TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Thomas Johann Seebeck Department of Electronics

IEE70LT

Jüri Bogatkin 144168IVEM

LOW COST HIGH PRECISION DEVICE FOR 3D ORIENTATION

Master's Thesis

Supervisor:	Alar Kuusik
	PhD
	Senior Research Scientist
Co-Supervisor:	Olev Märtens
	PhD
	Lead Research Scientist

Tallinn 2016

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Thomas Johann Seebecki elektroonikainstituut

IEE70LT

Jüri Bogatkin 144168IVEM

SOODNE JA TÄPNE 3D ORIENTATSIOONI MÄÄRAMISE SEADE

Magistritöö

Juhendaja:	Alar Kuusik
	PhD
	Vanemteadur
Kaasjuhendaja:	Olev Märtens
	PhD
	Juhtivteadur

Tallinn 2016

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Jüri Bogatkin

30.05.2016

Abstract

IMUs (Inertial Measurement Units) are widely spread. Off-the-shelf, low-cost, precise modules, which are easily connectable to a PC are rarely found.

This thesis represents the development and evaluation of 3D orientation IMU module. During the thesis hardware, firmware and demo software design processes are described. Also an economical overview is given and some testing is covered.

In the beginning of the thesis a market overview is represented, which is followed by specifications for the system. According to the specifications hardware and firmware are developed. In the center of the design is a Bosch BNO055 integral circuit, which internally calculates orientation data. The firmware serves as bridge between the IC and the communication interface. The designed demo software represents an interactive 3D virtual object and displays measured angles in degrees according to the physical device.

As a result of this thesis a fully functional system was developed. The device was evaluated and the correct functionality verified. The accordance to specification was largely met with small exceptions. Thorough testing and some future development is yet to be done.

This thesis is written in English language and is 63 pages long, including 6 chapters, 36 figures and 8 tables.

Annotatsioon

Soodne ja täpne 3D orientatsiooni määramise seade

IMU’d ehk Inertsiaalsed andurid on laialt levinud. Valmiskujul on soodsaid, täpseid ja lihtsalt arvutite külge integreeritavaid mooduleid vähe pakkumisel. Töös kirjeldatakse OEM seadme projekteerimist, mis autori hinnangute kohaselt rahuldab robootika, VR (virtuaalreaalsuse), videomängu juhtimisseadmete ja drooni valdkondade vajadusi.

Antud töös käsitletakse 3D orientatsiooni mooduli väljatöötamist ja esialgset katsetamist. Töö jooksul antakse ülevaade riistvara, püsivara ja demo tarkvara arendamisest. Samuti tuuakse välja majanduslikud kalkulatsioonid ja räägitakse testimisest ning selle tulemustest.

Pärast sissejuhatust räägitakse kõigepealt 3D orientatsiooni mõõtmise rakendustest, seejärel antakse ülevaade turu hetkeseisust ja sealt liigutakse süsteemi spetsifikatsioonide juurde. Vastavalt nõuetele töötatakse välja riistvara ja seadme püsivara. Riistvara keskmeks on Bosch BNO055 integraalskeem, mis on sisemiselt võimeline arvutama ruumi nurki ja väljastama neid Kvaternioonide või Euleri nurkadena. Püsivara peamiseks ülesandeks on töötada „sillana“, edastades BNO055’st välja loetud infot arvutile (või mõnele muule arvutusplatvormile). „9DoF Smart Board“ moodul võimaldab edastada nii ruumi nurkade kui ka töötlemata 9-telje infot. Arendatud demo tarkvara representeerib interaktiivset 3D objekti, mille pöörlemine järgib füüsilise seadme rotatsiooni ruumis. Lisaks kuvab tarkvara iga ruumi nurga info numbrilisel kujul.

Magistritöö tulemusel valmis täisfunktsionaalne süsteem. Katsetest võib järeldada, et seade toimib nii nagu projekteeritud ja vastab enamjaolt väljatöötatud nõutele. Põhjalikum testimine ja mõningane arendus ootavad veel ees.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 63 leheküljel, 6 peatükki, 36 joonist, 8 tabelit.

List of abbreviations and terms

6DoF	6 Degrees of Freedom
9DoF	Nine Degrees of Freedom
AHRS	Attitude and Heading Reference System
ASF	Atmel Software Framework
BOM	Bill of Materials
CAD	Computer-aided Design
CDC	Communication Device Class
DFU	Device Firmware Upgrade
ESD	Electrostatic Discharge
IC	Integrated Circuit
IMU	Inertial Measurement Unit
LDO	Low-dropout Regulator
LED	Light-emitting Diode
LUFA	Lightweight USB Framework for AVRs
mA	milliampers
MARG	Magnetic, Angular Rate, and Gravity
MAX	Maximum
MCU	Microcontroller Unit
MEMS	Microelectromechanical Systems
MIN	Minimum
PCB	Printed Circuit Board
PDI	Program and Debug Interface
PLL	Phase Locked Loop
PPM	Parts Per Million
PSRR	Power Supply Rejection Ratio
PTC	Positive Temperature Coefficient
SoC	System On a Chip
TUT	Tallinn University of Technology

TYP	Typical
UART	Universal Asynchronous Receiver/Transmitter
uF / μF	microFarads
USB	Universal Serial Bus
VI	Virtual Instrument

Table of contents

List of figures

List of tables

1 Introduction

IMUs (Inertial Measurement Units) are practically surrounding us in every step of the way nowadays. You can find these devices in your mobile phone, in your car, in your game controller etc. When talking about IMUs we mostly think about accelerometer, gyroscope and magnetometer (or in other words electronic compass). Many of these devices have been in use for hundreds of years. Historically they have been large mechanical or electromechanical systems. Only recently, from the beginning of the 90's, they have developed into MEMS which are small and can be fit almost anywhere. MEMS are often produced in a very small packages which surface area can be measured in square millimeters. Although it is quite easy to measure one precise parameter in a short amount of time (acceleration, rotational speed or strength of a magnetic field) a problem arises when you want to measure an object's exact orientation in 3D space.

Practice shows that 2 or 3 of these sensors have to be combined to get usable orientation data. This is done by using sensor fusion algorithms. There are basically 3 types of products on the market. The main shortcoming of lower end is a need for hardware and software development. The high-end devices that already output 3D orientation data are really expensive (starting from 800€) and are not very user friendly.

Not finding a plug-and-play type of 3D orientation device which would be inexpensive, ELIKO decided to create one their own. The goal was to have a small device that can be used for robot navigation and other use cases where 3D orientation or precise angle measurement in 3D space is needed.

The goal of this thesis is to give an overview and describe the development process of an absolute orientation module by the author when working in ELIKO Competence Center.

1.1 Applications of the 3D Orientation Sensing

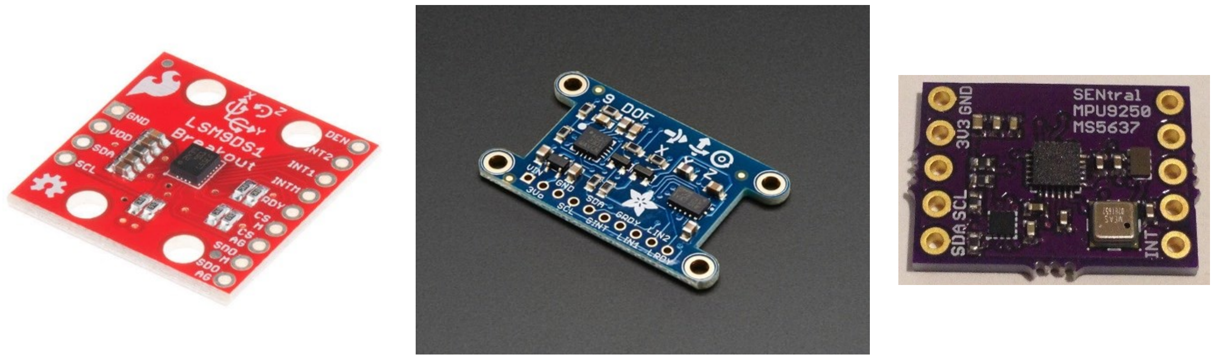
There are many applications for 3D Orientation Sensing. Most of the fields of use are listed below.

UAVs; Spacecraft; VR and head tracking; Robot navigation systems; Mobile phones; Sports Technology; Motion Capture; Music Technology (wah-wah pedal, volume control etc.); Pedometer; Smart Watch; Sports Watch;

1.2 Products on the Market

There are many IMU products available on the market – in a form of integrated circuits, platforms, evaluation kits, ready to use systems etc. Although there are many IMUs available there are far less products which output 3D orientation data. The latter are the main interest regarding to this thesis. Mentioned systems basically fall into two different categories – modules which already output 3D orientation data or combined hardware and software solutions which deliver 3D orientation as an end result.

There are many products which output 6-axis or 9-axis sensor data. These are usually PCB's which incorporate accelerometer and a gyroscope in the first case or accelerometer, gyroscope and magnetometer alternatively (9DoF). Measured 6DoF or 9DoF sensor data can be combined into 3D orientation information using the appropriate algorithms. These algorithms are usually called sensor fusion or orientation filter algorithms.



(a) Sparkfun LSM9DS1 Breakout (b) Adafruit's 9DoF IMU Breakout (c) Tindie's "Ultimate Sensor Fusion Solution"

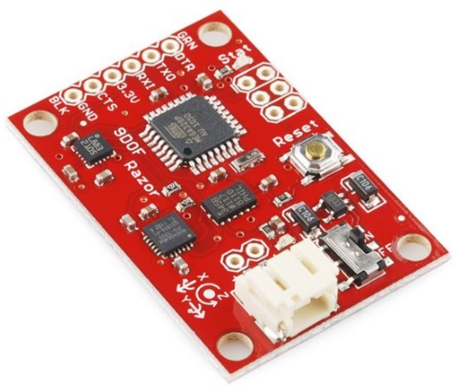
Figure 1. Examples of IMU Breakout Boards.

Some of the products have sensors onboard and do not incorporate any microcontroller (or DSP). These can be called "breakout" style boards. For example Sparkfun's LSM9DS1 Breakout [Figure 1(a)], Adafruit's "9DoF IMU Breakout" [Figure 1 (b)], Tindie's "Ultimate Sensor Fusion Solution" [Figure 1 (c)] [1] [2] [3] . There are of course more of similar products available on the market (these are just a few examples). The breakout board style products need an external computational platform, which can communicate with the onboard sensors, collect the data and calculate orientation information based upon the measurements.

The drawbacks for such "breakout" board products are:

- Need for external computational platform
- Need for sensor fusion algorithms
- May need extra communication "bridge" (for example I2C to USB or other)
- Cost around 15 - 40€
- May need knowledge of electronics
- May need knowledge of embedded programming

The second type of products are the ones that involve onboard computational device (usually microcontroller or DSP), but still need to be programmed by the user – for example Sparkfun's 9DoF Razor IMU [4] or Tindie's 9 DOF IMU with LSM9DS1 and Atmega328p [5] (Figure 2).


(a) Sparkfun's "9DoF Razor IMU"

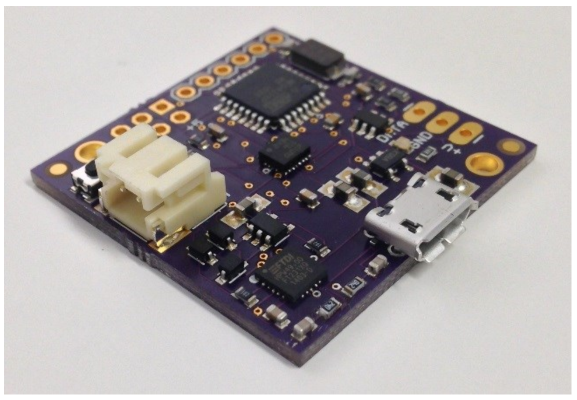

(b) Tindie's 9 DOF IMU with LSM9DS1 and Atmega328p

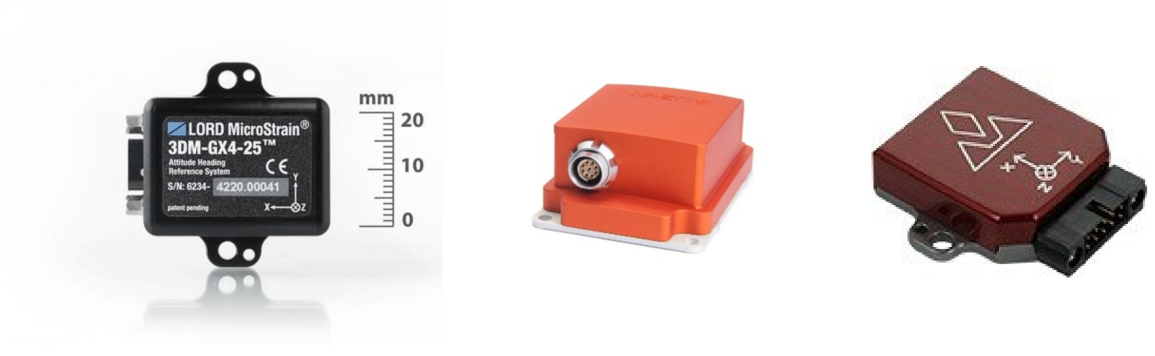
Figure 2. Examples of IMU + MCU Type Products

These boards have higher integration as they include sensors and a microcontroller onboard. Some of similar products come with firmware examples and some do not. So ultimately it is still a user responsibility to have some level of knowledge about embedded programming and electronics. Hardware wise there are all the components on these platforms to create 3D absolute orientation device.

The drawbacks of IMU + MCU type of products products:

- May need firmware (user / client responsibility)
- May need knowledge of electronics from user / client
- May need knowledge of embedded programming from user / client
- Higher cost than “breakout” board style products. Around 60 - 300€.

The last category are the readymade solutions. These products incorporate all the needed hardware and software, and output 3D orientation data in a form of Euler [6] angles or quaternions [7] . Another difference to the devices mentioned above is that they are usually put into specially designed enclosures making them a completely finished products. Such systems are manufactured by Xsens, MicroStrain, VectorNav, Intersens, PNI and Crossbow etc. Figure 3 represents some examples.



(a) MicroStrain’s 3DM-GX4 -25 (b) Xsens’ MTi-10-IMU (c) VectorNav’s VN-100 Rugged

Figure 3. Examples of Readymade 3D Orientation Modules

The drawbacks of the readymade 3D orientation modules:

- High price. 800 - 5000€ (or more)
- Some modules need manual calibration
- Custom communication interfaces, not easily usable

The most important setback of the complete modules is their high price, starting from 800€ and going up to 5000€ or more. And still some of them need manual calibration or are very sensitive to magnetic distortions. Regardless of the high price some measurements have shown that the accuracy of such modules may be comparable to the solutions which integrate sensors + MCU or even worse.

1.3 The Problem

One of the research and development projects in ELIKO is aiming to create an autonomous vehicle. This vehicle uses mainly GPS and image processing for navigation. To increase the accuracy and make the navigation still operational, if GPS signal is lost, an AHRS (Attitude and Heading Reference Systems) is needed. This was the main motivator for researching the available solutions on the market. After doing the market research and knowing the drawbacks of the existing systems ELIKO decided to create its own product. Although the motivation came mainly from the autonomous vehicle project it was kept in mind that the end solution could be sold as a separate product and used for other applications.

1.4 Solution and Specifications for the System

After knowing the requirements of the autonomous vehicle and the setbacks of the existing solutions on the market, the main criteria for were put on a paper.

General specifications for the product.

- Output's 3D orientation data, in quaternion or Euler angle format
- Low price. In the range of 20 - 50€
- Small physical size
- Should be as accurate as possible
- Small or no angular drifting in time

Technical Specifications.

After working out the general requirements specific technical details were established.

- Communication
    a) Virtual COM port (USB 2.0)

    b) UART / Regular COM port (115200 baud)

- Simultaneous communication over USB and UART. If not possible, can be chosen with a jumper
- Timestamp resolution: at least 1ms
- Jumper selectable output formats
    a) Quaternion
    b) Raw Data
- Quaternion
    - Data output format: q_1, q_2, q_3, q_4, timestamp
    - Sample rate: at least 100Hz
    - q_1...q_2 - quaternion
- Raw Data
    - Data output format: a_x, a_y, a_z, g_x, g_y, g_z, m_x, m_y,m_z, timestamp
    - Sample rate: at least 1kHz
    - a_x...a_z – raw acceleration (g)
    - g_x...g_z – raw gyroscope (deg / sec)
    - m_x...m_z – raw magnetometer
- COM to Sensor bridge for changing settings

2 Hardware Design

This section gives an overview of the overall hardware design and components. The hardware was designed according to the specification presented in section 1.4 and consists of 2 mature parts – electrical schematic and PCB layout. Both parts were designed using Altium Designer software.

The system hardware is composed of the following parts (Figure 4): Absolute Orientation IMU; Microcontroller; Connector for PDI programming and user configurations; Micro USB Connector; UART Connector; Automatic power selector; Voltage Regulator (LDO) and LED Indicators.

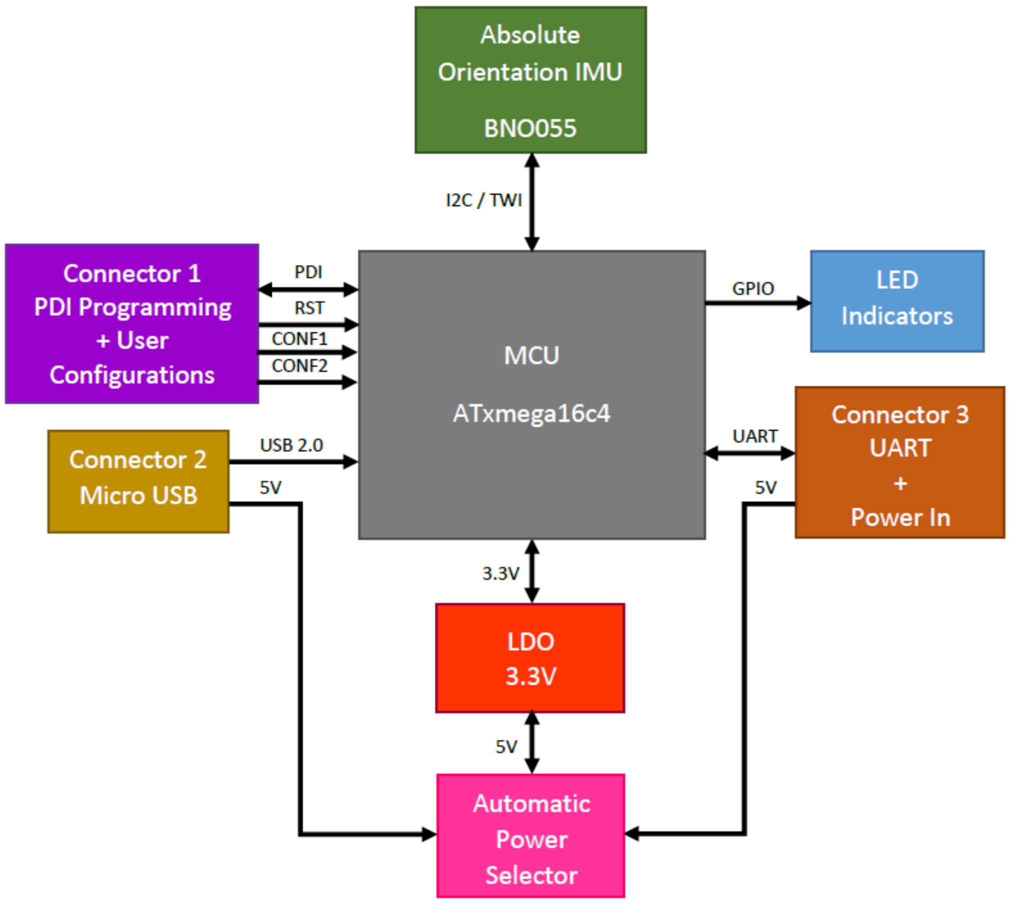


Figure 4. Hardware Block Diagram Developed for the Project

2.1 Electrical Specifications for the Device

The main application for our device was foreseen as a part of a bigger system which can communicate with the module over USB or UART connection and also produce enough power. Although the board can be used with battery power it was not considered when designing the 9DoF Absolute Orientation Module. Therefore the current consumption is specified mostly by the USB 2.0 specification [8] .which is up to 500mA. The supply voltage also follows the USB 2.0 specification. The requirements were produced so that it would be possible to supply the board from any other external power source. Table 1 lists all the electrical specifications.

Table 1 Electrical Specification of USB2.0 interface

Parameter	Symbol	Min	Typ	Max	Units
Supply Voltage	V_{DD}	4.4	5.0	5.25	V
Total Supply Current	I_{DD}	-	-	500	mA

During hardware design there were 3 main aspects that were considered in every step of the way:

- The device should be as cheap as possible
- The device should be as small as possible
- The device should consume reasonable amount of power

As we were not designing a special low power device the power consumption was not optimized to minimum possible value.

2.2 Electrical Schematic Design

The electrical schematic was designed using Altium Designer Software. The whole system was built around the Bosch BNO055 absolute orientation sensor. In the time of creating this system it was one of a kind IC available on the market. The SoC contains Bosch Sensortec sensor fusion software, in a single package outputting already pre-calculated orientation information, which is not available in other IMU ICs. The electrical environment surrounding the sensor was basically created just to support the BNO055 functionality. Although the BNO055 [9] incorporates 32-bit ARM cortex M0+ microcontroller and UART connection, the system does not output measurement information automatically nor with a given frequency. It is also not possible for a user to program the ARM cortex M0+ MCU which is integrated into the BNO055 system in package. Furthermore the sensor needs to be configured first and after that the measurement information can be read from corresponding output registers [9] . This basically leaves the user 2 choices – use an MCU or any other external computing platform (like PC) which can communicate with the BNO055 and run needed algorithms. In our case the goal was to create a small plug-and-play type of product that is very easy to use for a customer, making an MCU the obvious choice.

2.2.1 9DoF IMU

The Bosch BNO055 datasheet [9] includes application diagrams for using the IC. The schematic was created following these guidelines.

Under this sub-task the biggest decision was choosing the interface type for MCU communication. According to the specifications the speed for reading the raw data had to be as fast as possible- at least 1 kHz. The possible protocols were I^2C and UART. After reading the datasheet and completing the calculations it was clear that I^2C can produce faster data rates than UART. Therefore I^2C was used to communicate with the MCU. The created electrical schematic is given in Figure 5.

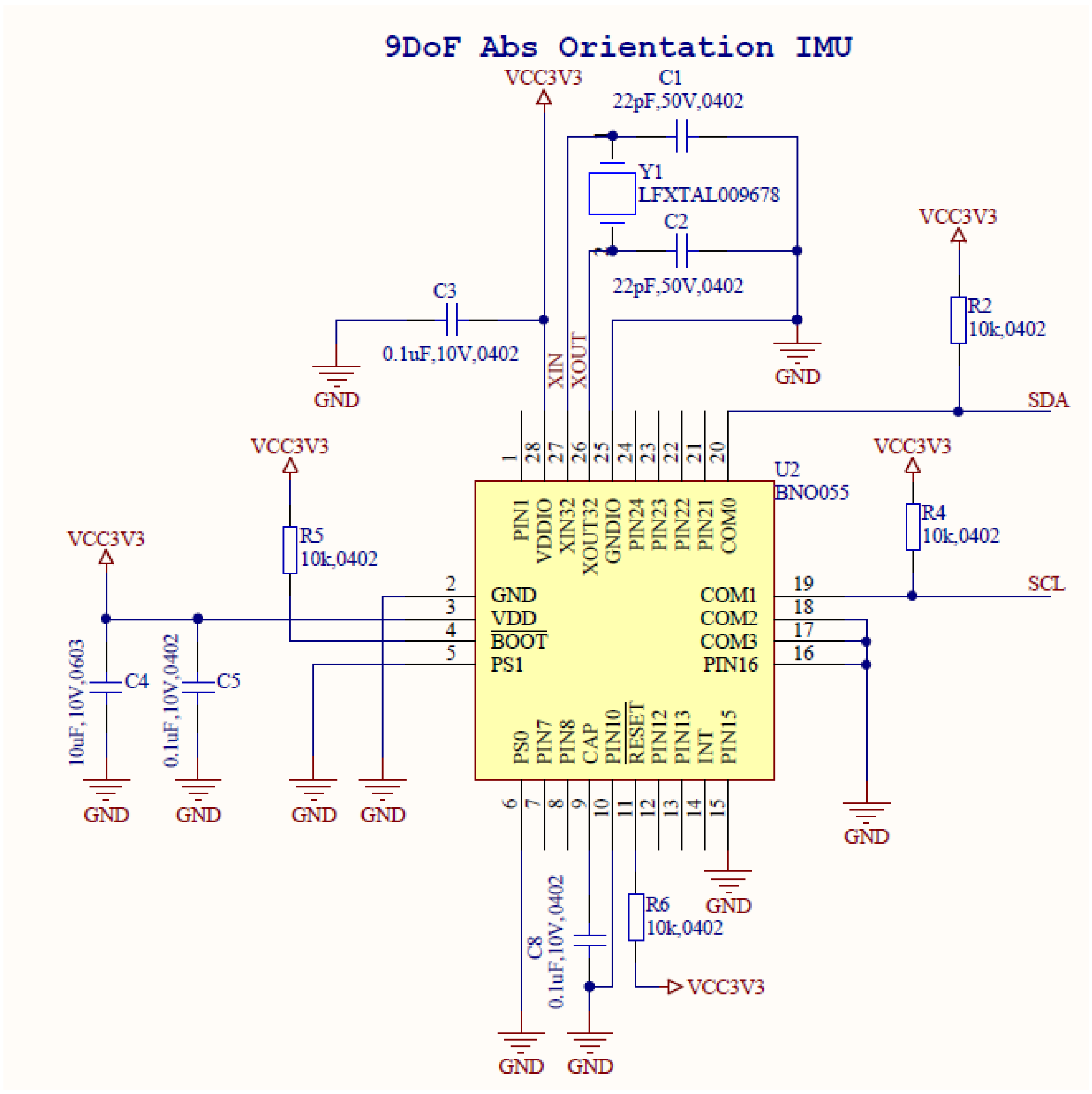


Figure 5. Circuit for BNO055 IMU

2.2.2 Microcontroller

The MCU was picked according to the following criteria:

- Inexpensive
- Widely available off-the-self product
- Atmel Microcontroller
- Widely available off-the-self product
- USB 2.0 Interface
- UART Interface
- Hardware I2C controller

- Some general purpose I/O pins for user configurations

When choosing the microcontroller unit the processing speed and power were not very critical because most of the processing is already done inside the BNO055 chip. The microcontroller serves more like a “bridge” between IMU and corresponding communication interfaces – USB 2.0 and / or UART. Atmel microcontroller was chosen because of the pre-existing in-house knowledge and experience in ELIKO. The cheap price and low processing power requirements lead to 8-bit solution as this was enough for our application.

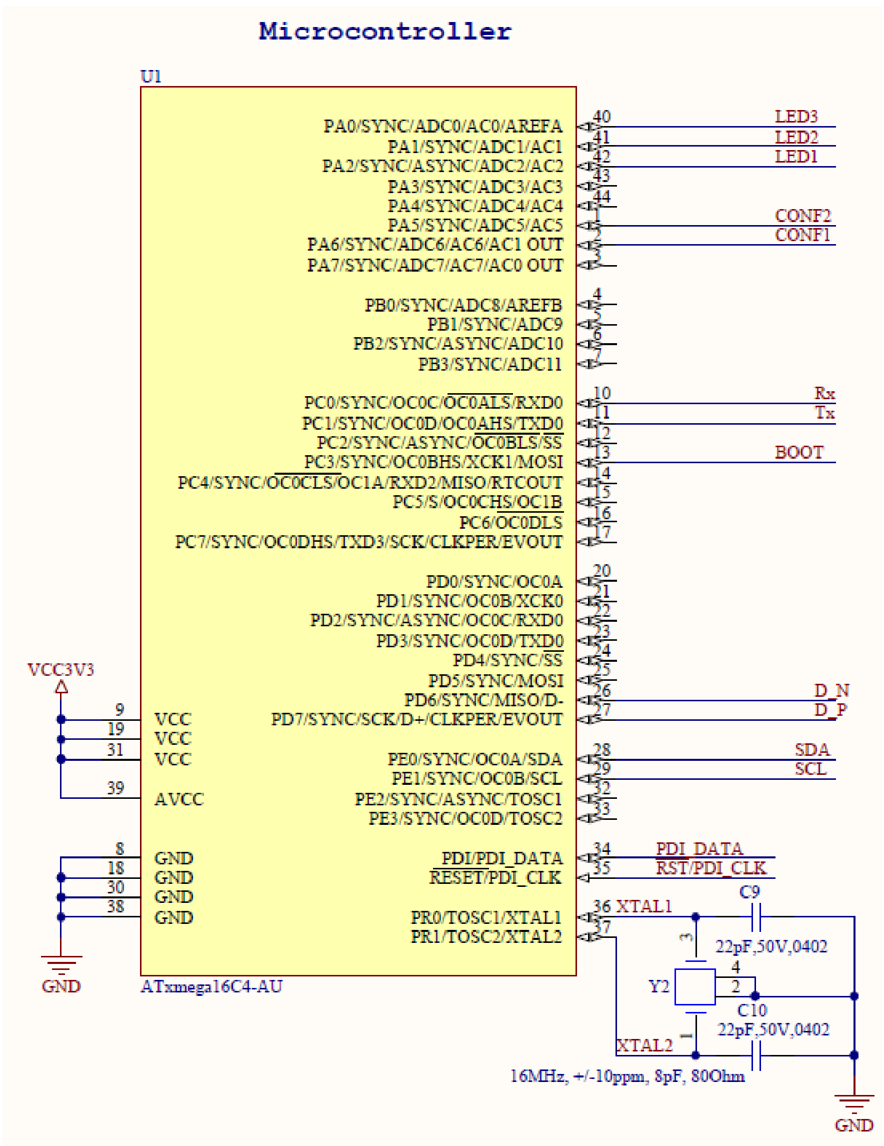


Figure 6. Schematic Diagram Created for the ATxmega16C4 Microcontroller

Considering all the requirements ATxmega16C4 [10] was chosen for our solution, which is an 8-bit, 32MHz, 16KB flash, 44 pin, TQFP microcontroller. Although the same MCU is available in smaller packages, TQFP was the cheapest one and gives and extra value as it is easier to solder by hand for prototyping. Some key features of the chosen microcontroller are listed below.

- 16K - 32KB of In-System Self-Programmable Flash
- USB 2.0 full speed (12Mbps) and low speed (1.5Mbps) device compliant
- Three USARTs with IrDA support for one USART
- Two two-wire interfaces with dual address match (I2C and SMBus compatible)
- Operating Voltage
    - 1.6 – 3.6V
- Operating frequency
    - 0 – 12MHz from 1.6V
    - 0 – 32MHz from 2.7V

The end result containing the microcontroller part for the Absolute Orientation Module is given in Figure 6.

One more thing worth mentioning here is the external crystal which was chosen to generate the clock frequency for the ATxmega16C4 MCU. Atmel AVR XMEGA C4 devices have a flexible clock system supporting a large number of clock sources (Figure 7). It incorporates both accurate internal oscillators and external crystal oscillator and resonator support. A high-frequency phase locked loop (PLL) and clock prescalers can be used to generate a wide range of clock frequencies.

The external crystal oscillator was chosen because it can generate more accurate clock for the MCU than the internal ones. A cheap, small-size and quite accurate crystal was chosen for the task – Epson TSX-3225. It is a 3.2 X 2.5mm, 16MHz crystal oscillator.

Internal oscillators' accuracy varies from ± 0.5% to ± 30% (specific information can be found from datasheet [10] page 76) as the external crystal accuracy is ± 10ppm or ± 0.001%, it is 500 times more accurate than the most accurate clock built inside the microcontroller. C9 and C10 are load capacitors for the crystal oscillator.

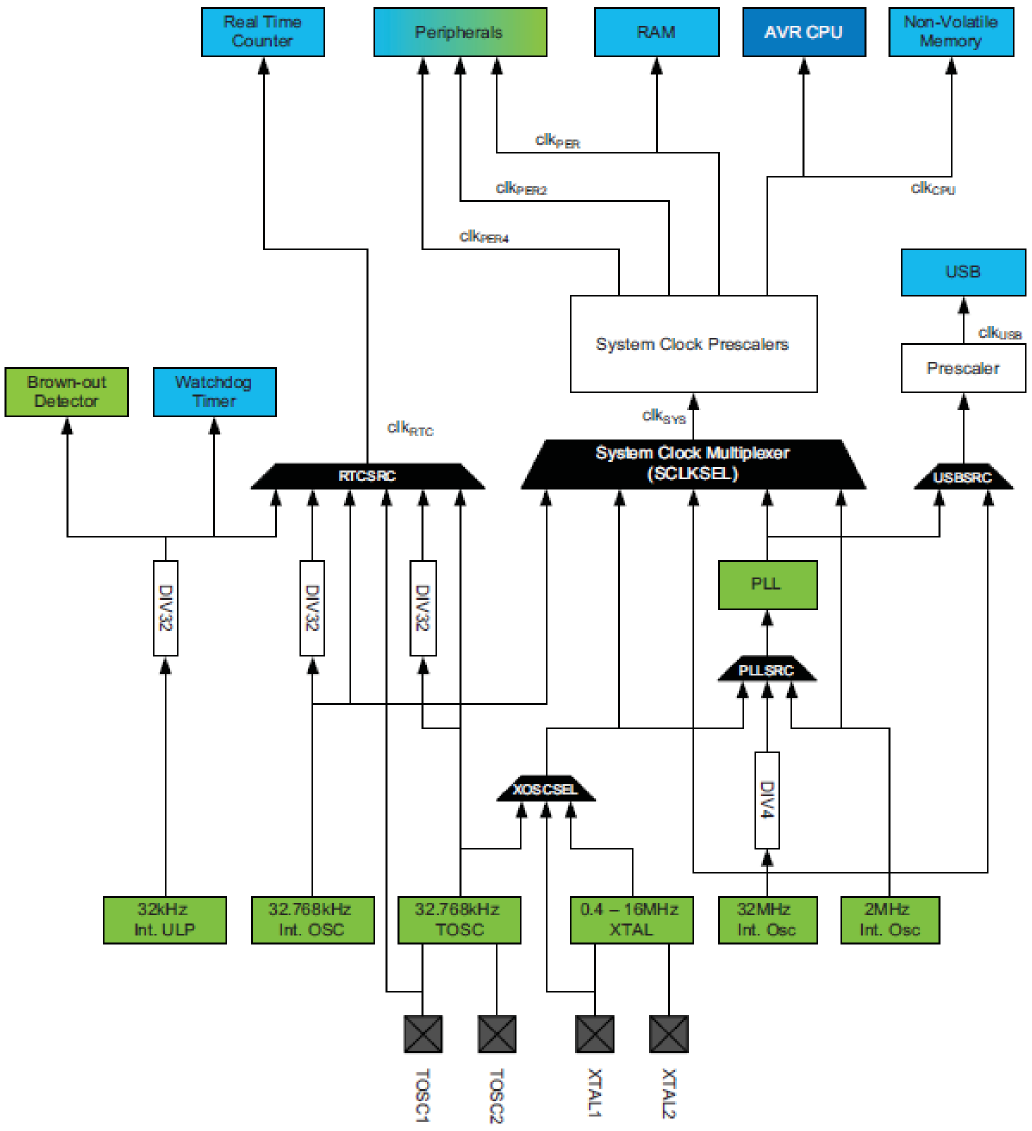


Figure 7. ATxmega16C4 Clock System, Clock Sources, and Clock Distribution [10]

The other connections to the microcontroller are covered in the upcoming chapters.

2.2.3 Communication Interfaces

Two interfaces were chosen for communication with external devices – USB 2.0 and UART (1.4).

As one of the criteria was the device to be as small as possible a micro USB connector was a logical choice. After making some research about USB hardware design [11] a schematic seen in Figure 8 was created. R1 and R3 are termination resistors for USB impedance matching. The USB specification requires a USB device to have a 45 Ω single-ended or 90 Ω differential impedance. This impedance includes the impedance of the USB transceiver. As the impedance of the USB transceiver is lower than the requirement of the USB specification, 22 Ω resistors (R1, R3) are used to bring the device impedance within the required specification limits.

USB allows the application to have an open connection to the external environment, which could expose the entire system to ESD. Even though Atmel AVR XMEGA Microcontroller embed on-chip ESD protection [12] , it is recommended to increase ESD protection on the USB D+ and D- lines using dedicated transient suppressors, Z1 and Z2 are specifically used for that purpose [13] . These devices protect equipment to IEC61000-4-2, Level 4 (±8 kV Contact / ±15 kV Air Discharge) ESD specifications and are especially targeted to protect USB and similar data communication lines.

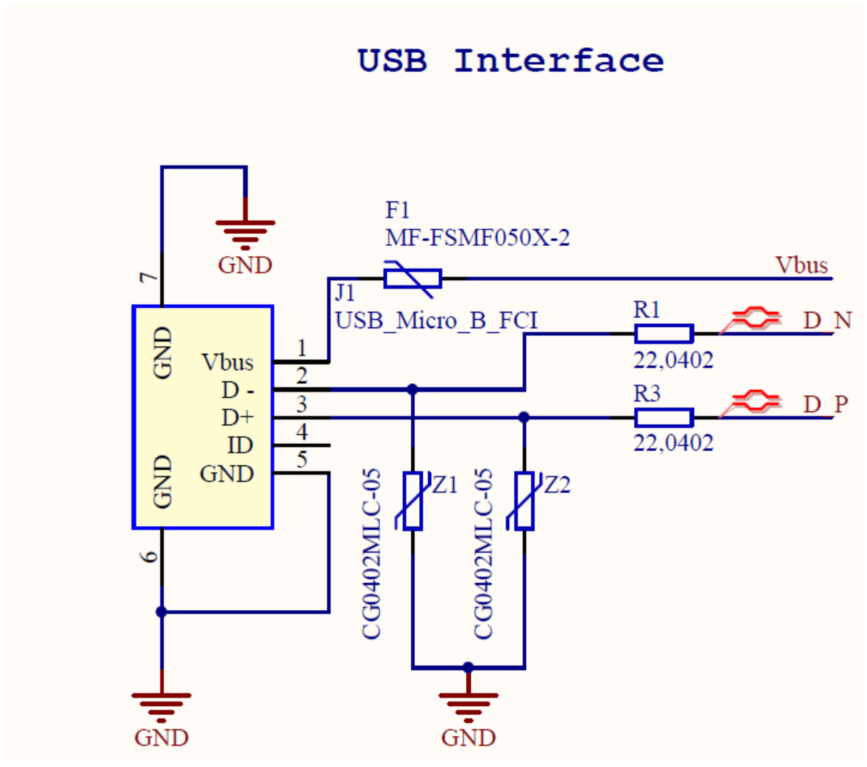


Figure 8. USB Interface Schematic Developed for the Device

According to the USB specifications mentioned in 2.12.2 the maximum current is 500mA. Therefore to protect circuitry from both sides a 500mA holding current PTC resettable fuse was used – F1 [14] . The D_N and D_P lines are defined as differential pair and are connected to dedicated MCU pins (Figure 6).

A 2.54mm pin header was chosen for UART connections (Figure 9). This kind of solution is widely used in the industry. Resistors R12 and R13 are protecting the microcontroller I/O pins from excessive current if short should take place. Pin 5 is for external power connection.

An extra pin was added to the connector for starting the MCU in USB boot loader mode. This USB boot loader [15] allows performing in-system programming (ISP) from a USB host controller without removing the part from the system, and without any external programming interface other than the USB connector. Once the part is configured with USB DFU boot loader application, the boot loader execution can be forced at power-on by connecting a specific pin to ground. By default for ATxmega16C4 this pin is PC3.

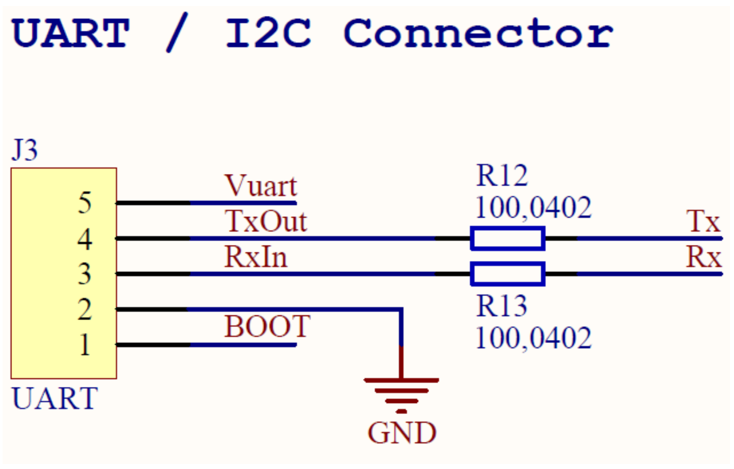


Figure 9. UART Connector Schematic Diagram

Later in the process (when the physical device was ready) it was discovered that the UART Tx and Rx pins were connected to wrong pins, because during the hardware design process outdated MCU datasheet was used. The outdated datasheet did not affect other MCU connections or functionality.

2.2.4 Inputs / Outputs and Configuration

The only external option for programming the ATxmega16C4 MCU (if the USB DFU boot loader is not flashed in) is the Program and Debug Interface or simply PDI. It is an Atmel 2-wire proprietary interface. The pins required for programming via the PDI interface are detailed in Table 2.

Table 2. Pins Required for Program and Debug Interface [15]

PDI Signal Name	Signal description	Direction from programmer	Pin name on XMEGA device
PDI_CLK	PDI Clock Signal	Output	RESET
PDI_DATA	PDI Data Signal (bi-directional)	Bi-directional	TEST
GROUND (0V)	Target / Programmer Signal GROUND	Passive	GND
Vcc	Target / Programmer Vcc Supply	Passive	VCC

The PDI connector was created according to XMEGA application note [16] seen in Figure 10.

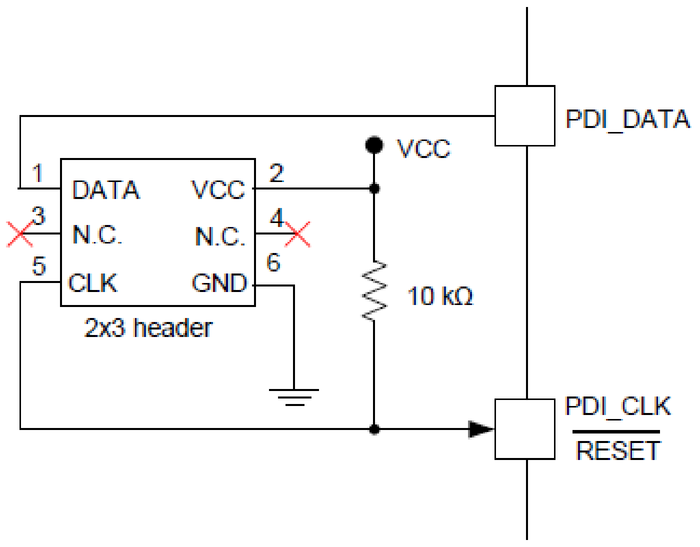


Figure 10. Example Schematic for PDI Interface [16]

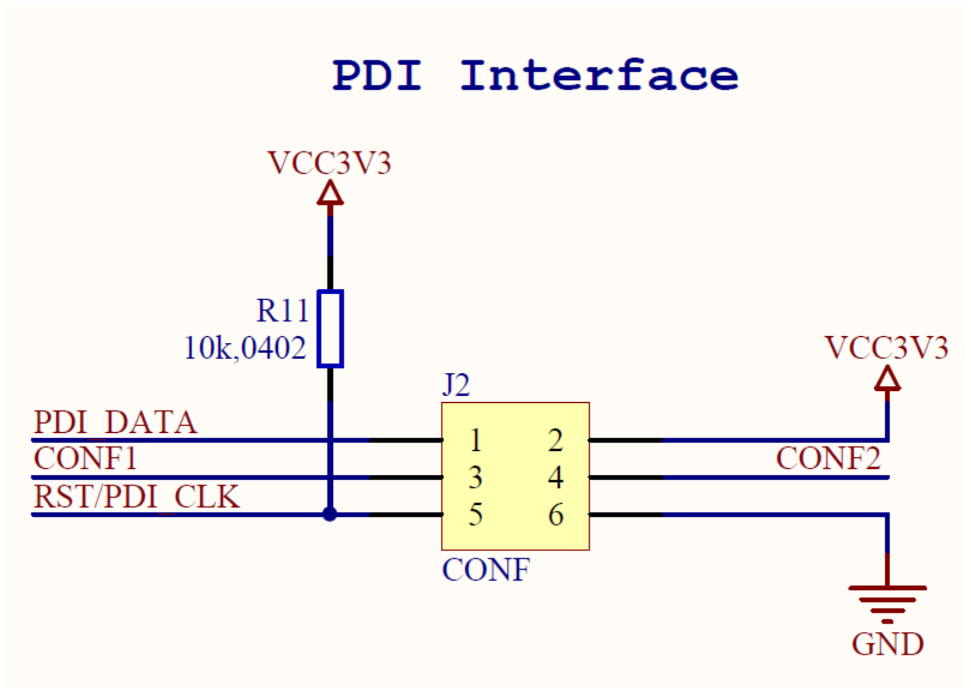


Figure 11. PDI Interface and User Configuration Pins

As seen in Figure 11 two pins are unused in the example schematic. These “unused” pins were connected to MCU in purpose to cleverly create user configuration pins. One of the requirements in the specification was the device to be as small as possible and reusing the pins from 2 x 3 header reduces the needed board area. By using a jumper over pins 2 - 4, 4 - 6 or 1-3, 3-5 it is possible to create different combinations.

Three LEDs were added to schematic to indicate different events to the user - Figure 12. At this point it was clear that the supply voltage is 3.3V, which also means that the high level for MCU outputs is 3.3V. The supply voltage selection process is described in chapter 2.2.5 which talks about power. Microcontroller pins PA0, PA1 and PA2 were chosen for the LED connections (Figure 6), because they did not have any other special functions that would be needed for the system.

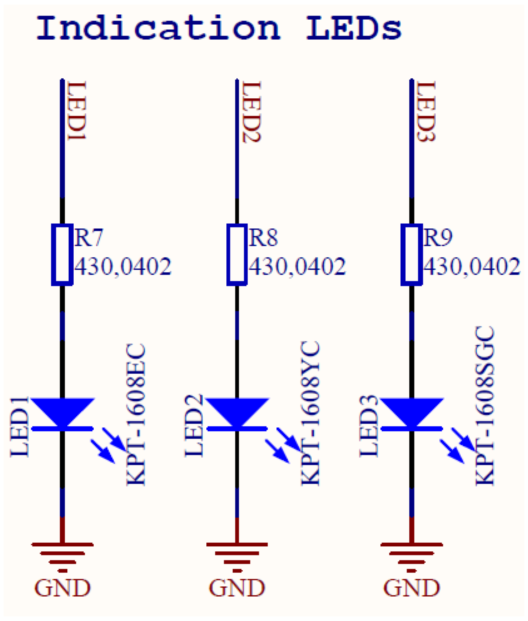


Figure 12. LED Indicators Circuit Diagram

A 0603 package red, green and yellow color LEDs were picked for the task. The typical forward voltage for these LEDs varies from 2.0 - 2.2V.

Applying Kirchhoff's voltage law and Ohm's law one can simply calculate the value for needed resistors. According to the Kirchhoff's voltage law:

$$V_{CC} = V_R + V_{LED} \tag{1}$$

Therefore the voltage which will remain on the resistor can be found from the following equation:

$$V_R = V_{CC} - V_{LED} \tag{2}$$

Where:

V_{CC} – Supply Voltage

V_{LED} – Voltage Drop on the LED

V_R – Voltage Drop on the Resistor

As $I_{CC} = I_R = I_{LED}$ the Ohm's law can be written in the following form:

$$I_R = I_{LED} = \frac{V_R}{R} \tag{3}$$

Where:

I_R – Current through the Resistor

I_{LED} – Current through the LED

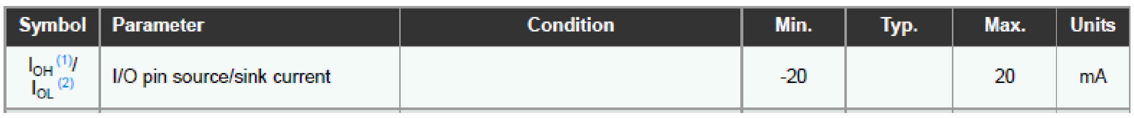
V_R – Voltage Drop on the Resistor

R – Resistor resistance

Combining the equations (2) and (3):

$$R = \frac{V_{CC} - V_{LED}}{I_{LED}} \tag{4}$$

LED voltages vary from 2.0 - 2.2V. As this is pretty small difference it can be considered that V_{LED} = 2.0V. It is known that the V_{CC} = 3.3V and the forward current for LEDs is a schematic designer choice. The LEDs datasheets only specify the maximum DC Forward Current which is for different colors in the range of 25mA - 30mA. Very often a 10mA current is chosen for LEDs, but based upon authors experience and taking account the fact that it is a good idea to minimize the current consumption a 3mA current was chosen. MCU I/O pin source/sink current [10] has to be taken account also, which is -20mA to +20mA (Figure 13) in our case, therefore exceeding the needs.



Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
I_{OH} (1)/ I_{OL} (2)	I/O pin source/sink current		-20		20	mA

Figure 13. MCU I/O pin source/sink current capability

After knowing all values for every variable resistance is calculated as follows.

$$R = \frac{3.3 - 2.0}{0.003} = \frac{1.3}{0.003} = 433.33\Omega \qquad (5)$$

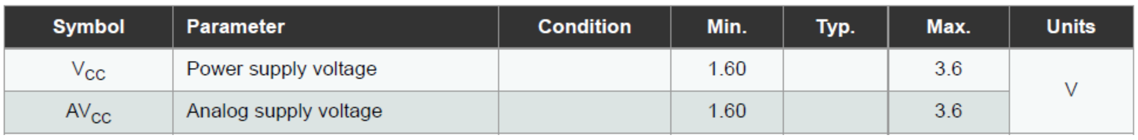
The closest nominal resistor value is 430Ω for the calculated resistance. Although there are slight differences between the red, green and yellow LEDs a resistor with a same value was chosen for all of them to minimize the components base.

It was taken into account that if after testing the LED brightness will be found too dim the resistance value can be changed to a smaller one.

2.2.5 Power

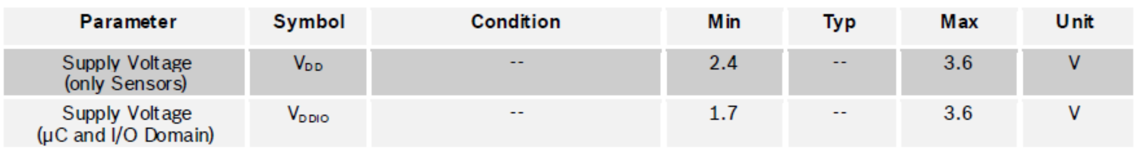
In this chapter the power electronics part of the schematic will be covered. The first question regarding power was the supply voltage selection.

There are only 2 critical parts which dictate the voltage selection – Atxmega16C4 and BNO055. To choose supply voltage the datasheets [9] [10] of these 2 components had to be taken into account. Figure 14 and Figure 15 are extracts of the operating supply voltage ranges from ICs datasheets.



Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
V_{CC}	Power supply voltage		1.60		3.6	V
AV_{CC}	Analog supply voltage		1.60		3.6	

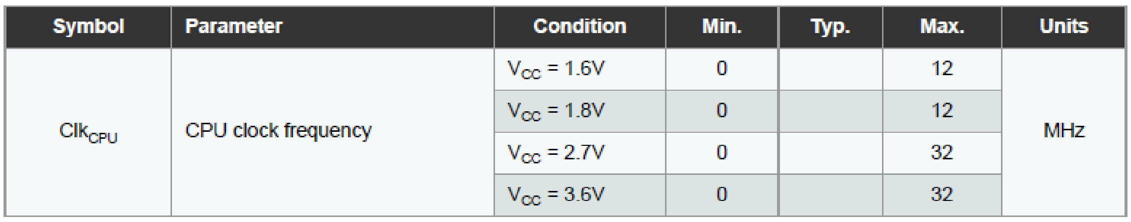
Figure 14. ATxmega16C4 Operating Voltage Conditions



Parameter	Symbol	Condition	Min	Typ	Max	Unit
Supply Voltage (only Sensors)	V_{DD}	--	2.4	--	3.6	V
Supply Voltage (µC and I/O Domain)	V_{DDIO}	--	1.7	--	3.6	V

Figure 15. BNO055 Operating Voltage Conditions

As 2 constraints for the device were inexpensiveness and a small size - 2.1 - it made sense to optimize the voltage to 1 value for both devices. Later means that only one voltage source is needed, translating into 1 component and less room on the board. It can be seen from the Tables (Figure 14 and Figure 15) that the maximum operating voltage for both devices is the same – 3.6V, but it is wise to use lower value if possible. The minimum possible value would be 2.4V. A 3.3V level is very commonly used and as there are many devices also communicating with UART with this level it was logical to use this voltage value for supplying the MCU and IMU. Another condition taken from the MCU datasheet was the clock frequency which can have maximum value, only if the supply voltage is 2.7V or higher (Figure 16).



Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
Clk_{CPU}	CPU clock frequency	V_{CC} = 1.6V	0		12	MHz
		V_{CC} = 1.8V	0		12	
		V_{CC} = 2.7V	0		32	
		V_{CC} = 3.6V	0		32	

Figure 16. Relationship between MCU Operating Voltage and Clock Frequency

Before choosing a voltage regulator type and voltage regulator itself, the whole current consumption of the schematic had to be calculated.

Table 3. Summary of Current Consumption

Device	Maximum Current Consumption [mA]
Atmel ATXmega16C4	12.0
Bosch BNO055	12.3
LED Indicator - Yellow	10.0
LED Indicator - Red	10.0
LED Indicator - Green	10.0
Power LED - Green	10.0
Overall Sum	**64.3**

Table 3 contains summary of maximum current consumption of the components. The information for ATXmega16C4 and BNO055 are taken from the datasheets [9] [10] and although the calculated current consumption for LEDs was 3mA, the possibility of increasing the LED currents was considered (in case they are not bright enough). When it is consider that the LEDs are consuming 3mA then the result is 36.3mA for the overall sum. But at this point the highest possible realistic current consumption had to be considered, therefore LED currents were theoretically increased up to 10mA, increasing the total current consumption up to 64.3mA.

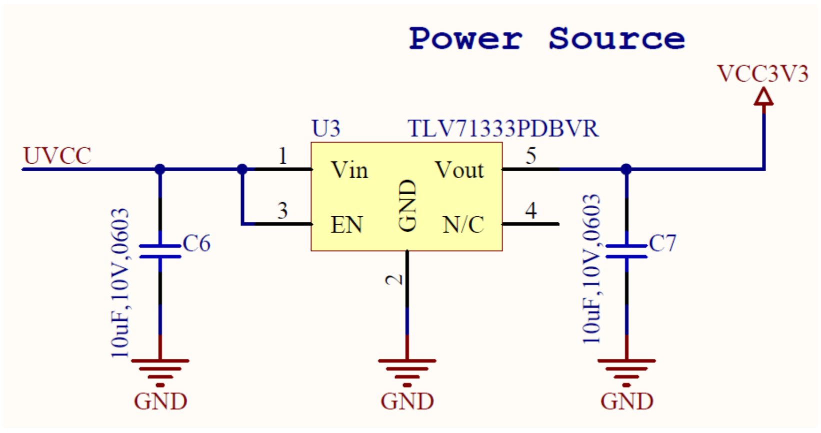


Figure 17. LDO Circuit Diagram

Linear regulators are recommended when the voltage of the source powering the regulator is near the regulator's output voltage and the circuit being powered does not care about the efficiency of the regulator. In our case these conditions were met. Therefore LDO seemed to be a good option. Other criteria that were considered were: a small package; possible to solder by hand; cheap price. After search and filtering from Farnell by the latter criteria, Texas Instruments TLV71333 [17] was chosen for the power source.

Here are some key features of the TLV71333:

- Input Voltage Range: 1.4 V to 5.5 V
- Maximum Output Current 150mA
- 5-Pin SOT-23
- Accuracy: 1%
- High PSRR: 65 dB at 1 kHz

It can be seen from the specifications that the input voltage falls into the wanted range and the output current capability exceeds the maximum needed current. In the worst case there will be still 150 - 64.3 = 85.7mA headroom. It is always a good idea to have a reserve in your system.

LDO circuit diagram is given in Figure 17. C6 and C7 are 10uF input and output capacitors needed mostly for LDO stable operation.

The last thing that had to be solved for the schematic was the presence of 2 possible supply points – USB and external power pin. It is not a good practice to just connect power sources in parallel, this can create a number of different problems – reverse current flow into power supply output; unevenly distributed current consumption; potential differences between 2 load points etc.

Input power OR-ing is a common practice to solve these issues. The simplest way is to use diodes which cathodes are tied together. OR-ing circuit created for the device can be seen in Figure 18.

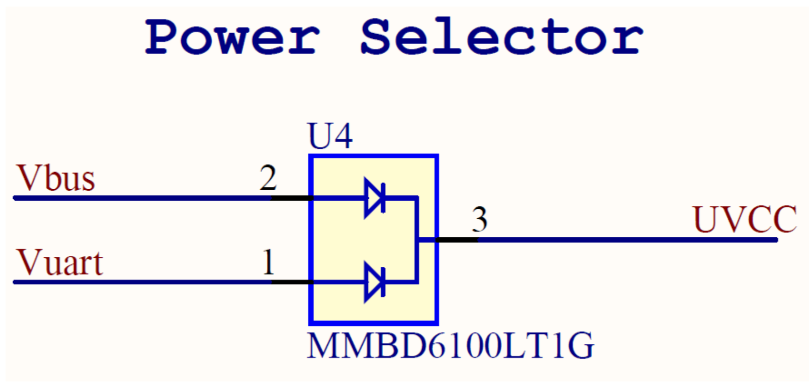


Figure 18. Automatic Power Selector for the Device

When using OR-ing:

- The power supplies are isolated from each other
- There is no backwards current flow
- Only 1 supply source will be selected at once

In real life 2 power sources do not have exactly the same voltage value – there will be a difference although it may be very small. This means that one of the diodes will be reverse biased at all times and the other one forward biased.

A regular diode dual package was chosen for the job. It means that the max voltage drop can be up to 1.1V. This means that the LDO input voltage will be as low as 3.9V. According to the LDO datasheet the input voltage has to be at least 0.5V higher than the nominal output value, in our case 3.3V. 3.9 - 3.3 = 0.6V which is still acceptable. The positive side is that there will be less power dissipated as heat on the LDO.

2.3 Layout Design

This section describes the development process and the results of PCB layout design. The PCB was designed using Altium Designer Software. Mentioned CAD software has all the needed tools to create circuit boards and it was a logical choice to make the schematic and board design using the same tool.

The PCB design process was guided by the same 3 main aspects presented in section 2.1:

- The device should be as cheap as possible

- The device should be as small as possible
- The device should consume reasonable amount of power

One more condition which was added to the list was the possibility to mount the developed module to some other device or system (using screws).

The first decision that had to be made when starting the PCB design process was the layer count. 4-layer PCB prices can be 4-5 times higher than the 2-layer ones. Keeping in mind that the module under development had to be cheap and after evaluating the complexity of the schematic, it was clear that 2-layer PCB is enough and the right choice for this design.

There was no reason to create 2-sided PCB therefore the components were placed on one side of the board. The board was developed around the BNO055 IC just like the schematic. The guidelines for creating circuit board for Bosch BNO055 are gathered into one document - Handling, soldering & mounting instructions [18] . When creating the printed circuit board these instructions were taken into account.

The design rules have to follow the specific manufacturers' capabilities and guidelines. In some cases it can be vice versa – if something very precise is needed then the designer has to find a manufacturer who is capable of producing this. In our case we chose Kamitra as the local manufacturer (in Estonia), but at the same time keeping in mind the possibility of producing the PCBs in China. There are many PCB manufacturers nowadays, located in China, who produce good quality circuit boards for a cheap price.

After placing all the components and optimizing their position and orientation it was clear that the board measurements can be reduced down to 35 x 25mm. 4 screw wholes for M2.5 standard were added to the corners of the board.

The selecting process of the components was already described in the schematic design section 2.2. This was derived mostly by 2 aspects – they needed to be small but also possible to solder by hand.

The end result can be seen in Figure 19, Figure 20, which represent the top and bottom view of the designed circuit board.

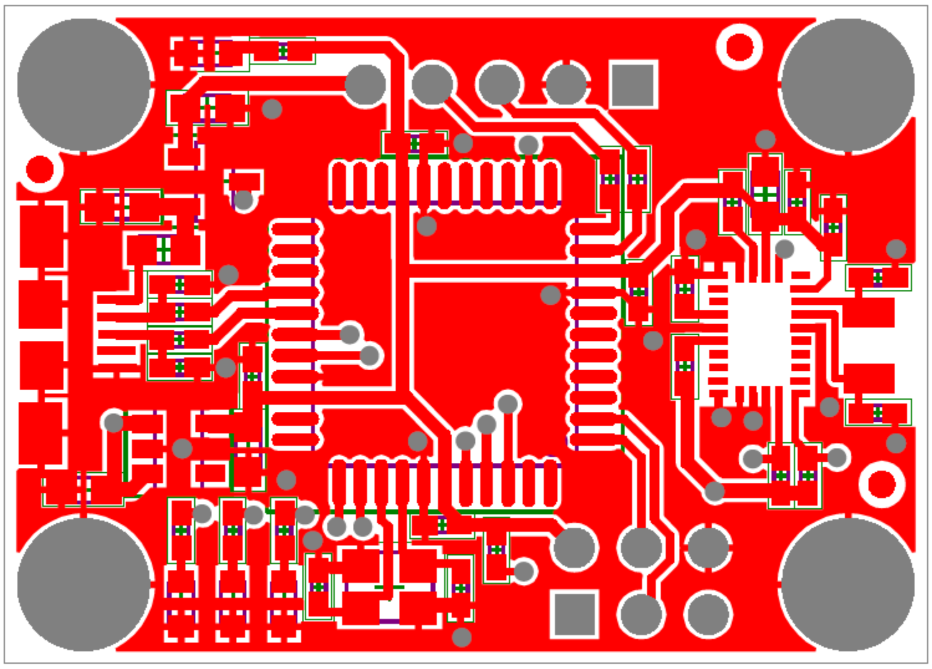


Figure 19. PCB Top Layer

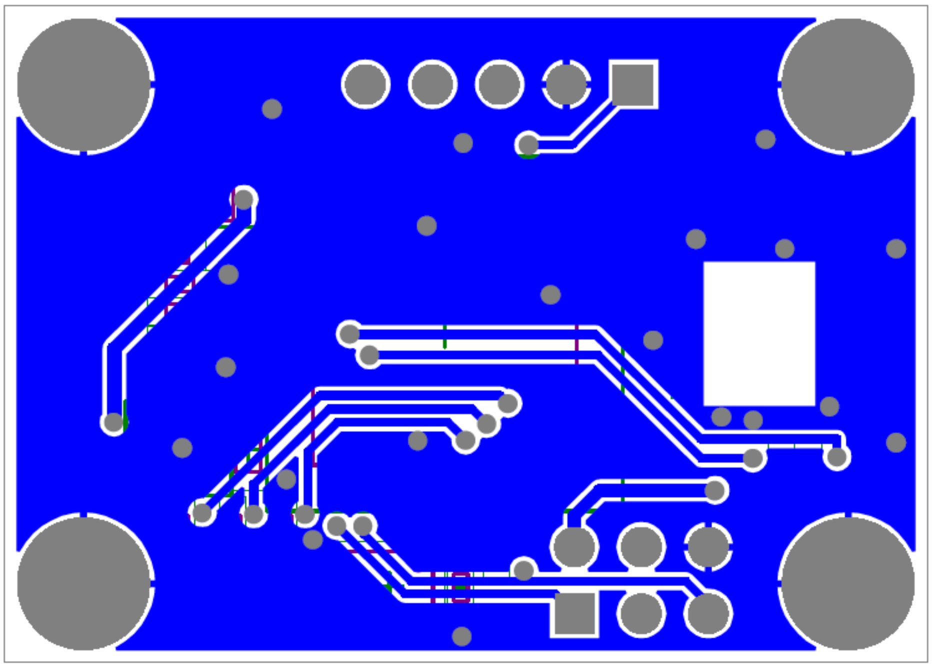


Figure 20. PCB Bottom Layer

2.4 Hardware Development Results

As a result of hardware development a fully functional platform was created according to the specifications listed in sections 1.4 and 2.1. In the core of the hardware design is the Bosch BNO055 9DoF sensor IC which is able to output pre-calculated orientation data. Remaining schematic supports this main functionality. PCB layout was created and a physical board was produced according to the design. The end result can be seen in Figure 21.

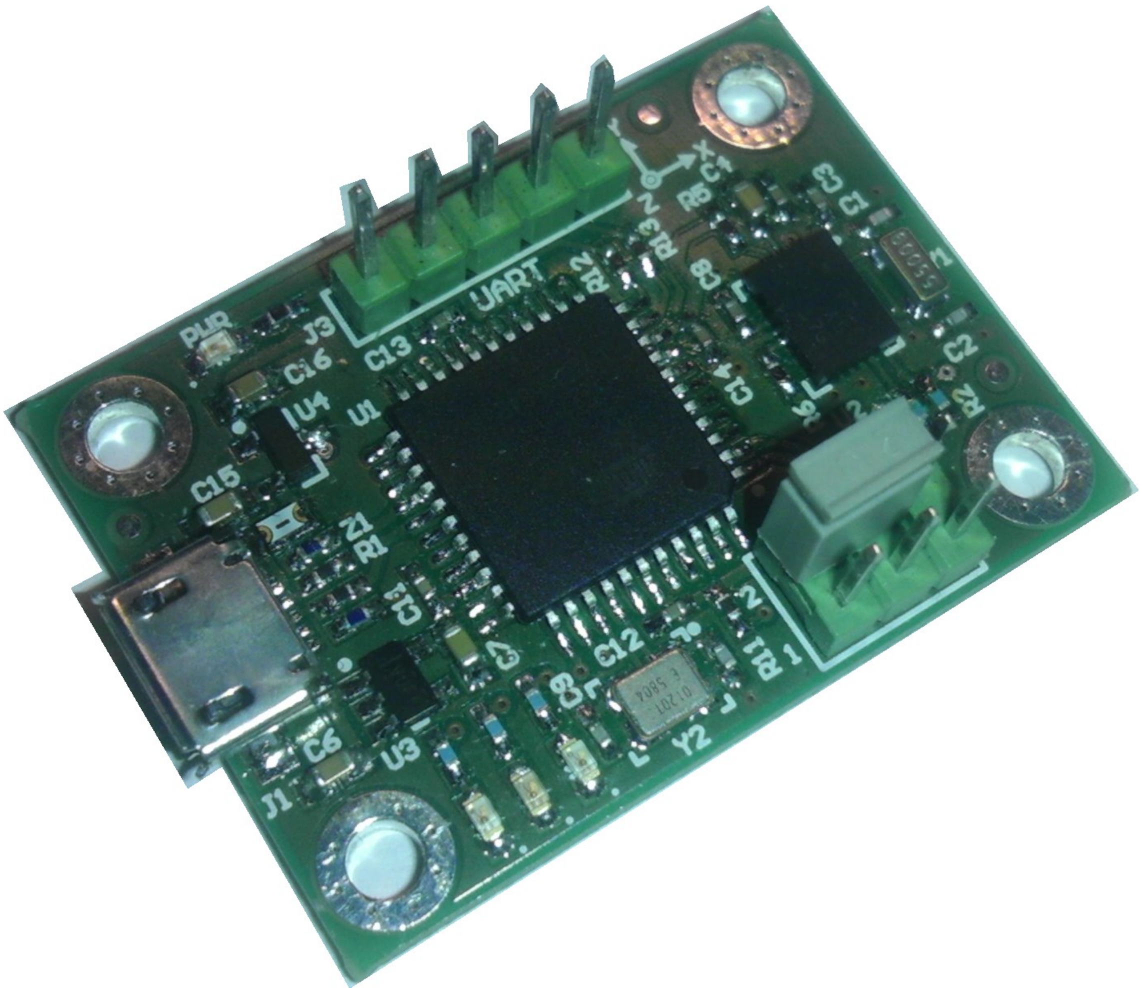


Figure 21. Hardware Development Result – Physical Device

3 Software Design

This section of the thesis will give an overview of the software part of the project. The software is divided into 2 main parts – firmware and demo software. The firmware is the part of the software that is written and running on the microcontroller (ATXmega16C4 in our case). The demo software is demonstrating how a 3D virtual object can be manipulated based upon the data which the module outputs.

The firmware is an essential part of the project and makes the device function as designed. The demo software on the other hand is demonstrating that the module itself is giving out usable and correct information about the orientation in 3D space.

3.1 Firmware Design

The firmware was designed according to the specifications listed in section 1.4 and was written in C programming language using AtmelStudio 6.2 as the programming environment. Below are the most important components of the firmware:

- Main Function
- Bosch BNO055 Driver
- USB (LUFA) Driver
- I2C / TWI (LUFA) Driver
- GPIO Controller / Driver
- Timer / Counter Driver

3.1.1 The Main Function

Let's start by describing the firmware in a top-down manner. It is well-known that every C program has a primary (main) function. It is the core of every C program and a common custom. Below is the flowchart diagram representing the firmware main code functionality (Figure 22).

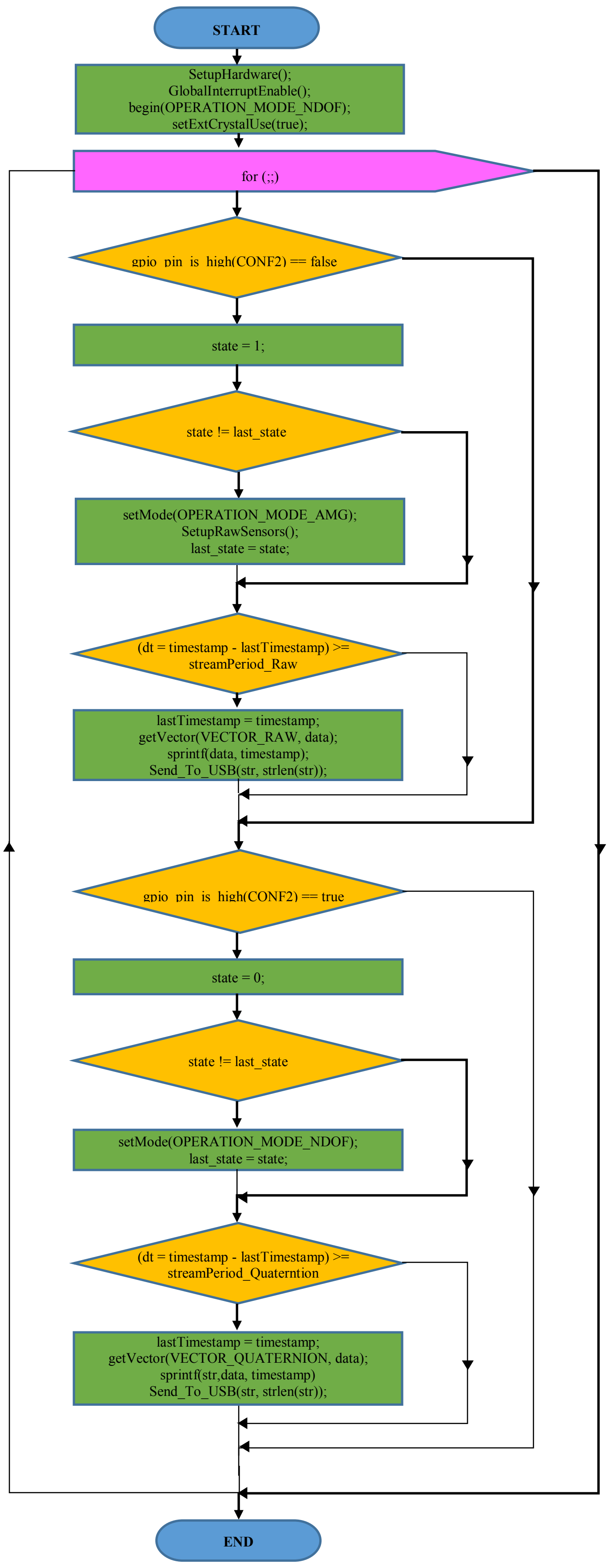

START
SetupHardware();
GlobalInterruptEnable();
begin(OPERATION_MODE_NDOF);
setExtCrystalUse(true);
for (;;)
gpio_pin_is_high(CONF2) == false
state = 1;
state != last_state
setMode(OPERATION_MODE_AMG);
SetupRawSensors();
last_state = state;
(dt = timestamp - lastTimestamp) >= streamPeriod_Raw
lastTimestamp = timestamp;
getVector(VECTOR_RAW, data);
sprintf(data, timestamp);
Send_To_USB(str, strlen(str));
gpio_pin_is_high(CONF2) == true
state = 0;
state != last_state
setMode(OPERATION_MODE_NDOF);
last_state = state;
(dt = timestamp - lastTimestamp) >= streamPeriod_Quaterntion
lastTimestamp = timestamp;
getVector(VECTOR_QUATERNION, data);
sprintf(str,data, timestamp)
Send_To_USB(str, strlen(str));
END

Figure 22. Flowchart of the Main Function

The whole idea of the main function is to read either raw sensor data or pre-calculated quaternion data from Bosch BNO055 and send the data with timestamp to virtual serial port / USB host device.

The main function's logic is quite simple. The code starts by configuring ATXmega16C4 hardware and initializing the Bosch BNO055 sensor. Then an endless FOR-loop is entered inside which there are 2 possible routes. The path which will be selected is determined by CONF2 pin, which can be high or low. The CONF2 value is configured by the user shorting appropriate pins with a jumper (J2 pins 2-4 or 4-6). After entering one of the possible IF-statements, the code checks if this is the first time for the code to enter (state != last_state) and if it's true the BNO055 is reconfigured to output either quaternion or raw sensor data. Next the streaming period is checked and following code part is entered only if specific amount of time is passed. When period is true `getVector()` reads either raw sensors data or pre-calculated quaternion data from Bosch BNO055 and after that the data is formatted into string including timestamp and sent over USB to host device - `Send_To_USB()`. The latter continues until the device is powered off.

3.1.2 Bosch BNO055 Driver

There is an official driver available for BNO055 on Bosch page [25] . At least at the time of developing the project, the official driver code was overly complicated, hard to understand and not easily usable. Therefore other possible solutions were considered. This led to Adafruit BNO055 driver, which is written in C++ [26] .

These 2 possible drivers were first tested on Arduino 9 Axes Motion Shield, which also uses BNO055 as an onboard sensor [28] There are examples from Arduino Corporation which are using the Bosch official driver and also Adafruit has developed examples which can be run on Arduino using 9 Axes Motion Shield. Testing on Arduino and Motion Shield gave us confidence about the orientation accuracy and usability, but on the other hand it was also noticed that the libraries do not support configuring and reading data from separate sensors – accelerometer, gyroscope and magnetometer. Moreover the functionality related to raw sensor data is partially or fully missing from these libraries. Therefore to test reading the raw sensor information one of the Adafruit's examples was modified by the author. After knowing the drawbacks of existing libraries it was clear that a custom driver has to be written.

The BNO055 driver that was developed using C programming language like the rest of the code. It contains register definitions, configuration settings, BNO055 operating functions and I2C / TWI higher level functions. Below is a list of functions which have been implemented in the driver (Figure 23).

```
uint8_t begin (bno055_opmode_t mode);
void setMode (bno055_opmode_t mode);
void setExtCrystalUse (uint8_t usextal);
int8_t getTemp(void);
void getRawSensors(int16_t * buffer);
void getEuler(int16_t * buffer);
void getVector(bno055_vector_type_t vector_type, int16_t * buffer);
uint8_t setAccMode(bno055_acc_opr_mode_t acc_mode);
uint8_t setAccConf(bno055_acc_range_t range, bno055_acc_bw_t bw,
bno055_acc_opr_mode_t acc_mode);
uint8_t setGyroConf(bno055_gyro_range_t range ,bno055_gyro_bw_t bw
,bno055_gyro_opr_mode_t gyro_mode);
uint8_t setMagConf(bno055_mag_rate_t rate, bno055_mag_opr_mode_t mag_opr_mode,
bno055_mag_pwr_mode_t mag_pwr_mode);
uint8_t setUnits(bno055_unit_t ori, bno055_unit_t temp_unit, bno055_unit_t
eul_unit, bno055_unit_t gyr_unit, bno055_unit_t acc_unit);
void switchPage(bno055_page_t page);
uint8_t BNO055_read_byte(uint8_t reg);
uint8_t BNO055_write_byte(bno055_reg_t reg, uint8_t value);
uint8_t BNO055_read_bytes(bno055_reg_t reg, uint8_t * packet, uint8_t len);
```

Figure 23. Functions Developed for BNO055 Driver

Many of these functions are used in the main code. Some functions incorporate the other functions that are listed above – i.e. the functions related to I2C / TWI communication.

3.1.3 USB CDC (LUFA) Driver

The firmware was built upon a USB virtual serial example. The virtual serial (also referred to as virtual COM port or virtual RS-232) belongs under USB CDC [19] [20] identifier. The functionality of USB devices is defined by class codes, communicated to the USB host to affect the loading of suitable software driver modules for each connected device. This provides for adaptability and device independence of the host to support new devices from different manufacturers. The virtual serial actually means that devices from both sides (the module and PC) emulate more or less RS-232 standard compliant communication. The virtual serial [21] communication is very often used and it can make developing applications easier than using standard USB functions. The latter may be quite difficult and could mean going through a lot of documentation, especially from the USB host side. This eventually leads to longer development time and higher cost of the whole system.

As it is not wise to "reinvent the wheel" it was a good idea to see if there is already a library or driver existing for implementing virtual serial communication on ATXmega16C4. After doing the research, there were basically two good options – Atmel ASF USB or LUFA USB driver [22] , both supporting the XMEGA architecture. Going deeper and analyzing the code and ease of use a decision was made to employ the LUFA driver.

The virtual serial code for ATXmega16C4 was mostly used according to the example and the only thing to point out here would be the *Send_To_USB(char *buf, char len)* function, which was organized into separate function by the author.

3.1.4 I2C / TWI (LUFA) Driver

There were 2 options for connecting BNO055 to ATXmega16C4 – UART and I2C communication interface. So the main question was which one would be faster. Therefore maximum possible data reading rate for raw (9-axis data) was calculated [39] [40] .

According to the BNO055 datasheet [9] the maximum I2C clock frequency is 400kHz. This can be used reversibly with 400bits / s. There are 9-axis, for every axis there is and LSB and MSB register. In the beginning of I2C communication slave 7-bits address is sent. Also start, stop and acknowledgement bits have to be taken into account. Altogether data bits with overhead sums up to 182 bits per reading cycle.

I2C:
$$f_S = \frac{R}{N} = \frac{400 bit/s}{182 bits} = 2198 Hz \tag{6}$$

UART:
$$f_S = \frac{R}{N} = \frac{115200 bit/s}{192 bits} = 600 Hz \tag{7}$$

The maximum bit rate for UART is 115200 bit / s. In case of calculating the real output data rate from the sensor, a command sequence, read response and read acknowledgment / failure bits have to be taken into account. This all results in 192 bits per every reading cycle. Equations 6 and 7 were used to calculate the data output rate.

Calculation proved that I2C interface would be the faster option and meeting the 1kHz requirement for reading raw data therefore it was chosen for communication between the MCU and BNO055 IC.

After doing the research about available drivers which support XMEGA architecture, there were again two good options left – to use Atmel ASF official TWI driver [23] or LUFA TWI driver [24] . Looking deeper into these alternatives it was more convincing to use LUFA TWI driver as it was understandable and easily usable. The LUFA library contains pre-written high level functions for initializing, reading and writing TWI packets. These high level functions were incorporated into the BNO055 driver.

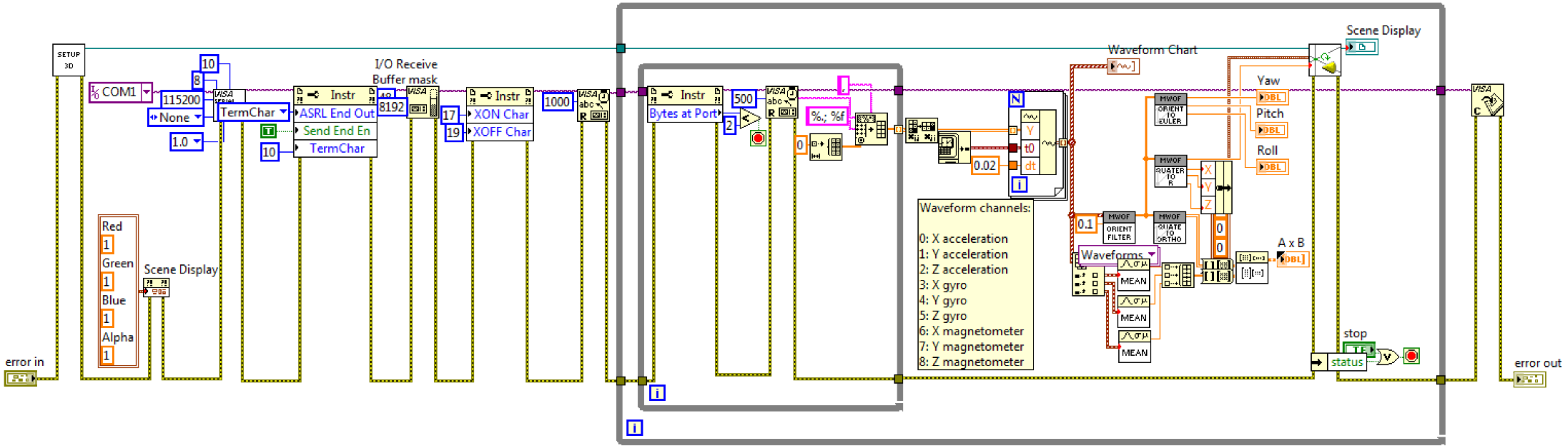
3.1.5 I2C Clock Stretching Issue

During firmware development an I2C clock stretching issue was discovered which limits the raw data communication bandwidth down to 150Hz maximum instead 1 kHz. Thus a decision was made to lower the frequency down to the same rate as for quaternion output - 100Hz. An I2C slave is allowed to hold down the clock if it needs to reduce the bus speed [27] . This means that the raw data rate was not able to meet the specifications.

3.2 Demo Software

This section will give an overview about the demo software. The demo software was developed using LabVIEW (short for Laboratory Virtual Instrument Engineering Workbench), which is a system-design platform and development environment for a visual programming language from National Instruments [30] . LabVIEW is commonly used for data acquisition, instrument control, and industrial automation, but can be used for other tasks. LabVIEW ties the creation of user interfaces (called front panels) into the development cycle. LabVIEW programs/subroutines are called virtual instruments (VIs) or sometimes SubVIs. Each VI has three components: a block diagram, a front panel and a connector panel. The last is used to represent the VI in the block diagrams of other, calling VIs. The front panel is built using controls and indicators.

The demo software is based on the open source LabVIEW project from Roger Isaksson [29] , which is, in turn adaption of Sebastian Madgwick's open source C# AHRS library. In 2009 Sebastian Madgwick developed an IMU and AHRS sensor fusion algorithm as part of his Ph.D research at the University of Bristol [37] . The original code by Roger Isaksson is presented in Figure 24. The code works in the following manner. First the 9-axis (3-axis acceleration, 3-axis gyroscope and 3-axis of magnetometer) data is read in from COM port, then the data enters VI called ORIENT FILTER, which calculatesquaternions from the raw sensor data. After that the quaternion data is fed into 3 subVIs – ORIENT TO EULER, QUATERNION TO R and QUATERNION TO ORTHO.



Waveform channels:
0: X acceleration
1: Y acceleration
2: Z acceleration
3: X gyro
4: Y gyro
5: Z gyro
6: X magnetometer
7: Y magnetometer
8: Z magnetometer

Figure 24. LabVIEW Block Diagram of Roger Isaksson's Orientation VI

The ORIENT FILTER VI is the one that actually incorporates the Sebastian Madgwick's algorithms and is used to calculate quaternions from raw sensors information. ORIENT TO EULER calculates Euler angles (yaw, pitch and roll) from quaternions, these angles are displayed to the user on the front panel of the virtual instrument. QUATERNION TO R takes quaternions as input and outputs x,y,z coordinates and an angle for drawing a 3D object. The x,y,z coordinates and an angle are connected to Set Rotation VI (as inputs) which uses axis–angle representation to manipulate 3D scene object in LabVIEW and performs an absolute rotation from the object's initial position (Figure 25).

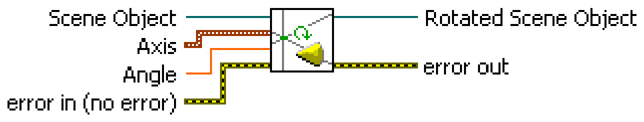


Figure 25. Set Rotation VI

Finally we have the QUATERNION TO ORTHO VI, which calculates orthogonal quaternion matrix from quaternion data. This VI has no importance in this context and therefore will not be considered.

It is quite easy to convert quaternion data to axis-angle representation, using equations (8), (9), (10) and (11) which are represented below [31] [32] .

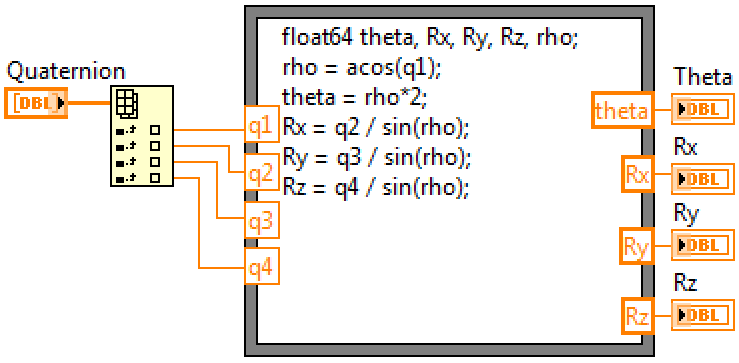
$$q_w = q_1 = \cos\left(\frac{\theta}{2}\right) \Leftrightarrow \frac{\theta}{2} = \cos^{-1} q_w \tag{8}$$

$$q_x = q_2 = x \cdot \sin\left(\frac{\theta}{2}\right) \Leftrightarrow x = \frac{q_x}{\sin\left(\frac{\theta}{2}\right)} \tag{9}$$

$$q_y = q_3 = y \cdot \sin\left(\frac{\theta}{2}\right) \Leftrightarrow y = \frac{q_y}{\sin\left(\frac{\theta}{2}\right)} \tag{10}$$

$$q_z = q_4 = z \cdot \sin\left(\frac{\theta}{2}\right) \Leftrightarrow z = \frac{q_z}{\sin\left(\frac{\theta}{2}\right)} \tag{11}$$

The QUATERNION TO R is using exactly these equations to convert quaternions to axis-angle representation. The internal block diagram is present in Figure 26.



```
float64 theta, Rx, Ry, Rz, rho;
rho = acos(q1);
theta = rho*2;
Rx = q2 / sin(rho);
Ry = q3 / sin(rho);
Rz = q4 / sin(rho);
```

Figure 26. Internal Block Diagram of Quaternion To R

The $\frac{\theta}{2}$ is replaced with ρ for code simplification (in mathematical form $\rho = \frac{\theta}{2}$). Below is the mathematical equivalents (equations (12), (13), (14), (15) and (16)) for the code seen in Figure 26.

$$\rho = \cos^{-1} q_1 \tag{12}$$

$$\theta = 2 \cdot \rho \tag{13}$$

$$x = \frac{q_2}{\sin \rho} \tag{14}$$

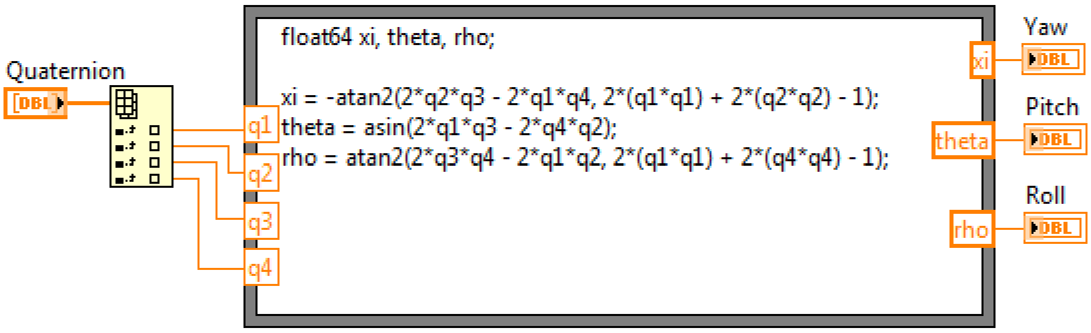
$$y = \frac{q_3}{\sin \rho} \tag{15}$$

$$z = \frac{q_4}{\sin \rho} \tag{16}$$

ORIENT TO EULER converts quaternion data into Euler angles (yaw, pitch, roll) using following equations [33] [37] :

$$\begin{bmatrix} \psi \\ \theta \\ \phi \end{bmatrix} = \begin{bmatrix} -arctan\dfrac{2(q_2q_3 - q_1q_4)}{2(q_1^2 + q_2^2) - 1} \\ arcsin\big(2(q_1q_3 - q_4q_2)\big) \\ arctan\dfrac{2(q_3q_4 - q_1q_2)}{2(q_1^2 - q_4^2) - 1} \end{bmatrix} \tag{17}$$

The block diagram for Orientation to Euler VI is presented in Figure 27.



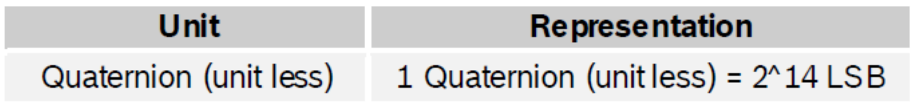
```
float64 xi, theta, rho;

xi = -atan2(2*q2*q3 - 2*q1*q4, 2*(q1*q1) + 2*(q2*q2) - 1);
theta = asin(2*q1*q3 - 2*q4*q2);
rho = atan2(2*q3*q4 - 2*q1*q2, 2*(q1*q1) + 2*(q4*q4) - 1);
```

Figure 27. Block Diagram of Orientation To Euler VI

It can be seen from the figure that atan2 function is used instead of arctangent. The atan2 is the arctangent function with two arguments. The purpose of using two arguments instead of one is to gather information on the signs of the inputs in order to return the appropriate quadrant of the computed angle, which is not possible for the single-argument arctangent function. It also avoids the problems of division by zero [34] .

The LabVIEW block diagram of the demo software can be seen in Figure 29. The demo software starts by reading in quaternion from COM port. The read in data is formatted as string. “Spreadsheet String To Array Function” is used to convert the data into int16 type quaternion array.

Next the 16-bit quaternions converted into floating point numbers by dividing them with 16384 (which is 2^14). The latter comes from BNO055 datasheet which states that 1 quaternion = 2^14LSB (Figure 28). After the last step the floating point quaternion data is fed into ORIENT TO EULER and QUATERNION TO R SubVIs, which were described earlier.



Unit	Representation
Quaternion (unit less)	1 Quaternion (unit less) = 2^14 LSB

Figure 28. Quaternion accuracy according to the BNO055 Datasheet

The QUATERNION TO R is again used to control the 3D object and ORIENT TO EULER calculates angles of the module. As in the original the angles were given in radians, the conversion into degrees was added for a better and quicker judgement of the angle.

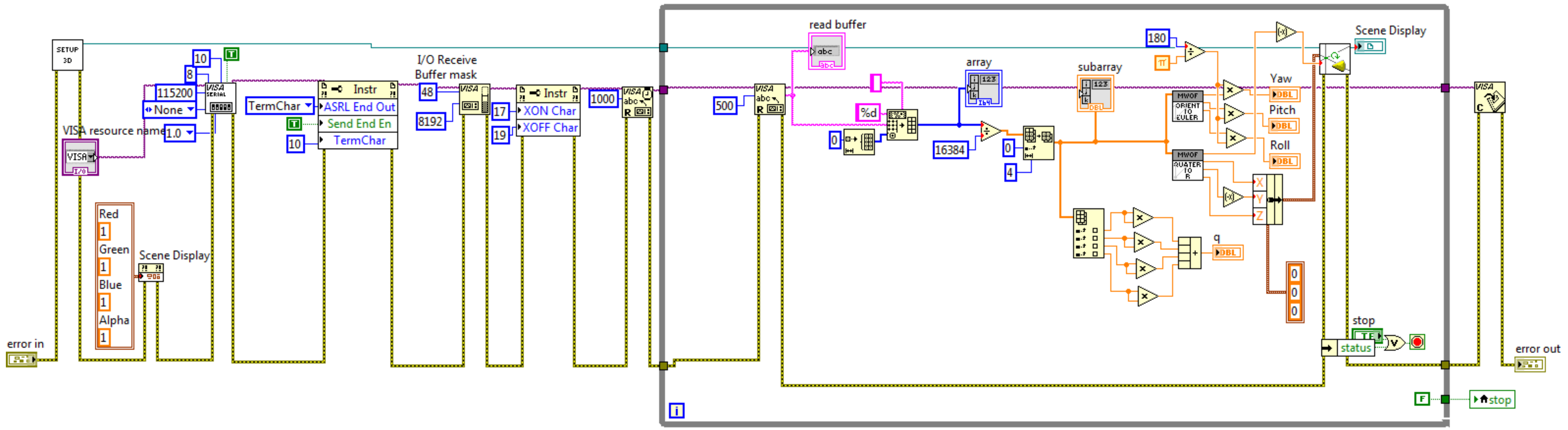
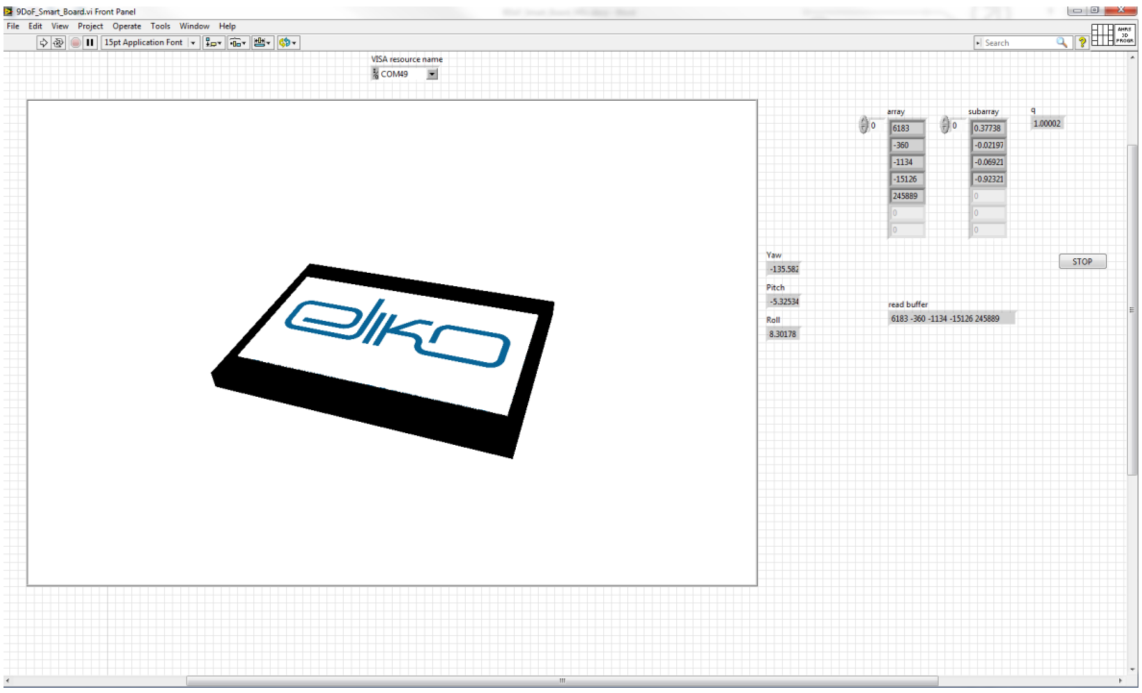


Figure 29. LabVIEW Block Diagram of 9DoF Smart Board Demo

All the functionality that is related to raw (9-axis) sensor data was removed in this version as the incoming data is already in quaternion format. QUATERNION TO ORTHO was also removed, because it was not needed.

The front panel of the VI has a really simple design – in the heart of it is the 3D object which is rotated according to the physical module. Above the 3D object window there is a COM port selector. On the right Yaw, Pitch and Roll are displayed in degrees. Also some assisting information is displayed on the right hand side and finally a "STOP" button to turn off the program.


9DoF_Smart_Board.vi Front Panel
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
STOP

VISA resource name: COM49

array: 6183, -360, -1134, -15126, 245889, 0, 0

subarray: 0.37738, -0.02197, -0.06921, -0.92321, 0, 0, 0

q: 1.00002

Yaw: -135.582

Pitch: -5.32534

Roll: 8.30178

read buffer: 6183 -360 -1134 -15126 245889

Figure 30. Front Panel of 9DoF Smart Board Demo

3.3 Software Development Results

A firmware and demo software were developed for the 3D Orientation Device. The firmware was designed according to the specification in section 1.4. As a result a custom driver was created for the BNO055 9DoF Sensor (for controlling and reading data). Also a main program was created which purpose is to read measurement results from BNO055 and send the data to USB host device. The outcome is corresponding to the requirements excluding the UART connections which will be added in the future development.

A LabVIEW based demo software was created to demonstrate the functionality of the 3D orientation module. It reads in the quaternion data from the device, convert it to axis-angle representation, which is used to manipulate virtual 3D object on a PC screen.

4 Economic Overview

Economically the goal was to create a product which would have a low price, comparable to breakout board style solutions, and functionality, which can be comparable to industrial grade devices. All the electronic components were picked so that their price-quality ratio would be optimal. Table 4 summarizes the cost of the components and the total amount spent for the electrical parts.

Table 4. Simplified BOM with Prices

#	Description	Comments	Qty	Price for 1 piece, €	Price per Board, €	Percentage, %
1	Capacitor	22pF,50V,0402	4	0.008	0.032	0.18%
2	Capacitor	0.1uF,10V,0402	7	0.005	0.035	0.19%
3	Capacitor	10uF,10V,0603	5	0.089	0.445	2.44%
4	FUSE, RESETTABLE PTC	MF-FSMF050X-2	1	0.46	0.46	2.52%
5	MICRO USB, 2.0 TYPE B	USB_Micro_B_FCI	1	0.58	0.58	3.18%
6	Header, 3-Pin, Dual row	CONF	1	0.46	0.46	2.52%
7	1 x 4 Header, 2.54mm pitch	UART	1	0.26	0.26	1.42%
8	LED, 0603, RED, 15MCD, 625NM	KPT-1608EC	1	0.08	0.08	0.44%
9	LED, 0603, YELLOW, 8MCD, 588NM	KPT-1608YC	1	0.071	0.071	0.39%
10	LED, GREEN, 12MCD, 568NM, 0603	KPT-1608SGC	2	0.075	0.15	0.82%
11	Mounting Hole	MH	4	0	0	0.00%
12	resistor	22,0402	2	0.012	0.024	0.13%
13	resistor	10k,0402	5	0.009	0.045	0.25%
14	resistor	430,0402	4	0.004	0.016	0.09%
15	resistor	100,0402	2	0.009	0.018	0.10%
16	Microcontroller AVR XMEGA	ATxmega16C4-AU	1	1.71	1.71	9.37%
17	Absolute Orientation 9-Axis Sensor	BNO055	1	11.06	11.06	60.61%
18	LDO, FIXED, 3.3V, 0.15A, SOT-23-5	TLV71333PDBVR	1	1.05	1.05	5.75%
19	Diode - Dual Common Cathode	MMBD6100LT1G	1	0.077	0.077	0.42%
20	CRYSTAL, 32.768KHZ, 12.5PF, SMD	LFXTAL009678	1	0.73	0.73	4.00%
21	16MHz Crystal Oscillator	16MHz, +/-10ppm, 8pF	1	0.598	0.598	3.28%
22	ESD Protection Device	CG0402MLC-05	2	0.173	0.346	1.90%
					18.247	**100.00%**

As it can be seen from Table 4, BNO055 is the most expensive component, forming about 61% of the whole cost. At the same time it is the most important component and, as it is a unique product, can't be replaced. MCU ATxmega is on the second place with 9.37%. The microcontroller was already picked by its low price and is not easily replaceable with cheaper alternatives. It can be considered to remove some of the components (which are not critical) or replace some of them with cheaper alternatives to reduce the self-cost.

To calculate the consumer price of for the product materials, PCB production, PCB assembly, cost of labor (development cost), desired profit, reseller profit and taxes must be considered. In this case the price is calculated with very low volumes (10 - 20 pieces of components, PCB etc), giving the worst case estimation. The volume discounts will increase the profit margin, but the formation of the end price will be calculated from worst case scenario.

Nowadays it is best practice to order PCB manufacturing from China, because the prices compared to local (Estonian) manufactures are about 10 times cheaper (or more). There are many different companies worldwide, PCB Shopper webpage [35] conveniently summarizes prices of most PCB manufacturers. Therefore it is a good source for getting comparison about the prices and reputation of the companies. The price estimation for calculating the consumer price of "9DoF Smart Board" was taken from PCB Shopper website. The author of this work is confident about the price estimations found from mentioned website, because he has used it before and also services of different China manufacturers which are listed on PCB shopper webpage - Itead Studio, ShenZhen2U, Elecrow etc. The lowest price with regular shipping is around 12€, with DHL shipping 26€, thus making 1.2 - 2.5€ for the unit price.

The components must be also assembled to the PCB, this is the next cost source. Low volumes can be hand soldered, larger amounts must be handled out of the house. This can be again ordered from Chinese companies, who offer PCB assembly as a service [36] . According to different sources this will cost around 200 - 800€, if 100 or more units are ordered. This means 2 - 8€ per device.

Next and very important part of product end price is the desired profit, which is usually 10 – 20% for electronics hardware, reseller or distributor adds another 30% and in Estonia 20% turnover tax must be added.

Table 5 summarizes all the parts that form purchase price for end user. It can be seen that the purchase price comes around 53€, so we can say that the product can be safely sold for 55€. The overall price is little bit higher than expected. The desirable price range was

Table 5. Summary of Purchase Price Formation for “9DoF Smart Board”

Description	Cost, €
Materials	18.000
PCB Production	2.500
PCB Assembly	8.000
Desired Profit (20%)	5.700
Distributor profit (30%)	10.260
Taxes	8.892
Purchase price	**53.352**

20 - 40€. If it is possible to lower some costs (for example the PCB Assembly from 8 to 5€) then the price will be 47.77€. That number is much closer to the desired pricing and probably 49€ would be a good price for the 3D orientation device.

The developer salary during the development was 1000€ net per month, which means 1676€ payroll fund for the employer. The project development lasted about 2 months which makes 3352€ for the total labor cost. In case of 20% profit 3352 / 5.7 = 588 units must be sold to break even (that is when distributing the product through reseller). In case the devices are sold straight from the company to end-customers there will be 15.96€ profit per every unit, which means that the break-even point will come at 3352 / 15.96 = 210 units. In reality it will be probably somewhere in-between, because some of the modules will be sold by distributers and some of them will be sold straight to the clients.

4.1 Economic Results

One of the main goals was to create a device which has a low cost. All the materials were chosen optimizing price-quality ratio. After adding production costs, profit and tax, the end price little bit exceeds the target. It is about 55€ instead of 20 – 40€ range. It is possible to lower the price by cutting some costs to 49€ or even less.

5 Evaluation

This chapter will give an overview about the first evaluation of the 3D module. Advanced and systematic testing is yet to be done in the future.

The verification of the orientation data was done by using the demo software. To verify if the output angles correlate with a physical object's orientation a cuboid was used. The device was put inside a matchbox and rotated around every axes (Figure 31). The angles were checked from the calculated Euler data (pitch, roll, yaw) and from visual inspection of the virtual 3D object's orientation to confirm the correlation.

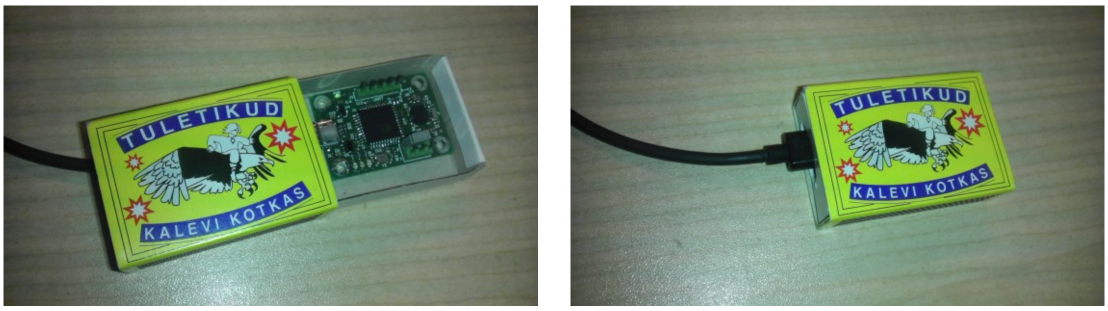


Figure 31. "9DoF Smart Board" Put Inside a Matchbox for Testing

The module inside the matchbox was rotated by 90° steps, at each step the values were recorded and put into a table. After marking down the values, difference between steps, difference from the starting point, absolute distance and error from the target value was calculated (in degrees).

Table 6. Measured and calculated values for Yaw-axis

	0 to right, °	90 to right, °	180 to right, °	270 to right, °
Yaw (measured)	-122.0	145.8	54.2	-36.0
Delta	0.0	92.2	91.6	90.2
Diff	0.0	-267.8	-176.2	-86.0
Abs dist.	0.0	92.2	183.8	274.0
Error	0.0	2.2	3.8	4.0

Table 7. Measured and calculated values for Pitch-axis

	0 to right, °	90 to right, °	180 to right, °	270 to right, °
Pitch (measured)	-2.2	85.4	3.9	-85.9
Delta	0.0	87.6	81.5	89.8
Diff	0.0	-87.6	-6.1	83.7
Abs dist.	0.0	87.6	173.9	276.3
Error	0.0	-2.4	-6.1	6.3

Table 8. Measured and calculated values for Roll-axis

	0 to right, °	90 to right, °	180 to right, °	270 to right, °
Roll (measured)	-0.3	83.7	176.1	-96.1
Delta	0.0	84.0	92.4	87.8
Diff	0.0	-84.0	-176.4	95.8
Abs dist.	0.0	84.0	183.6	264.2
Error	0.0	-6.0	3.6	-5.8

Table 6, Table 7 and Table 8 represent the measured data for Yaw-, Pitch- and Roll-axis accordingly. The ranges for the axis are different and listed below.

Yaw: -180…180°

Pitch: -90…90°

Roll: -180…180°

As it can be seen from the tables above the measured values are generally close to desired values, differing from the target up to 6.3° (in absolute value). Also the visual inspection gives overall impression that the 3D object simulation is following the orientation of the physical device. Depending on the application up to 6.3° absolute error may be acceptable or it can be too large. The setup was not accurate enough to make any final conclusions about the errors. This was just to get a general estimation if the device is working correctly. It was discovered during testing that matchbox is not suitable for testing because it is differing quite a lot from ideal cuboid. Specifically the box is too soft and does not have angles that are firmly 90°. A minor movement (like bending of the sides) can change the angle several degrees. Therefore a better cuboid enclosure should be found for next testing.

A known issue for 3D orientation sensors is an angles drift. The gyroscope zero bias will drift over time, with temperature and with motion. The MARG implementation usually incorporates magnetic distortion and gyroscope bias drift compensation [37] . This drift was tested with specially made firmware which read out both gyroscope and Euler angles data from BNO055.

The test was done as follows. Two devices were put laying on top of a foam, on a table to isolate modules from outside vibration noise. During test no one moved closer than 2m from the desk. There were two tests – first lasted 2 minutes and the second one 8 minutes. Below is the configuration information and equation (18) which was used to calculate the total gyroscope drift. Figures Figure 32, Figure 33, Figure 34 and Figure 35 below represent the test results for Test1 and Test2.

Configurations

Operation mode - OPERATION_MODE_NDOF (see BNO055 operation modes for more information)
sample rate - 100Hz
Gyroscope range – ± 2000°/s (default for MODE_NDOF)
Acceleration range - ±4g (default for MODE_NDOF)

Test Equations

Total change in angle using gyroscope

$$\sum_{k=0}^{n} x_k * \frac{2000}{32767} * \Delta t \tag{18}$$

N – sample count
x_k – Gyroscope sensor raw output data
Δt – time difference between samples

Test 1

Test duration 2min

Module 1:

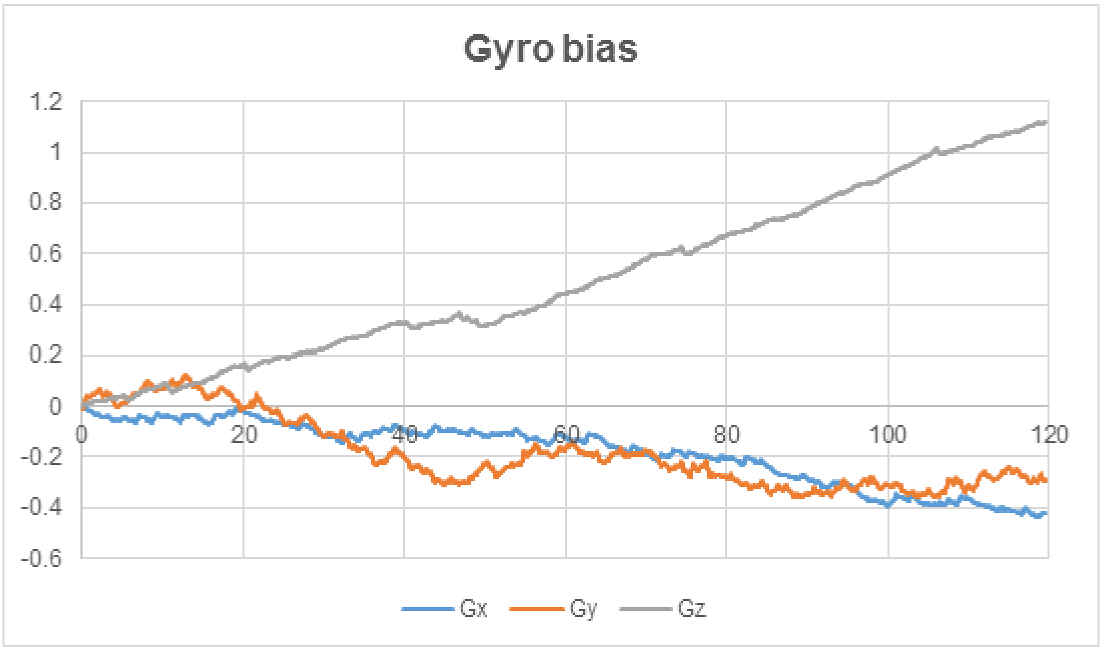


Figure 32. Test1, Module 1: Gyro Bias Drift

Calculated Gyro bias for 1 hour
Gx: -12.78°/h
Gy: -8.92°/h
Gz: 33.68°/h

No change in Euler angles.

Test 1

Test duration 2min

Module 2:

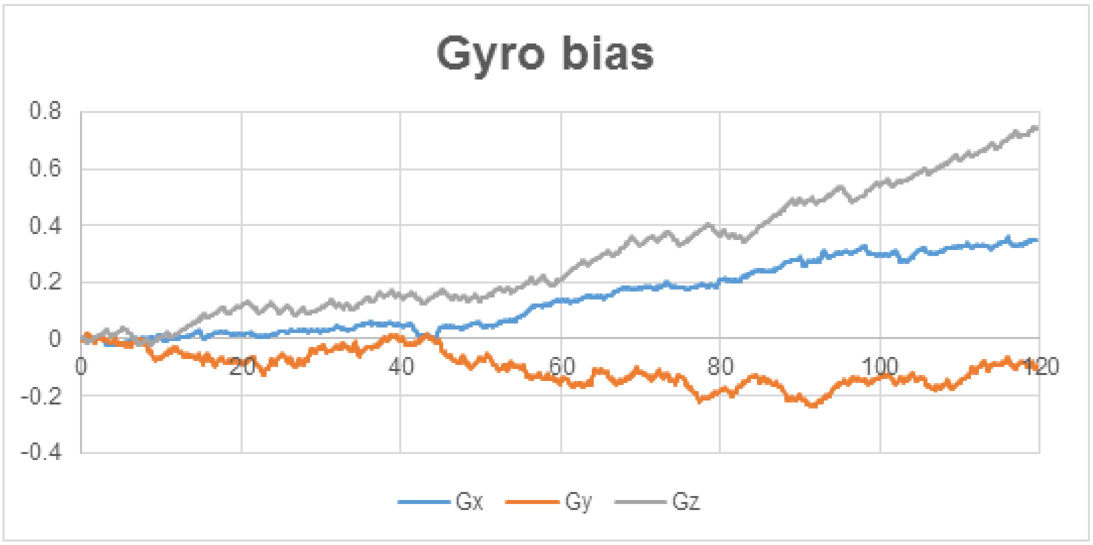


Figure 33. Test1, Module 2: Gyro Bias Drift

Calculated Gyro bias for 1 hour
Gx: 10.60°/h
Gy: -2.91°/h
Gz: 22.52°/h

No change in Euler angles.

Test 2

Test duration 8min

Module 1:

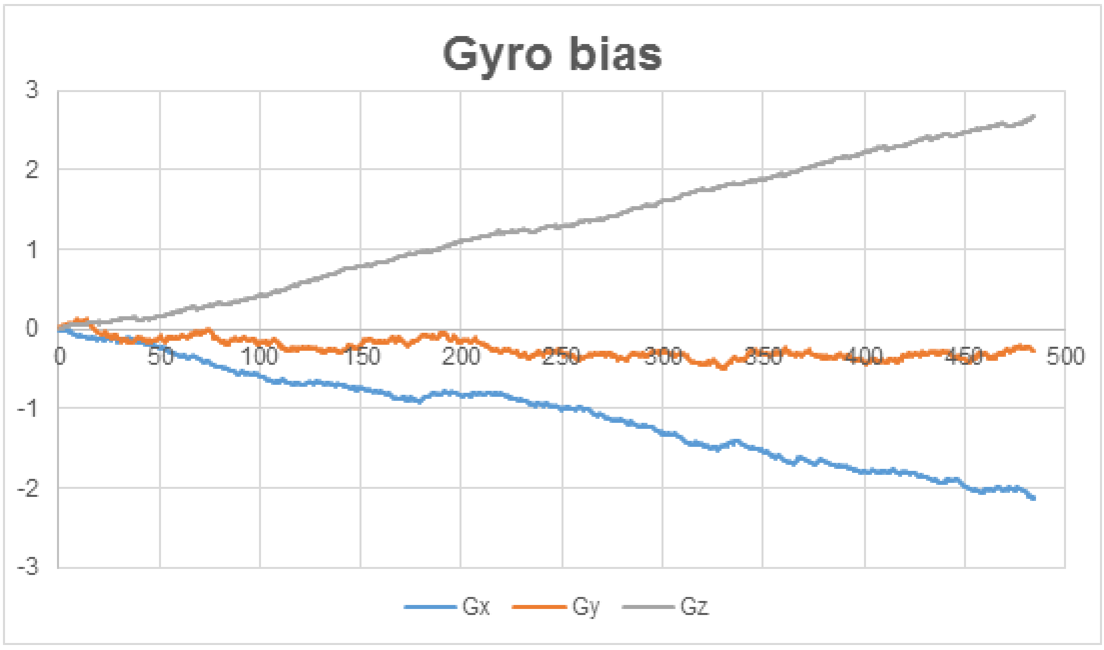


Figure 34. Test2, Module 1: Gyro Bias Drift

Calculated Gyro bias for 1 hour
Gx: -15.80°/h
Gy: -2.00°/h
Gz: 19.88°/h

No change in Euler angles.

Test 2

Test duration 8min

Module 2:

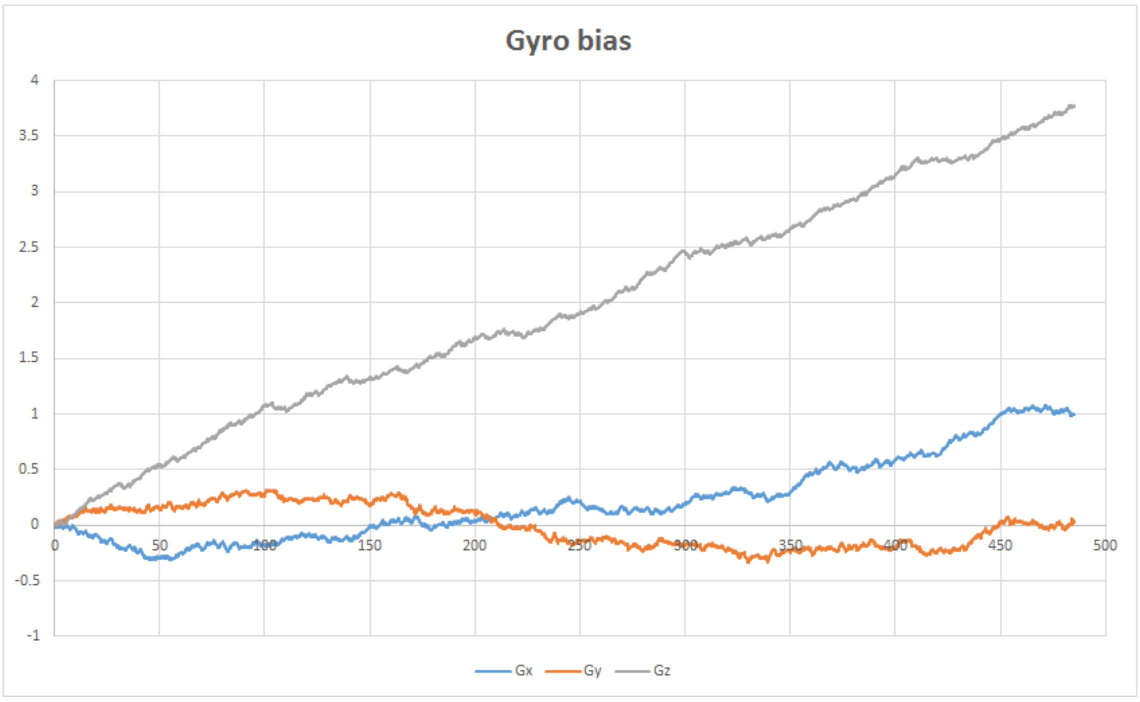


Figure 35. Test2, Module 2: Gyro Bias Drift

Calculated Gyro bias for 1 hour
Gx: 7.37°/h
Gy: 0.25°/h
Gz: 27.90°/h

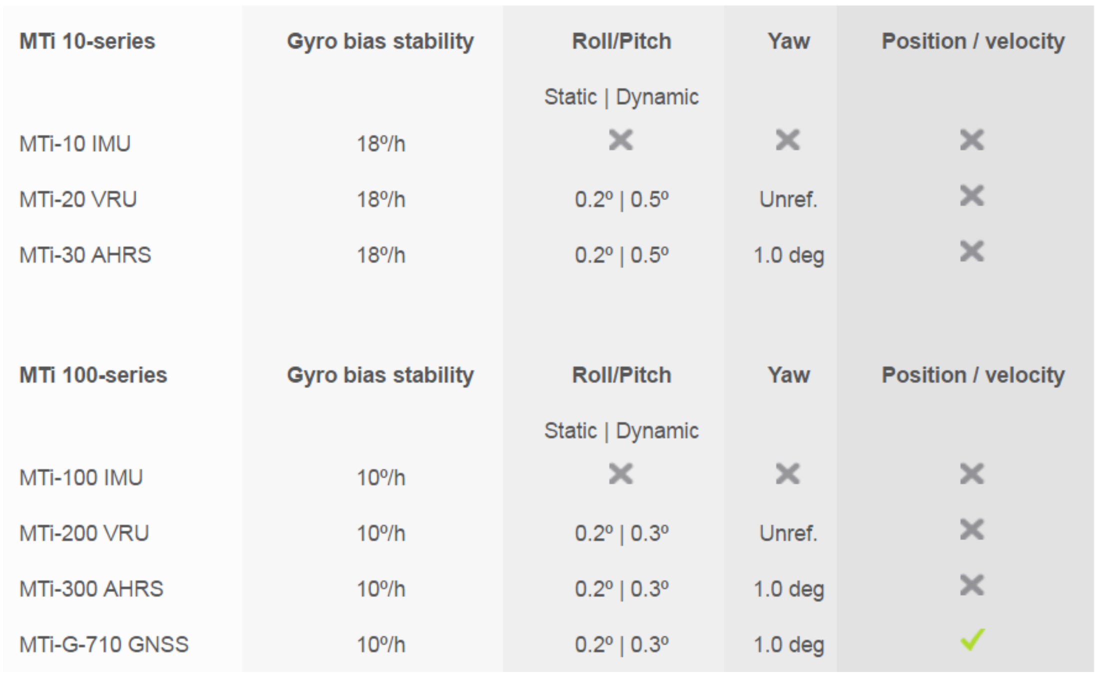
Euler angles

Yaw: no change
Pitch: -0.0625°
Roll: 0.25°

It can be concluded from the gathered data that Euler angles are quite stable, which is a positive result. Gyroscope Z value seem to have quite big offset drift (20°…34°). Gyroscope X and gyroscope Y differ a lot comparing the tests. Therefore calculated gyro bias for 1 hour is not valid for X- and Y-axis and the result for these axis is inconclusive.

Figure 36 represents Xsens' summary of their products, listing gyro bias stability and Euler angles static and dynamic accuracy. Comparing the measurement results with

Xsens data we can say that Z-axis gyro bias drift is higher (magnitude is similar, which again validates the information). According to the measurements “9DoF Smart Board” Euler angle accuracy is better having almost no change in angle values (has to be verified with more measurements).



MTi 10-series	Gyro bias stability	Roll/Pitch	Yaw	Position / velocity
		Static | Dynamic		
MTi-10 IMU	18º/h	✕	✕	✕
MTi-20 VRU	18º/h	0.2º | 0.5º	Unref.	✕
MTi-30 AHRS	18º/h	0.2º | 0.5º	1.0 deg	✕

MTi 100-series	Gyro bias stability	Roll/Pitch	Yaw	Position / velocity
		Static | Dynamic		
MTi-100 IMU	10º/h	✕	✕	✕
MTi-200 VRU	10º/h	0.2º | 0.3º	Unref.	✕
MTi-300 AHRS	10º/h	0.2º | 0.3º	1.0 deg	✕
MTi-G-710 GNSS	10º/h	0.2º | 0.3º	1.0 deg	✓

Figure 36. Xsens Product’s Overview [38]

5.1 Evaluation Results

The evaluation and tests for the device can be summarized as follows. Generally the module’s angular output correlates to physical object’s rotations in 3D room. Measurements and visual inspection of simulated cuboid verify this. The angular error was up to 6.3° in absolute value, which has to be checked, because may have been caused by test setup.

Drift measurements resulted in very stable Euler angle output data, although at the same time gyroscope z-axis drift is about 20…30°/h. It was not possible to make conclusions for x and y-axis gyroscope drift as the tests gave very different results.

Overall impression is positive, because the angular data correlates to the real world and is stable in time. More tests are yet to be made.

6 Summary

The purpose of this thesis was to create a low cost high precision 3D orientation device. A fully functional systems was developed during the thesis. Special hardware and firmware were developed according to the specifications laid out in the beginning of the design process. Additionally a software was designed for demonstration and testing purposes.

The created module has a Bosch BNO055 9DoF in its core, which pre-calculates orientation data. The module is able to output 3D orientation information in a form of quaternions or output raw 9-axis data, according to user configurations. The created hardware fulfills specification up to 95%, the shortfall being UART communication which pins are misplaced because of using an old datasheet version. The latter mistake will be fixed in the next version.

A firmware was created which main purpose is to act as a “bridge” between the sensor IC and USB or UART communication ports. The developed firmware is working oraccording to the requirements sending the user selected information to communication ports. Some future work is left to be done in this field – USB/UART to I2C bridge for example, LED indications. During testing an I2C clock stretching issue was discovered which limits the raw data communication bandwidth down to 150Hz max instead 1kHz. Thus a decision was made to lower the frequency down to 100Hz.

One of the main goals was to create a device which has a low cost. All the materials were chosen optimizing price-quality ratio. After adding production costs, profit and tax, the end price exceeds the target. It is about 55€ instead of 20 – 40€ range. It is possible to lower the price by cutting some costs.

After evaluation and some tests overall impression is positive, because the angular data correlates to the real world and is stable in time. The angular error was up to 6.3° in absolute value, which has to be checked, because it may have been caused by test setup. Drift measurements resulted in very stable Euler angle output data, although at the same time gyroscope z-axis drift is about 20…30°/h. Generally more tests have to be made before making any final conclusions.

Overall it can be said that most of the goals were met during this thesis. There are some minor mistakes that will be corrected. The module will be developed further and updated in future.

I hereby thank my supervisor Alar Kuusik, co-supervisor Olev Märtens, coworker Arthur Randjärv, ELIKO, Thomas Johann Seebeck Department of Electronics and my family for assistance and support.

References

[1] SparkFun 9 Degrees of Freedom IMU Breakout - LSM9DS1. [WWW] https://www.sparkfun.com/products/13284

[2] ADAFRUIT 9-DOF ACCEL/MAG/GYRO+TEMP BREAKOUT BOARD - LSM9DS0. [WWW] https://www.adafruit.com/products/2021

[3] Tindie Ultimate Sensor Fusion Solution. [WWW] https://www.tindie.com/products/onehorse/ultimate-sensor-fusion-solution/

[4] Sparkfun's 9 Degrees of Freedom - Razor IMU. [WWW] https://www.sparkfun.com/products/10736

[5] Tindie's 9 DOF IMU with LSM9DS1 and Atmega328p. [WWW] https://www.tindie.com/products/Afritlabs/9-dof-imu-with-lsm9ds1-and-atmega328p/

[6] Euler angles. [WWW] https://en.wikipedia.org/wiki/Euler_angles

[7] Quaternion. [WWW] https://en.wikipedia.org/wiki/Quaternion

[8] USB 2.0 Specification. [WWW] http://www.usb.org/developers/docs/usb20_docs/usb_20_040816.zip (Rev. 1.0, March 25, 2016)

[9] Bosch BNO055 Datasheet. [WWW] https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST_BNO055_DS000_13.pdf (Rev 1.3, August, 2015)

[10] ATxmega16C4-ATxmega32C4 Datasheet. [WWW] http://www.atmel.com/images/atmel-8493-8-and-32-bit-avr-xmega-microcontrollers-atxmega16c4-atxmega32c4_datasheet.pdf (12/2014)

[11] USB Hardware Design Guide. [WWW] https://www.silabs.com/Support%20Documents/TechnicalDocs/AN0046.pdf (2013-09-16 - an0046_Rev1.01)

[12] Atmel AVR1017: XMEGA - USB Hardware Design Recommendations. [WWW] http://www.atmel.com/images/doc8388.pdf (07/2011)

[13] BOURNS CG0402MLC-05LG, ESD Protectors Datasheet. [WWW] https://www.bourns.com/pdfs/MLC.pdf (June 8, 2011)

[14] BOURNS MF-FSMF Series - PTC Resettable Fuses Datasheet. [WWW] https://www.bourns.com/data/global/pdfs/mffsmf.pdf (June 8, 2011)

[15] APPLICATION NOTE, Atmel AVR1916: USB DFU Boot Loader for XMEGA. [WWW] http://www.atmel.com/images/doc8429.pdf (08/2012)

[16] APPLICATION NOTE, Atmel AT01607: XMEGA C Schematic Checklist. [WWW] http://www.atmel.com/Images/Atmel-8466-AVR-AT01607-XMEGA-C-Schematic-Checklist_Application-Note.pdf

[17] Texas Instruments, TLV713 Datasheet. [WWW] http://www.ti.com/lit/ds/symlink/tlv713p.pdf

[18] Bosch BNO055 Handling, soldering & mounting instructions. [WWW] https://ae-bst.resource.bosch.com/media/_tech/media/others/BST-BNO055-HS000-00.pdf

[19] USB Wikipedia. [WWW] https://en.wikipedia.org/wiki/USB

[20] USB-IF Device Class Documents. [WWW] http://www.usb.org/developers/docs/devclass_docs/

[21] Serial Port Wikipedia. [WWW] https://en.wikipedia.org/wiki/Serial_port

[22] LUFA. [WWW] http://www.fourwalledcubicle.com/LUFA.php

[23] ASF Quick start guide for XMEGA TWI driver. [WWW] http://asf.atmel.com/docs/latest/xmegaa/html/xmega_twi_quickstart.html

[24] LUFA TWI Peripheral Driver (XMEGA). [WWW] http://www.fourwalledcubicle.com/files/LUFA/Doc/130901/html/group___group___t_w_i___x_m_e_g_a.html

[25] Bosch BNO055 Driver. [WWW] https://github.com/BoschSensortec/BNO055_driver

[26] Bosch Adafruit BNO055 Driver. [WWW] https://github.com/adafruit/Adafruit_BNO055

[27] I2C Clock Stretching. [WWW] http://www.i2c-bus.org/clock-stretching/

[28] Arduino 9 Axes Motion Shield. [WWW] http://www.arduino.org/products/shields/5-arduino-shields/arduino-9-axes-motion-shield

[29] Quaternion based AHRS LabVIEW Library. [WWW] https://decibel.ni.com/content/docs/DOC-18964

[30] LabVIEW Wikipedia. [WWW] https://en.wikipedia.org/wiki/LabVIEW#Release_history

[31] Quaternion. [WWW] http://wiki.alioth.net/index.php/Quaternion

[32] Rotation formalisms in three dimensions, Wikipedia. [WWW] https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions#Euler_angles_.28z-x-z_extrinsic.29_.E2.86.92_Quaternion

[33] Conversion between quaternions and Euler angles, Wikipedia. [WWW] https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles

[34] Atan2 Wikipedia. [WWW] https://en.wikipedia.org/wiki/Atan2

[35] PCB Shopper Webpage. [WWW] http://pcbshopper.com/

[36] Shenzhen2u, PCB Assemly Service. [WWW] http://www.shenzhen2u.com/PCB-Assembly

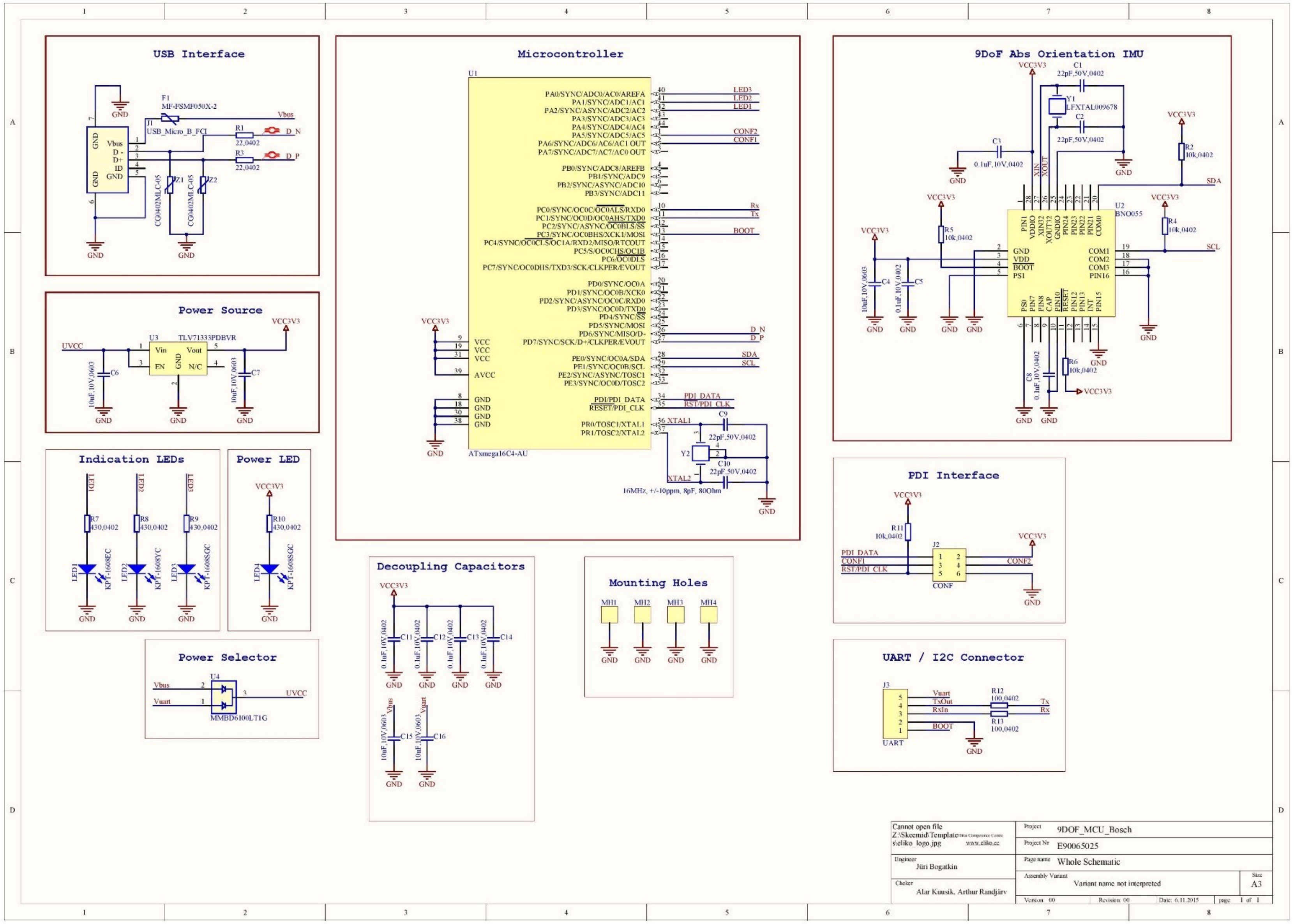
[37] An efficient orientation filter for inertial and inertial/magnetic sensor arrays Sebastian O.H. Madgwick. [WWW] http://www.x-io.co.uk/res/doc/madgwick_internal_report.pdf

[38] Xsens Products. [WWW] https://www.xsens.com/products/mti-10-series/

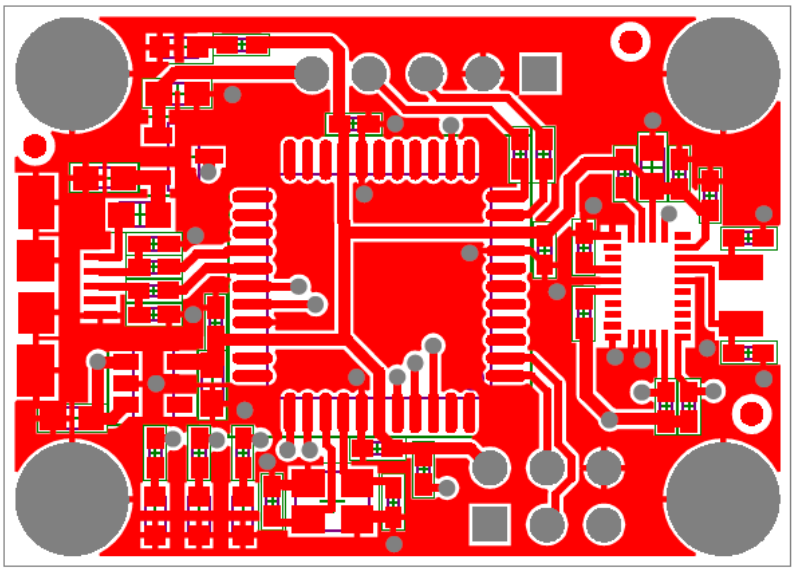
[39] Symbol rate Wikipedia. [WWW] https://en.wikipedia.org/wiki/Symbol_rate

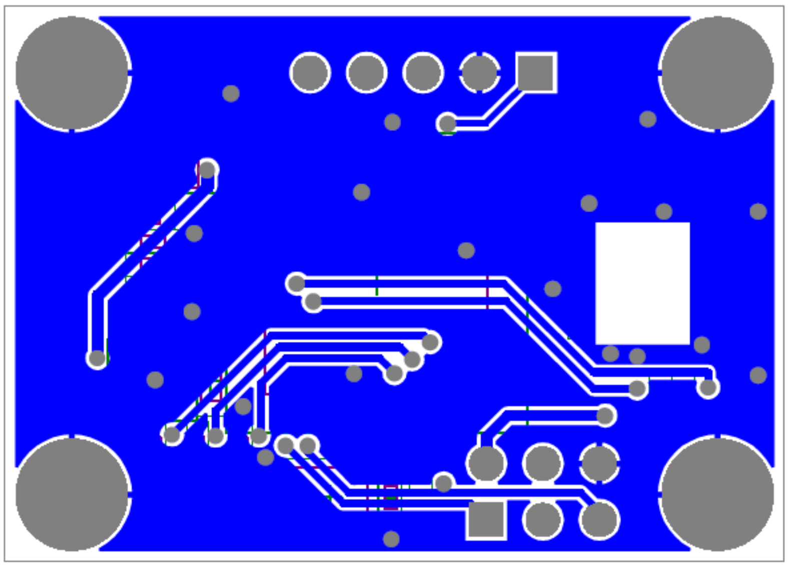
[40] Bit Rate Wikipedia. [WWW] https://en.wikipedia.org/wiki/Bit_rate

Appendix 1 – Electrical Schematic Design for the “9DoF Smart Board”

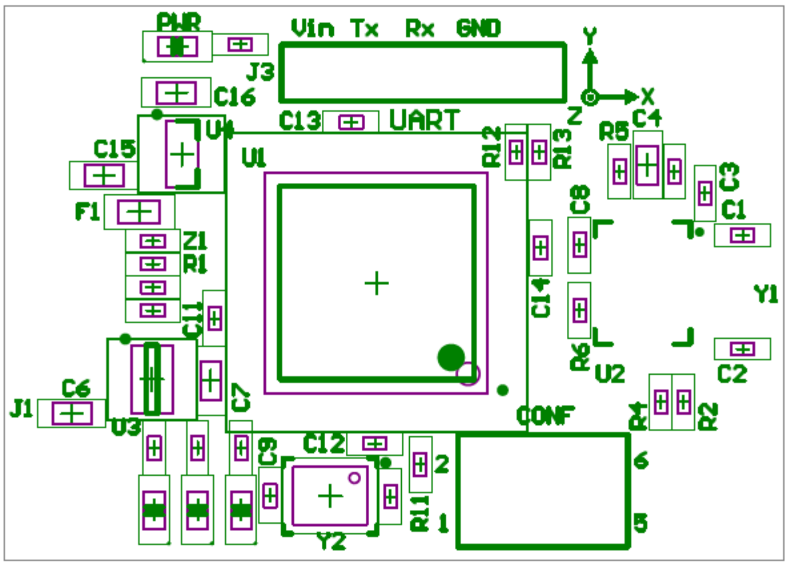


Appendix 2 – PCB Top and Bottom Layer Design

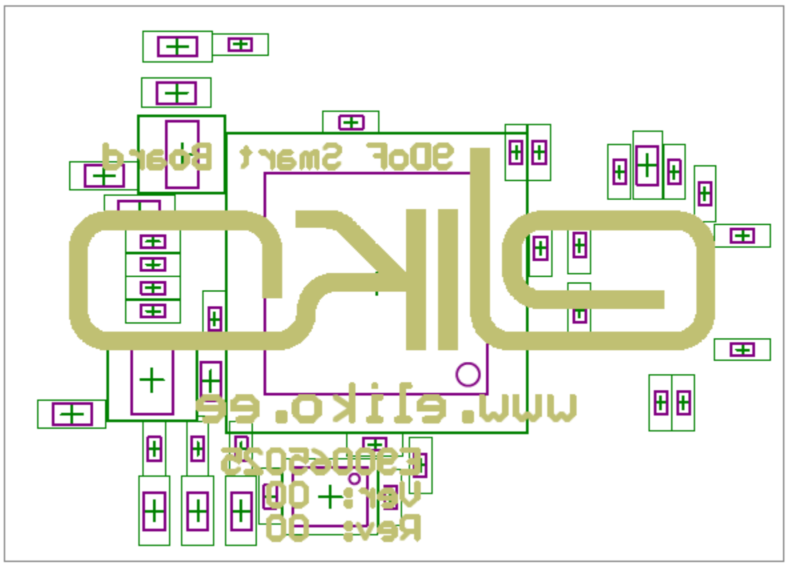

Top Layer


Bottom Layer

Appendix 3 – PCB Top and Bottom Silkscreen Overlay Design

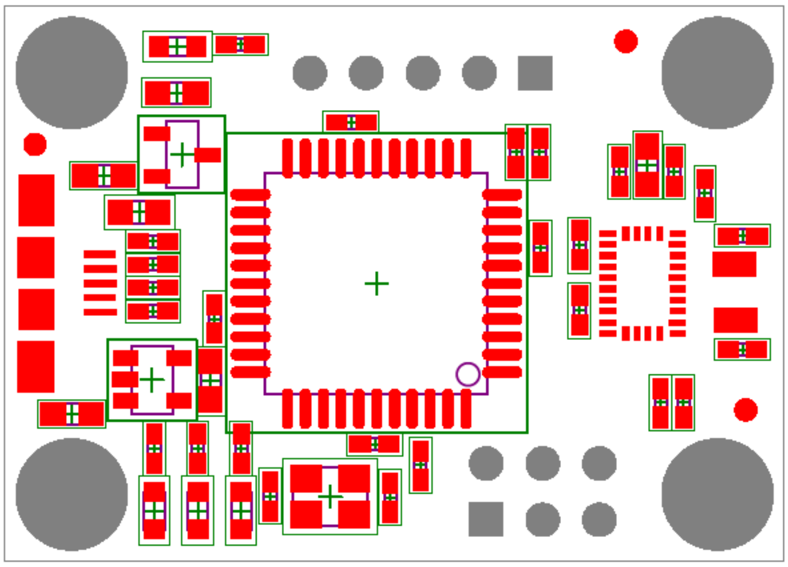


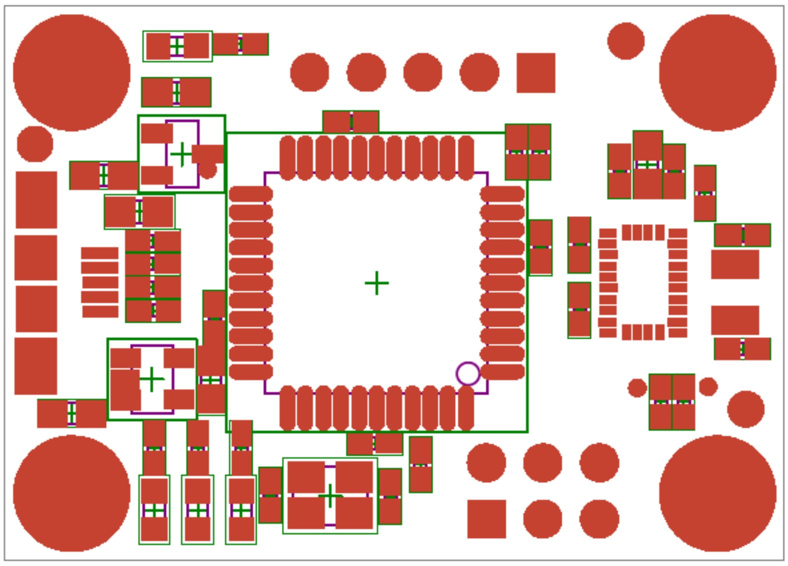
Top Silkscreen Overlay



Bottom Silkscreen Overlay

Appendix 4 – PCB Top Pad Master and Top Solder Mask Design

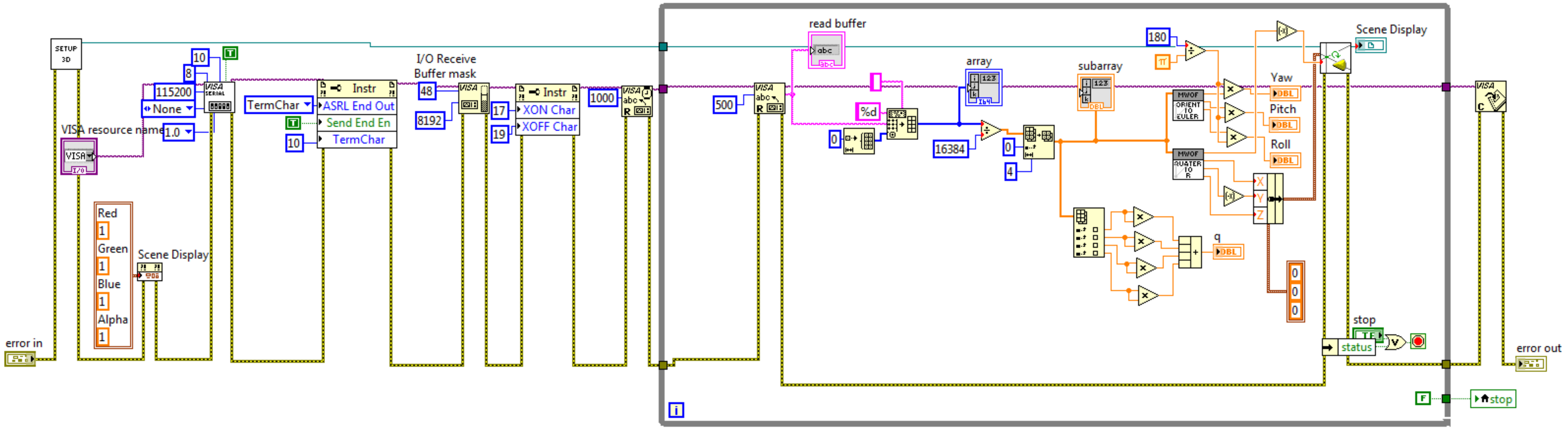

Top Pad Master


Top Solder Mask

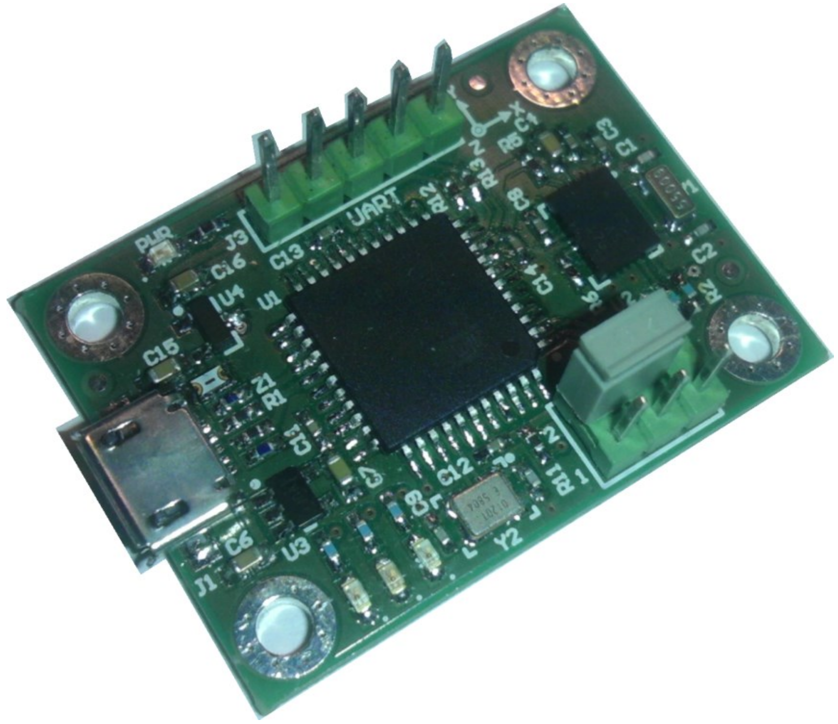
Appendix 5 – Main Function Program Code

```c
#include "VirtualSerial.h"

static CDC_LineEncoding_t LineEncoding = { .BaudRateBPS = 0,
                                           .CharFormat  = CDC_LINEENCODING_OneStopBit,
                                           .ParityType  = CDC_PARITY_None,
                                           .DataBits    = 8                            };

/** Main program entry point. This routine contains the overall program flow, including initial
 *  setup of all components and the main program loop.
 */

uint32_t timestamp = 0;
uint32_t lastTimestamp = 0;
int16_t data[9] = {0};
char str[100];
volatile uint8_t conf = 0;
volatile uint8_t conf2 = 0;
volatile uint8_t conf3 = 0;
volatile uint8_t conf4 = 0;
volatile uint8_t conf5 = 0;
volatile uint8_t state = 2;
volatile uint8_t last_state = 3;
const uint8_t streamPeriod_Euler = 10;
const uint8_t streamPeriod_Raw = 10;
const uint8_t streamPeriod_Quaterntion = 10;
uint8_t dt = 0; // time difference

int main(void)
{
        SetupHardware();
        GlobalInterruptEnable();

        /* EULER mode */
        Delay_MS(100);
        begin(OPERATION_MODE_NDOF);

        /* Use BNO055 external crystal oscillator */
        setExtCrystalUse(true);

        for (;;)
        {
                USB_USBTask();

                /* Raw output */
                if(gpio_pin_is_high(CONF2) == false)
                {
                        state = 1;
                        // be sure to write the configuration just once
                        if(state != last_state)
                        {
                                    BNO055_write_byte(BNO055_SYS_TRIGGER_ADDR, 0x20);
                                    while(BNO055_read_byte(BNO055_CHIP_ID_ADDR)!= BNO055_ID)
                                    {
                                            Delay_MS(10);
                                    }
                                    Delay_MS(50);
                                        BNO055_write_byte(BNO055_SYS_TRIGGER_ADDR, 0x00);
                                        Delay_MS(1000);

                                        // setExtCrystalUse(true);

                                // Enter raw data mode
                                setMode(OPERATION_MODE_AMG);
                                conf2 = BNO055_read_byte(BNO055_OPR_MODE_ADDR);
                                // Configure sensors
                                SetupRawSensors();


                                last_state = state;

                            setExtCrystalUse(true);

                        }

                        if((dt = timestamp - lastTimestamp) >= streamPeriod_Raw)
                        {
                                lastTimestamp = timestamp;

                                    ioport_set_pin_high(BOOT);

                                        // get raw sensor values
                                        getVector(VECTOR_RAW, data);

                                        ioport_set_pin_low(BOOT);


                                        sprintf(str, "%d %d %d %d %d %d %d %d %d %lu\r\n",
data[0], data[1], data[2], data[6], data[7], data[8], data[3], data[4], data[5], timestamp);

                                        Send_To_USB(str, strlen(str));
                        }
                }

                /* Quaternion */

                if(gpio_pin_is_high(CONF2) == true)
                                {
                        state = 0;
                        // be sure to write the configuration just once
                        if(state != last_state)
                        {
                                    BNO055_write_byte(BNO055_SYS_TRIGGER_ADDR, 0x20);
                                    while(BNO055_read_byte(BNO055_CHIP_ID_ADDR)!= BNO055_ID)
                                    {
                                            Delay_MS(10);
                                    }
                                    Delay_MS(50);
                                    BNO055_write_byte(BNO055_SYS_TRIGGER_ADDR, 0x00);
                                    Delay_MS(1000);

                                    setMode(OPERATION_MODE_NDOF);
                                    conf2 = BNO055_read_byte(BNO055_OPR_MODE_ADDR);

                                    setExtCrystalUse(true);

                                last_state = state;
                        }

                        if((dt = timestamp - lastTimestamp) >= streamPeriod_Quaterntion)
                        {
                                lastTimestamp = timestamp;


                        getVector(VECTOR_QUATERNION, data);
                        sprintf(str, "%d %d %d %d %lu\r\n", data[0], data[1], data[2], data[3],
timestamp);

                        Send_To_USB(str, strlen(str));

                        }

                }

        }
}
```

Appendix 6 – LabVIEW Block Diagram of Developed Demo Software



Appendix 7 – Images of the Resulted Physical Device



Top View

Bottom View