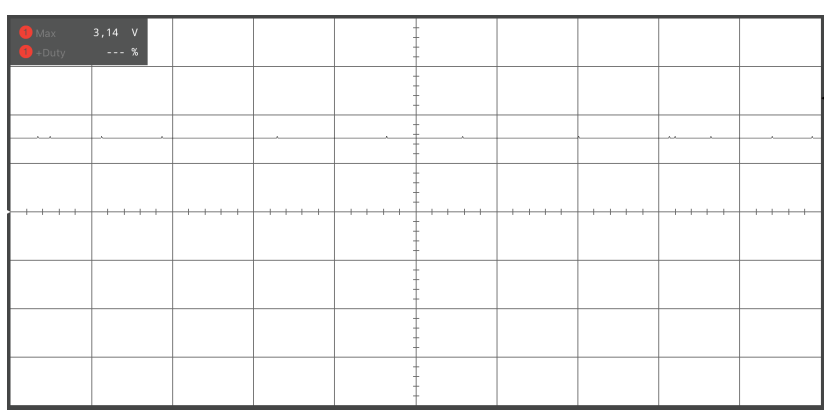


Figure 4.19: Approximately 3.1V output after LP-filter with duty cycle 100%

appended and commented in appendix (see appendix Arduino code C.2.1).

4.4.6 Sleep mode

The sleep mode works exactly the same as for the master (see section 4.3.10) besides the initial threshold.

4.5 Power consumption

Since the setup will be powered by a battery the total power consumption is investigated. The setup for the basic measurements are shown in figure 4.20.

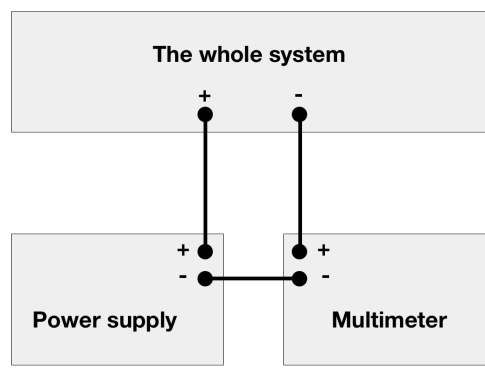


Figure 4.20: Measurement setup for power consumption of the total setup.

The Arduino is connected to an external power source. Between the source and the system the multimeter is connected to measure the current. In this case the Arduino was set to go in and out off sleep mode to easily observe how the current will change while measuring. The current changed as followed.

- Sleep mode enabled: 35mA
- Only FSRs and vibrators active: 110mA
- Full activity including movement of the prosthesis: 180mA

A calculation of the power consumption is shown in equation 4.4-4.11.

$$P = UI \tag{4.4}$$

$$U = 5V \tag{4.5}$$

$$I_1 = 35mA \tag{4.6}$$

$$I_2 = 110mA \tag{4.7}$$

$$I_2 = 180mA \tag{4.8}$$

$$P_1 = 5V * 35mA \approx 0.18W \tag{4.9}$$

$$P_2 = 5V * 110mA = 0.55W \tag{4.10}$$

$$P_1 = 5V * 180mA = 0.9W \tag{4.11}$$

4.6 Transfer time of the setup

It is of great interest to investigate how long it takes for the system to process one cycle of code so that the system is rapid enough so the sensory feedback feels natural for the patient. In order to ascertain that one full cycle is completed the digital port in the Arduino were toggled and observed on the oscilloscope.

4.6.1 Time Measurements

The Arduinos master code is written in a way that both FSRs and EMG values will be read so the masters total time for running once will contain all sensor values. The block diagram shows the tasks that are being done to investigate one loop (see Fig. 4.21). The digital port D3 is used to toggle a high and low signal in the master (see code in appendix C.1.1) so the total time of setup can be analyzed in the oscilloscope.

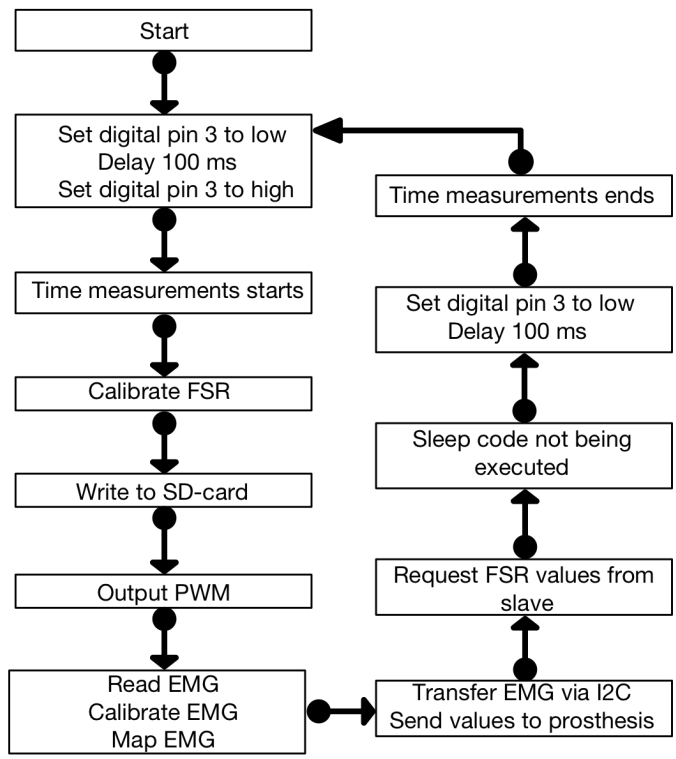


Figure 4.21: The block diagram of the loop in the time measurement.

The total time for the process is shown in figure 4.22. The process time for the master is measured to 5.6 ms and the time measurement for the slave is 0.9 ms. For the slave the toggle pin for the time measurement is the digital pin 4 (see Arduino code appendix C.2.1). Since there is a threshold value on the EMG values in the slave, the request transfer of the force sensors will not be run. The slaves process

time is merely reading FSR values, run the request function called from the master and deliver the EMG values to the hand on the digital outputs. The toggle in the master is on digital pin 3 and to force the code to run, the EMG values are above the threshold.

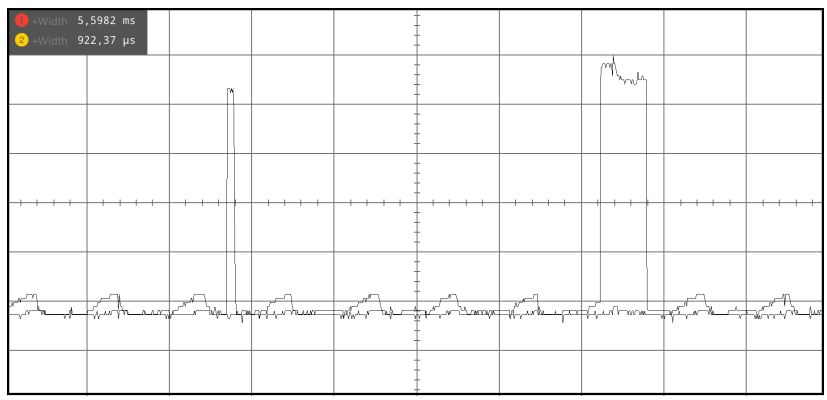


Figure 4.22: Measured time for the whole system. Master has a transfer time of $\approx 5.6ms$ (graph to the right) and the slave has a transfer time of $\approx 0.9ms$ (graph to the left).

SD card time measurements

The standard SD card library from Arduino was used with an older micro SD card initially. The old SD card without classification was compared to a newer one with class 10 classification. The old card (see Fig. 4.23) had a transfer time of 10ms. The newer card had a transfer time of 5.6 ms (see Fig. 4.24).

The time for the transfer without SD-card read or write (see Fig. 4.25) is 609 μs. When opening the SD card breakout board but not writing or opening the SD card, meaning, if no SD-card is inserted the transfer time is shown in figure 4.26. To sum up the experiment the total transfer time is 609 μs - 10 ms depending if it is needed to log information to the SD card. Since the calibration values are stored in the SD card it is of best interest to have one inserted at all times.

The biggest improvement regarding the SD card time measurements was made with changing the SD library that was originally used and thus decreasing the transfer time

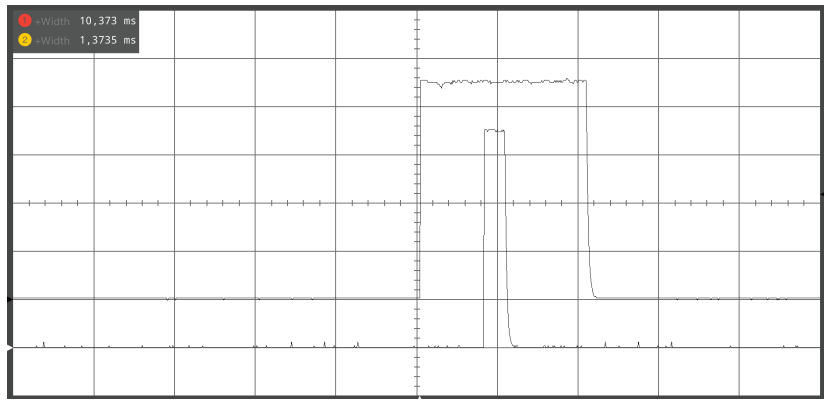


Figure 4.23: Process time for the loop with the old SD-card: master (upper graph) ≈10ms, slave (lower graph) ≈1.4ms

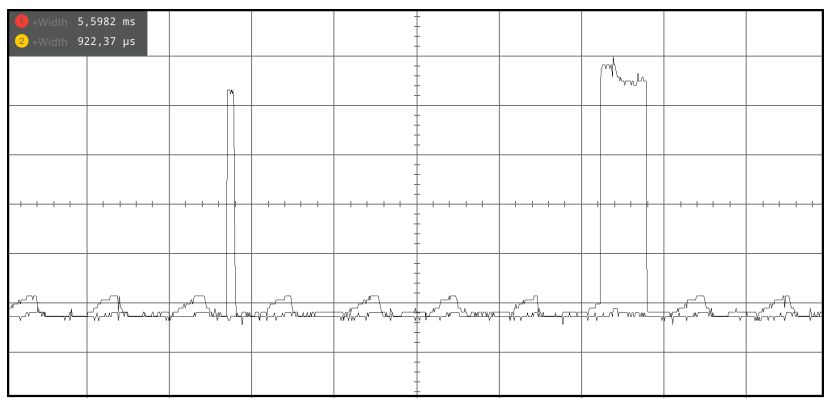


Figure 4.24: Process time for the loop with the new SD-card class 10: Master(graph to the right) ≈5.6ms, Slave (graph to the left) ≈0.9ms

from approximately 100 ms to below 10ms with the new SDfat library [57].

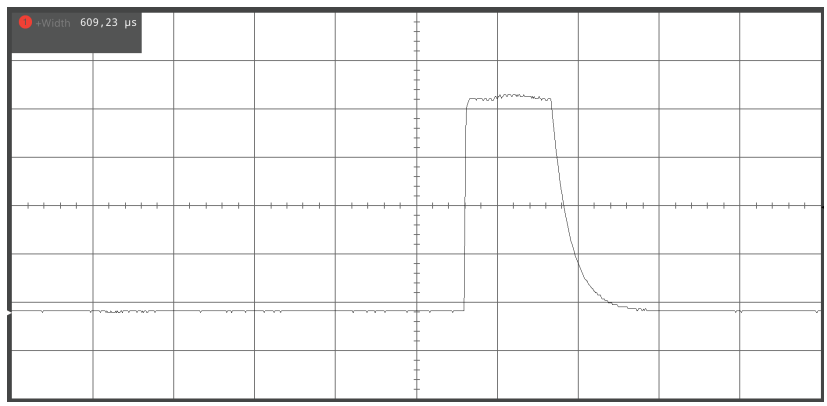


Figure 4.25: Process time for the master loop without writing to the SD-card breakout board, $\approx 600\mu$s.

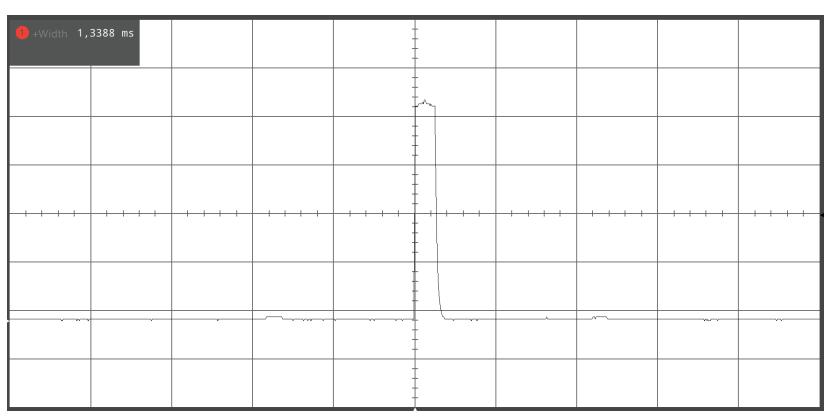


Figure 4.26: Time for writing to the SD-card breakout board but not to the SD-card, 1.3ms.

SD Card storage

The data is saved to the SD card as following:

[year/month/day hour:minutes:seconds-
EMG1;EMG2;FSR1;FSR2;FSR3;FSR4;FSR5;]

There are 14 characters and one new line (2 bytes) and it takes up totally 16 bytes. Available space on the SD-card is 62 864 228 352 bytes (64 Gigabyte). The maximum number of lines theoretically stored on the SD-card are:

$$\frac{62864228352}{16} = 3929014272 lines. \tag{4.12}$$

One line takes approximately $5.6ms$ according to figure 4.23. The following equation 4.13 is presenting for the time to run out of free space on the SD card.

$$3929014272 * 5.6 * 10^{-}3 \approx 2.2 * 10^{7} s \approx 366707 min \tag{4.13}$$

$$\approx 6112h \approx 255 days. \tag{4.14}$$

The calculation predicts that the SD card could run with the system continuous for 255 days. With sleep mode implemented the system do not run 24h each day, so the system should be able to run even longer than the predicted calculated days.

4.7 Result

This section will sum up the experimental chapter and compile the most important progress made during the different approaches. The chapter will present the finished product as is, how fast it is, how robust and how functional it turned out to be.

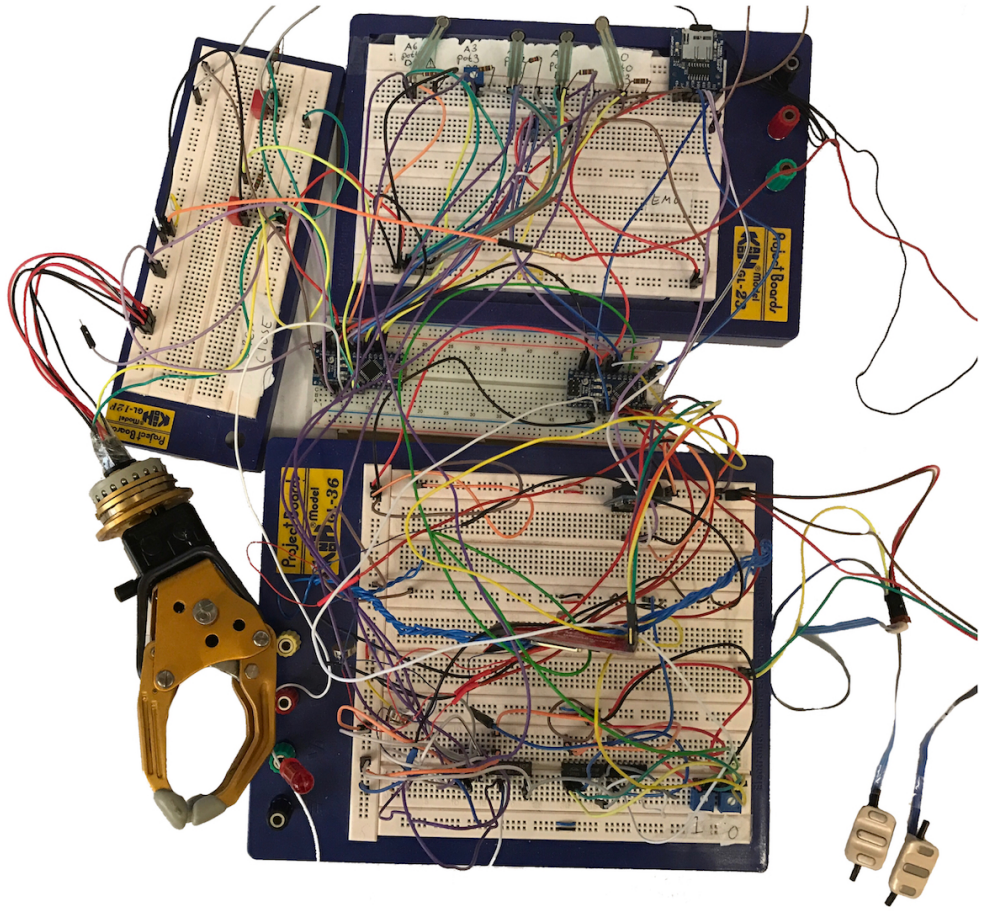


Figure 4.27: The total setup.

4.7.1 The Complete Setup

The complete setup (see Fig. 4.27) contains a fully functional sensory feedback system which was the main purpose of this master thesis. The whole setup (see Fig. 4.28) will be described from the FSRs and the hand to the left in the figure.

The FSRs and the prosthetic hand is connected to the slave Arduino. The slave communicates via I^2C to the master Arduino. The Bluetooth dongle is connected (via SoftwareSerial) to the master for calibration purposes, the EMG sensors are connected to the analog inputs, the data logger is connected via SPI, the RTC is connected via I^2C and the sensory feedback vibrators are connected to the digital output of the master Arduino. The idea with the

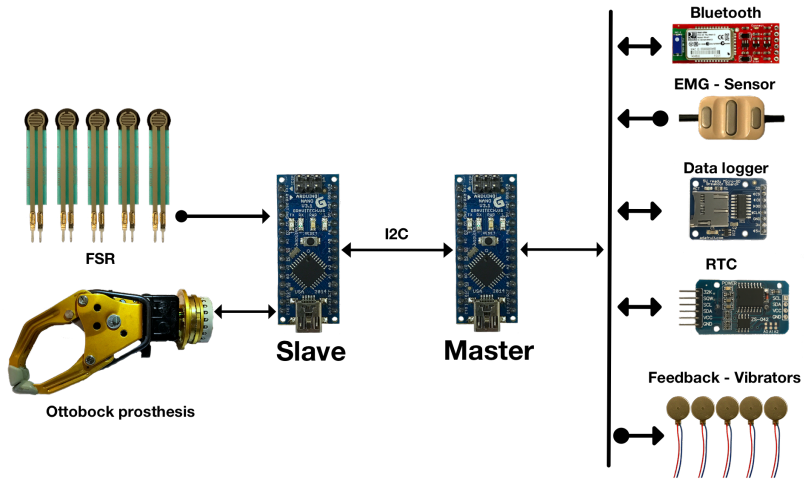


Figure 4.28: Graphical view of total setup.

calibration mode that is implemented was born after working with LabVIEW [16].

In addition to all the hardware some software features are also added such as the sleep mode, mapping values according to the hardware and storing these in text files (see Fig. 4.29).

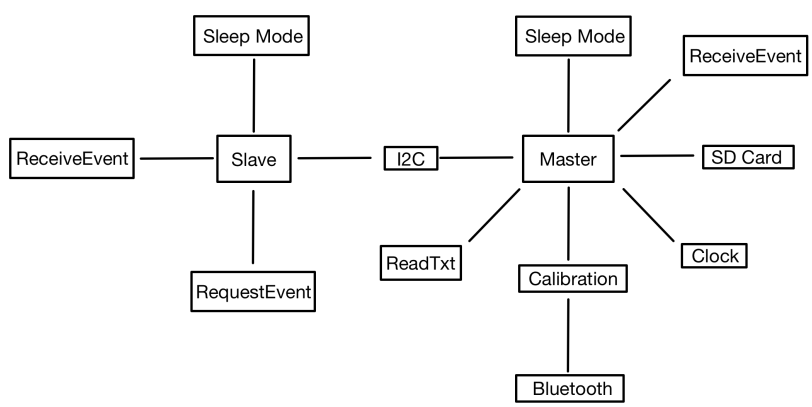


Figure 4.29: Graphical view of the programing setup.

The setup with its I^2C transfer is quite rapid. Around 6 ms is the response time for a signal from the EMG, until the user is given the sensory feedback. The I^2C interface itself has a transfer time of about 1 ms which is fast enough, the theoretical I^2C transfer is set to 400KHz.

Since the setup will be powered with an external power source (battery) the sleep mode was implemented. The current drawn from the circuit was at approximately 180mA and dropped to about 35mA when sleep mode was enabled.

The output data is written to the SD card and the print was minimized in a format as short as possible but at the same time readable for future observation. The result of storing activity data is restricted up to 255 days running 24h a day which will give enough time to adjust the system and more suitable for the person using it. Of course the system will not run every hour of the day, so it would be be considered confident to assume that the storage would not run out for a year.

Chapter 5

Discussion

The Arduino Nano board provided an ample of possibilities since the Arduino has a large community and there are vast examples and inspirations available online. It is easy to implement code and by using a Stand-alone Arduino a good test base was conducted. By introducing two microcontrollers in the hand prosthesis setup, it opened up a world of new possibilities to connect and implement a diversity of new sensors. Only a fraction of them were implemented in this project.

5.1 Transfer

There were different approaches in how the actual transfer between the microcontrollers would occur. SPI which is the fastest transfer method had one significant issue. The number of pins required for SPI were too big. It requires three lines and each slave requires a specific SS (slave select). Since the connection between the two Arduino boards has a maximum of 4 connections in the coaxial cable the number of pins requested for the SPI are to many (4 wires for communication, the +5V and GND). The second method of transfers was to use the SoftwareSerial interface which creates virtual Tx and Rx ports. This method worked well but there was an issue regarding the timing. Since SoftwareSerial does not have a built-in feature for ACK and NACK when data is received such code needed to be implemented. There had to be a clock line controlling when data is sent and a data line for sending the data. This was

solved by using $ as a start bit and # as a stop bit. The issue about the timing still occurred although since there needed to be a clock, controlling if the microcontroller had processed the data. This is why the I^2C interface made its appearance. As described in theory the I^2C combines the best of SoftwareSerial and SPI. It almost has the speed of SPI, uses the same amount of pins as SoftwareSerial but also has ACK and NACK bits included in its feature. Another good feature with the I^2C interface is that it wakes up the Arduino from sleep while requesting to transfer. Unfortunately there was an issue while trying to claim the I^2C bus from several devices. For this reason the clock stamp is only read once, while the Arduino boots and this is the reason for having a slave-master constellation.

5.1.1 Time consuming tasks

The first library used for implementing the SD card was the basic SD card library which is a default library in Arduino. Instead, a new library called the SDFat library was introduced. The desirable transfer time would be under 10 ms in order to get a robust and overall fast system which was obtained eventually. The 10 ms transfer time was set as a limit so that the feedback would not be delayed to the user. It is of great significance to mention that the standard SD card library in Arduino had a very time consuming interface. The SDFat library took the 100 ms transfer time down to a whopping 6 ms. It is not clear how that library is implemented rather than that it solved the time consuming issue. Nevertheless some minor time saving solutions such as switching the old SD card to a new CLASS 10 card saved about 3 ms. Since then the Arduino code has been rewritten more than once in order to make it as robust and rapid as possible. This is shown in figure 4.24 where the total time for the setup is 10 ms with the new SD card and the new library but without the code changes to improve the rapidity. The smaller adjustments in the code made it drop from 10 ms to about 6 ms.

5.2 Errors

Even when not considering that the Ottobock hand prosthesis used in this project was only functional from time to time because the connection pin did not fit precisely, there was a reading error while reading and transferring the EMG values. The issue was that the microcontroller was reading the values to fast and thus sending an faulty value on the I^2C. The faulty value of 0 made the motor lag in the prosthetic hand. The solution was too have a "dummy read" in the masters code (see Arduino code appendix C.1.1). This means that the value was read twice to give the microcontroller enough time to process and store the correct value. Another error that occurred was the amount of storage available on the microprocessor. This is explained in section memory and this is the reason for changing the print() to printf(). Regardless, the amount of storage still available on the masters Arduino is close to its maximum.

5.3 Calibration

The threshold values (see section 4.3.2) are values set to fit the hardware used in this project. These values could be set in the calibration mode and stored in text files as the calibration for the sensors so they could be changed through the Bluetooth interface, in case different hardware is used.

Besides this the only calibration done in the code is a linear calibration. This could be misleading if the actual hardware connected for example has an exponential curve.

5.4 PWM

The different PWM ports on the Arduino was based on different timers, causing them to have different output frequencies. In reality this means that the PWM will have different pulse width. With some testing this was not noticeable for the user but nevertheless well worth mention-

ing. The internal timers could be reset by a divisor, but not so the frequencies align perfectly, besides a reset of the internal timer would affect functions such as the delay() function.

5.5 Power consumption

It would be a good idea to calculate how power consuming the setup in reality is and how it would drain the battery. Since this master thesis was centered around developing a sensory feedback system, it never came to the point where the setup was fully tested with the external power source. The aim with the power efficient idea was to make the circuit as low power consuming as possible. The most efficient power saving would be done by implementing the PCB with the MIC5055 voltage regulator and thus removing the internal Arduino regulator as well with its USB interface regulator.

5.6 Interrupt

The interrupt function is set to go off when a LOW pin is detected on the interrupt pin (D2). This was created with a NOR-gate and in the theory the authors state that only a low pin could wake up the microcontroller but actually there was a data sheet bug in the ATmel data sheet. Any type of interrupt (Rising edge/ Falling edge / Low level / Any logical change) could be used to wake up the microcontroller [58].

Chapter 6

Conclusion

This project was a further development on ideas that our supervisor Christian Antfolk and his colleagues at the department of Biomedical Engineering in Lund had developed during years of research within the subject. The conclusion is that it is conceivable to create a fully functional, robust, rapid and rendering sensory feedback system on a basic scale. The most important matter is that introducing microcontrollers in a smart manner opens up new possibilities for the prosthesis as a whole. Possibilities to log information, increase the number of sensors and the variation of sensors added to the prosthesis. Even tough the setup never came to the point where we manufactured the actual PCB, the schematics are made and hold potential of working rather well. This would decrease size, power consumption and give the user a more user friendly device. Furthermore the Arduino setup was never clinical tested with any test person besides ourselves. Nevertheless we felt that the sensory feedback actually gave good and sufficient feedback in the testing that was made. For even more resolution and increased quality different sensors, with more precision could be used. This project merely scrapes the surface on available sensors.

6.1 Further work

Even though this project resulted in a successful, functioning sensory feedback system, it needs to be evaluated more thoroughly and preferably on amputees. This requires an

investigation that follows the federal policy for the protection of human subjects. Even though one can try to figure out what a patient or user would prefer in a sensory feedback system, to conduct a truly accurate investigation, would be with authentic test subjects. The system also needs to be further developed and built on printed circuit boards in order to really decrease the power consumption and to optimize the size and shape of the circuits. Besides this, there are almost endless options to add to the setup. The Bluetooth could be put in off mode while calibration state is over in order to save even more power. All the breakout boards could also be PCB manufactured and set to off when they are not used. The code probably needs an overview and could be even smarter written. Many of these improvements would arise during the PCB testing and the with the patient testings but we will leave that for future developers.

Appendix A

Data sheets

A.1 Arduino Nano Schematic

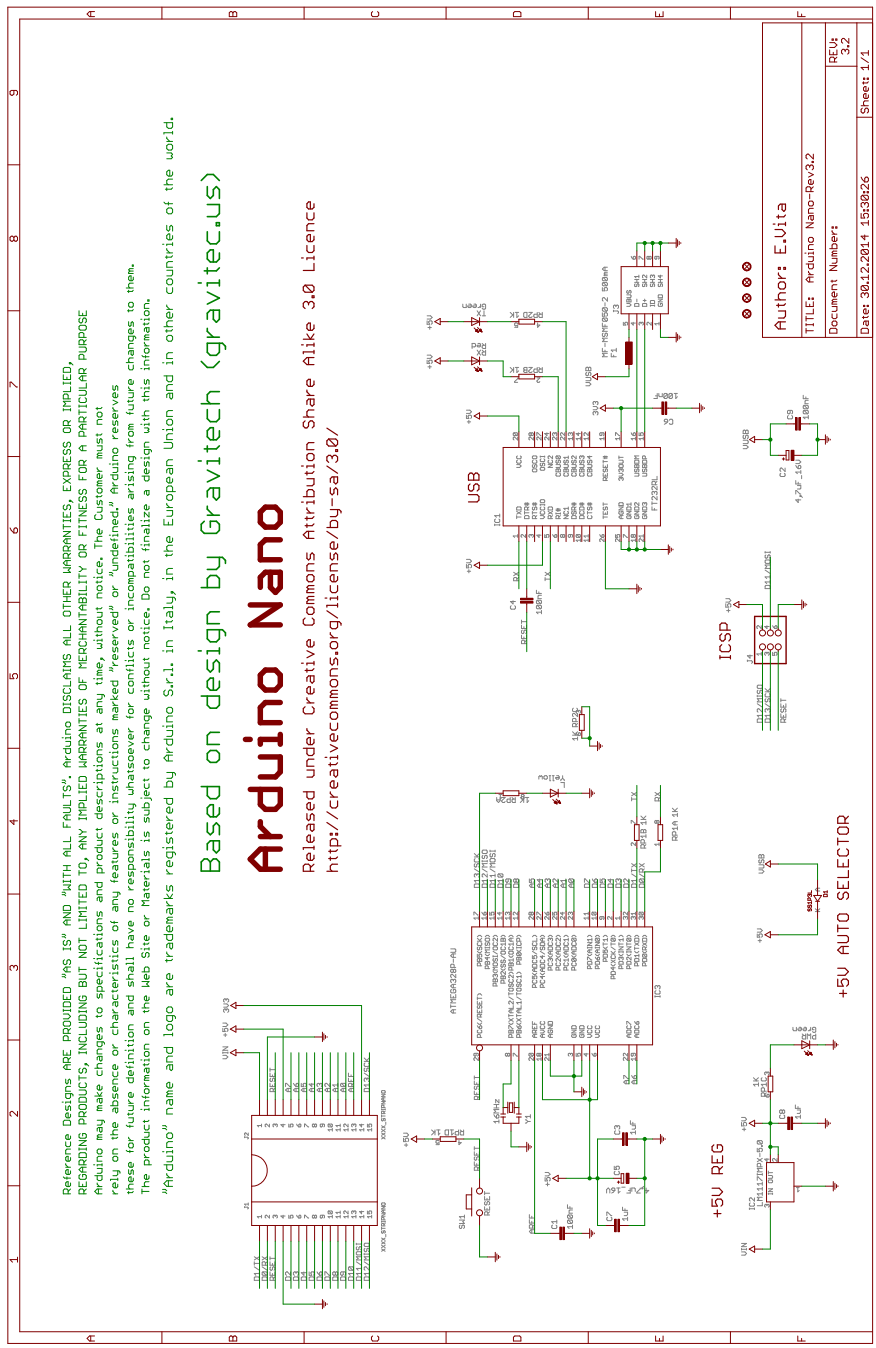


Figure A.1: Arduino Nano Schematic [59].

A.2 MC14001B data sheet

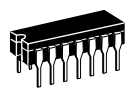
MOTOROLA
SEMICONDUCTOR TECHNICAL DATA

MC14001B
Quad 2-Input NOR Gate
MC14002B
Dual 4-Input NOR Gate
MC14011B
Quad 2-Input NAND Gate
MC14012B
Dual 4-Input NAND Gate
MC14023B
Triple 3-Input NAND Gate
MC14025B
Triple 3-Input NOR Gate
MC14068B
8-Input NAND Gate
MC14071B
Quad 2-Input OR Gate
MC14072B
Dual 4-Input OR Gate
MC14073B
Triple 3-Input AND Gate
MC14075B
Triple 3-Input OR Gate
MC14078B
8-Input NOR Gate
MC14081B
Quad 2-Input AND Gate
MC14082B
Dual 4-Input AND Gate

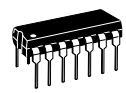
B-Suffix Series CMOS Gates

The B Series logic gates are constructed with P and N channel enhancement mode devices in a single monolithic structure (Complementary MOS). Their primary use is where low power dissipation and/or high noise immunity is desired.

- Supply Voltage Range = 3.0 Vdc to 18 Vdc
- All Outputs Buffered
- Capable of Driving Two Low–power TTL Loads or One Low–power Schottky TTL Load Over the Rated Temperature Range.
- Double Diode Protection on All Inputs Except: Triple Diode Protection on MC14011B and MC14081B
- Pin–for–Pin Replacements for Corresponding CD4000 Series B Suffix Devices (Exceptions: MC14068B and MC14078B)



L SUFFIX
CERAMIC
CASE 632



P SUFFIX
PLASTIC
CASE 646

D SUFFIX
SOIC
CASE 751A

ORDERING INFORMATION

MC14XXXBCP	Plastic
MC14XXXBCL	Ceramic
MC14XXXBD	SOIC

T_A = – 55° to 125°C for all packages.

MAXIMUM RATINGS* (Voltages Referenced to V_{SS})

Symbol	Parameter	Value	Unit
V_{DD}	DC Supply Voltage	– 0.5 to + 18.0	V
V_{in}, V_{out}	Input or Output Voltage (DC or Transient)	– 0.5 to V_{DD} + 0.5	V
I_{in}, I_{out}	Input or Output Current (DC or Transient), per Pin	± 10	mA
P_D	Power Dissipation, per Package†	500	mW
T_{stg}	Storage Temperature	– 65 to + 150	°C
T_L	Lead Temperature (8–Second Soldering)	260	°C

* Maximum Ratings are those values beyond which damage to the device may occur.
†Temperature Derating:
Plastic "P and D/DW" Packages: – 7.0 mW/°C From 65°C To 125°C
Ceramic "L" Packages: – 12 mW/°C From 100°C To 125°C

This device contains protection circuitry to guard against damage due to high static voltages or electric fields. However, precautions must be taken to avoid applications of any voltage higher than maximum rated voltages to this high-impedance circuit. For proper operation, V_{in} and V_{out} should be constrained to the range $V_{SS} \le (V_{in}$ or $V_{out}) \le V_{DD}$.
Unused inputs must always be tied to an appropriate logic voltage level (e.g., either V_{SS} or V_{DD}). Unused outputs must be left open.

REV 3
1/94

MOTOROLA

Figure A.2: MC14001B quad 2-input NOR gate[60].

A.3 MC14001B data sheet

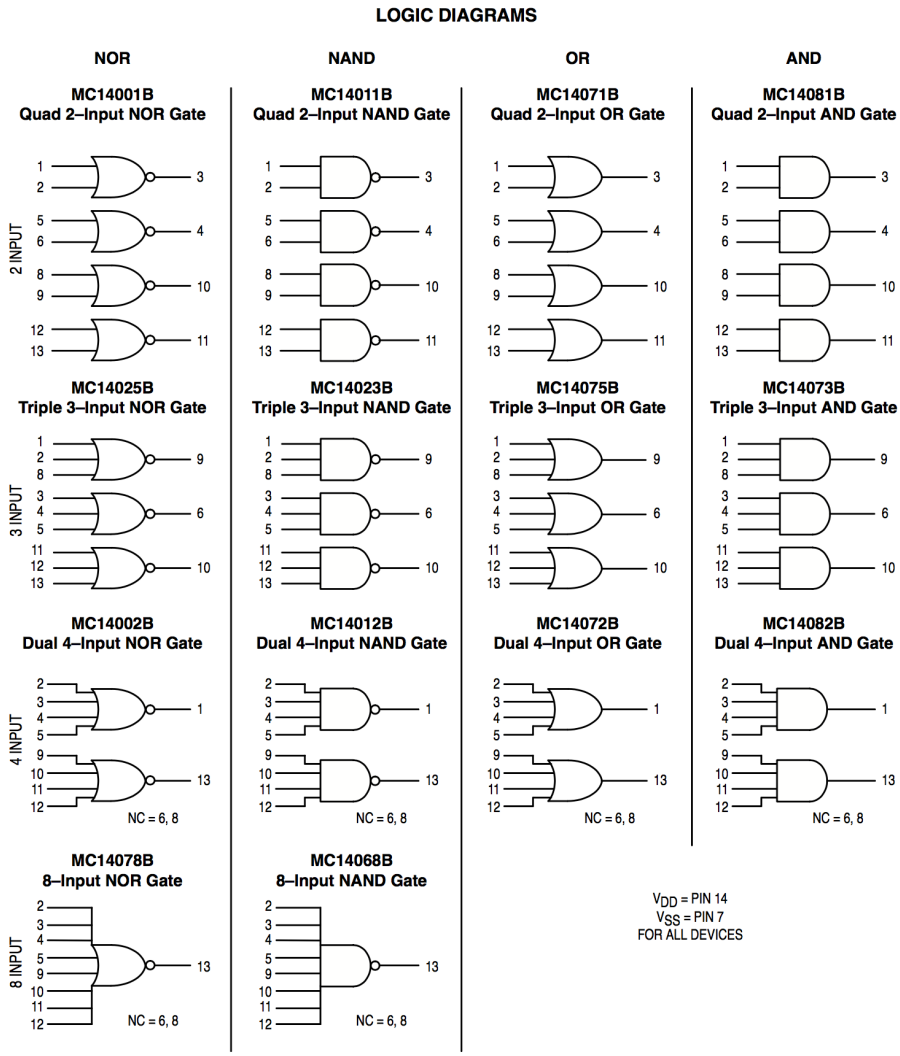


Figure A.3: MC14001B quad 2-input NOR gate[60].

A.4 TC4071BP data sheet

TOSHIBA TC4071BP/BF/BFN

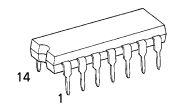
TOSHIBA CMOS DIGITAL INTEGRATED CIRCUIT SILICON MONOLITHIC

TC4071BP, TC4071BF, TC4071BFN

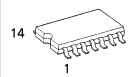
(Note) The JEDEC SOP (FN) is not available in Japan.

TC4071B QUAD 2 INPUT OR GATE

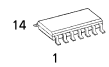
TC4071B is positive logic OR gates with two inputs respectively. As all the outputs of gates are equipped with the buffer circuits of inverters, the input / output propagation characteristic has been improved and the variation of propagation time caused by increase of load capacity is kept minimum.



P (DIP14-P-300-2.54)
Weight : 0.96g (Typ.)



F (SOP14-P-300-1.27)
Weight : 0.18g (Typ.)



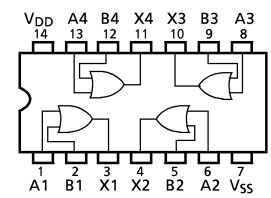
FN (SOL14-P-150-1.27)
Weight : 0.12g (Typ.)

MAXIMUM RATINGS

CHARACTERISTIC	SYMBOL	RATING	UNIT
DC Supply Voltage	V_{DD}	$V_{SS}-0.5\sim V_{SS}+20$	V
Input Voltage	V_{IN}	$V_{SS}-0.5\sim V_{DD}+0.5$	V
Output Voltage	V_{OUT}	$V_{SS}-0.5\sim V_{DD}+0.5$	V
DC Input Current	I_{IN}	± 10	mA
Power Dissipation	P_D	300 (DIP) / 180 (SOIC)	mW
Operating Temperature Range	T_{opr}	− 40~85	°C
Storage Temperature Range	T_{stg}	− 65~150	°C

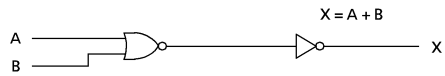
PIN ASSIGNMENT (TOP VIEW)

TC4071B



LOGIC DIAGRAM

1 / 4 TC4071B



980910EBA2

● TOSHIBA is continually working to improve the quality and the reliability of its products. Nevertheless, semiconductor devices in general can malfunction or fail due to their inherent electrical sensitivity and vulnerability to physical stress. It is the responsibility of the buyer, when utilizing TOSHIBA products, to observe standards of safety, and to avoid situations in which a malfunction or failure of a TOSHIBA product could cause loss of human life, bodily injury or damage to property. In developing your designs, please ensure that TOSHIBA products are used within specified operating ranges as set forth in the most recent products specifications. Also, please keep in mind the precautions and conditions set forth in the TOSHIBA Semiconductor Reliability Handbook.

1999-08-20 1/5

Figure A.4: TC4071BP quad 2-input or gate[61].

A.5 Hand prosthesis data sheet

1 2 3 4 5 6 7 8 9

646D44 647H326

8E38=6 System Electric Hand DMC plus

With quick-disconnect wrist

Suitable for all amputation levels, except wrist disarticulation.
Passive wrist rotation with ratchet lock (can be replaced by the 11S30 Friction Ring).
The DMC plus control features a DMC and DMC plus control mode. The desired control mode is selected with the integrated 13E185 Function Plug. In DMC plus control mode, after gripping once with maximum grip force, a higher signal is required to open the hand. This reduces the risk of opening the hand with undesired muscle signals. In this system, two independent measurement and control systems proportionally control gripping speed as well as gripping force. Gripping speed and gripping force are determined by the strength of the muscle signal. The System Electric Hand DMC plus can be operated with the 757B35=* MyoEnergy Integral, the 757B20/757B21 EnergyPack or the 757B15 X-ChangePack.

It features a central coaxial plug connection, automatic shut-off electronics and integrated on-off switch, low-friction bevel gear, positive back lock and system inner hand. An integrated slip clutch allows the hand to be opened in case of power supply or myoelectric control failure.

Article number	Side	Size	Inner hand
8E38=6-L7	left (L)	7	8X18=L7
8E38=6-L7 1/4	left (L)	7 1/4	8X18=L7 1/4
8E38=6-L7 3/4	left (L)	7 3/4	8X18=L7 3/4
8E38=6-L8 1/4	left (L)	8 1/4	8X18=L8 1/4
8E38=6-R7	right (R)	7	8X18=R7
8E38=6-R7 1/4	right (R)	7 1/4	8X18=R7 1/4
8E38=6-R7 3/4	right (R)	7 3/4	8X18=R7 3/4
8E38=6-R8 1/4	right (R)	8 1/4	8X18=R8 1/4

Technical Data

Reference number	8E38=6	8E38=6
Size	7	7 1/4, 7 3/4, 8 1/4
Operating voltage	6/7.2 V	6/7.2 V
Opening width	79 mm	100 mm
Proportional gripping force	0-90 N	0-90 N
Proportional speed	15-130 mm/sec	15-130 mm/sec
Weight with system inner hand	355 g	457 g
for	Women, Adolescents	Women, Adolescents, Men

- The electrodes must be adjusted with the 757M11 MyoBoy!
- For the suitable system prosthetic glove, see the pages 55-58

Particularly short and light: the new Electric Hand size 7.

It closes the gap between the Electric Hand 2000 hand system for children and the well-known System Electric Hands for adults. It is particularly suitable for the fitting of adolescents or women with small, dainty hands.

The new Electric Hand size 7 is offered with the well-known Digital Twin and DMC plus control systems.

Figure A.5: Ottobock hand prosthesis specifications[62].

A.6 13E200 EMG-Sensor data sheet

13E200 Electrode

Electrode cable connection with IDC termination, with **13E153 Electrode Accessories**

These MyoBock electrodes are particularly sensitive in the range of low muscle signals. The change in amplification now takes place logarithmically, which enables enhanced differentiation of the signal level, particularly in the range of high muscle signals.

Thanks to modern frequency shielding and filtering technologies, it is significantly less sensitive to low and high frequency interferences that are emitted, for example, by mobile phones or shopping centre security systems.

The electrode contacts are made from **pure titanium** and as they do not contain nickel, they are also suitable for people with allergies.

The electrode accessories 13E201 is included in the scope of delivery of the electrode.

The frequency filter's full protection effect will only be provided if the mains frequency and filter frequency are identical.

647H490

Article number	13E200=50	13E200=60
Weight	4.5 g	
HZ	50	60
Operating voltage	4.8 - 7.2 V	
Dimensions LxWxH	27x18x9.5 mm	
Frequency bandwidth	90 - 450 Hz	
Ambient temperature	-15 bis +60 °C	

- Use 633F11 Silicone Grease to seal the plug connections. Remove any excessive grease after connecting the electrode cable.
- For accessories for vacuum forming of inner sockets, see the pages 86, 262, 264

Figure A.6: 13E200 EMG-Sensor specifications [62].

A.7 Adafruit micro sd breackout board

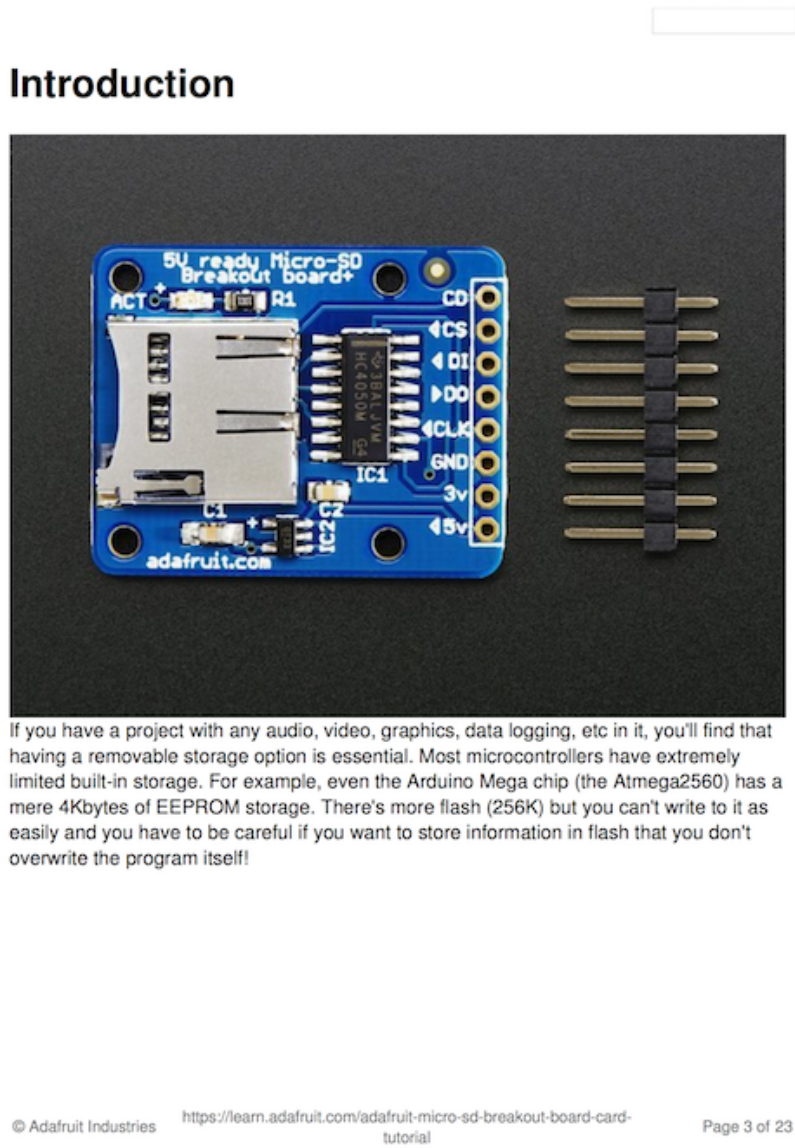


Figure A.7: Data logger [63]

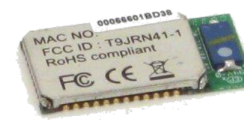
A.8 Bluetooth-RN-41 data sheet

ROVING networks
WIRELESS FOR LESS

RN-41

www.rovingnetworks.com DS-RN41-V3.1 6/27/2011

Class 1 Bluetooth® Module



Features

- Fully qualified Bluetooth 2.1/2.0/1.2/1.1 module
- Bluetooth v2.0+EDR support
- Postage stamp sized form factor, 13.4mm x 25.8 mm x2mm
- Low power *(30mA connected,, <10mA sniff mode)*
- UART (SPP or HCI) and USB (HCI only) data connection interfaces.
- Sustained SPP data rates - 240Kbps (slave), 300Kbps (master)
- HCI data rates - 1.5Mbps sustained, 3.0Mbps burst in HCI mode
- Embedded Bluetooth stack profiles included *(requires no host stack)*: GAP, SDP, RFCOMM and L2CAP protocols, with SPP and DUN profile support.
- Bluetooth SIG Qualified, End Product Listing
- Castellated SMT pads for easy and reliable PCB mounting
- Class 1 high power amplifier with on board ceramic RF chip antenna.
  - Certifications: FCC, ICS, CE
  - Environmentally friendly, RoHS compliant

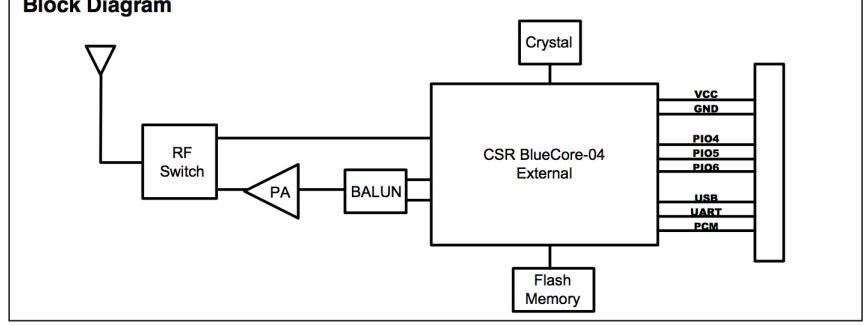
Applications

- Cable replacement
- Barcode scanners
- Measurement and monitoring systems
- Industrial sensors and controls
- Medical devices
- Asset tacking

Description

The RN41 is a small form factor, low power, highly economic Bluetooth radio for OEM's adding wireless capability to their products. The RN41 supports multiple interface protocols, is simple to design in and fully certified, making it a complete embedded Bluetooth solution. With its high performance on chip antenna and support for Bluetooth® Enhanced Data Rate (EDR), the RN41 delivers up to 3 Mbps data rate for distances to 100M.. The RN41 is the perfect product for engineers wanting to add wireless capability to their product but don't want to spend significant time and money developing Bluetooth specific hardware and software.

Block Diagram



809 University Avenue • Los Gatos, CA 95032 • Tel (408) 395-6539 • info@RovingNetworks.com

Figure A.8: Bluetooth-RN-41 [64]

A.9 Force sensor data sheet

INTERLINK ELECTRONICS
Sensor Technologies

FSR 400
P/N: 30-49649

Application Information

FSRs are two-wire devices with a resistance that depends on applied force.

For specific application needs please contact Interlink Electronics support team. An integration guide is also available.

For a simple force-to-voltage conversion, the FSR device is tied to a measuring resistor in a voltage divider configuration (see Figure 3). The output is described by the equation:

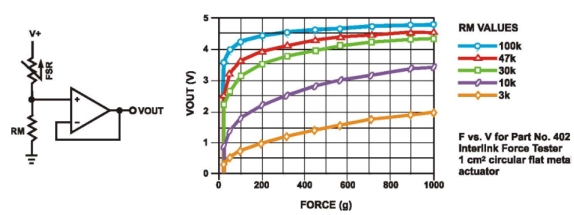
$$V_{OUT} = \frac{R_M V +}{(R_M + R_{FSR})}$$

In the shown configuration, the output voltage increases with increasing force. If R_{FSR} and R_M are swapped, the output swing will decrease with increasing force.

The measuring resistor, R_M, is chosen to maximize the desired force sensitivity range and to limit current. Depending on the impedance requirements of the measuring circuit, the voltage divider could be followed by an op-amp.

A family of force vs. V_{OUT} curves is shown on the graph below for a standard FSR in a voltage divider configuration with various R_M resistors. A (V+) of +5V was used for these examples.

Figure 3



Contact Us

United States
Corporate Offices
Interlink Electronics, Inc.
546 Flynn Road
Camarillo, CA 93012, USA
Phone: +1-805-484-8855
Fax: +1-805-484-9457
Web: www.interlinkelectronics.com
Sales and support:
fsr@interlinkelectronics.com

Japan
Japan Sales Office
Phone: +81-45-263-6500
Fax: +81-45-263-6501
Web: www.interlinkelec.co.jp

Korea
Korea Sales Office
Phone: +82 10 8776 1972

Figure A.9: Force sensor specifications[65].

A.10 RTC-DS3231 data sheet

DS3231 **Extremely Accurate I²C-Integrated RTC/TCXO/Crystal**

General Description

The DS3231 is a low-cost, extremely accurate I²C real-time clock (RTC) with an integrated temperature-compensated crystal oscillator (TCXO) and crystal. The device incorporates a battery input, and maintains accurate timekeeping when main power to the device is interrupted. The integration of the crystal resonator enhances the long-term accuracy of the device as well as reduces the piece-part count in a manufacturing line. The DS3231 is available in commercial and industrial temperature ranges, and is offered in a 16-pin, 300-mil SO package.

The RTC maintains seconds, minutes, hours, day, date, month, and year information. The date at the end of the month is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with an AM/PM indicator. Two programmable time-of-day alarms and a programmable square-wave output are provided. Address and data are transferred serially through an I²C bidirectional bus.

A precision temperature-compensated voltage reference and comparator circuit monitors the status of V_{CC} to detect power failures, to provide a reset output, and to automatically switch to the backup supply when necessary. Additionally, the $\overline{RST}$ pin is monitored as a pushbutton input for generating a µP reset.

Benefits and Features

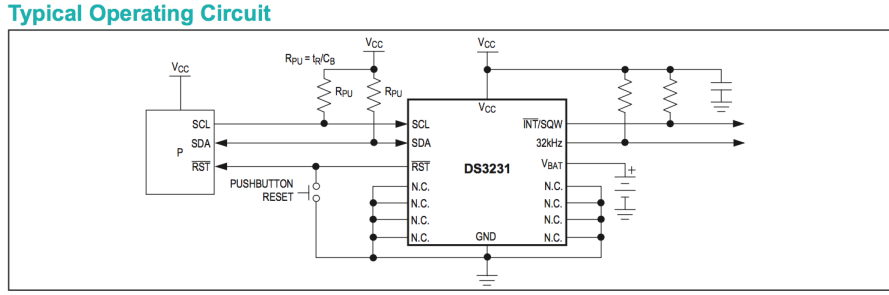
- Highly Accurate RTC Completely Manages All Timekeeping Functions
  - Real-Time Clock Counts Seconds, Minutes, Hours, Date of the Month, Month, Day of the Week, and Year, with Leap-Year Compensation Valid Up to 2100
  - Accuracy ±2ppm from 0°C to +40°C
  - Accuracy ±3.5ppm from -40°C to +85°C
  - Digital Temp Sensor Output: ±3°C Accuracy
  - Register for Aging Trim
  - $\overline{RST}$ Output/Pushbutton Reset Debounce Input
  - Two Time-of-Day Alarms
  - Programmable Square-Wave Output Signal
- Simple Serial Interface Connects to Most Microcontrollers
  - Fast (400kHz) I²C Interface
- Battery-Backup Input for Continuous Timekeeping
  - Low Power Operation Extends Battery-Backup Run Time
  - 3.3V Operation
- Operating Temperature Ranges: Commercial (0°C to +70°C) and Industrial (-40°C to +85°C)
- Underwriters Laboratories® (UL) Recognized

Applications

- Servers
- Telematics
- Utility Power Meters
- GPS

Ordering Information and Pin Configuration appear at end of data sheet.

Typical Operating Circuit



Underwriters Laboratories is a registered certification mark of Underwriters Laboratories Inc.

maxim integrated™

19-5170; Rev 10; 3/15

Figure A.10: RTC-DS3231 specifications[66].

A.11 mic5205 data sheet

$V_{IN(max)}$ = 6.23V

Therefore, a 3.3V application at 150mA of output current can accept a maximum input voltage of 6.2V in a SOT-23-5 package. For a full discussion of heat sinking and thermal effects on voltage regulators, refer to the Regulator Thermals section of Micrel's *Designing with Low-Dropout Voltage Regulators* handbook.

Fixed Regulator Applications

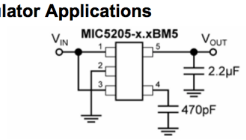


Figure 1. Ultra-Low-Noise Fixed Voltage Application

Figure 1 includes a 470pF capacitor for low-noise operation and shows EN (pin 3) connected to IN (pin 1) for an application where enable/shutdown is not required. C_{OUT} = 2.2µF minimum.

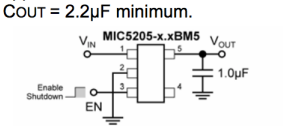


Figure 2. Low-Noise Fixed Voltage Application

Figure 2 is an example of a low-noise configuration where C_{BYP} is not required. C_{OUT} = 1µF minimum.

Adjustable Regulator Applications

The MIC5205BM5 can be adjusted to a specific output voltage by using two external resistors (Figure 3). The resistors set the output voltage based on the following equation:

$$V_{OUT} = 1.242V \times \left(\frac{R2}{R1} + 1\right)$$

This equation is correct due to the configuration of the bandgap reference. The bandgap voltage is relative to the output, as seen in the block diagram. Traditional regulators normally have the reference voltage relative to ground and have a different V_{OUT} equation.

Resistor values are not critical because ADJ (adjust) has a high input impedance, but for best results use resistors of 470kΩ or less. A capacitor from ADJ to ground provides greatly improved noise performance.

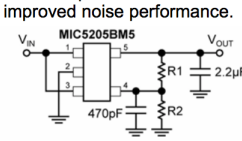


Figure 3. Ultra-Low-Noise

Adjustable Voltage Application

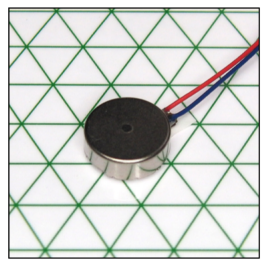
Figure 3 includes the optional 470pF noise bypass capacitor from ADJ to GND to reduce output noise.

Dual-Supply Operation

When used in dual supply systems where the regulator load is returned to a negative supply, the output voltage must be diode clamped to ground.

Figure A.11: Mic5205 specifications [67].

A.12 310-101 data sheet



10mm Vibration Motor - 3mm Type
Shown on 6mm Isometric Grid

PRECISION MICRODRIVES™

Product Data Sheet
Pico Vibe™
10mm Vibration Motor - 3mm Type

Model: 310-101

Ordering Information

The model number 310-101 fully defines the model, variant and additional features of the product. Please quote this number when ordering.
For stocked types, testing and evaluation samples can be ordered directly through our online store.

Datasheet Versions

It is our intention to provide our customers with the best information available to ensure the successful integration between our products and your application. Therefore, our publications will be updated and enhanced as improvements to the data and product updates are introduced.
To obtain the most up-to-date version of this datasheet, please visit our website at:
www.precisionmicrodrives.com
The version number of this datasheet can be found on the bottom left hand corner of any page of the datasheet and is referenced with an ascending R-number (e.g. R002 is newer than R001). Please contact us if you require a copy of the engineering change notice between revisions.

If you have any questions, suggestions or comments regarding this publication or need technical assistance, please contact us via email at:
enquiries@precisionmicrodrives.com or call us on +44 (0) 1932 252 482

Key Features

Body Diameter:	10 mm [+/- 0.1]
Body Length:	3.4 mm [+/- 0.1]
Rated Operating Voltage:	3 V
Rated Vibration Speed:	12,200 rpm [+/- 2,500]
Typical Rated Operating Current:	63 mA
Typical Normalised Amplitude:	1.38 G

Typical Vibration Motor Performance Characteristics

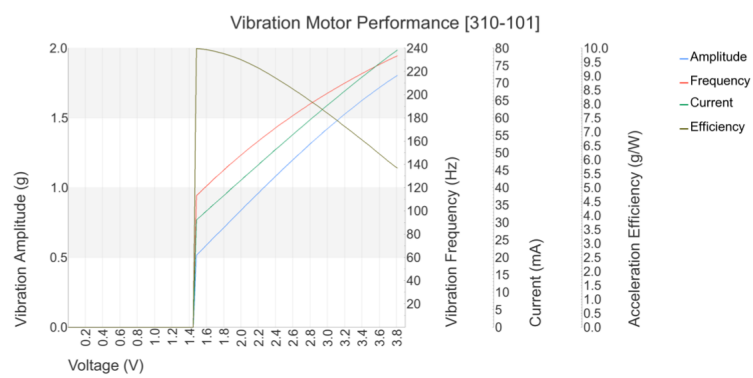


Figure A.12: 310-101 specifications[46].

Appendix B

LabVIEW-Arduino Code

B.1 LabVIEW code

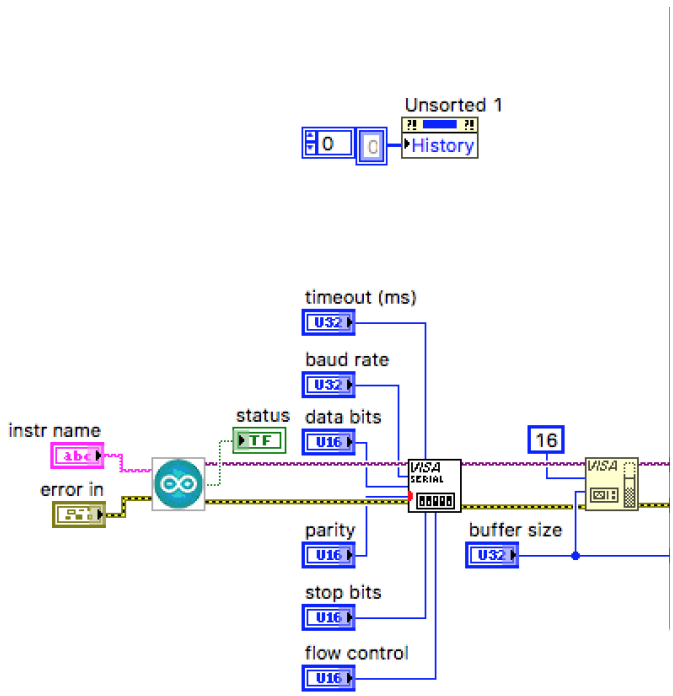


Figure B.1: Connection from Arduino to LabView [50].

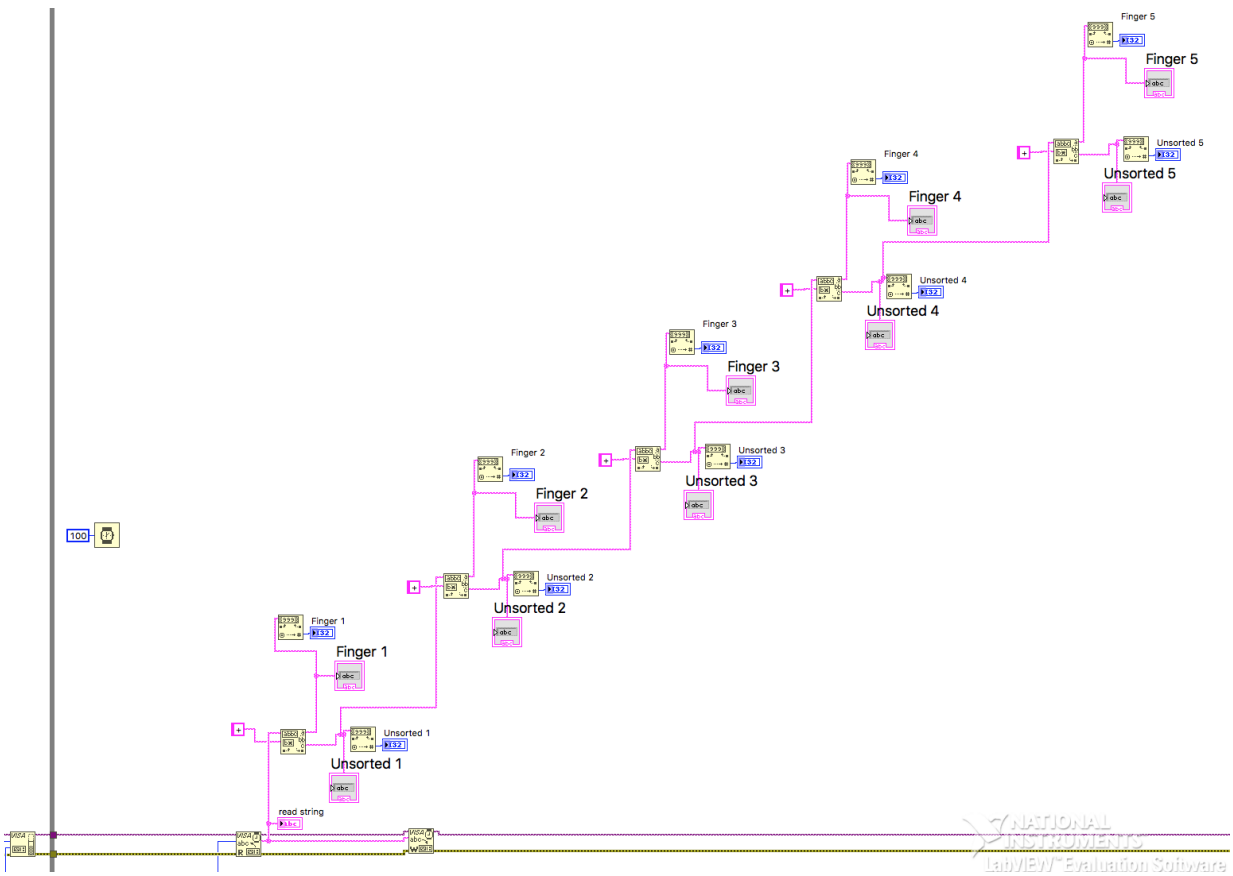


Figure B.2: Data sorted out from the Arduino for each finger.

B.2 Arduino

```
//*************Arduino Labview***************

//----------Gobal Varibles----------------
int sensorValue0,sensorValue1,sensorValue2;
int sensorValue3,sensorValue4;
int outputValue0,outputValue1,outputValue2;
int outputValue3,outputValue4;
int a0,a1,a2,a3,a4;
//----------------------------------------
void setup() {

  //Baude Rate set.
  Serial.begin(9600);
}
void loop() {
//---Read, write and mapping sensor values---

  //Reading sensor.
  sensorValue0 = analogRead(0);

  //Mapping values suitble for PWM.
  outputValue0=map(sensorValue0,0,1023,0,255);

  //Prints out the value on Serial for labview
  //to catch up.
  Serial.print(outputValue0);

  //Create an easier envorement for labview
  //to read characters, otherwies error
  //messege occures.
  delay(10);

  //Sort out the different values with a character
  Serial.print("+");

  sensorValue1 = analogRead(1);
  outputValue1 = map(sensorValue1, 0, 1023, 0, 255);
  Serial.print(outputValue1);

  delay(10);
  Serial.print("+");

  sensorValue2 = analogRead(2);
  outputValue2 = map(sensorValue2, 0, 1023, 0, 255);
  Serial.print(outputValue2);

  delay(10);
  Serial.print("+");

  sensorValue3 = analogRead(3);
  outputValue3 = map(sensorValue3, 0, 1023, 0, 255);
  Serial.print(outputValue3);

  delay(10);
  Serial.print("+");

  //Reading sensor 4.
  sensorValue4 = analogRead(4);

  //Mapping values suitble for pwm.
  outputValue4=map(sensorValue4,0,1023,0,255);
  Serial.println(outputValue4);

 //------Serial Values from Labview-----------
 //Values from Labview written in to
 //the Arduino.
 while( Serial.available() ){
    a0 = Serial.parseInt();
    a1 = Serial.parseInt();
    a2 = Serial.parseInt();
    a3 = Serial.parseInt();
    a4 = Serial.parseInt();
}
//-------------------PWM-------------------------
  //PWM to a feeback module.
  analogWrite(5,a0);
  analogWrite(6,a1);
  analogWrite(9,a2);
  analogWrite(10,a3);
  analogWrite(11,a4);
```

```
  delay(150);
}
```

Appendix C

Arduino code

C.1 Master Arduino Code

C.1.1 Main code

```
//********* I2CBus MASTER47 (#0) *************
//Master code created by Alexander Jordansson and Nima Haji sheykhi
 // 18/11/2016

//**********Including libraries**************
#include <Wire.h>
// Wire library uses the I2C
#include <avr/sleep.h>
// SleepMode library
#include <avr/interrupt.h>
// Interrupt library
#include <avr/power.h>
#include <avr/io.h>
#include <SoftwareSerial.h>
// The Bluetooth input and output
#include <SPI.h>
// Sd card uses SPI
//#include <SD.h>
// Sd card library (OLD ONE)
#include "RTClib.h"
// RTC library
RTC_DS3231 rtc;
// RTC is the DS3231
#include <string.h>
// String library for storing values
#include "SdFat.h"
// SDFat library
SdFat SD;
// Constant to write is as the SD library.

//***********Global declaration**************
int pot0, pot1;
// The Emg values
byte dataPot[5];
// Array for the received data from device #8 (SLAVE)
byte dataEmg[2];
// Array to store and send the EMG values
SoftwareSerial bluetooth (7,8);
// RX, TX (Bluetooth input)
byte state;
// While loop stuff
double k = 1;
// The amplification of the force sensors
double c = 0;
// C- value (y-axis) for force sensors
double ke = 1;
// The amplification of the EMG sensors
double ce = 0;
// Ce- value (y-axis) for EMG sensors
```

```
double v = 1.25;
// Voltage output to the prosthetic hand
String time = "";
// The string to store the time
String readString;
// String to store the incoming values from Bluetooth
String sk, sc, ske, sce, sv;
// sk(StringK-value), sc(StringC-value), ske(StringKe-value),
      sce(StringCe-value),sv(StringVoltage-value)
String sy, sm, sd, sh, smm, ss;
// sy(StringYear), sm(StringMonth), sd(StringDay), sh(StringHour), smm(StringMinute),
      ss(StringSecond)
short ind1, ind2, ind3, ind4, ind5, ind6;
// for locating values between commas.
boolean loop1, loop2;
// for the whiles in the calibration loops
boolean flag = false;
// for the whiles in the calibration loops
boolean t = false;
// write time to SD-card.
int y,m,d,h,mi,se;
// time stamp values b is bluetooth-loop.
int g = 0;
// Bluetooth loop setting
byte r = 5;
// Calibration setting
byte sleep = 0;
// Sleep counter

//******************Setup******************
void setup() {
  Wire.begin(0);
  // Set i2c bus address 0 for Master
  Wire.setClock(400000L);
  // Set the baudRate to 400 000 (400KHz)
  Serial.begin(115200);
  // start the serial communication
  Wire.onReceive(receiveEvent);
  // Start the wire to register events on receive
  Wire.onRequest(requestEvent);
  // Start the wire to register events on request
  bluetooth.begin(115200);
  // Start the Bluetooth
  SD.begin(4, SPI_FULL_SPEED);
  // Set the SPI speed to FULL speed.
  rtc.begin();
  // Begin the Real Time Clock
  //clock();
  readTxt();
  // Read text files that contain calibration values
  calibration();
  // Calibrate accordingly to the text files
  clock();
  // Start the Clock.
  pinMode(2, INPUT_PULLUP);
  // Interrupt pin set to register interrupts
  attachInterrupt(0,wakeUpNow, LOW);
  // Attach a low interrupt

}
void wakeUpNow(){
  //DO NOTHING AS ISR
}

//*****************Loop*********************
void loop() {
  //digitalWrite(3,LOW);
  // Toggling for oscilloscope measurements
  //delay(100);
  //digitalWrite(3, HIGH);

  /*The output. The master calibrates the FSR:s with the saved calibration-values
  on each value in the array before it is send to the digital
  ports 3,5,6,9,10 which are PWM ports to the vibrators.*/

  dataPot[0] = dataPot[0]*k + c;
  //adjust the force sensor values with k and c calibration.
  dataPot[1] = dataPot[1]*k + c;
  dataPot[2] = dataPot[2]*k + c;
  dataPot[3] = dataPot[3]*k + c;
  dataPot[4] = dataPot[4]*k + c;
  SDcardadd();
  // Add the values to the SD-card
  analogWrite(3,dataPot[0]);
  //Send to the analog PWM outputs.
```

```
  analogWrite(5,dataPot[1]);
  analogWrite(6,dataPot[2]);
  analogWrite(9,dataPot[3]);
  analogWrite(10,dataPot[4]);

//************* Reading EMG values ***********
/* The EMG-values are read on the analog ports 0 and 1. This value
  is divided by 4 so it is send as byte and added in an array and add the calibration values. */
  analogRead(A0);
  // Dummyread to give the Arduino time to read more accurate
  dataEmg[0] = analogRead(A0)/4;
  dataEmg[0] = dataEmg[0] * ke + ce;
  dataEmg[1] = analogRead(A1)/4;
  dataEmg[1] = dataEmg[1] * ke + ce;
  dataEmg[0] = map(dataEmg[0], 0, 255, 0, 65.5*v);
  // Map the value so it delivers the correct voltage output
  dataEmg[1] = map(dataEmg[1], 0, 255, 0, 65.5*v);

   /*The I2C interface. If the EMG values are bigger than the threshold value(30) the
         transmission
   is started between slave and master (0). 2 bytes of data is sent (the dataEmg array).
   The master request five bytes from slave device #8 and write the incoming data in an array.
   So the potential FSR values are not forgotten.
   If the values are below the threshold value the transmission is still done, so the correct
  values are transmitted 1 time before the sleep constant is high enough to enter sleepMode. */

  if(dataEmg[0]>=30 || dataEmg[1]>=30){
    digitalWrite(2,LOW);
    // force the interrupt pin 2 to be low so attach interrupt doesn't go off.
    Wire.beginTransmission(8);
    Wire.write(dataEmg, 2);
    Wire.endTransmission(8);
    sleep = 0;
    // Sleep counter

    Wire.requestFrom(8, 5);
    while (Wire.available()) {
      for(byte i = 0; i < 5; i++){
        dataPot [i] = Wire.read();
        // add each wire.read to the array
      }
    }
  }

  else{
    digitalWrite(2,LOW);
//    Wire.beginTransmission(8);
//    Wire.write(dataEmg, 2);
//    Wire.endTransmission(8);
    sleep++;
    Serial.println(sleep);
    if(sleep > 10){
      Wire.beginTransmission(8);
      Wire.write(dataEmg, 2);
      Wire.endTransmission(8);
      analogWrite(3,0);
      //Send to the analog PWM outputs as zero so no floating values are left on the output.
      analogWrite(5,0);
      analogWrite(6,0);
      analogWrite(9,0);
      analogWrite(10,0);
      Serial.println("I sleep");
      delay(2);
      sleepNow();
    }
  }

  //digitalWrite(3, LOW);
  // Toggling for oscilloscope measurements
  //delay(100);
}
```

C.1.2 SD-Card

```
void SDcardadd(){

    //*************SD card stuff*************
  /*Make an empty string (dataString) and add all the different sensor values
  separated with ";". These values are stored in a file called datalog.txt.
```

```
 Type the string on the serial window. Add the time stamp ONCE int the beginning. */

   String dataString = "";
   if(t==false){
   dataString += String(time);
   t = true;
   }
   dataString += String(dataEmg[0]);
   dataString += ";";
   dataString += String(dataEmg[1]);
   dataString += ";";
   dataString += String(dataPot[0]);
   dataString += ";";
   dataString += String(dataPot[1]);
   dataString += ";";
   dataString += String(dataPot[2]);
   dataString += ";";
   dataString += String(dataPot[3]);
   dataString += ";";
   dataString += String(dataPot[4]);
   dataString += ";";
   File dataFile = SD.open("datalog.txt", FILE_WRITE); //write the values to the Sd-card.
   if (dataFile) {
     dataFile.println(dataString);
     dataFile.close();
   }
   Serial.println(dataString);

}
```

C.1.3 Bluetooth

```
 //********** Bluetooth interface************
 /*Listen to the Bluetooth port (softwareserial 7 and 8) and if
 anything (0,1,2) is typed enter different modes. This is just used in calibration purpose.
 State 0 is the time setting, state 1 is the EMG calibration and the
 force sensor calibration, 2 is a control setting which checks: RTC, timesetting,
 datalog.txt, cal.txt, calE.txt and Sd-card. In state 1 new k-values/ke-values and c-values
       /ce-values
 are implemented and written to the SD-card. The potential old values are
 erased. Calibration as kx+c, Enter values as: k, c, ke, ce (0-9999999) voltage out to hand[V]
       (0-3V).
 For every typed character a print is made to see the progress of
 the incoming characters. When * is typed the user is asked is everything is okey and is asked
 to type y or Y for yes and n or N for N. If no, the user is sent back in the loop to type new
       values
 if yes the program continous.
 Please observe that the user is asked to type ONE character at a time to give the bluetooth
 enough time to process the incomming character i.e to type 1,2,3,4, 1.25* the user needs to
       type:
 1 [press enter] , [press enter] 2 [press enter] , [press enter] 3 [press enter] , [press
       enter]
 4 [press enter] * [press enter] and do not type to fast.
 */

void bluetoothCal(){
   //bluetooth.println("Type 0 to calibrate timesettings and w to calibrate EMG and force
         sensors");

 bluetooth.println(F("Press 0 TS, 1 SC or 2 C, q quit "));
   while(g == 0){
     if(bluetooth.available()>0){
     state = bluetooth.read();
     }

 /* The TIME-calibration settings*/
 if(state == '0' ){
   if(flag == false){
     bluetooth.println(F("Set the time YYYY,MM,DD,HH,MM,SS"));
     loop1 = false;
       while(loop1 == false){
         if (bluetooth.available()) {
           char d = bluetooth.read(); //Reads one byte from bluetooth buffer and stores in d
                 until * is recieved else add in in the readString string.
           if (d == '*') {
             ind1 = readString.indexOf(',');
             // Finds the location of the first ,
             sy = readString.substring(0, ind1);
             // Captures first data String (sy = StringYear)
```

```
            y = sy.toFloat();
            // Convert String value to double and store.
            ind2 = readString.indexOf(',', ind1+1 );
            // Finds the location of the second ,
            sm = readString.substring(ind1+1, ind2+1);
            // Captures second data String (sm = StringMonth)
            m = sm.toFloat();
            // Convert String value to double and store.
            ind3 = readString.indexOf(',', ind2+1 );
            // Finds the location of the third ,
            sd = readString.substring(ind2+1, ind3+1);
            // Captures third data String (sd = StringDay)
            d = sd.toFloat();
            // Convert String value to double and store.
            ind4 = readString.indexOf(',', ind3+1 );
            // Finds the location of the forth ,
            sh = readString.substring(ind3+1);
            // Captures forth data String (sh = StringHour)
            h = sh.toFloat();
            // Convert String value to double and store
            ind5 = readString.indexOf(',', ind4+1 );
            // Finds the location of the fifth ,
            smm = readString.substring(ind4+1);
            // Captures fifth data String (smm = StringmMinute)
            mi = smm.toFloat();
            // Convert String value to double and store
            ind6 = readString.indexOf(',', ind5+1 );
            // Finds remaining part of data after last ,
            ss = readString.substring(ind5+1);
            // Captures sixth data String (ss = StringSecond)
            se = ss.toFloat();
            // Convert String value to double and store
            bluetooth.println(F("YYYY,MM,DD,HH,MM,SS Are these ok? Y/N"));
            bluetooth.println(readString);
            // Write the values to the Bluetooth for user to see.
            readString="";
            // Clears variable readstring for new input
            loop2 = false;
            while(loop2 == false){
              if(bluetooth.available()>0){
                char key = bluetooth.read();
                // Reads one character from Bluetooth (Y/y or N/n)
                if(key == 'Y' || key == 'y'){
                  loop2 = true;
                  loop1 = true;
                  bluetooth.println(F("Date is set!"));
                  time = "";
                  rtc.adjust(DateTime(y, m, d, h, mi, se));
                  // Adjust the time in the #include "RTClib.h"
                  bluetooth.println(F("Press 0 for timesettings, 1 for sensor calibration or 2
                        for control, q for quit "));
                  state = 0;
                  // Leave the setting.

                }
                if(key == 'N' || key == 'n'){
                  loop2 = true;
                  // Leave the Yes or No loop
                  loop1 = false;
                  // Enter the the loop to type new values
                  bluetooth.print(F("Enter new time: YYYY,MM,DD,HH,MM,SS"));
                }
              }
            }

          }else {
            readString += d;
            // Add all the input characters in the readString until character * is typed.
            bluetooth.println(readString);
            // Print every input value so the user can see what is stored.
          }

        }
      }
    }
  }
  /* The Sensor settings*/
  if(state == '1'){
    if(flag == false){
      bluetooth.println(F("Calibrate kx+c, Enter values as: k, c, ke, ce, v"));
      bluetooth.println(F("1 char at a time"));
      loop1 = false;
        while(loop1 == false){
          if (bluetooth.available()) {
            char d = bluetooth.read();
```

```
if (d == '*') {
  ind1 = readString.indexOf(',');
  sk = readString.substring(0, ind1);
  k = sk.toFloat();
  ind2 = readString.indexOf(',', ind1+1 );
  sc = readString.substring(ind1+1, ind2+1);
  c = sc.toFloat();
  ind3 = readString.indexOf(',', ind2+1 );
  ske = readString.substring(ind2+1, ind3+1);
  ke = ske.toFloat();
  ind4 = readString.indexOf(',', ind3+1 );
  sce = readString.substring(ind3+1, ind4+1);
  ce = sce.toFloat();
  ind5 = readString.indexOf(',', ind4+1 );
  // Finds the location of the fifth ,
  sv = readString.substring(ind4+1);
  // Captures fifth data String (smm = StringmMinute)
  v = sv.toFloat();
  bluetooth.println(F("Try the sensor, press enter and blankspace for new values, q
        for quit"));
  int g = 0;
  while(g == 0){
    if (bluetooth.available()>0) {
      //bluetooth.available();
      char d = bluetooth.read();
      dataEmg[0] = analogRead(A0)/4;
      dataEmg[0] = dataEmg[0] * ke + ce;
      dataEmg[1] = analogRead(A1)/4;
      dataEmg[1] = dataEmg[1] * ke + ce;
      Wire.requestFrom(8, 5);
      while (Wire.available()) {
      // Check if data is available on the I2C wire.
        for(int i = 0; i < 5; i++){
          dataPot [i] = Wire.read();
          // Add each wire.read to the array
         }
      }
      dataPot[0] = dataPot[0] * k + c;
      dataPot[1] = dataPot[1] * k + c;
      dataPot[2] = dataPot[2] * k + c;
      dataPot[3] = dataPot[3] * k + c;
      dataPot[4] = dataPot[4] * k + c;
      bluetooth.println(F("FS 0 Expected value (0-255)"));
      bluetooth.println(dataPot[0]);
      bluetooth.println(F("FS 1 Expected value (0-255)"));
      bluetooth.println(dataPot[1]);
      bluetooth.println(F("FS 2 Expected value (0-255)"));
      bluetooth.println(dataPot[2]);
      bluetooth.println(F("FS 3 Expected value (0-255)"));
      bluetooth.println(dataPot[3]);
      bluetooth.println(F("FS 4 Expected value (0-255)"));
      bluetooth.println(dataPot[4]);
      bluetooth.println(F(""));
      bluetooth.println(F("EMG 0 Expected value (0-255)"));
      bluetooth.println(dataEmg[0]);
      bluetooth.println(F("EMG 1 Expected value (0-255)"));
      bluetooth.println(dataEmg[1]);
      bluetooth.println(F(""));
      bluetooth.println(F("Expected voltage out"));
      bluetooth.println(v);
      bluetooth.println(F(""));
      if(d == 'q'){

        bluetooth.println(F("k, c, ke, ce, v Are these ok? Y/N"));
        bluetooth.println(readString);
        g = 1;
      }
    }
  }
  readString="";
  loop2 = false;
  while(loop2 == false){

    if(bluetooth.available()>0){
      char key = bluetooth.read();
      if(key == 'Y' || key == 'y'){
        loop2 = true;
        loop1 = true;
        bluetooth.println(F("K, c, ke, ce, v values are set!"));

        SD.remove("cal.txt");
        //remove the txt file on sd card.
        File dataFile2 = SD.open("cal.txt", FILE_WRITE);
        // Create new txt file on sd card.
        if (dataFile2) {
```

```
                //try to open it
                  dataFile2.println(k);
                  //store k value
                  dataFile2.println(c);
                  //store c value
                  dataFile2.close();
                  //close the file
                }else {
                  bluetooth.println(F("error opening cal.txt"));
                  // Error message if not able to open txt file.
                 }
                SD.remove("calE.txt");
                dataFile2 = SD.open("calE.txt", FILE_WRITE);
                if (dataFile2) {
                  dataFile2.println(ke);
                  dataFile2.println(ce);
                  dataFile2.close();
                 }
                else {
                  bluetooth.println(F("error opening calE.txt"));
                 }
                SD.remove("v.txt");
                dataFile2 = SD.open("v.txt", FILE_WRITE);
                if (dataFile2) {
                  dataFile2.println(v);
                  dataFile2.close();
                 }
                else {
                  bluetooth.println(F("error opening v.txt"));
                 }
                bluetooth.println(F("Press 0 for timesettings, 1 for sensor calibration or 2
                      for control, q for quit "));
                state = 0;

               }
               if(key == 'N' || key == 'n'){
                 loop2 = true;
                 loop1 = false;
                 SD.remove("cal.txt");
                 SD.remove("calE.txt");
                 bluetooth.println(F("Enter new values as k, c, ke, ce, v"));
               }
             }
           }

           }else {
             readString += d;
             bluetooth.println(readString);
           }
         }
      }
    }
  }
/* check settings*/
if(state == '2'){
  if(flag == false){
    bluetooth.println(F("Controlling RTC, timesetting, datalog.txt, cal.txt, calE.txt and
          Sd-card"));
    bluetooth.println(F("Controlling RTC card..."));
    if (!rtc.begin()) {
    //if rtc.begin does not run print couldn find RTC. Else ok.
      bluetooth.println(F("Couldn't find RTC"));
    }else{
      bluetooth.println(F("RTC ok, time is set to: "));
      clock();
      bluetooth.println(time);
      //Print the time (the time String)
      time = "";
    }
    File dataFile = SD.open("datalog.txt");
    //try to open txt files and just close them
    if (dataFile) {
      dataFile.close();
      bluetooth.println(F("datlog.txt check ok"));
    }
    else {
      bluetooth.println(F("datlog.txt not available"));
    }
    dataFile = SD.open("cal.txt");
    //try to open the calibration txt:s
    if (dataFile) {
       bluetooth.println(F("cal.txt, check ok. cal.txt contains: "));
       bluetooth.println(F("k: "));
       bluetooth.println(k);
       // print the k value
```

```
        bluetooth.println(F("c: "));
        bluetooth.println(c);
        // print the c value
       dataFile.close();
      }else {
       bluetooth.println(F("cal.txt not found"));
      }
      dataFile = SD.open("calE.txt");
      if (dataFile) {
        bluetooth.println(F("calE.txt, check ok. calE.txt contains: "));
        bluetooth.println(F("ke: "));
        bluetooth.println(ke);
        // print the k value
        bluetooth.println(F("ce: "));
        bluetooth.println(ce);
        // print the k value
       dataFile.close();
      }else {
       bluetooth.println(F("calE.txt not found"));
      }
      dataFile = SD.open("v.txt");
      if (dataFile) {
        bluetooth.println(F("v.txt, check ok. v.txt contains: "));
        bluetooth.println(F("v: "));
        bluetooth.println(v);
        // print the v value
       dataFile.close();
      }else {
       bluetooth.println(F("v.txt not found"));
      }
      bluetooth.println(F("Initializing SD card..."));
      if(!SD.begin(4, SPI_FULL_SPEED)){
       bluetooth.println(F("Card setup failed"));
       state = 0;
       }else{
       bluetooth.println(F("Card initialized"));
       bluetooth.println(F("Control finished"));
       bluetooth.println(F("Press 0 for timesettings, 1 for sensor calibration or 2 for
             control, q for quit "));
       state = 0;

      }
     }
   }
   if(state == 'q'){
     bluetooth.println(F("Quit"));
     g = 1;
   }
     }

 }
```

C.1.4 Calibration

```
    //************* Calibration **************
  /* The user has 5 seconds to enter the calibration mode when typing the letter c as in
        calibration. The user is sent to the bluetoothCal() file */

void calibration(){

  for(byte u = 0; u<5; u++){
    delay(1000);
    bluetooth.println("Press c for calibration, You have ");
    bluetooth.println(r);
    r = r-1;
    if(bluetooth.available()>0){
      state = bluetooth.read();
    }
    if(state == 'c'){
      bluetoothCal();
      break;
    }
  }
}
```

C.1.5 Clock

```
//*************** Time setup *****************
  /* When calling rtc.now() the RTC_DS3231 is called in the RTClib.h
   library. now.year and so on called the year, month, day... The if statement is just
    extra control that the RTC started properly. If the year, month or day has wrong values,
   try and start the RTC again. Then append everything in a string called time. */

void clock(){

  DateTime now = rtc.now();
  y = now.year();
  m = now.month();
  d = now.day();
  h = now.hour();
  mi = now.minute();
  se = now.second();
  if(y<2016 || m> 12 || d>31){
    DateTime now = rtc.now();
    y = now.year();
    m = now.month();
    d = now.day();
    h = now.hour();
    mi = now.minute();
    se = now.second();
  }
  time += String(y);
  time += String("/");
  time += String(m);
  time += String("/");
  time += String(d);
  time += String(" ");
  time += String(h);
  time += String(":");
  time += String(mi);
  time += String(":");
  time += String(se);
  time += String("-");
}
```

C.1.6 Read text files

```
  //************ READING TEXT FILES **********
  /* Open the text file CalE.txt, parse out the values ke and ce and close the file.
  Open the text file Cal.txt, parse out the values k and c and close the file.
  Open the text file v.txt, parse out the v value */

 void readTxt(){

  File dataFile2 = SD.open("calE.txt");
  if(dataFile2){
    ke = dataFile2.parseFloat();
    ce = dataFile2.parseFloat();
    dataFile2.close();
  }
  dataFile2 = SD.open("cal.txt");
  if(dataFile2){
    k = dataFile2.parseFloat();
    c = dataFile2.parseFloat();
    dataFile2.close();
  }
    dataFile2 = SD.open("v.txt");
  if(dataFile2){
    v = dataFile2.parseFloat();
    dataFile2.close();
  }
 }
```

C.1.7 Receive Event

```
//***** When receive data is received ********
/*When slave request to send data, be ready to recieve file bytes of data (dataPot) on the I2C
      */

void receiveEvent(int howMany) {
  sleep = 0;
  while (Wire.available()) {
  //check if data is available on the I2C wire.
    for(int i = 0; i < 5; i++){
    dataPot [i] = Wire.read();
    // add each wire.read to the array
    //sleep = 0;
    //Serial.println(dataPot[i]);
    }
  }

}
```

C.1.8 Request Event

```
//NOT USED

void requestEvent(){
  sleep = 0;
  analogRead(A0);
  //dummyread
  dataEmg[0] = analogRead(A0)/4;
  dataEmg[0] = dataEmg[0] * ke + ce;
  dataEmg[1] = analogRead(A1)/4;
  dataEmg[1] = dataEmg[1] * ke + ce;
  dataEmg[0] = map(dataEmg[0], 0, 255, 0, 65.5*v);
  // Map so max gives output of v
  dataEmg[1] = map(dataEmg[1], 0, 255, 0, 65.5*v);
  Wire.write(dataEmg,2);
}
```

C.1.9 Sleep Mode

```
//*************** SLEEPMODE *****************
/*The different sleep modes are chosen at first. The most aggressive sleep mode,
      "SLEEP_MODE_PWR_DOWN" is set. Sleep is
enabled before disabled if an interrupt occurs. The interrupt is then detached so a new
      interrupt can be attached.*/

 void sleepNow(void){
   // sleep mode is set here
   //set_sleep_mode(SLEEP_MODE_IDLE);
   //set_sleep_mode(SLEEP_MODE_ADC);
   //set_sleep_mode(SLEEP_MODE_PWR_SAVE);
   //set_sleep_mode(SLEEP_MODE_STANDBY);
   set_sleep_mode(SLEEP_MODE_PWR_DOWN);
   //Most aggressive sleep
   sleep_enable();
   // enables the sleep bit in the mcucr register
   attachInterrupt(0,wakeUpNow, LOW);
   //Attach low to interrupt
   sleep_mode(); // Arduino sleeping
   // THE PROGRAM CONTINUES FROM HERE AFTER WAKING UP
   sleep_disable();
   // disable sleep
  detachInterrupt(0);
  // disables interrupt 0 on pin 2 so the
  // wakeUpNow code will not be executed
  // during normal running time.
 }
```

C.2 Slave Arduino code

C.2.1 Main code

```
//*********** I2CBus SLAVE (#8) **************
//Slave code created by Alexander Jordansson and Nima Haji sheykhi
// 11/11/2016

//*********** including libraries ***********
#include <Wire.h>
//wire library uses the I2C
#include <avr/sleep.h>
#include <avr/interrupt.h>
#include <avr/power.h>
#include <avr/io.h>
#include <avr/wdt.h>
//#include <SoftwareSerial.h>

//************ Global declaration ************
int pot0 = 10, pot1, pot2, pot3, pot4;
//Potentiometers
byte dataPot[5];
// Array to store received data from FSRs
byte dataEmg[2];
// Array to store received data from Master (#0)
byte sleep = 0;
//Constant for controlling when to enter sleep mode

//***************** Setup ********************
void setup() {
  Serial.begin(115200);
  // Begin serial communication
  Wire.begin(8);
  // Set i2c bus with address #8 (SLAVE)
  Wire.setClock(400000L);
  // Set the baud rate to 400 000 (400 KHz)
  Wire.onReceive(receiveEvent);
  // Start the wire to register events on Receive
  Wire.onRequest(requestEvent);
  // Start the wire i2c on request
  pinMode(5, OUTPUT);
  // EMG output for opening the hand
  pinMode(6, OUTPUT);
  // EMG output for closing the hand
  pinMode(2, INPUT_PULLUP);
  // Interrupt pin for external interrupts
  attachInterrupt(0, wakeUpNow, LOW);
  // Attached interrupt for sleep mode

}
void wakeUpNow(){
  //Do nothing
}

//***************** Loop ********************
void loop() {
  //digitalWrite(4,LOW);
  // Toggeling for oscilloscope measurments
  //delay(100);
  //digitalWrite(4, HIGH);

  //**** Reading FSR values ****
/* The Force sensor-values are read on the analog ports 0,1,2,3,4.
These values are divided by 4 so it is send as one byte and added in an array*/
  dataPot[0] = analogRead(A0)/4;
  dataPot[1] = analogRead(A1)/4;
  dataPot[2] = analogRead(A2)/4;
  dataPot[3] = analogRead(A3)/4;
  dataPot[4] = analogRead(A6)/4;
  Serial.println(dataEmg[0]);
  // Debugging purposes
  Serial.println(dataEmg[1]);
  // Debugging purposes

   /*The I2C interface. If the force sensor values are larger than the threshold value(5) and
        the EMG signals are smaller
  than 30 the transmission is started between slave (8) and master (0). Five bytes of data is
       sent (the dataPot array). The sleep counter is set to 0.
  Otherwise the sleep counter is added with 1 before entering the sleep code.
  Then the EMG values are send to the digital ports 5,6 for opening the hand.
```

```
  If the values are below och above threshold, the sleep counter increases, when it hits 10
        laps, the output against the hand
  is set to 0 before the sleep mode goes on*/

  if(dataEmg[0]<30 && dataEmg[1]<30 && (dataPot[0]> 5 || dataPot[1]> 5 || dataPot[2]> 5 ||
        dataPot[3]> 5 || dataPot[4]> 5)){
    digitalWrite(2,LOW);
    // force the interrupt pin 2 to be low so that attach interrupt doesn't go off.
    Wire.beginTransmission(0);
    Wire.write(dataPot, 5);
    Wire.endTransmission();
    sleep = 0;
  }
  else{
    digitalWrite(2,LOW);
    Serial.println(sleep);
    sleep++;
    if(sleep > 10){
      analogWrite(5,0);
      analogWrite(6,0);
      Serial.println("I sleep");
      delay(2);
      //so "??" is not written to the sd card
      sleepNow();
    }
  }
  analogWrite(5,dataEmg[0]);
  analogWrite(6,dataEmg[1]);

  //digitalWrite(4,LOW);
  // Toggling for oscilloscope measurements.
  //delay(100);
}
```

C.2.2 Receive Event

```
//************* When recieved ***************
/*When Master request to send data, listen on the Wire and add the EMG values to the Array.
      Reset the sleepCounter. */

void receiveEvent(int howMany) {
// this function is registered as an event, see setup()
    while (Wire.available()) {
      for(byte i = 0; i < 2; i++){
      dataEmg [i] = Wire.read();
      // add each wire.read to the array
       sleep = 0;
      //Serial.println("i recieve");
      //Serial.println(dataEmg[i]);
      }
   }
}
```

C.2.3 Request Event

```
//********* When requested to send ***********
/*When Master request to receive data, write on the Wire and add the Force sensors on the wire.
      Reset the sleepCounter. */

void requestEvent(){
  dataPot[0] = analogRead(A0)/4;
  dataPot[1] = analogRead(A1)/4;
  dataPot[2] = analogRead(A2)/4;
  dataPot[3] = analogRead(A3)/4;
  dataPot[4] = analogRead(A6)/4;
  Wire.write(dataPot,5);
  sleep = 0;
}
```

C.2.4 Sleep Mode

```
//*************** SLEEPMODE ****************
/*The different sleep modes are chosen at first. The most aggressive sleep mode,
      "SLEEP_MODE_PWR_DOWN" is set. Sleep is
enabled before disabled if an interrupt occurs. The interrupt is then detached so a new
      interrupt can be attached.*/

 void sleepNow(){

   // sleep mode is set here
   //set_sleep_mode(SLEEP_MODE_IDLE);
   // Least aggressive sleep
   //set_sleep_mode(SLEEP_MODE_ADC);
   //set_sleep_mode(SLEEP_MODE_PWR_SAVE);
   //set_sleep_mode(SLEEP_MODE_STANDBY);
   set_sleep_mode(SLEEP_MODE_PWR_DOWN);
   // Most aggressive sleep
   //noInterrupts();
   sleep_enable();
   // Enables the sleep bit in the mcucr register
   attachInterrupt(0,wakeUpNow, LOW);
   // Attach low to interrupt
   sleep_mode();
   // Arduino sleeping
   sleep_disable();
   // Disable sleep
  detachInterrupt(0);
  // Disables interrupt on pin 2 (0) so the it can attach a new interrupt.
 }
```

Appendix D

Schematics and PCB

D.1 Arduino master schematic

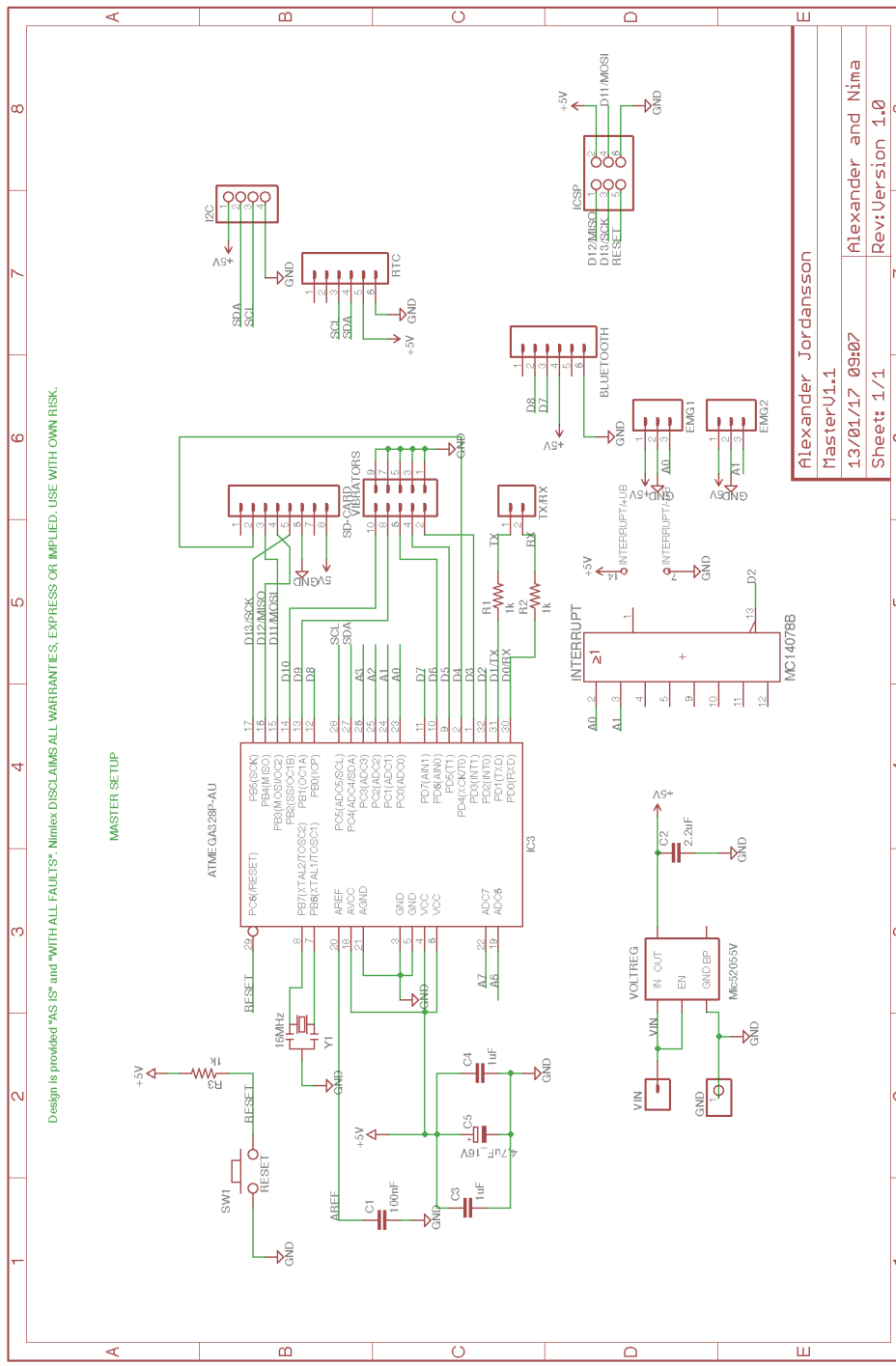


Figure D.1: Arduino master device Schematic.

D.2 Arduino master PCB

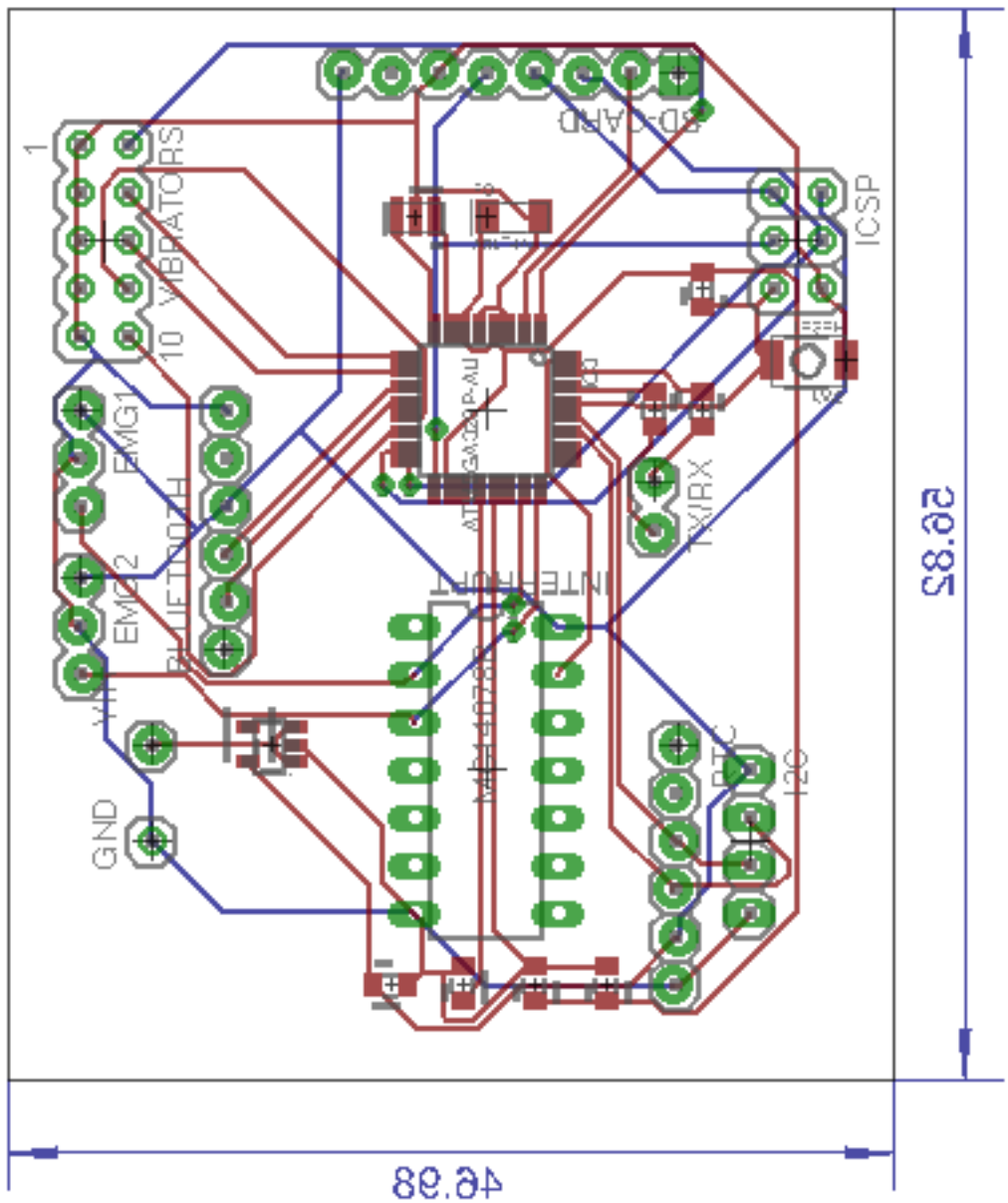


Figure D.2: Arduino Master PCB.

D.3 Arduino slave schematic

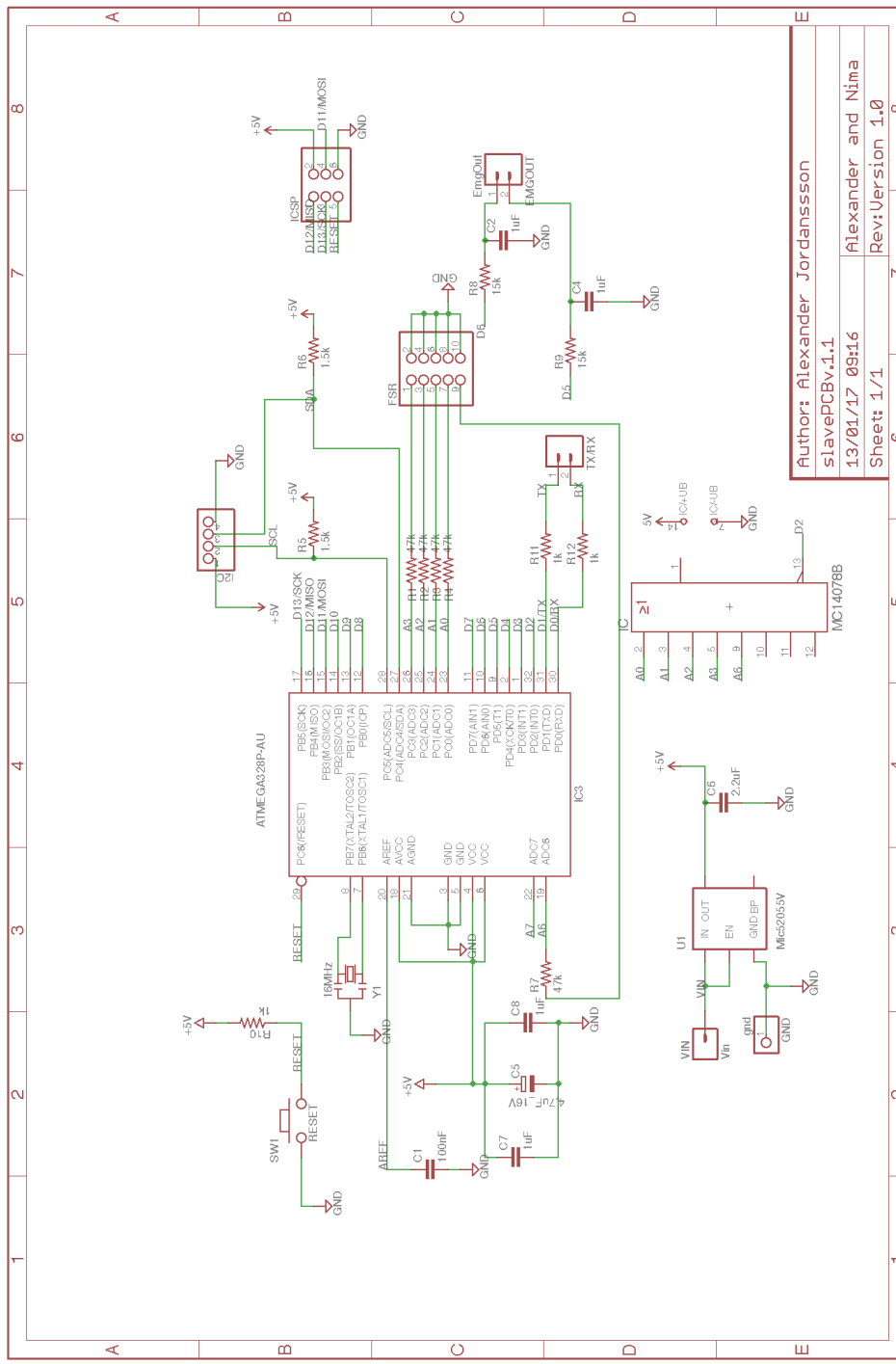


Figure D.3: Arduino slave device Schematic.

D.4 Arduino slave PCB

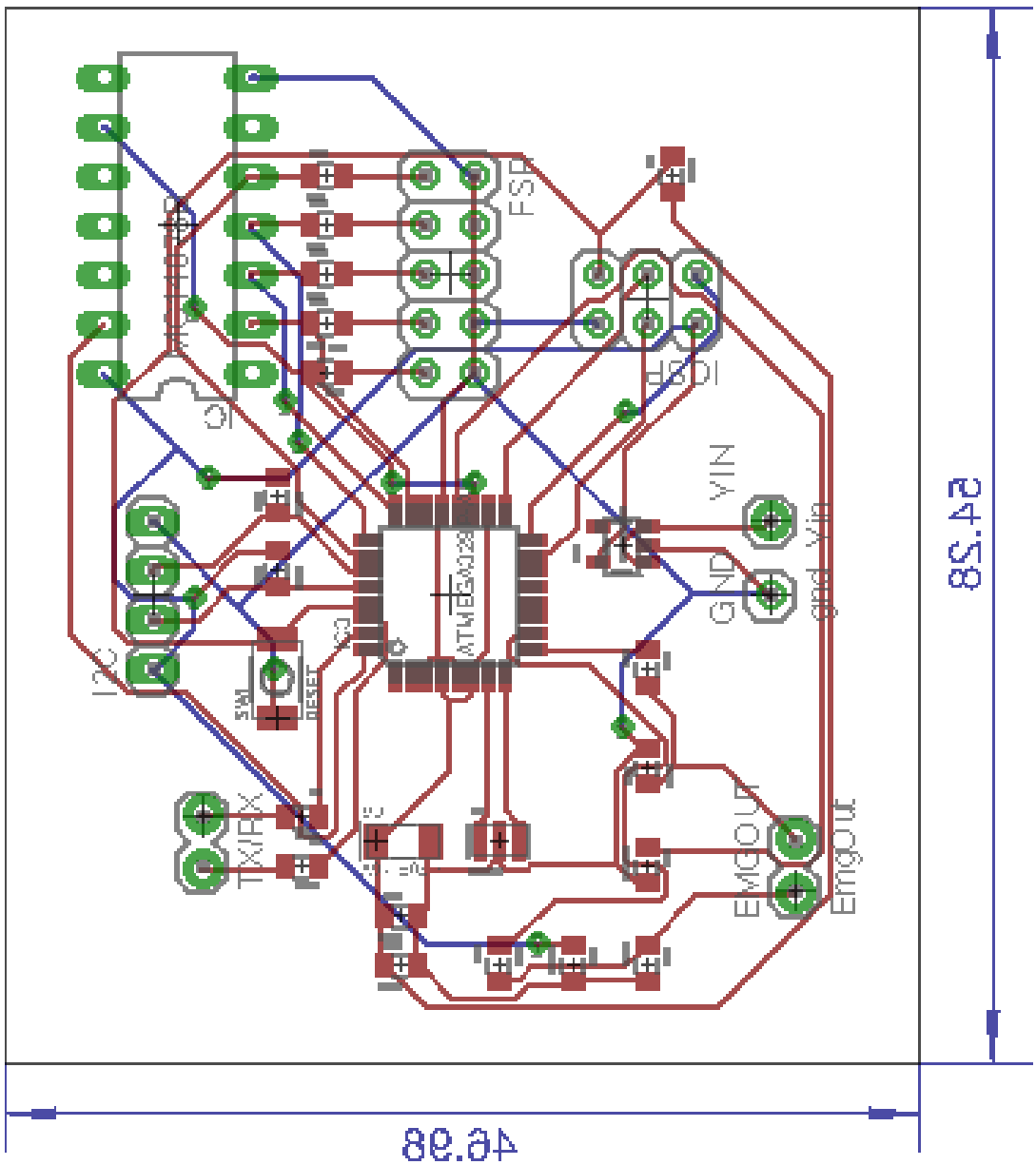


Figure D.4: Arduino slave PCB.

References

[1] Antfolk, *On sensory feedback in hand prostheses*. Lund University, 2012.

[2] M. Zecca, S. Micera, M. C. Carrozza, and P. Dario, "Control of multifunctional prosthetic hands by processing the electromyographic signal," *Crit Rev Biomed Eng*, vol. 30, no. 4-6, pp. 459–85, 2002.

[3] G. F. Shannon, "A myoelectrically-controlled prosthesis with sensory feedback," *Med Biol Eng Comput*, vol. 17, pp. 73–80, Jan 1979.

[4] A. Ninu, S. Dosen, S. Muceli, F. Rattay, H. Dietl, and D. Farina, "Closed-loop control of grasping with a myoelectric hand prosthesis: Which are the relevant feedback variables for force control?," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 22, pp. 1041–1052, Sept 2014.

[5] C. Pylatiuk, S. Schulz, and L. Döderlein, "Results of an internet survey of myoelectric prosthetic hand users," *Prosthet Orthot Int*, vol. 31, pp. 362–70, Dec 2007.

[6] A. Fougner, O. Stavdahl, P. J. Kyberd, Y. G. Losier, and P. A. Parker, "Control of upper limb prostheses: terminology and proportional myoelectric control-a review," *IEEE Trans Neural Syst Rehabil Eng*, vol. 20, pp. 663–77, Sep 2012.

[7] C. Antfolk, C. Balkenius, B. Rosén, G. Lundborg, and F. Sebelius, "Smarthand tactile display: A new concept for providing sensory feedback in hand prostheses," *Journal of Plastic Surgery and Hand Surgery*, vol. 44, no. 1, pp. 50–53, 2010.

[8] C. Antfolk, C. Balkenius, G. Lundborg, B. Rosén, and F. Sebelius, "Design and technical construction of a tactile display for sensory feedback in a hand prosthesis system," *BioMedical Engineering OnLine*, vol. 9, no. 1, p. 50, 2010.

[9] B. Rosén, H. H. Ehrsson, C. Antfolk, C. Cipriani, F. Sebelius, and G. Lundborg, "Referral of sensation to an advanced humanoid robotic hand prosthesis," *Scandinavian Journal of Plastic and Reconstructive Surgery and Hand Surgery*, vol. 43, no. 5, pp. 260–266, 2009.

[10] S.-O. F. F. S. G. L. B. R. Christian Antfolk, Anders Björkman, "Sensory feedback from a prosthetic hand based on air-mediated pressure from the hand to the forearm skin," *Journal of rehabilitation medicine.*, vol. 44, no. 8, p. 702–707, 2012.

[11] C. Antfolk, M. D'Alonzo, M. Controzzi, G. Lundborg, B. Rosen, F. Sebelius, and C. Cipriani, "Artificial redirection of sensation from prosthetic fingers to the phantom hand map on transradial amputees: Vibrotactile versus mechanotactile sensory feedback," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 21, pp. 112–120, Jan 2013.

[12] H. J. Witteveen, H. S. Rietman, and P. H. Veltink, "Vibrotactile grasping force and hand aperture feedback for myoelectric forearm prosthesis users," *Prosthetics and Orthotics International*, vol. 39, no. 3, pp. 204–212, 2015.

[13] A. J. Thurston, "Paré and prosthetics: the early history of artificial limbs," *ANZ journal of surgery*, vol. 77, no. 12, pp. 1114–1119, 2007.

[14] J. R. Kirkup, *A history of limb amputation*. Springer Science & Business Media, pp. 1,2007.

[15] G. Lundborg, "Tomorrow's artificial hand," *Scandinavian Journal of Plastic and Reconstructive Surgery and Hand Surgery*, vol. 34, no. 2, pp. 97–100, 2000.

[16] Antfolk, *On sensory feedback in hand prostheses*. Lund University, pp. 4, 10 ,2012.

[17] P. D. Marasco, A. E. Schultz, and T. A. Kuiken, "Sensory capacity of reinnervated skin after redirection of amputated upper limb nerves to the chest," *Brain*, p. awp082, 2009.

[18] U. Wijk and I. Carlsson, "Forearm amputees' views of prosthesis use and sensory feedback," *Journal of Hand Therapy*, vol. 28, no. 3, pp. 269 – 278, 2015.

[19] Arduino, "Arduino board Nano." `https://www.arduino.cc/en/Main/ArduinoBoardNano`, 2016.

[20] B. A. Forouzan, "Global edition," in *Data Communications and Networking*, ch. 4, p. 97, McGraw-Hill Education, 2012.

[21] Arduino, "Baud rate." `https://www.arduino.cc/en/serial/begin`, 2017.

[22] R. Meier, "Roger Meier's Freeware." `http://freeware.the-meiers.org/`, 2016.

[23] C. Mitter, "Communication device having asynchronous data transmission via symmetrical serial interface," Mar. 6 2007. US Patent 7,188,207.

[24] B. A. Forouzan, "Global edition," in *Data Communications and Networking*, ch. 4, pp. 126–129, McGraw-Hill Education, 2012.

[25] Motorola, "SPI Block Guide, V03.06." `https://web.archive.org/web/20150413003534/http://www.ee.nmt.edu/~teare/ee308l/datasheets/S12SPIV3.pdf`, pp. 15, Online accessed 2017-01-17.

[26] Arduino, "A Brief Introduction to the Serial Peripheral Interface (SPI)." `https://www.arduino.cc/en/Reference/SPI`, 2016.

[27] Arduino, "SoftwareSerial Library." `https://www.arduino.cc/en/Reference/softwareSerial`, Online accessed 2017-01-12.

[28] N. Semiconductors, "i2C-bus specification and user manual." `http://www.nxp.com/documents/user_`

manual/UM10204.pdf, pp. 3-22 Online accessed 2017-01-17.

[29] SFUptownMaker, "i2C." https://learn.sparkfun.com/tutorials/i2c, Online accessed 2017-01-26.

[30] Atmel, "ATmega328-328P Datasheet." http://www.atmel.com/Images/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, pp. 39-41, 62, 64-65, Online accessed 2017-01-17.

[31] N. Gammon, "Interrupts." http://gammon.com.au/interrupts, Online accessed 2017-01-19.

[32] Arduino, "Interrupts." http://playground.arduino.cc/Code/Interrupts, 2017.

[33] N. Instruments, "PWM." https://www.ni.com/tutorial/2991/en/, Online accessed 2017-01-02.

[34] Arduino, "PWM." https://www.arduino.cc/en/Tutorial/PWM, Online accessed 2017-01-30.

[35] Arduino, "Memory." https://www.arduino.cc/en/Tutorial/Memory, Online accessed 2017-01-19.

[36] B. Earl, "Memories of an Arduino." https://cdn-learn.adafruit.com/downloads/pdf/memories-of-an-arduino.pdf, pp.8-9,19,Online accessed 2017-01-19.

[37] Arduino, "Memory." http://playground.arduino.cc/Learning/Memory, Online accessed 2017-01-19.

[38] Arduino, "EEPROM Library." https://www.arduino.cc/en/Reference/EEPROM, Online accessed 2017-01-19.

[39] S. E. Kim, R. S. List, and S. A. Kellar, "Process of vertically stacking multiple wafers supporting different active integrated circuit (ic) devices," Jan. 2 2007. US Patent 7,157,787.

[40] Adafruit, "Data logger." https://www.adafruit.com/product/254, 2016.

[41] dpreview staff, "Three Giants to develop new "Secure Memory Card"." https://www.dpreview.com/articles/6861681955/newmemory, Online accessed 2016-12-16.

[42] C.-J. Chen, "Memory card for integrating the usb interface and an adaptor for the memory card," June 13 2006. US Patent 7,062,585.

[43] K. . Co., "RTC." https://www.kjell.com/se/sortiment/el-verktyg/elektronik/arduino/moduler/realtidsklocka-for-arduino-p87984, Online accessed 2017-01-12.

[44] H. Haiplik and C. R. Graham, "Real time clock," Apr. 20 2010. US Patent 7,702,943.

[45] Sparkfun, "Bluetooth Mate Gold." https://www.sparkfun.com/products/12580, Online accessed 2017-01-19.

[46] P. M. drivers, "310-101." https://www.precisionmicrodrives.com/sites/default/files/310-101-datasheet.pdf, Online accessed 2017-01-17.

[47] pcb global, "Printed Circuit Board History and Evolution." http://www.pcbglobal.com/7/blog.html, Online accessed 2017-01-20.

[48] Autodesk, "Eagle." http://www.autodesk.com/products/eagle/overview, Online accessed 2017-01-20.

[49] N. N. Instruments, "Labview." http://sweden.ni.com/labview, Online accessed 2016-12-13.

[50] physiclight, "ARDUINO SERIAL WRITE AND READ USING LABVIEW." https://physicslight.wordpress.com/2014/07/14/arduino-labview-serial-write-read/, 2016.

[51] physiclight, "Arduino serial write and read using LabVIEW." https://github.com/marcomauro/Arduino-LabVIEW, 2016.

[52] Arduino, "I^2C protocol." https://www.arduino.cc/en/Reference/Wire, Online accessed 2017-01-23.

[53] Arduino, "analogWrite()." https://www.arduino.cc/en/Reference/analogWrite, Online accessed 2017-01-23.

[54] GitHub, "RTClib.h library." https://github.com/adafruit/RTClib, Online accessed 2017-01-24.

[55] R. Keim, "Low-Pass Filter a PWM Signal into an Analog Voltage." http://www.allaboutcircuits.com/technical-articles/low-pass-filter-a-pwm-signal-into-an-analog-voltage/, Online accessed 2017-01-23.

[56] A. about circuits, "Low-pass Filters." http://www.allaboutcircuits.com/textbook/alternating-current/chpt-8/low-pass-filters/, Online accessed 2017-01-30.

[57] greiman/SdFat, "Arduino FAT16/FAT32 Library." https://github.com/greiman/SdFat, Online accessed 2017-01-20.

[58] N. Gammon, "Interrupt bug." http://www.gammon.com.au/interrupts, Online accessed 2017-01-24.

[59] Arduino, "Arduino Nano PCB." http://download.arduino.org/products/NANO/Arduino%20Nano-Rev3.2-SCH.pdf, Online accessed 2017-01-23.

[60] Motorola, "B-Sufflix Series CMOS Gates." http://pdf1.alldatasheet.com/datasheet-pdf/view/90023/MOTOROLA/MC14001B.html, pp. 1-2, Online accessed 2017-01-26.

[61] TOSHIBA, "TC4071BP Datasheet - Toshiba Semiconductor." http://pdf1.alldatasheet.com/datasheet-pdf/view/31643/TOSHIBA/TC4071BP.html, pp. 1, Online accessed 2017-01-26.

[62] Ottobock, "Prosthetics." http://www.ottobock.se/media/local_media_1/bu-prosthetics/

proteskatalog-övre-extremitet-2.pdf, pp. 34,85,Online accessed 2017-01-19.

[63] L. Ada, "Micro SD Card Breakout Board Tutorial." https://cdn-learn.adafruit.com/downloads/pdf/adafruit-micro-sd-breakout-board-card-tutorial.pdf, pp. 3, 6, Online accessed 2017-01-26.

[64] RN-41, "Bluetooth RN-41." http://cdn.sparkfun.com/datasheets/Wireless/Bluetooth/Bluetooth-RN-41-DS.pdf, pp. 1, Online accessed 2017-01-26.

[65] I. Electronics, "FSR 400 data sheet." https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/2010-10-26-DataSheet-FSR400-Layout2.pdf, Online accessed 2017-01-12.

[66] M. integrated, "DS3231." https://datasheets.maximintegrated.com/en/ds/DS3231.pdf, pp. 1, Online accessed 2017-01-26.

[67] Micrel, "MIC5205." http://ww1.microchip.com/downloads/en/DeviceDoc/mic5205.pdf, pp. 9, Online accessed 2017-01-26.