HSN

Sindre Marius Schulstock

Data Cloud Platform for Data Management, Logging, Control and Monitoring
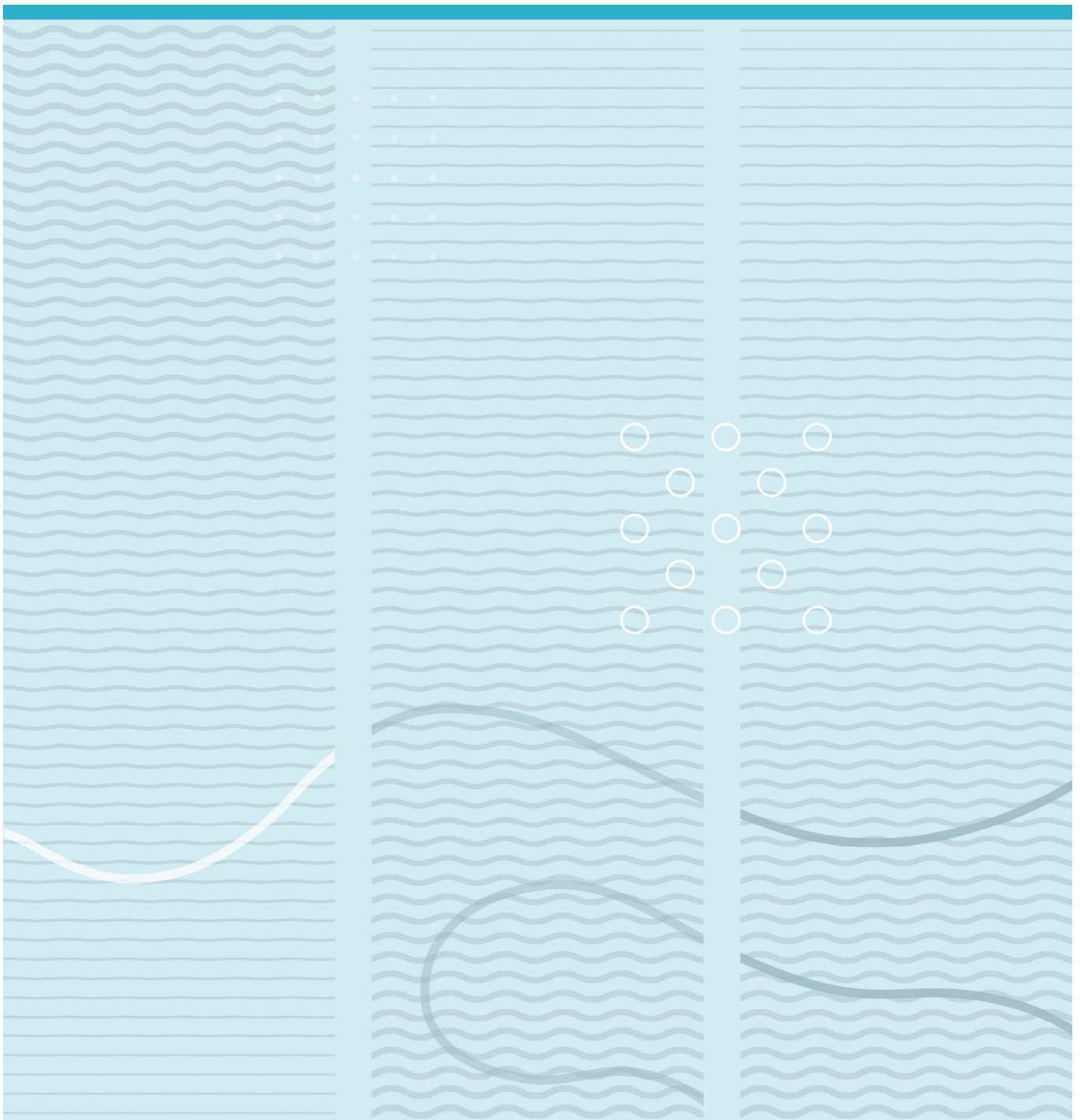
This thesis is worth 30 study points

University College of Southeast Norway

www.usn.no

Course: FMH606 Master's Thesis, 2016

Title: Data Cloud Platform for Data Management, Logging, Control and Monitoring

Number of pages: 148

Student:	Sindre Marius Schulstock
Supervisor:	Hans-Petter Halvorsen
External partner:	National Instrument
Availability:	Open

Approved for archiving: _____

(supervisor signature)

Summary:

It has become more and more common in today's society to implement automated systems for monitoring and controlling objects remotely across existing network infrastructures.

In this master's thesis, design and implementation of an open platform for data management, logging, control and monitoring was to be developed based on the IoT concept.

Prototypes for sensor nodes and data hubs was built using different types of hardware found by research, and programed with Arduino and LabVIEW software.

The security aspect was addressed for highlighting the security level.

Complete monitoring system was developed with its appurtenant sensor nodes and data hubs installed in a test environment at Telemark University College. The monitoring system displayed a stable system over a five days period with decent measurement data.

The system was developed for potential further development.

Preface

This thesis is the final project as part of the master's degree in System and Control Engineering at University College of Southeast Norway (HSN). The thesis is written during the fall semester of 2016.

I wish to thank my supervisor Hans-Petter Halvorsen for guidance and close supervision throughout the project.

My gratitude goes to family and friends for support and encouragement throughout the thesis.

Porsgrunn, 9 December 2016

Sindre Marius Schulstock

Nomenclature

Arduino IDE	Arduino Integrated Development Environment
GBP	Great Britain pound
GPU	Graphics Processing Unit
GUI	Graphical user interface
GPIO	General-purpose input/output
IoT	Internet of Things
I/O	Inputs/Outputs
I2C	Inter-Integrated Circuit
Intel NUC	Intel Next Unit of Computing
Kr	Norwegian Kroner (currency)
LXDE	Lightweight X11 Desktop Environment
NTC	Negative Temperature Coefficient
OS	Operating system
Pa	Pascal
ppm	Parts per million
PWM	Pulse-width modulation
P2P	Peer-to-peer
RAM	Random-access memory
RH	Relative humidity
SoC	System on chip
Soft-AP	Software Enabled Access Point
SPI	Serial Peripheral Interface Bus
TWI	Two Wire Interface
UPS	Uninterruptible power supply
URL	Uniform Resource Locator
USD	United Sates dollar
VAT	Value added tax

Table of Contents

1 Introduction

The main goal for this thesis is to design and implement an open platform for data management, logging, control and monitoring based on the concept of IoT (Internet of things). Use of programming languages such as LabVIEW and C/C++ and the development of a functioning prototype for testing, will be an essential part in this master thesis.

IoT is the concept of letting objects be controlled remotely across existing network infrastructures, which then gives opportunities for more direct integration between the physical world and computer-based systems, which again can improve efficiency and also give economic benefits.

The focus for this thesis will be the development of sensor nodes for retrieving temperature readings and other additional data, such as humidity, pressure, co2, etc. A data hub will gather the data from each of the sensor nodes and upload this to a local database. This data shall then be available using a computer or tablet/smartphone, giving the options to gather historical data for analyzes or for controlling components such as heaters, doors, lights, etc. in for example a private home. The key features will be to find simple but robust solutions that are not expensive, making it more appealing for everyday people to buy and implement in their house. Necessary research regarding hardware and software options is essential before any development of prototypes can begin.

The security aspects regarding the concept of IoT will as part of this thesis be discussed in a briefly manner, since more and more of our daily lives are involved in the use of internet and the risks that others are stealing private information.

The report is divided into the following chapters:

- Chapter. 2: Background
- Chapter. 3: Security
- Chapter. 4: Communication Protocols
- Chapter. 5: Software
- Chapter. 6: Hardware
- Chapter. 7: Power options
- Chapter. 8: Selected components
- Chapter. 9: Development
- Chapter. 10: Testing

2 Background

This background chapter, as part of a master thesis provided by the University College of Southeast Norway, will introduce a general problem description about automated systems regarding the IoT concept. This chapter will describe the redefining/adjustments to some of the tasks due to areas of interests and limitations.

2.1 General Problem Description

In today's society, it is becoming more and more common to implement automated systems. Advances in wireless communications and IoT has made it possible to create simple automated systems at affordable prices [1]. There are already today a wide selection of different systems, but the prices are still high. These systems are also not normally open for modification by the end user, which is the opposite of the IoT concept.

In this thesis, the IoT concept will be exploited using cheaper solutions with components such as Raspberry PI, Arduino Uno, Genuino MKR1000, etc. These components are single-board microcontrollers/computers using open-source software with many possibilities for modifications based on user preferences. For making the automated system accessible remotely for control and monitoring, the concept of cloud services will also be exploited so the end-user can monitor and control the different components remotely from any location where there is internet available. A database, as part of the cloud services, will store the data for logging and analyzing purposes.

The system will be set up as an overarching system, see Figure 2-1, where the nodes collect data such as temperature, humidity, pressure, etc. These nodes will send this data to a data hub, using bluetooth or WI-FI, for temporary storing and onward transmission to an online database using Cloud Services. The data from the database will be accessible for monitoring using either a computer or a mobile device connected to the Cloud Service, which again will give the end-user the option to log and analyze data collected from each node. Area of use for this system can be either a private home or an industrial area, depending on what the end-user wants.
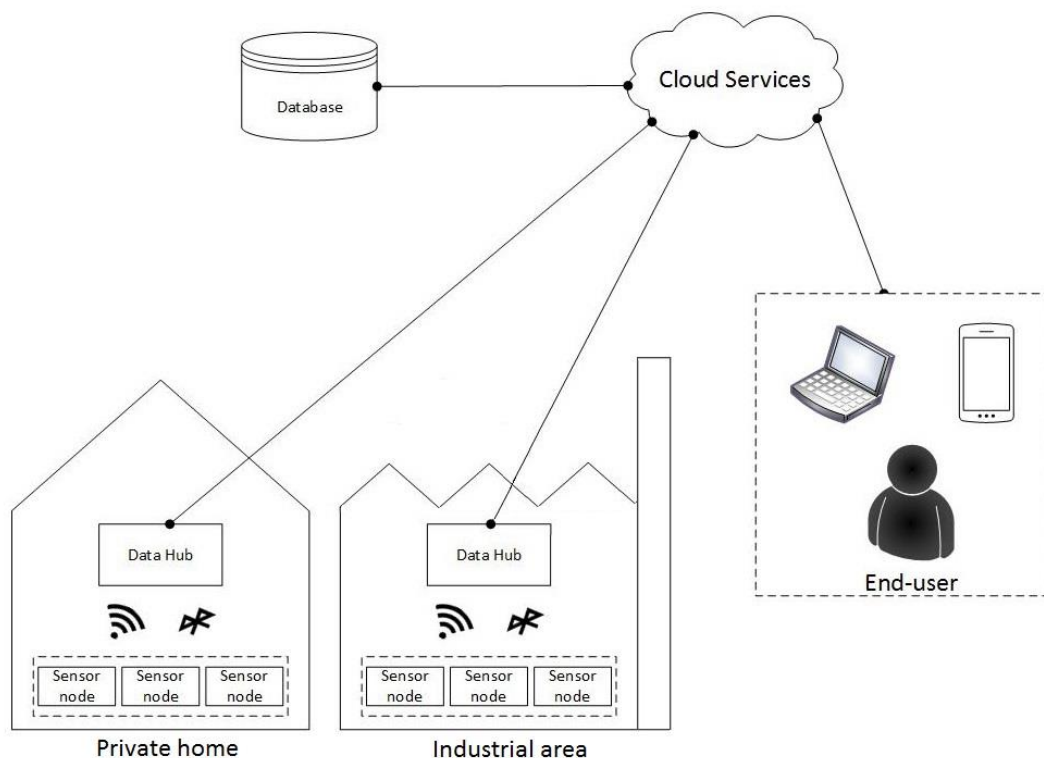
Figure 2-1 Overarching system

2.2 Redefined/adjusted tasks

The redefined/adjusted tasks based on area of interest and limitations are as following:

- Develop a simple embedded system for showing temperature readings from a temperature sensor which lights up a LED when the temperature exceeds a certain limit. The system must operate using LabVIEW LINX application uploaded to a Raspberry Pi 3, documented in a systematic procedure (user manual).
- Develop a simple LabVIEW application for transferring temperature readings from a Raspberry Pi 3 to a computer using bluetooth and WI-FI communication.
- Develop the sensor node further by implementing additional sensors such as:
  - Humidity (%)
  - Co2 level (ppm)
  - Pressure (Pa)
- Develop the data hub further by implementing the option to receive data from multiple sensor nodes and storing this in a local SQL server.
  Create a simple GUI for displaying the real-time data.
- Develop alternatives for sensor nodes using hardware such as:
  - Arduino Pro Mini
  - Arduino MKR1000
  This will imply the use of different programming languages.
- Develop alternatives for the data hub using hardware such as:
  - Intel NUC

- Exploit the use of LabVIEW Web Services to create a data management and monitoring interface, using software such as:
  - LabVIEW Data Dashboard
  - Simple HTML page
- Build prototypes for the sensor node and data hub wrapped in a commercial casing.
- Set up a test environment for trail/testing.
- Document hardware and software used for creating the sensor nodes and data hubs.
- Explain the communications protocols used for the sensor node and data hubs.
- Discuss briefly the challenges associated with security regarding IoT.
- Conduct research and tests to determine the power consumption for each type of sensor node and data hub.

3 Security

This chapter will briefly describe the security aspect of the IoT concept regarding the devices themselves and the network they are connected to. A security assessment towards the system developed as part of this master thesis will also be giving, highlighting the potential security risks.

3.1 IoT devices

The era of IoT, where more and more devices are digitally connected, including your home, office, cars and so forth [2]. With the wide deployment of Wi-Fi networks and the IoT growing in a fast pace, the security aspect is becoming more vital since the number of attractive targets for cybercriminals are increasing. Some frightening vulnerabilities found on IoT devices have brought IoT security up the stack of issues that need to be addressed as quickly as possible. Situations such as IoT baby monitors being hacked, internet-connected cars being compromised, wearables such as smartwatches being monitored, is just some examples of security breaches discovered recently.

Several measures are already being taken to prevent security breaches at the device level, and efforts of tackling major disasters before they happen. Security firms and manufacturers are working together to secure the IoT world before it spins out of control. Companies such as "Gemalto" is planning to use their experience from mobile payments to help secure IoT devices, offering their "Secure Element" (SE) technology. This technology uses a tamper-resistant component that are embedded into devices to enable advanced digital security via encryption of sensitive data. Other companies such as Microsoft is also entering the fray and has promised to add "BitLocker encryption" and "Secure Boot technology" to the Windows 10 IoT for devices and platforms such as Raspberry Pi. These two technologies encrypts the entire disk volume and secures the boot up procedure, which prevent device hijacking. Security updates for IoT devices is also a safety measure that can keep the devices secure, but proper safeguards must then be put in place to prevent updating interfaces from becoming security holes themselves.

3.2 IoT network

Security cannot only be tackled regarding IoT devices [2]. It is simply not enough to ensure that only the devices are secured from hijacking. Gateways that connect the IoT devices to a network must also be secured as well as the devices themselves. The IoT devices will normally go through a one-time authentication process, in contrast to a human-controlled device, making them perfect sources of infiltration into a company's network. Another security concern is also where the IoT data is being stored. Massive data breaches and data theft has been seen in the recent years by corporate hackers and industrial spies who rely on big data to make profits. Therefore, an increased effort to ensure that privacy of consumers and the functionality of businesses and corporations are being secured is also vital.

3.3 Security assessment

The use of sensor nodes and data hubs on a wireless network or by using bluetooth communication comes with a certain security risk.

The wireless network, which the nodes and hubs are connected to, will be secured by using wireless encryption (WPA2) and with a short range of around 20 meters indoors, as described in chapter 4.2. Direct possible security intrusions will in such case be towards the network (gaining access to the data stored) through an internet connection (indirectly), making it a breach on a higher level then emphasized in this thesis.

Bluetooth communication, for sending/receiving data as an alternative to Wi-Fi, is secured using "trusted devices" for exchanging data without asking permission and in same time decline other devices trying to connect. Using Service/Device-level security to protect bluetooth devices from unauthorized data transmissions and with the devices configured as "non-discoverable" helps avoiding other devices from connecting. The limited range of approximately 10 meters, in this case, also makes it hard for other users to connect directly to any of the devices, and again, the possible breach will be through an internet connection (indirectly), on a higher level then emphasized in this thesis.

The risk plot shown in Figure 3-1 with its impact and probability level, is created based on a scenario of an intruder who specifically want access this system. Therefore, the probability and impact is defined high in this case, but not in an overall risk view regarding IoT systems around the world.
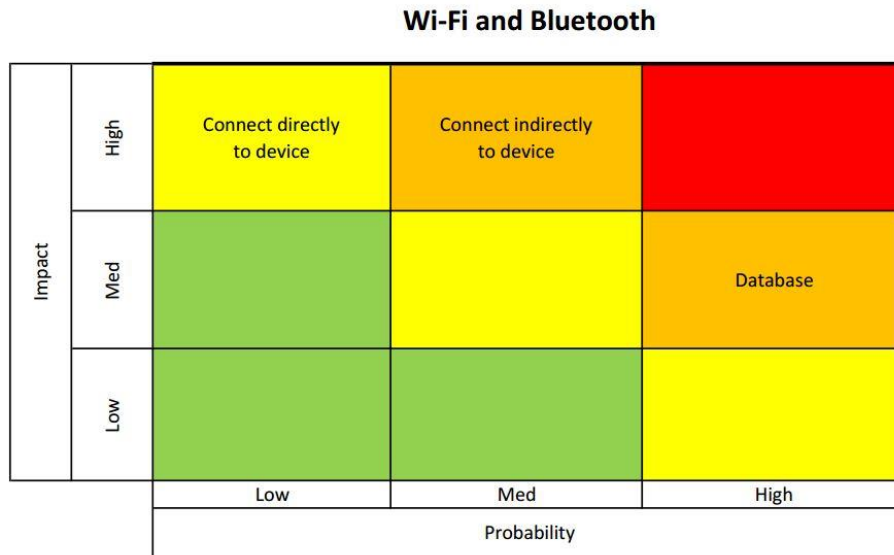

Figure 3-1 Risk plot for Wi-Fi and Bluetooth communication

4 Communication protocols

This chapter will give a general description regarding the different communication protocols used for the sensor nodes and data hubs. The Bluetooth and Wi-Fi protocol are used for transferring the measurement data between the sensor nodes and the data hubs, while the SPI and I2C are used for the components inside the sensor nodes.

4.1 Bluetooth

Bluetooth is a wireless standard for exchanging data over short distances (2.4GHz to 2.485GHz) from both fixed, mobile devices and personal area networks (PAN's) [3]. It was originally conceived as a wireless alternative to serial communication "RS-323" data cables, with the option of connecting several devices and in this way overcoming the problems of synchronization.

Bluetooth operates at frequencies between 2402MHz and 2480MHz (or 2400MHz and 2483.5MHz); with the inclusion of 2 MHz wide at bottom end an 3.5MHz wide at the top known as guard band (preventing interference). The data transmitted is divided into packets and is transmitted in one of the 79 designated bluetooth channels. Each channel has a bandwidth of 1 MHz and usually performs 800 hops per second.

Bluetooth communication protocol is a packet-based protocol with a master-slave structure, where one master may communicate with up to seven slaves. All the devices share the master's clock, and packets are exchange based on the ticks from this clock. The master transmits packets in even slots and deceives in odd slots, while the slave receives in even slots and transmits in odd slots. Due to the concept of using radio (broadcast) communication, there is no need for visual line of sight between the devices. However, there must be a quasi-optical (signals in the terahertz region of the electromagnetic spectrum) wireless path available. The range is power-class-dependent and the official classes have ranges as shown in Table 4-1. Bluetooth is mainly intended for portable equipment and a typical two devices connected with minimal configuration, while WI-FI is more suited for applications where high speeds are required, typically network access through an access node.

In this thesis, bluetooth communication protocol will be used for sending measurement data between the sensor node and the data hub, as an alternative to WI-FI.

Table 4-1 Official bluetooth classes [3]

Class	Max permitted power	Typical range
1	100 mW	~100 m
2	2.5 mW	~10 m
3	1 mW	~1 m
4	0.5 mW	~0.5 m

4.2 Wi-Fi

Wi-Fi is a technology that connects electronics devices to a wireless LAN (WLAN) network, mainly using the radio bandwidth of 2.4 GHz or 5GHz [4]. WLAN allows any device within its range to connect and access its resources, usually by entering a Wi-Fi password. The WLAN protocol is based on the 802.11 standards developed by the Institute of Electrical and Electronics Engineers (IEEE).

Most technical devices in today's society has a built-in Wi-Fi module and connects to the internet via a WLAN network. Access points (or hotspots) that the devices connect to, has a range of typically 20 meters indoors, but can be extended by using multiple overlapping access points. Wi-Fi is considered less secure than wired connection (Ethernet cable), since a potential intruder does not need a "physical" connection, and it is therefore developed several types of encrypted technologies (WEP, WPA, WPA2, etc.) to secure networks.

Wi-Fi communication protocol uses two-way radio communication to send data packets as a radio signal using a wireless adapter. A wireless router receives the radio transmission and decodes it back to data packets. The response (data packets returned) is then replied back in the same manner using radio signals. The signal range of the radio signal depends on the frequency band, radio power and antenna gain/type. A typical access point with a stock antenna might have a range of 100 meter, but with an external semi-parabolic antenna, the range can be up to 20 miles.

In this thesis, the devices will be indoors where it is expected that the range will be around 20 meters. The Wi-Fi communication protocol will be used for sending measurement data between the sensor node and the data hub, by connecting both device to the same router (WLAN). This will be the primary communication protocol used for exchanging data.

4.3 Serial Peripheral Interface (SPI)

The Serial Peripheral Interface (SPI) bus is a synchronous serial communication interface primarily used in embedded systems, for short distance communication [5]. The SPI interface developed by Motorola, are typically used in applications such as Secure Digital cards (SD, SDHC, and SDXC) and liquid crystal displays (electronic visual display). SPI device communicates in full duplex mode (communication in both directions simultaneously), using a master-slave architecture with a single master. Multiple slave device are supported through selection with individual "slave select" lines. Figure 4-1 shows an example of a "Single Master to Single Slave bus".
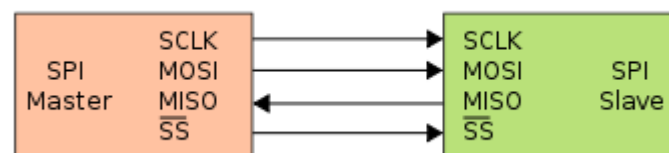
Figure 4-1 SPI bus example [5]

The data transmission begins with bus master configuring the clock frequency. The master then selects the slave device with a logic level 0 on the selected line. During each clock cycle, the master sends a bit on the MOSI line and the slave reads it. At the same time, the slave sends a bit on the MISO line that the master reads. Transmissions normally involve two shifts registers, one in the master and on in the slave. Data is usually shifted out with the most-

significant bit first and then a new least-significant bit into the same register. At the same time this occurs, data from the counterpart is shifted into the least-significant bit register. After the register bits have been shifted, master and slave have exchanged the register values. Transmission like this often consist of 8-bit words, but may also be 16-bit or 12-bit words. Figure 4-2 shows an example of a typical hardware setup using two shift registers to form an inter-chip circular buffer.
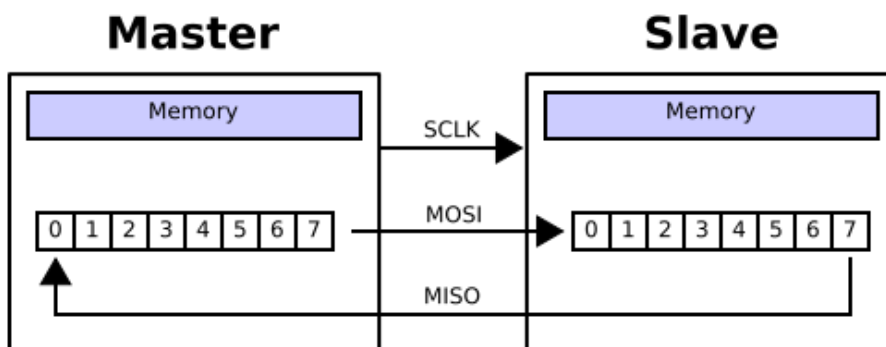
Figure 4-2 Shift registers forming an inter-chip circular buffer [5]

In addition to setting the clock frequency as mentioned, the master must also configure the clock polarity and phase with respect to the data. A timing diagram shown in Figure 4-3 represents a set of signals in a time domain containing multiple rows. This adds more flexibility to the communication channel between the master and slave and is often referred to as "modes", which are commonly numbered with CPOL as high order bit and CPHA as low order bit. Table 4-2 shows the numbering of the SPI modes.

In this thesis, the SPI communication protocol will be used for retrieving measurement data from an Analog/Digital converter chip MCP3002.
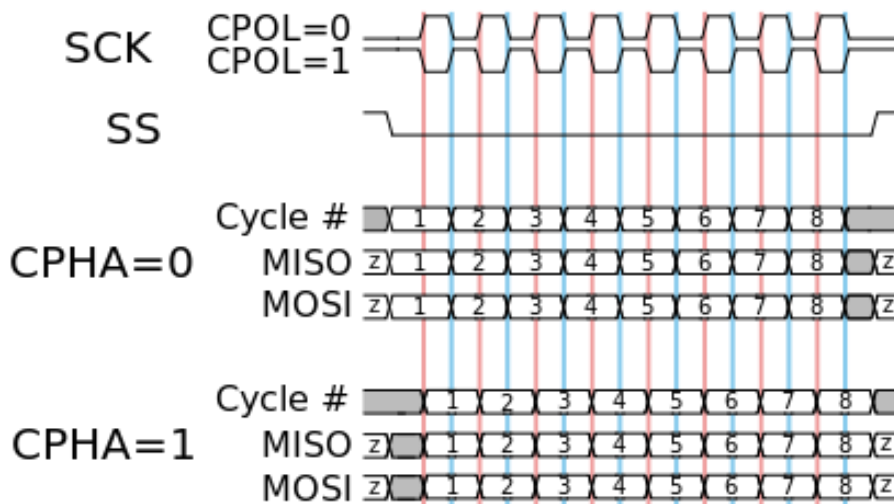
Figure 4-3 Timing diagram for clock polarity and phase [5]

Table 4-2 SPI modes numbering [5]

SPI Mode	Clock Polarity (CPOL/CKP)	Clock Phase (CPHA)	Clock Edge (CKE/NCPHA)
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	0

4.4 Inter-Integrated Circuit (I2C)

Inter-Integrated Circuit (I2C) is a "multi-master, multi-slave, single-ended, serial computer bus" used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication [6]. I2C uses only two bidirectional open-drain lines, Serial Data Line (SDA) and Serial Clock line (SCL), in addition with a "pullup" resistor. Typical voltage used for I2C are +5V or +3.3V. The I2C has a 7-bit or 10-bit address space (physical entity) and a bus speed of 100 Kbit/s in "standard" mode and 10Kbit/s in "low-speed" mode. The bit rates is quoted for the transaction between master and slave without the clock stretching. The maximum number of nodes is limited by the address space and the bus capacitance (400pF), thus restricting practical communication distance to a few meters. Due to relatively high impedance and low noise immunity, a common ground is required. Figure 4-4 shows an example of I2C interface with one master, three slave nodes and pullup resistors.
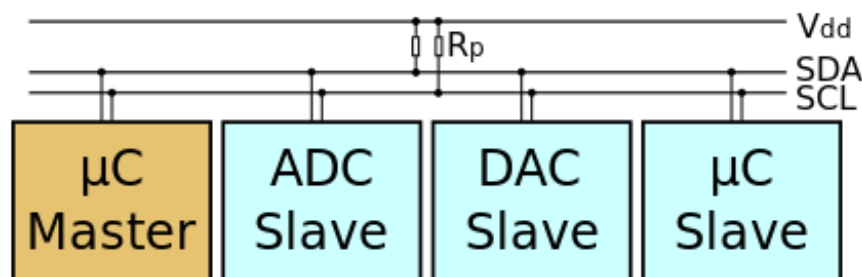
Figure 4-4 I2C interface with one master, three slave nodes and pullup resistors [6]

The bus has two roles for nodes: Master node is the node that generate the clock and initiates the communication with the slaves. Slave node is the node that receives the clock and responds when addressed by the master. The master is initially in "master transmit mode" and sends a start bit followed by a 7-bit address of the slave it wishes to communicate with, followed by a single bit representation for either write(0) or read(0) from the slave. If the slave is located on the bus, it will respond with an acknowledge bit (ACK-bit) for that particular address. If acknowledgement is received, the master will go into transmit or receive mode while the slave continues in its complementary mode either (receive or transmit). The address and the data bytes are sent as "most significant bit" first. The "start bit" is indicated by a "high-to-low" transition of SDA with SCL high, and the "stop bit" is indicated by "low-to-high" transition of SDA with SCL high. The rest of the transitions takes place with SCL low. Basic types of messages, each beginning with START and end with STOP, are:

"Master-transmitter to Slave-receiver", "Slave-transmitter to Master-receiver" and "Bi-directional (Read/Write) in same data transfer". Like the SPI communication protocol, I2C also use a timing diagram for data transfer as shown in Figure 4-5.

In this thesis, I2C communication protocol will be used for retrieving measurement data from I2C supported components such as the digital thermal sensor TC74A0.
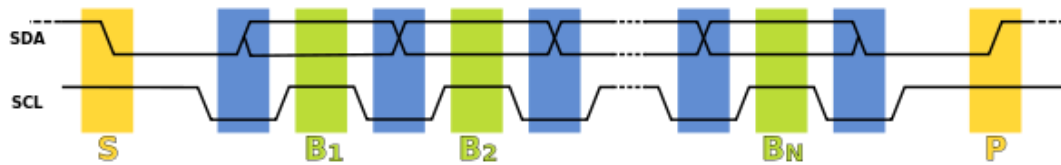

Figure 4-5 Timing diagram (Data transfer sequence) [6]

5 Software

This chapter will describe the software used for developing the applications for the sensor nodes and data hubs. The subchapters will give a general explanation for each of the software used, with figures showing examples of the interface and syntax.

5.1 LabVIEW

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) is a system-design platform and development environment for visual programming language, created by National Instrument [7]. The programming language (graphical language) is named "G" and was originally released for Apple computers. LabVIEW is most commonly used for data acquisitions, instrument control and industrial automation, supported on variety of OS, including Microsoft Windows, UNIX, Linux, and Mac OS. The graphical approach allows nonprogrammer to build programs by dragging and dropping virtual representations of lab equipment.

The graphical language is a dataflow programming language, where the execution is determined by the structure of a graphical block diagram. Different function nodes are connected by drawing wires that propagate variables that nodes can execute as soon as all its input data is available. G code can execute inherently in parallel by exploiting a built-in scheduler. LabVIEW creates what is known as virtual instruments (VIs) that consists of the components: "Block diagram", "Front panel" and "Connector panel". The front panel is built using controls and indicators, where controls are inputs and indicators are outputs. The block diagram contains the graphical source code where the objects in the front panel appears and where the structure/functions are created. The connector panel is used for representing the VIs created and to be used in the block diagram, most commonly associated with SubVIs. An example of a front panel and block diagram can be seen in Figure 5-1 and Figure 5-2.

To sum up some of the main features regarding advantages and disadvantages using LabVIEW:

- Includes extensive support for interfacing with other devices
- Graphical code compiling
- Parallel programming
- No need for programming skills
- Dataflow programming is not preferred by skilled programmers
- Run-time engine is required to create standalone applications
- Applications run slower than hand coded native languages such as C
- Small applications still have to start runtime environment (large and slow task)
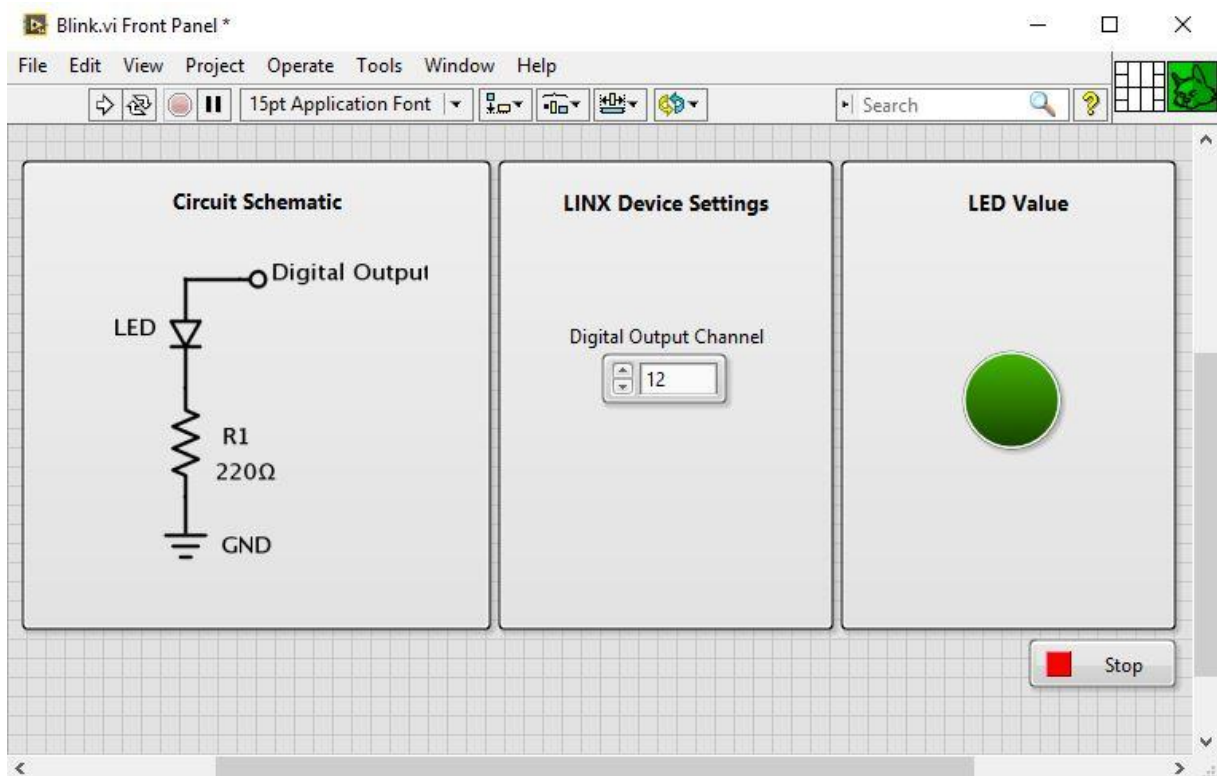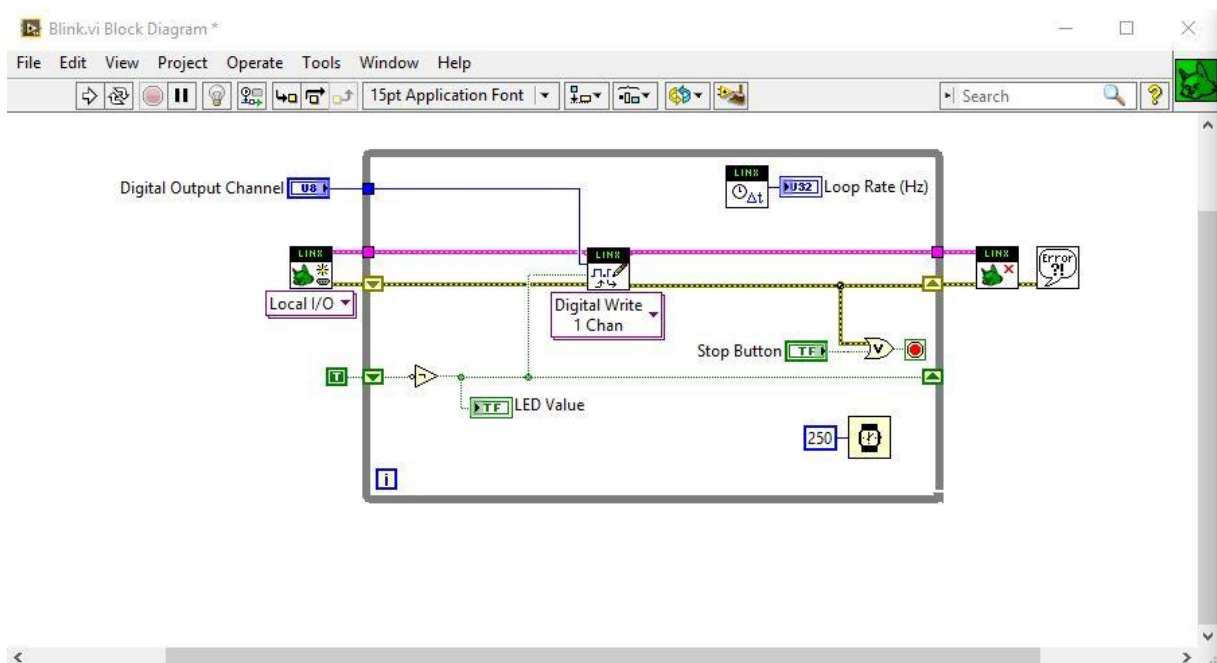
Figure 5-1 LabVIEW example of a Front panel

Figure 5-2 LabVIEW example of a Block diagram

5.1.1 Data Dashboard

Data dashboard, seen in Figure 5-3, is a program available for mobile devices that allow users to create custom portable views (GUI) for LabVIEW applications [8]. It is available for Apple iPad and Android tablets and is free of charge.
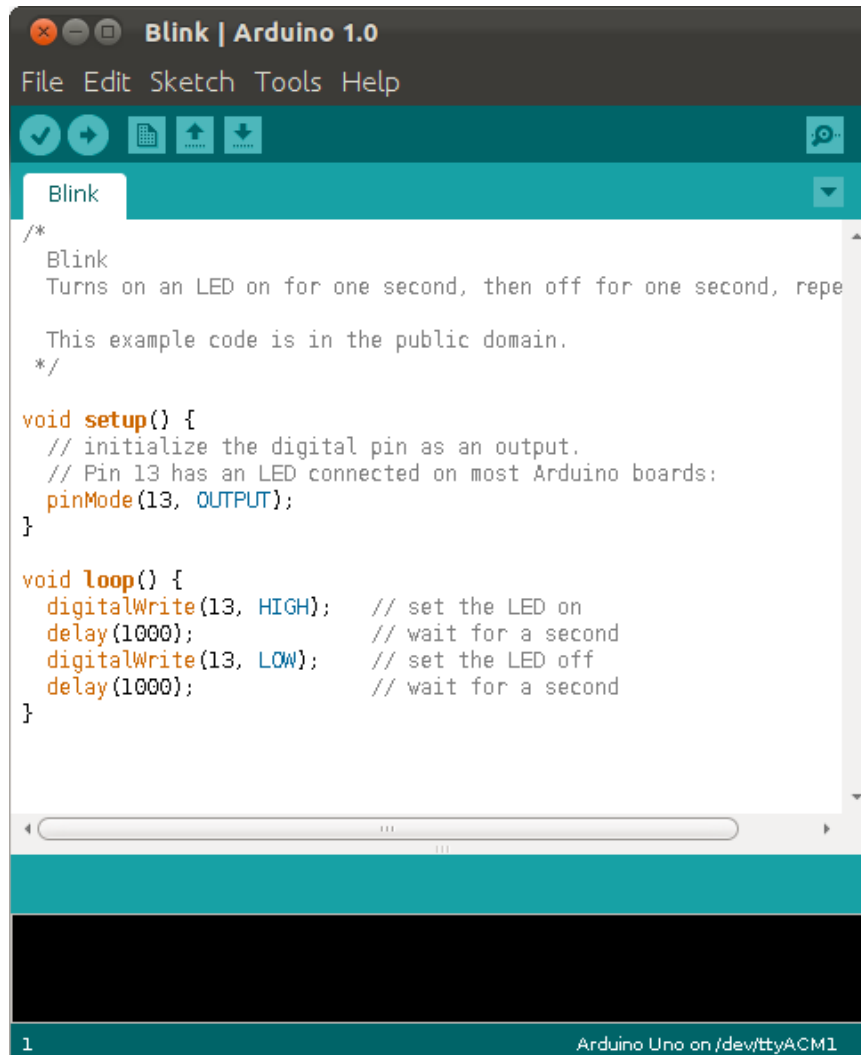
The application works in the same way as LabVIEW development by using graphical approach, letting the user drag and drop controllers and indicators into a Front panel. These controllers/indicators can be connected to a LabVIEW application through WI-FI using Web services or local variables. A range of different controllers and indicators can be implemented like for example charts, gauges, textboxes, LEDs, etc.

Figure 5-3 LabVIEW Data Dashboard [9]

5.2 Arduino IDE

Arduino hardware are programed using Arduino integrated development environment (IDE), which is a cross-platform application written in Java language [10]. It is designed for newcomers unfamiliar with software development and includes a code editor with features such as syntax highlighting, brace matching and automatic indications. A simple one-click mechanism is provides for compiling and uploading a program (sketch) to any Arduino board. The Arduino IDE supports the languages C and C++ and also supplies a software library called Wiring, providing common input/output procedures. Figure 5-4 shows a typical Arduino C/C++ sketch consisting of the two functions "setup()" and "loop()". When the code is compiled, it is converted into a text file in hexadecimal uploaded to the board's firmware.

Figure 5-4 Arduino IDE sketch example [10]

The Arduino IDE is a free to use software for Arduinos open-source hardware, with the advantages and disadvantages:

- Easy to use software with all Arduino boards supported
- Huge community (lots of examples online)
- Large assortment of included libraries (for different components)
- Easy debugging environment
- Libraries are optimized for easy usage, not efficiency
- Not fully suitable for industrial use
- Arduino IDE is very limited for skilled programmers

5.3 SQL

Structured Query Language (SQL), seen in Figure 5-5, is a "special-purpose programming language" designed for managing data in a relation database management system known as RDBMS [11]. SQL derives from the theoretical foundation of "relational model" with table and queries resulting in lists of rows, where the same row may occur multiple times and the order of rows can be employed in queries. The languages used in SQL is subdivided into server language elements such as "Clauses, Expressions, Predicates, Queries and Statements. The use of operators makes it possible to manipulate individual data items and returning a desired result. These operators are often part of a query, which is the most common operation in SQL. In summary, one can say that SQL is a mix of functionality, power and relative ease of use for most correlations, and the most common advantages and disadvantages are [12]:

- High speed
- No coding required
- Easy to learn
- Difficulty in interfacing (more complex)
- Development is not community driven
- Functionality heavily dependent on add-ons
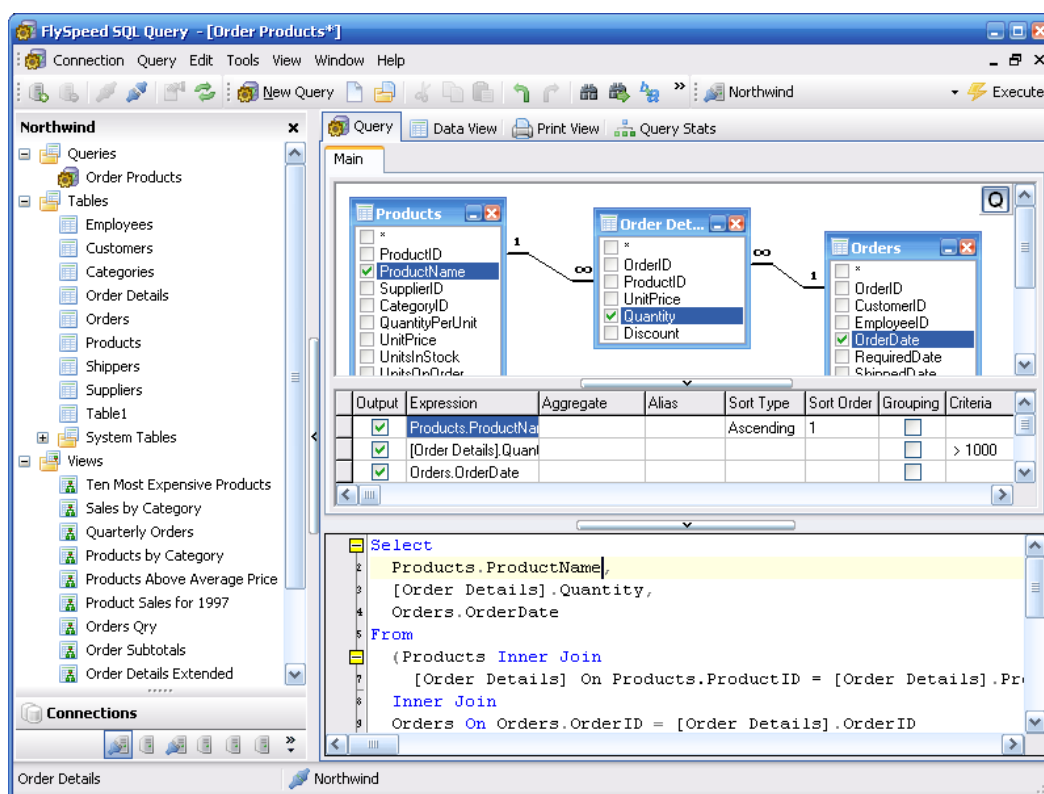
Figure 5-5 SQL Program example [13]

6 Hardware

This chapter will describe the hardware used for developing the sensor nodes and the data hubs. The subchapters, describing the hardware specifications, are separated into sensors, microcontrollers/computers, communication modules and mobile devices.

6.1 Sensors

In order to develop the sensor nodes, a variety of sensors was considered for measuring:

- Temperature (°C)
- Humidity (%)
- Barometric pressure (mbar)
- Co2 level (ppm)

Each type of sensors had to be supported by the microcontroller/computer and selected based on the criteria's:

- Functionality
- Measurement range
- Accuracy
- Communication protocol
- Price (Per piece)

The idea of the build-up for the sensor nodes consisting of the sensors with the microcontroller/computer, can be seen in Figure 6-1. The measurement sensors that are selected are described in subchapter 6.1.1 to 6.1.6, giving a brief overview of the most important details and where they are implemented.
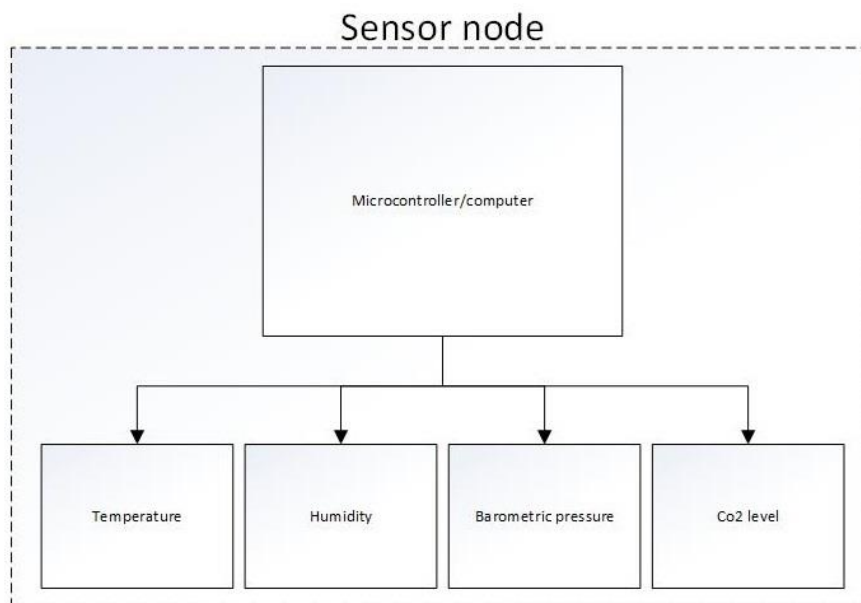
Figure 6-1 Build-up for the sensor nodes

6.1.1 Linear Active Thermistor (MCP9700AE)

Linear active Thermistors, shown in Figure 6-2, was used in the first version sensor node for measuring temperature by converting temperature to a proportional analog voltage. It is a low-cost, low-power sensor with an accuracy of ±2.0°C from 0°C to +70°C with a consumption (operating current) of 6 µA [14].

Figure 6-2 MCP9700A [15]

Unlike resistive sensors, such as thermistors mentioned in subchapter 6.1.2, it does not require an additional signal-conditions circuit. The temperature coefficients scale is 1°C/bit resolution for an 8-bit ADC with reference voltage of 2,5V. The sensor is also immune to the effects of parasitic capacitance and can therefore drive large capacitive loads. Temperature range is from - 40°C to + 125°C with a slope of 10.0mV/°C and DC offset of 500mV. The main features summed up for Linear Active Thermistors:

- Linear Temperature Slope
- Very low Operating Current
- No external Components Required

For more details, see datasheet in reference [14].

6.1.2 Non-Linear Thermistor (NTCLE100E3)

Non-Linear Thermistors (NTC), shown in Figure 6-3, was used in the first version sensor node as an alternative to the MCP9700AE. The NTC is a temperature-sensing element made of semiconductor material (sintered) for measuring small changes in temperature with a large change in resistance [16].

Figure 6-3 NTCLE100E3 [17]

NTC Thermistors are non-linear, meaning that the resistance characteristics alters with the temperature. As the temperature rises the resistance decreases. This manner is related to the constant β (beta), measured in Kelvin (°K).

For calculating the resistance at different temperatures, the "Steinhart & Hart equation", shown in equation (6.1), is optimal since it provides a closer approximation to the actual temperature than simpler equations [18].

$$\frac{1}{T} = A + B * \ln(R) + C[\ln(R)]^3 \tag{6.1}$$

Where T is the temperature [Kelvins], R is the resistance at T [ohm] and A, B, C is the Steinhart-Hart coefficients. The NTC used in this case, has the color code brown, black and orange. This implies that the resistance is 10 000 Ω at 25°C and that the beta value is 3977K (±0.75) between temperature endpoint 25°C and 85°C. Temperature range is - 40°C to + 125°C with a dissipation factor of 7.0mW/K. The main features summed up for the NTC Thermistor are:

- Good accuracy over a wide temperature range
- High stability over a long life
- Excellent price/performance ratio

For more details, see datasheet in reference [19].

6.1.3 Digital Thermal Sensor (TC74A0)

The digital thermal sensor, shown in Figure 6-4, was used in the further developed Raspberry Pi 3 sensor node and is a digital temperature sensors that is particularly suited for low cost and small form factor applications. It uses I2C communication and sends the temperature data as an 8-bit digital word. The temperature resolution is 1 °C with an 8.0 samples/sec conversion rate. During normal operation, the quiescent current is 200µA [20].

Figure 6-4 TC74A0 [21]

The operating temperature is - 40°C to + 125°C, with an accuracy of ±2.0°C from +25°C to +85°C and ±3.0°C from 0°C to +125°C. In temperature data registers, each unit value (two's complement) represent one degree Celsius, such that a reading of 0000 0000 corresponds to 0°C. The main features summed up for the digital thermal sensor are:

- Simple SMBus/I2C interface
- Low power consumption
- No need for calibration

For more details, see datasheet in reference [22].

6.1.4 Gas sensor (MQ-135)

The gas sensor, shown in Figure 6-5, is used in the MKR1000 sensor node and is a stable low cost electro-chemical sensor suitable for a wide range of gasses, including NH3, NOx, alcohol, benzene, smoke and Co2 [23]. It comes as a complete module with a small built-in heater. The analog output voltage is controlled by the concertation (ppm) of the measured gas.

Figure 6-5 MQ-135 [24]

The operating range is from 10ppm to 1000ppm, with sensor resistance depending on the gas measured. The output is an analog voltage controlled by the concentration of the gas measured. The main features summed up for the gas sensor are:

- Fast response
- High sensitivity
- Wide detection scope
- Stable and long life
- Simple drive circuit

For more details, see datasheet in reference [25].

6.1.5 Co2 sensor (MH-Z19)

The Co2 sensor shown in Figure 6-6, originally planned to be used for the Raspberry Pi 3 sensor node, is small size infrared gas module using non-dispersive infrared (NDIR) principle to detect Co2 [26]. It has a built-in temperature sensor for temperature compensation and has digital and analog voltage output.

Figure 6-6 MH-Z14 [26]

The operating range is 0ppm to 2000ppm with an accuracy of ±50ppm for Co2. The output from the sensor can be measured using PWM or UART and the output is controlled by the measured concentration of Co2. The main features summed up for the Co2 sensor are:

- High sensitivity, high resolution
- Low power consumption
- Easy interface (UART/PWM)

- Temperature compensation (Linear output)
- Good stability
- Log lifespan

For more details, see datasheet in reference [27].

6.1.6 Multi sensor BME280

The multi sensor BME280 shown in Figure 6-7 is used on the MKR1000 sensor node, and is the next-generation sensors from Bosch for measuring temperature, humidity and pressure with low altitude noise and fast conversion time. The sensor has built-in 3.3V regulator, I2C level shifter and pull-up resistors [28].

Figure 6-7 BME280 [29]

The operating temperature is -40°C to + 85°C with an accuracy of ±1.0°C. The operating humidity is 0% to 100% RH with an accuracy of ±3% (RH), and the operating pressure is 300hPa to 1100hPa with accuracy of ±1hPa. The chip uses I2C 7-bit address 0x76 [Hex] and takes measurements at less than 1mA, with idle state less than 5µA. The main features summed up for the multi sensor are:

- Multi sensor (Humidity, Pressure and Temperature)
- I2C/SPI interface
- Low power consumption
- Low noise measurements
- Factory-calibrated

For more details, see datasheet in reference [30].

6.2 Microcontroller/computer

The microcontrollers/computers for the sensor nodes and data hubs have mostly the same I/O module, built-in transceivers and programming language. However, there are some variations when it comes to firmware setup. During the following subchapters, a brief overview of the most important details and where they are implemented, will be given. The most suited microcontroller/computer is selected based on the criteria's:

- Functionality
- Inputs/Outputs
- Communication protocols
- Size
- Price

6.2.1 Arduino MKR1000

Arduino MKR1000 shown Figure 6-8 is used in the MKR1000 sensor node and is a powerful board based on the Arduino Zero combined with a WI-FI shield [31]. The microcontroller is based on the Atmel ATSAMW25 SoC (System on Chip) which is part of the SmartConnect family of Atmel Wireless devices. The design includes a Li-Pi charging circuit, allowing it to run on both battery power and external power supply (5V). When running on external power supply, the battery is charged up to 4.2V with a current half the nominal capacity. The chip is programmed by default to a maximum of 4 hours charging time before going into automatic sleep mode. Switching between power sources happens automatically when one of the power source are removed. The microcontroller is programmed using Arduino IDE.

Figure 6-8 Arduino MKR1000 [32]

The microcontroller has 8 specific digital I/O pins, 12 PWM support I/O pins, 7 analog inputs and 1 analog output. There is also support for different communication protocols, such as I2C, SPI, UART, etc. The WI-FI module is a low power Cryptochip for secure communication with SHA-256 certification. Table 6-1 shows the technical specifications for the microcontroller.

Table 6-1 Technical specifications for MKR1000 [31]

Microcontroller	SAMD21 Cortex-M0+ 32bit low power ARM MCU
Board Power Supply (USB/VIN)	5V
Supported Battery(*)	Li-Po single cell, 3.7V, 700mAh minimum
Circuit Operating Voltage	3.3V
Digital I/O Pins	8
PWM Pins	12 (0, 1, 2, 3, 4, 5, 6, 7, 8, 10, A3 - or 18 -, A4 -or 19)
UART	1
SPI	1
I2C	1
Analog Input Pins	7 (ADC 8/10/12 bit)
Analog Output Pins	1 (DAC 10 bit)

External Interrupts	8 (0, 1, 4, 5, 6, 7, 8, A1 -or 16-, A2 - or 17)
DC Current per I/O Pin	7 mA
Flash Memory	256 KB
SRAM	32 KB
EEPROM	no
Clock Speed	32.768 kHz (RTC), 48 MHz
LED_BUILTIN	6
LED_BUILTIN	6
Length	80 mm
Width	55 mm
Weight	32 gr.

6.2.2 Arduino Pro Mini

Arduino Pro Mini shown in Figure 6-9, used as an alternative to MKR1000, is a small size board microcontroller based on ATmega328 [33]. It is designed for semi-permanent installation in objects and comes in two versions, 3.3V 8MHz and 5V 16MHz. It is considered one of the best microcontroller for applications where power consumption, low price and support for various communication protocols, is important. Unlike the Arduino MKR1000, this microcontroller does not have a built-in Wi-Fi shield.
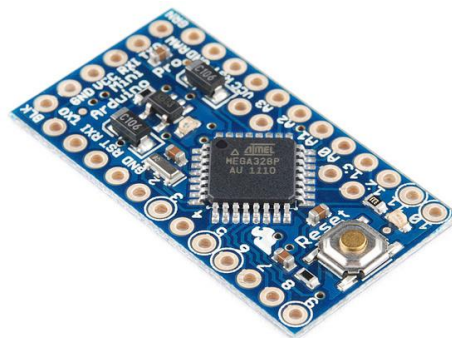
Figure 6-9 Arduino Pro Mini [34]

The board is powered with an FTDI cable or breakout board connected to a six-pin header. There is an integrated voltage regulator in the board, so it can accept voltage up to 12V. The Arduino Pro Mini is programed the same way as for the Arduino MKR1000, using Arduino IDE. The board has 14 digital I/O pins (6 of them with PWM), 6 analog inputs. There is also support for different communication protocols, such as I2C, SPI, UART, etc. Table 6-2 shows the technical specifications for the microcontroller.

Table 6-2 Technical specifications for Arduino Pro Mini [33].

Microcontroller	ATmega328
Board Power Supply	3.35 -12 V (3.3V model) 5 - 12 V (5V model)
Circuit Operating Voltage	3.3V or 5V (depending on model)
Digital I/O Pins	14
PWM Pins	6
UART	1
SPI	1
I2C	1
Analog Input Pins	6
External Interrupts	2
DC Current per I/O Pin	40 mA
Flash Memory	32KB of which 2 KB used by bootloader *
SRAM	2 KB *
EEPROM	1 KB
Clock Speed	8 MHz (3.3V versions) 16 MHz (5V versions)

6.2.3 Raspberry Pi

The Raspberry Pi, shown in Figure 6-10, is a single-board computer capable of running different types of operating systems (OS) and is available in three different versions (model A, B and B+) [35]. All the modules features a Broadcom system on chip (SoC), including an ARM compatible central processing unit (CPU) and graphics-processing unit (GPU). The OS and programs are all stored on an external micro SD card and applications such as Python, MatLAB, LabVIEW, C#, etc. are supported.

Figure 6-10 Raspberry Pi [35]

The newest version, Raspberry Pi 3 Model B, is the third generation Raspberry Pi, with integrated WI-FI and bluetooth module. It has 40 GPIO (General-purpose input/output), UART, I2C and SPI support. It is also possible to set up remote access, making it ideal for data hub applications. Table 6-3 shows the technical specifications for the computer.

Table 6-3 Technical specifications for Raspberry Pi 3 [36]

SoC	Broadcom BCM2837
CPU	4× ARM Cortex-A53, 1.2GHz
GPU	Broadcom VideoCore IV
RAM	1GB LPDDR2 (900 MHz)
Networking	10/100 Ethernet, 2.4GHz 802.11n wireless
Bluetooth	Bluetooth 4.1 Classic, Bluetooth Low Energy
Storage	microSD
GPIO:	40-pin header, populated
Ports:	HDMI, 3.5mm analogue audio-video jack, 4× USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)

6.2.4 Intel NUC

The Intel NUC (Next Unit of Computing), Figure 6-11, is a small-form-factor personal computer from Intel [37]. The computer is a so called "Barebone kit" consisting of the board, plastic case with a fan and an external power supply. The small design and its powerful hardware makes it ideal for usage where size and compactness matters. The NUC is customizable, letting the user choose what hardware to use, and is constructed for mounting on a wall or on the back of a computer screen. It has a great reputation for running quietly without overheating.

Figure 6-11 Intel NUC [38]

Intel NUC comes in various models with different hardware. The module dc53427HYE, which is selected for this thesis, has the technical specifications as shown in Table 6-4.

Table 6-4 Technical specifications for Intel NUC [39]

Processor	3rd generation Intel Core i5-3427U processor with active fan heat sink
Memory	Two SO-DIMM slots supporting up to 16GB of 1333/1600MHz DDR-3 memory
Chipset	Intel QS77 Express Chipset
Display	Dual Mini-DisplayPort 1.1.a ports HDMI port supporting 1.4a Supporting triple independent display capability
Audio	Intel High Definition Audio subsystem (8-channel (7.1) digital audio via HDMI 1.4a output and via one DisplayPort 1.1. connector
LAN support	Intel I218LM Gigabit Ethernet Controller
Peripheral interfaces	One USB 3.0 port (front panel) Two USB 3.0 ports (back panel) Two USB 2.0 ports (internal headers)
Expansion capabilities	Full size Mini PCI Express with mSATA support Half size PCI Express

For more details, see technical product specification in reference [39].

6.3 Communication modules

Some of the microcontrollers/computers mentioned in subchapter 8.2 do not have built-in Wi-Fi and Bluetooth modules, and therefore need extra modules which are described in this subchapter.

6.3.1 Bluetooth module (HC-05)

The bluetooth module HC-05, shown in Figure 6-12, is a low-cost single chip radio 2.4GHz system with an enhanced data rate of 3Mbps [40]. It is fully compliant with Bluetooth v2.0 hardware/devices and operates on 5V with a default baud rate of 9600.

Figure 6-12 HC-05 [40]

The module is based on the Cambridge Silicon Radio BC417 2.4 GHz bluetooth radio chip, with an external 8Mbit flash memory and works well with Arduino and other microcontrollers. The main features summed up for the bluetooth module are:

- Low Power operation
- Integrated antenna
- UART interface for changing baud rate
- Up to +4dBm RF transmitting power

For more details, see datasheet in reference [41].

6.3.2 Wi-Fi module (ESP8266)

The Wi-Fi module ESP8266, shown in Figure 6-13, is a low-cost SOC with 1MB built-in flash memory and integrated TCP/IP protocol stack [42]. The module operates on 3.3V and is capable of either hosting or offloading a Wi-Fi network application.
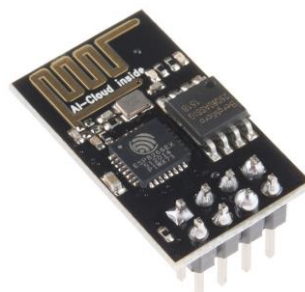
Figure 6-13 ESP8266 [42]

The module is pre-programmed with an AT command set firmware, making it ideal for Arduino devices. Its powerful on-board processing and storage capability makes the module integrable with sensors through its GPIOs.

The main features summed up for the Wi-Fi module are:

- Integrated TCP/IP protocol stack
- Integrated flash memory
- Standby power consumption
- Supports P2P and Soft-AP

For more details, see datasheet in reference [43].

6.4 Mobile device

For accessing the measurement data, using Web Services, only one mobile device was selected based on the support for LabVIEW Data Dashboard.

6.4.1 Apple iPad

The iPad, designed by Apple Incorporation, is an iOS-based line tablet computer with built-in WI-FI and bluetooth support [44]. The user interface is a multi-touch screen, including a virtual keyboard. There are six versions of the iPad and continues developing iOS versions. There are large selection of applications that can be downloaded to the iPad using Apple store, such as LabVIEW Data Dashboard 2.0 described in subchapter 5.1.1. The general technical specifications for the iPad model 2, which is used, are shown in Table 6-5.

Table 6-5 Technical specifications for iPad model 2 [45]

Models	WI-FI or WI-FI + 3G
Storage	16GB, 64GB or 128GB
Chipset	1GHz dual core Apple A5 custom-designed
Display	9.7 inch LED-backlit glossy widescreen (Multi-touch display)
Sensors	Camera (front and back) Accelerometer Ambient light sensor Gyroscope
Audio	3.5mm minijack Built-in speaker Microphone

For more details, see technical product specifications in reference [45].

7 Power options

This chapter will describe the power options for the sensor nodes and data hubs, separated into three subchapters. First, the fixed power supply, then the battery power supply and then estimated power consumption.

7.1 Fixed power supply

The MKR1000 and Raspberry Pi 3 sensor node and the Raspberry Pi 3 data hub have a 5 volt DC input, normally powered by an USB port. As a fixed power supply, a USB port is not considered as a suitable option, so the power grid will be used as a fixed power supply.

In Norway, the power grid is 230V 50Hz AC with a high level of reliability. The sensor nodes uses 5V DC as mentioned, which requires a power conversion/step down from 230V to 5V and from AC to DC. This conversion can easily be achieved by using a power adapter with an output of 5V DC 2.5A.

The Intel NUC data hub also needs a conversion from 230V 50Hz AC, but instead down to 19V DC. The computer is delivered with a power adapter with input range 100V to 240V AC, 50Hz to 60Hz and an output of 19V DC 3.42A.

7.2 Battery power supply

The option of running the MKR1000 and Raspberry Pi 3 on battery power is an alternative option, which makes the nodes more flexible/portable. However, power consumption because of hardware and software used, can make this option less optimal. The option of integrated charging circuits makes the transition between fixed and battery powered supply seamless.

MKR1000 offers an integrated charging circuit, Figure 7-1, fixed at 350mA with a charging time of four hours [46]. LiPo batteries between 1400mAh to 1500mAh is considered maximum, giving it an application run time of approximately 8 hours. This estimation is based on an application running continuously with an average current of 120mA. With energy saving principles such as sleep-mode and current regulators, the battery supply can last much longer.
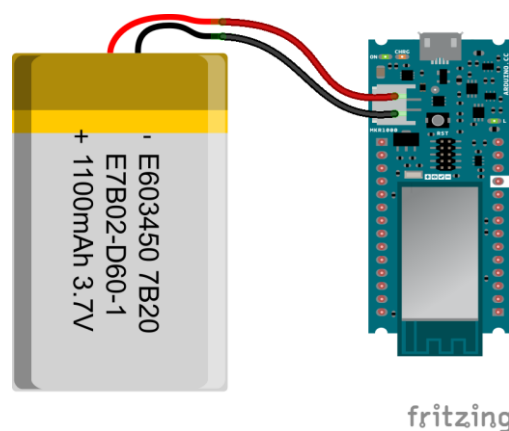

Figure 7-1 MKR1000 with LiPo battery [46]

Raspberry Pi 3 does not have an integrated charging circuit as the MR1000, but there are "battery expansion board", Figure 7-2, that can be implemented using the same micro USB

connection as when connected to a fixed power supply [47]. This makes the option easy to implement without having to solder or use other components in order to make the connection between the battery and the Raspberry Pi 3. The battery is 3800mAh Lithium based with output of 5.1V ±1V, giving it a lifetime of approximately 9 hours (running only the Raspberry Pi 3).
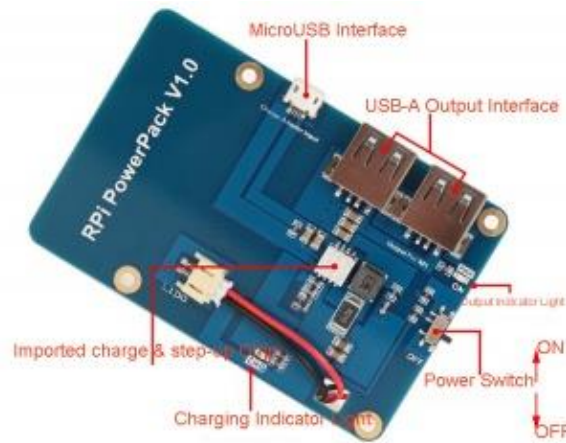
Figure 7-2 Battery expansion board for Raspberry Pi 3 [47]

The Intel NUC does not have an integrated charging circuit, but there is an option of using UPS. The "Fit-Uptime" UPS shown in Figure 7-3 from CompuLab, offers up to three hours of backup designed for Intel's NUCs [48]. Since there are no room for internal UPS circuits in the NUC, a compact, low-cost DC-powered UPS with instant switch between fixed and battery supply ensures that the NUC does not reset if a power failure occurs. This option can be seen suitable as a battery power supply for making the Intel NUC data hub more flexible in special events, but not as an option for making it more portable.

Figure 7-3 Fit-Uptime UPS from CompuLab [48]

7.3 Estimated power consumptions

The power consumption for electrical devices depends on the operations they are performing. In this case, components and program/script will determine the total consumption of each type of sensor node and data hub. Measurements of voltage, current and power will be documented during subchapter 10.3, but a general power consumption will be given now based on test data from online sites (performed by uncertified testing agencies). Note that the estimate power consumption is based on a midpoint value between lowest and highest current.

7.3.1 Arduino MKR1000

The Arduino MKR1000 power consumption based on a test from Arduino.cc, shows that the micro-controller itself use about 20mA in ideal state [46]. When running a sketch using the built-in Wi-Fi, the average current is around 120mA. The total current (I_T) can be found using the equation (7.1):

$$I_T = MKR1000_{Idling} + \frac{WiFi_{average} - MKR1000_{Idling}}{2} \tag{7.1}$$

With an idling state of 20mA and Wi-Fi average of 120mA, the total current, equation (7.2), becomes:

$$I_T = \left(20mA + \frac{120mA - 20mA}{2}\right) = 70mA \tag{7.2}$$

By using ohms law in equation (7.3):

$$P = U * I \tag{7.3}$$

With U = 5V and I = 0.07A, the total power consumption, equation (7.4), becomes:

$$P = 5V * 0.07A = 0.35W \tag{7.4}$$

7.3.2 Raspberry Pi 3

The Raspberry Pi 3 power consumption depends on the operations it is preforming. A test found from raspi.tv, seen in Figure 7-4, shows how different operations determines the consumption [49]. A simple test of the built-in Wi-Fi by downloading a 16mb file several times, adds an additional 40-100mA to the power consumption.

	Zero /mA	A+ /mA	A /mA	B+ /mA	B /mA	Pi2B /mA	Pi3B /mA
Idling	100	100	140	200	360	230	230
Loading LXDE	140	130	190	230	400	310	310
Watch 1080p Video	140	140	200	240	420	290	290
Shoot 1080p Video	n/a	230	320	330	480	350	350

Figure 7-4 Power consumption for all Raspberry Pi models [49]

Estimated power consumption based on available data from raspi.tv running graphical code using Wi-Fi, can be found using equation (7.5):

$$I_T = (Pi3_{Idling} + \frac{Pi3_{Loading\ LXDE} - Pi3_{Idling}}{2}) + (WiFi_{Low} + \frac{WiFi_{High} - WiFi_{Low}}{2}) \qquad (7.5)$$

With an idling current of 230mA, LXDE loading current of 310mA and Wi-Fi High/Low current of 100mA and 40mA respectively, the total current, equation (7.6), becomes:

$$I_T = \left(230mA + \frac{310mA - 230mA}{2}\right) + \left(40mA + \frac{100mA - 40mA}{2}\right) = 340mA \qquad (7.6)$$

Using same ohms law as mentioned earlier but with U = 5V and I = 0.34A, the total power consumption, equation (7.7), becomes:

$$P = 5V * 0.34A = 1.7W \qquad (7.7)$$

7.3.3 Intel NUC

The Intel NUC (dc53427HYE) is ranked as 39 out of 160 computers tested by The Power Consumption Database (TPCDB), which is an international site for collecting worldwide electricity usage data [50]. The computer tested is equipped with Intel Wi-Fi network card, 8 gigabyte of memory, mSATA solid-state disk and Windows 8 as OS. The measured power consumption shows an average power consumption of 8.5W at normal/Idle operation and a 24.5W at max (peaks).

8 Selected components

This chapter present the selected components needed for building the prototypes for the sensor nodes and data hubs. The first subchapter describes the selected sensors based on the criteria's in subchapter 6.1, while the next subchapter describes the selected microcontrollers/computers based on the criteria's in subchapter 6.2. The final subchapter presents the order lists for each type of sensor node and data hubs, with a price estimation for each component.

8.1 Sensors

The first sensor selected is the multi sensor "BME280" for temperature, humidity and barometric pressure readings, because of its easy I2C communication protocol. It contains three sensors, one for measuring temperature, one for measuring humidity and one for measuring barometric pressure. For more information, see chapter 6.1.6.

The second sensors selected is the "MQ135" for Co2 level readings, suitable for a wide range of gasses, due to its good sensitivity and low humidity/temperature dependency. It is also one of the cheapest options available, which offers strait forward readings using analog output. For information, see chapter 6.1.4.

The third sensor selected is the "TC74" for temperature reading. It uses I2C communication and is particularly suited for low cost and small form factor applications. It has a low power consumption and do not need any calibration. For more information, see chapter 6.1.3.

8.2 Microcontrollers/computers

Raspberry Pi 3 is one of the selected microcontrollers/computers due to its built-in support for WI-FI and Bluetooth. The option of running different types of OS with applications such as LabVIEW Run-Time is a huge advantage since it can run as an embedded system. The price is also fair considering it is a "credit card-sized single-board computer". For more information, see chapter 6.2.3.

Genuino MKR1000 is also one of the selected microcontrollers/computers because of its built-in support for WI-FI and its built-in Li-Po charging circuit, allowing it to run on battery power and external 5v. Its small size makes it ideal for creating small sensor nodes. For more information, see chapter 6.2.1.

8.3 Order lists

The order lists, shown in Table 8-1, Table 8-2, Table 8-3 and Table 8-4, presents the selected components, quantity and price per piece. The prices are based on components ordered from eBay and the total price for building each individual unit (prototype) is displayed in the bottom of each table. Most of the selected components are also available from Norwegian suppliers, but with higher prices due to VAT and shipping costs.

A total price estimation for all components from Norwegian suppliers and eBay is available in Appendix B.

Table 8-1 Order list for Sensor Node Raspberry Pi 3

Item	Quantity	Unit price (incl. VAT & Shipping)
Raspberry Pi 3 Model B	1	kr 247,37
Battery Expansion Board	1	kr 173,19
Micro USB Wall charger (EU plug)	1	kr 44,00
Micro SD 16GB (Class 10)	1	kr 41,18
Enclosure Box (129x41x67mm)	1	kr 54,38
Micro USB charging cable (10 cm)	1	kr 6,19
BME280	1	kr 54,54
MH-Z19	1	kr 244,13
	Total Price	kr 864,98

Table 8-2 Order list for Sensor Node MKR1000

Item	Quantity	Unit price (incl. VAT & Shipping)
Genuino MKR1000	1	kr 516,02
Li-polymer battery (3,7V 1200mAh)	1	kr 36,68
Micro USB Wall charger (EU plug)	1	kr 44,00
Enclosure Box (100x60x25mm)	1	kr 8,17
HC05	1	kr 27,12
BME280	1	kr 54,54
MQ-135	1	kr 28,61
	Total Price	kr 715,14

Table 8-3 Order list Data Hub Raspberry Pi 3

Item	Quantity	Unit price (incl. VAT & Shipping)
Raspberry Pi 3 Model B	1	kr 247,37
Micro SD 16GB (Class 10)	1	kr 41,18
Enclosure Box (Raspberry Pi 3)	1	kr 16,65
Micro USB Wall charger (EU plug)	1	kr 44,00
	Total Price	kr 349,20

Table 8-4 Order list Data Hub Intel NUC

Item	Quantity	Unit price (incl. VAT & Shipping)
Barebone Intel NUC (NUC5CPYH)	1	kr 1 841,00
	Total Price	kr 1 841,00

9 Development

This chapter describes the development of the sensor nodes and data hubs with figures and appendixes as documentation, separated into subchapters describing the wiring, the applications and the constructed unit. The sensor nodes and data hubs are developed using the software and hardware described in chapter 5 and chapter 6, and the communication protocols described in chapter 4.

9.1 Simple version of sensor node and data hub

Development of the prototypes for the measurement and monitoring system, started with simple versions of the sensor node and data hub for ensuring that the Raspberry Pi 3 hardware could run properly with LabVIEW LINX software.

First, the simple version sensor node was developed for demonstrating that a Raspberry Pi 3 could run as an embedded system, showing temperature readings using a MCP9700AE temperature sensor. The system operated using a LabVIEW application and a LED, which lighted up when a certain temperature level was reached. A simple systematic procedure (user manual) was created as documentation using PowerPoint and is located in Appendix H.

Next, the first version of a data hub was created for demonstrating that the Raspberry Pi 3, as embedded system using LabVIEW, can transfer temperature readings to a computer using bluetooth and Wi-Fi communication. However, it was later discovered that the bluetooth pallets in LabVIEW was not supported for Linux Jessie OS and that bluetooth communication was impossible until further support for Linux Jessie OS is given and it was therefore only created TCP (Wi-Fi) transmission for the temperature readings.

An overview of the simple version sensor node and data hub are shown in Figure 9-1.
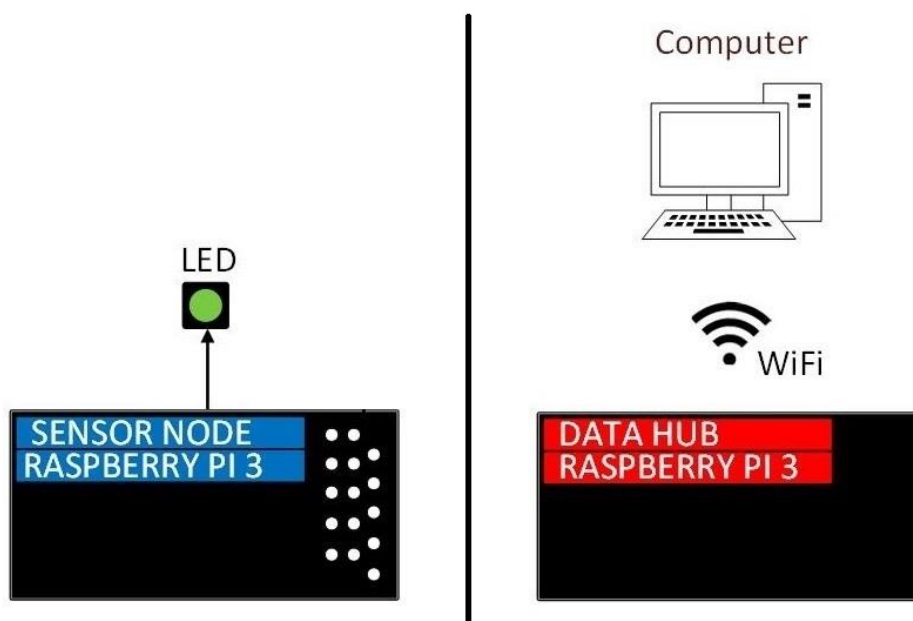

Figure 9-1 Overview of the simple version sensor node and data hub

9.1.1 Wiring for the simple versions sensor node

The wiring schematic for the simple version sensor node can be seen in Figure 9-2, showing how the temperature sensor (MCP9700AE) was connected to an analog to digital converter (MCP3002) for converting an analog signal to a digital signal. It also shows how the LED (green) is connected through a 220ohm resistor.

For the simple version data hub, there was no need for any wiring other than establishing a connection between the Raspberry Pi 3 and the computer using Wi-Fi.
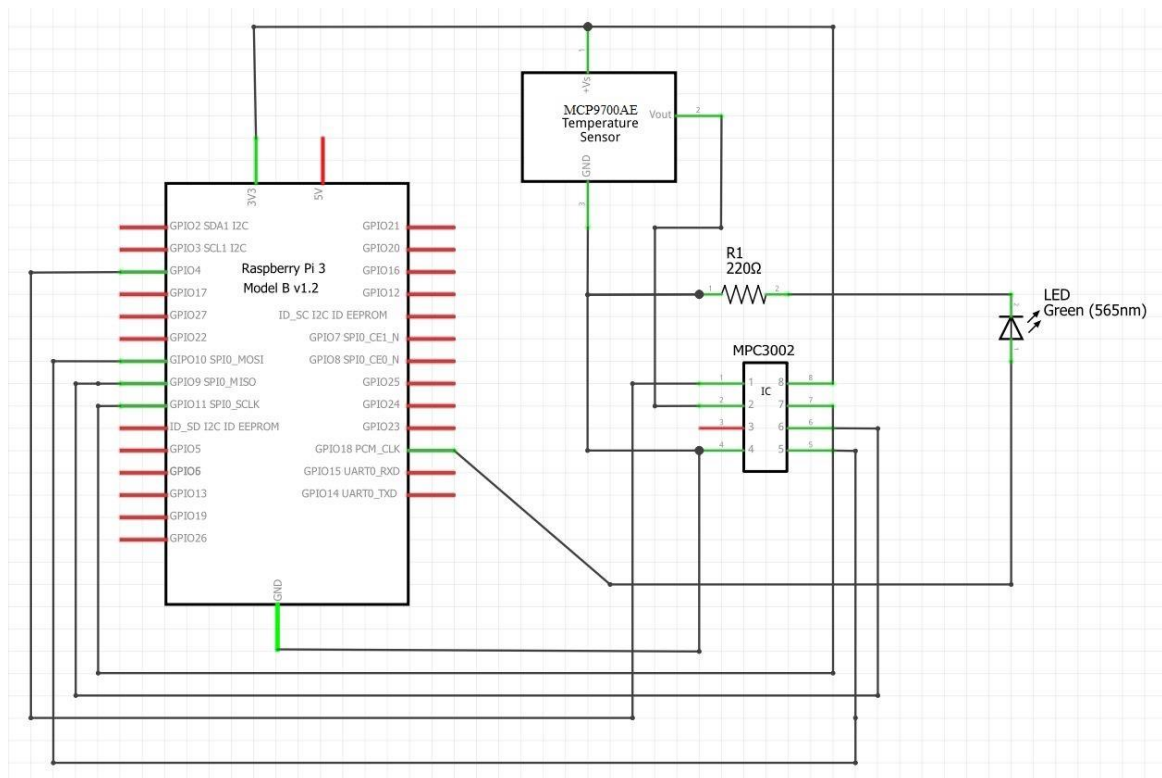

Figure 9-2 Wiring schematic for the simple version sensor node

9.1.2 LabVIEW applications for the simple versions sensor node and data hub

The LabVIEW application shown in Figure 9-3 (Appendix C) is for the simple version sensor node and consists of three SubVIs:

- GenBytes
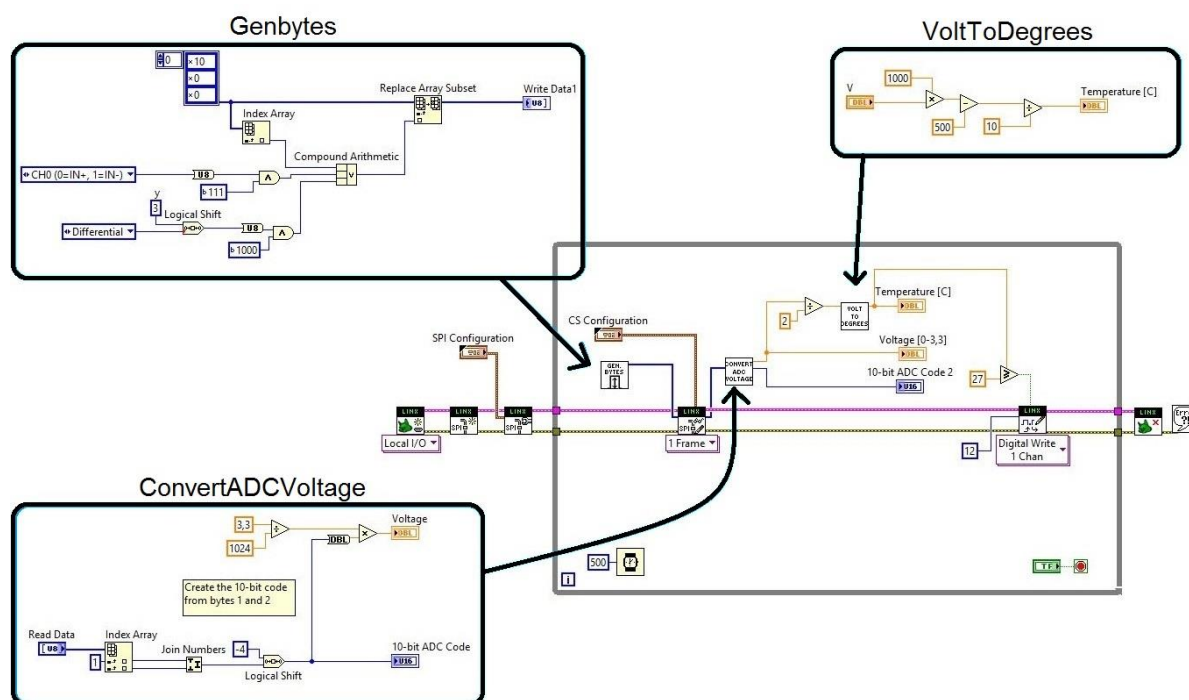- ConvertADCVoltage
- VoltToDegrees

Figure 9-3 LabVIEW application for first version of sensor node (Main VI)

"GenBytes" SubVI generates the necessary bytes to communicate with the MCP3002, based on configuration values found in the datasheet [51]. By using I2C communication for sending/retrieving data, a 10-bit code is retrieved and converted by "ConvertADCVoltage" SubVI to a voltage signal. Based on the reference voltage (3,3v or 5v), the voltage signal is then converted to a temperature value by "VoltToDegrees" SubVI. A simple logic "greater or equal" (set to 27) determines when the LED is turn on/off.

For the simple version data hub, two simple LabVIEW applications was created:

- TCP send
- TCP receive

"TCP send", Figure 9-4, listens for an incoming request (string) on a specified port. When a request is received, it replays by "writing" a string back that in this case is a constant value of "21" formatted into a string.
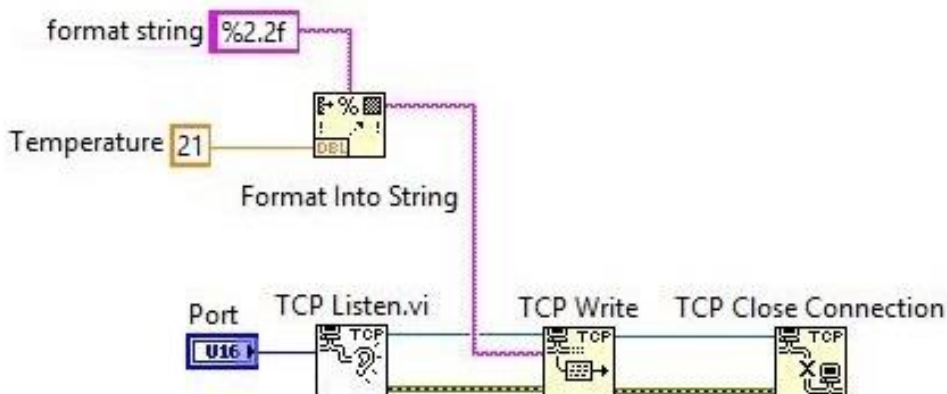

Figure 9-4 LabVIEW application for first version data hub (TCP send)

"TCP receive", Figure 9-5, writes an "empty" string (request) to a specified IP address and port, and receives a string in reply (1 byte). This byte is then converted/indexed into a byte array and displayed in the front panel.
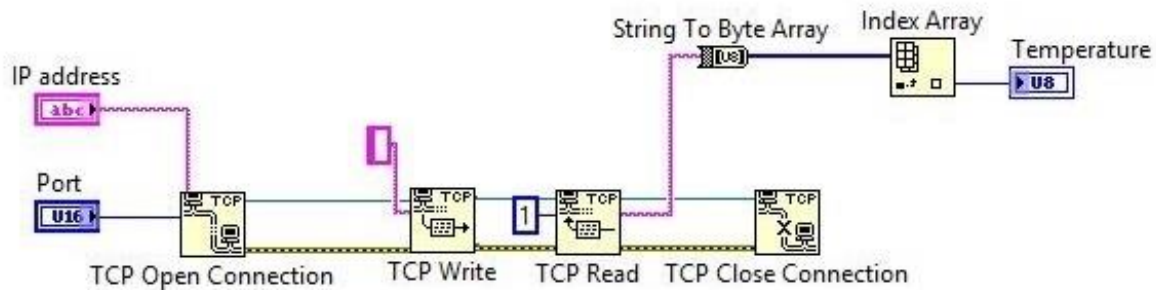
Figure 9-5 LabVIEW application for first version data hub (TCP receive)

9.1.3 Constructed sensor node (simple version)

The simple version of a sensor node was created with a breadboard for simplicity, as shown in Figure 9-6, with the components and application mentioned in subchapter 9.1.1 and subchapter 9.1.2 respectively. The simple version data hub was also created with the application mentioned in subchapter 9.1.2, but without any wiring as mentioned in subchapter 9.1.1.

Figure 9-6 Simple version sensor node constructed using a breadboard

9.2 Development of prototypes for the sensor nodes and data hubs

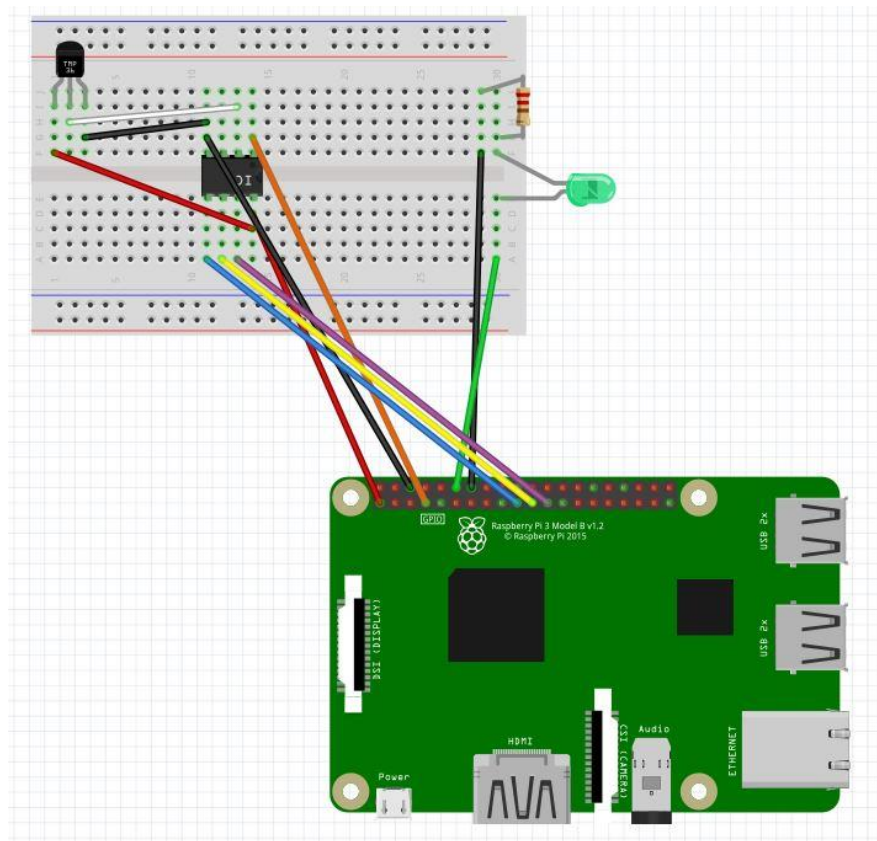The prototypes for the measurement and monitoring system was created as two types of sensors nodes and data hubs, using Arduino MKR1000, Raspberry Pi 3 and Intel NUC hardware, wrapped in a casing for commercial use. An overview of the complete system can be seen in Figure 9-7, showing how the sensor nodes and data hubs are connected using Wi-Fi and Bluetooth communication protocol. Note that the sensor nodes are either connected to the Raspberry Pi 3 or the Intel NUC data hub. It is possible to run both data hubs at the same time, but interruption(delay) in sampling may occur if both data hubs are requesting measurement data at the same time.
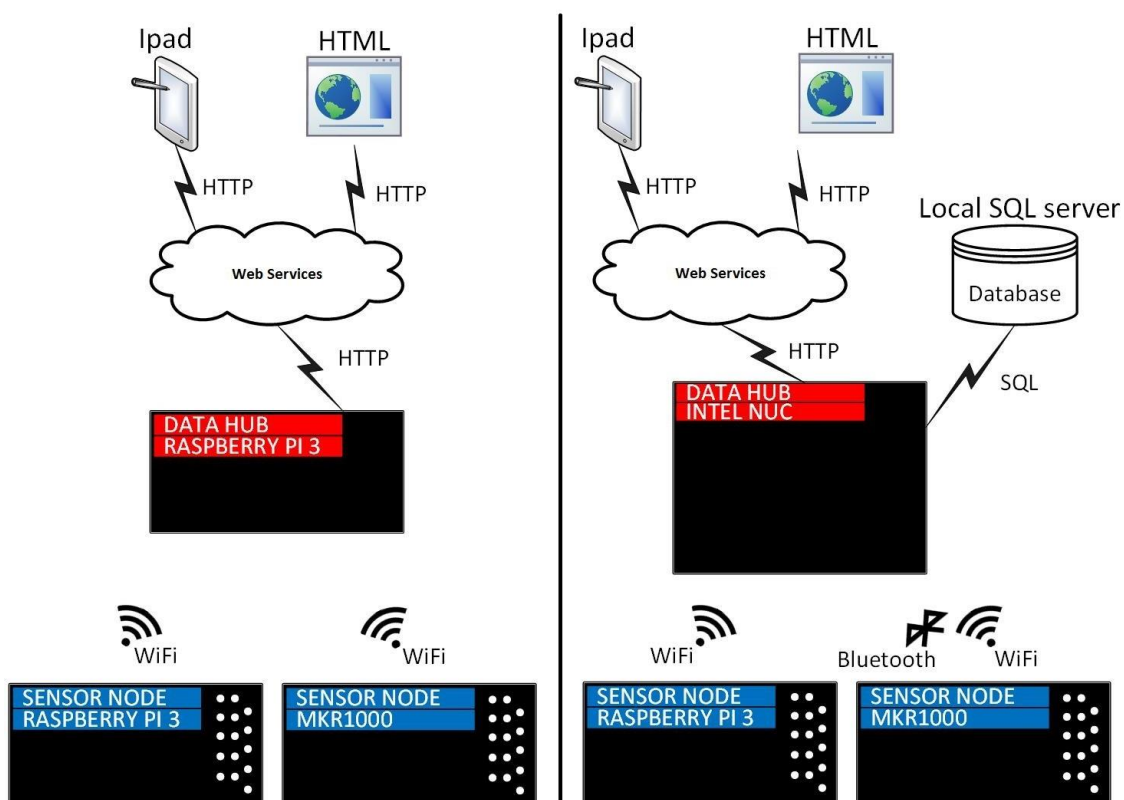
Figure 9-7 Overview of the measurement and monitoring system

The first developed sensor node prototype was the MKR1000 sensor node, which was developed using a MKR1000 microcontroller (subchapter 6.2.1) instead of a Raspberry Pi 3 computer. Two sensor, BME280 (subchapter 6.1.6) and MQ-135 (subchapter 6.1.4), was implemented and placed inside an enclosure box (100x60x25mm) along with the microcontroller. A 3,7V 1200mAh Li-polymer battery and a Bluetooth module HC-05 (subchapter 6.3.1) was later implemented, for giving the sensor node the feature of battery operation and communicating using Bluetooth communication protocol. The programming language used for the MKR1000 was C++ (Arduino IDE), which is opposite to the programming language used on the Raspberry Pi 3 (LabVIEW).

Second developed sensor node prototype was the Raspberry Pi 3, developed further from the simple version (subchapter 9.1) with the intensions of using two sensors, BME280 (subchapter 6.1.6) and MH-Z19 (subchapter 6.1.5) along with a battery expansion board. But

due to lack of support for "Custom Command" and the time for developing the libraries needed for the BME280 and MH-Z19, the sensors was replaced with the digital temperature sensor TC74 (chapter 6.1.3) and the battery expansion board removed. The enclosure box (129x41x67mm) was replaced with the same type used for the MKR1000 sensor node.

The third developed prototype was the Raspberry Pi 3 data hub, which was placed inside an enclosure box (100x60x25mm) without any extra components such as battery expansion board since the feature of battery operation was never intended for the data hubs.

For the Intel NUC data hub (subchapter 6.2.4), the unit was not developed in any other way than inserting RAM and Wi-Fi modules which was delivered with the computer.

9.2.1 Wiring for the prototypes

Wiring schematics for the MKR1000 and Raspberry Pi 3 sensor node prototype are created for showing how the components are connected to the microcontroller/computer. The data hubs on the other hand, both Intel NUC and Raspberry Pi 3, has no specific components connected and therefore no wiring schematics.

The wiring schematic for the MKR1000 sensor node prototype seen in Figure 9-8, shows how the BME280 and MQ135 sensors and HC-05 Bluetooth module are connected to the microcontroller.
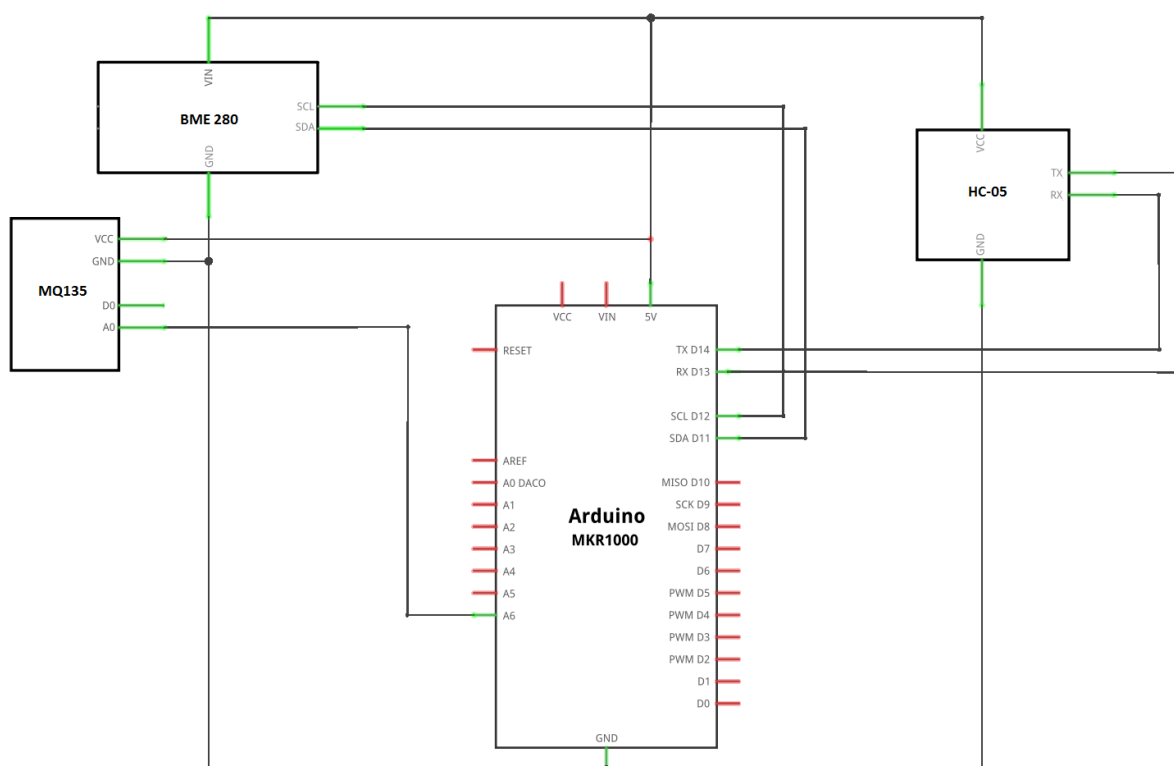

Figure 9-8 Wiring schematic for the MKR1000 sensor node prototype

Wiring schematic for the Raspberry Pi 3 sensor node prototype seen in Figure 9-9, shows how the TC74 sensor is connected to the computer.
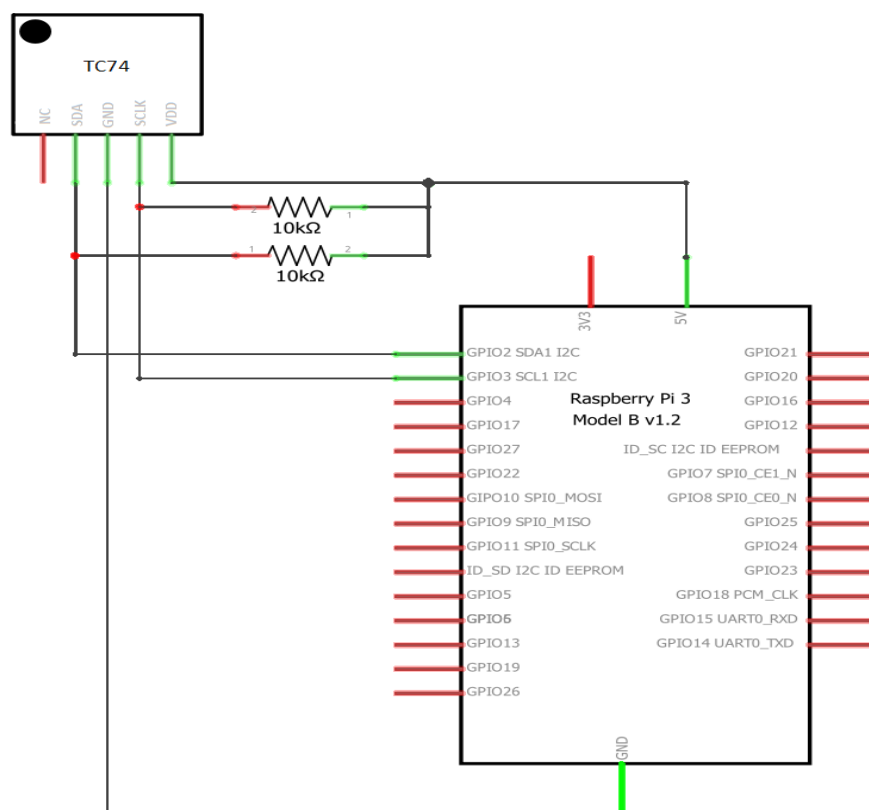
Figure 9-9 Wiring schematic for the Raspberry Pi 3 sensor node prototype

9.2.2 Arduino IDE application for the MKR1000 sensor node prototype

The applications developed for the sensor nodes prototype was developed using Arduino IDE and is described briefly during this subchapter. For more details with comments for this application, see full code in Appendix D.

The application for the MKR1000 sensor node consists of four parts:

- Include
- Global Variables
- Setup
- Loop

"Include" part includes all the libraries need, which are SPI, Wi-Fi, BME280 and MQ135. "Global Variables" part contains the variables for temperature, humidity, pressure, co2 level and Wi-Fi settings. It also contains the setting for the units for the BME280 (Pa, hPa, etc.). "Setup" part contains the methods that are enabled/started for the Bluetooth and Wi-Fi communication and for the BME280 sensor. "Loop" part, running with 2000ms delay per iteration, contains the code for retrieving sensor data and the code for the data using Bluetooth and Wi-Fi. Note that the data hub sends the request for data using either bluetooth or Wi-Fi, and the MKR1000 responds accordantly.

The application was also developed with two additional code tabs:

- SerialMonitor
- WiFiStatus

The "SerialMonitor" tab was for reading sensor data from a serial monitor in the Arduino IDE software, and the "WiFiStatus" tab was for retrieving Wi-Fi status such as IP address and signal strength. Both tabs was used during the development phase.

9.2.3 LabVIEW application for the Raspberry Pi 3 sensor node prototype

The applications developed for the sensor nodes prototype was developed using LabVIEW and is described briefly during this subchapter. For more details with comments for this application, see full code in Appendix E.

The application for the Raspberry Pi 3 sensor node, shown in Figure 9-10, consists of four SubVIs:

- I2C
- Bit_To_Temp
- TCP
- Bluetooth
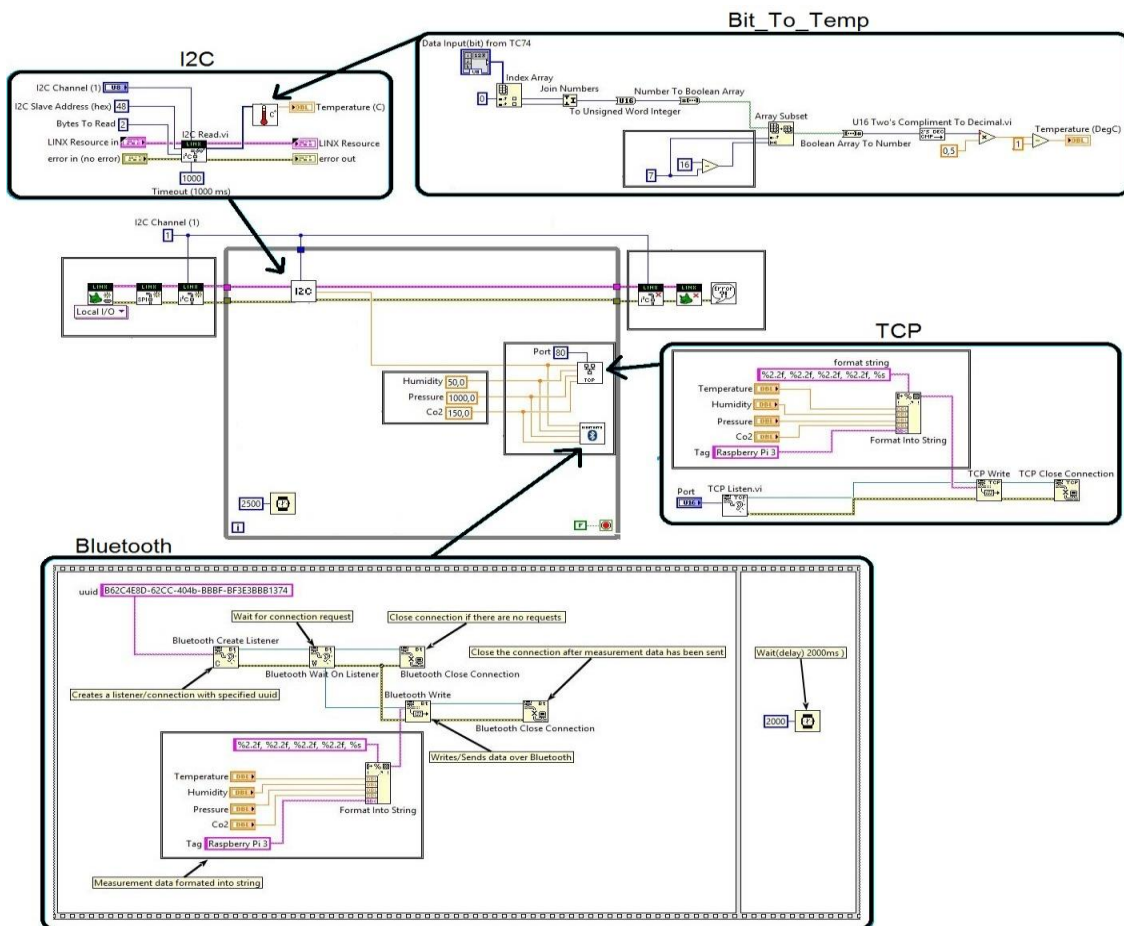
Figure 9-10 LabVIEW application for Raspberry Pi 3 sensor node (Main VI)

The "I2C" SubVI contains a LINX I2C Read pallet for retrieving the temperature data from the TC74. It also contains the "Bit_To_Temp" SubVI for converting the bit value retrieved into a temperature value in degrees Celsius and the "TCP" and "Bluetooth" SubVI located in the main VI sends the measurement data to the data hub. Note that only temperature data is

collected and that humidity, pressure and Co2 is set to constant values 50, 1000 and 150 respectively, since the sensors for measuring these units are not implemented.

9.2.4 LabVIEW application for the Intel NUC and Raspberry Pi 3 data hub prototypes

The applications developed for both the data hub prototypes was developed using LabVIEW and is described briefly during this subchapter. For more details with comments for these two applications, see full code in Appendix F and G.

The LabVIEW application for the Intel NUC data hub, shown in Figure 9-11, consists of four SubVIs:
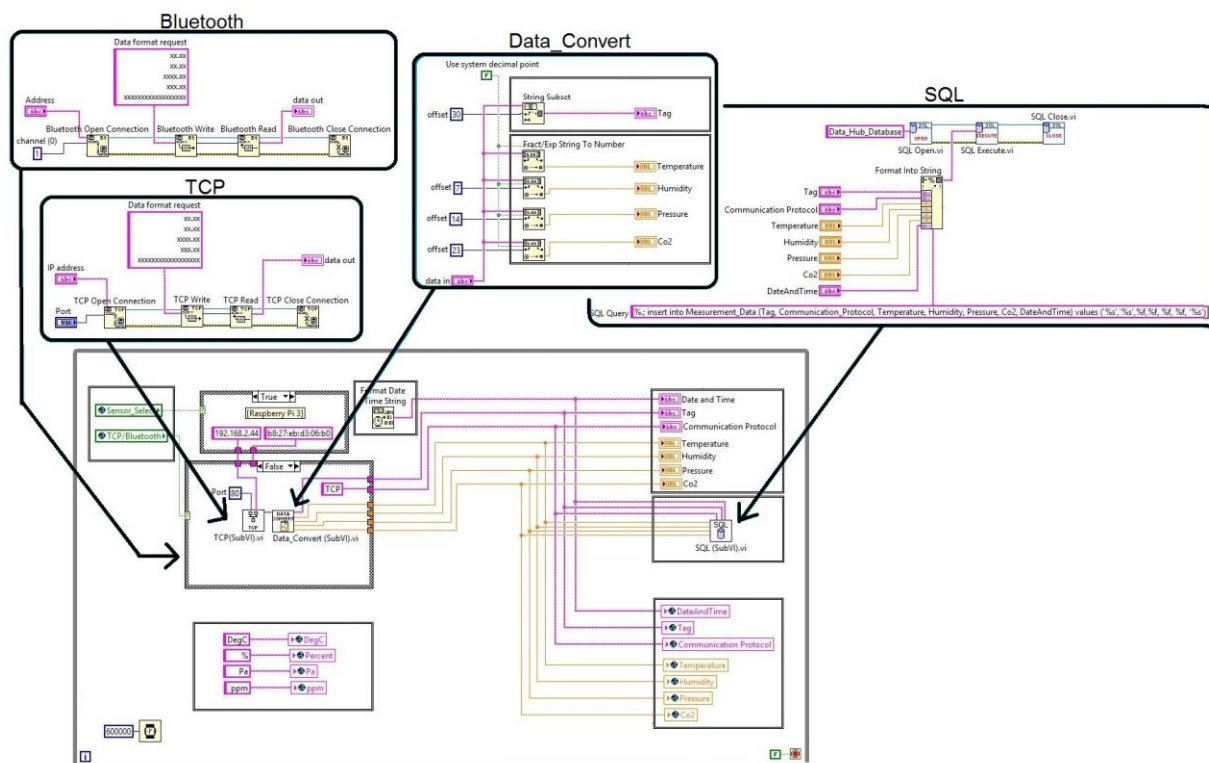
- TCP
- Bluetooth
- Data_Convert
- SQL

Figure 9-11 LabVIEW application for the Intel NUC data hub (Main VI)

"TCP" SubVI is similar to the one described in subchapter 9.1.2, but instead of an empty string, it has a specific data format for requesting measurement data. This format, Table 9-1, ensures that the data requested is in correct format for further use in Dashboard/HTML and SQL.

Table 9-1 Data format used in TCP and Bluetooth SubVI

String format	Value format	Sensor data
xx.xx	00.00	Temperature
xx.xx	00.00	Humidity
xxxx.xx	0000.00	Pressure
xxx.xx	000.00	Co2 level
xxxxxxxxxxxxxx	abc…	Tag

The IP address is selected using a "case structure" for switching between the MKR1000 and Raspberry Pi 3 sensor node. The "Bluetooth" SubVI is built the same way as for the "TCP" SubVI, with the same format request, but with bluetooth pallets and the device address selected the same way as for the IP address using the same "case structure". The "Data_Convert" SubVI separates and converts the received string from either the "TCP" or "Bluetooth" SubVI into decimal points and tag. This is sent to the indicators on the front panel, the global variables (Web Services) for the Dashboard/HTML page, and the "SQL" SubVI. The "SQL" SubVI inserts the measurement data, along with tag and time/date, into a table consisting of multiple rows, which operates only as local database (server).

The loop iteration in the main VI is set to 10 seconds for testing purposes, but normally operates with a 10-minute interval. As an extra feature, additional global variables are added for making it possible to change units for the measurement data without having to reprogram the sensor nodes. These are located in the lower left corner of the main VI.

The LabVIEW application for the Raspberry Pi 3 data hub is the same as for the Intel NUC data hub shown in Figure 9-11, but without the "SQL" SubVI. This SubVI is removed due to lack of support for SQL in Linux Jessie OS, and the measurement data is therefore only retrieved and presented in a real-time manner for the Dashboard/HTML page.

9.2.5 Constructed prototypes

The constructed prototypes for the sensor nodes and data hubs was wrapped in casings using mounting putty for holding components temporary in place. The components was placed in such way that more components can be implemented later on.

The MKR1000 sensor node was created as compact as possible with a separate chamber for the sensors, with holes drilled in the lid for creating airflow. The sensor node was equipped with the components mentioned in subchapter 9.2.1, shown in Figure 9-12.
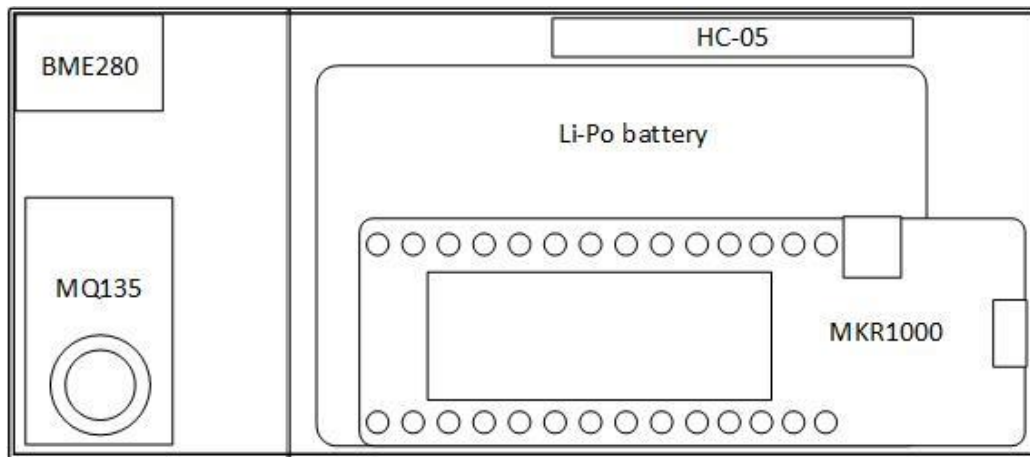
Figure 9-12 MKR1000 sensor node components overview

Wires where soldered to each component using different colors for giving a better overview. The color codes are as follow:

- Red = 5V
- Black = Ground
- Blue = Bluetooth (RX/TX)
- Green = I2C (SCL/SDA)
- White = Analog signal

The constructed prototype for the MKR1000 sensor node can be seen in Figure 9-13, showing the sensor node with/without the lid and with the different colors used for the wiring.
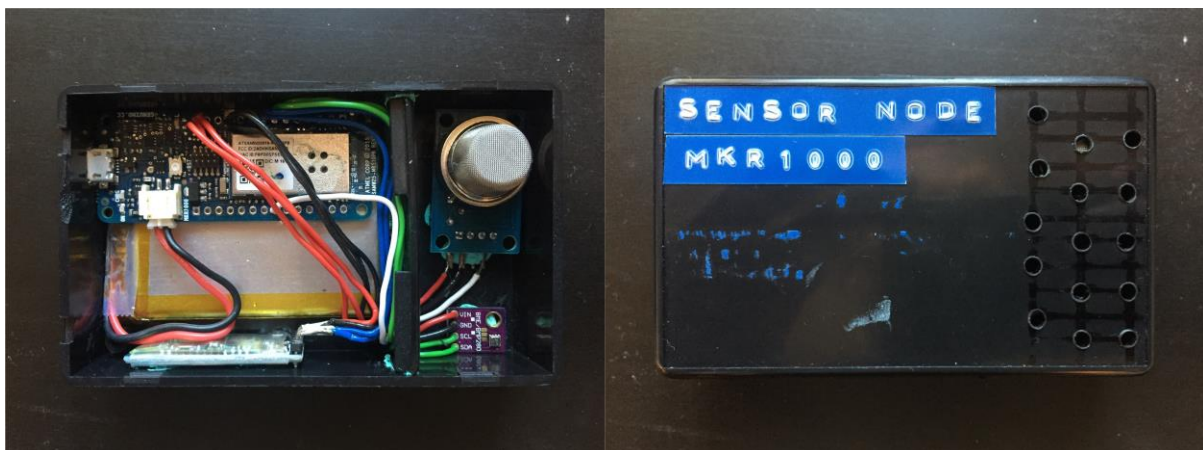
Figure 9-13 MKR1000 sensor node prototype

The Raspberry Pi 3 sensor node was also created as compact as possible, since the original layout with a battery expansion board was dismissed as mentioned in subchapter 9.2. Holes was drilled for creating airflow, but not to a separate chamber as for the MKR1000 sensor node, and the locations for the components are shown in Figure 9-14.
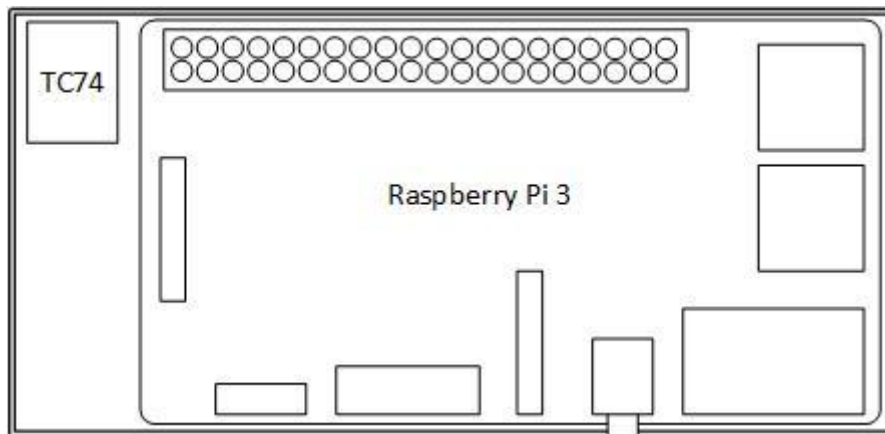
Figure 9-14 Raspberry Pi 3 sensor node components overview

Wires where soldered to each component, same as for the MKR1000 sensor node, using different colors for giving a better overview, and the color codes are the same as mentioned earlier for the MKR1000 sensor node.

The constructed prototype for the Raspberry Pi 3 sensor node can be seen in Figure 9-15, showing the sensor node with/without the lid and with the different colors used for the wiring. This sensor node prototype was developed with the digital temperature sensor (TC74) and not the actual intended multi sensor BME280 and Co2 sensor MH-Z19 as shown in Figure 9-16.
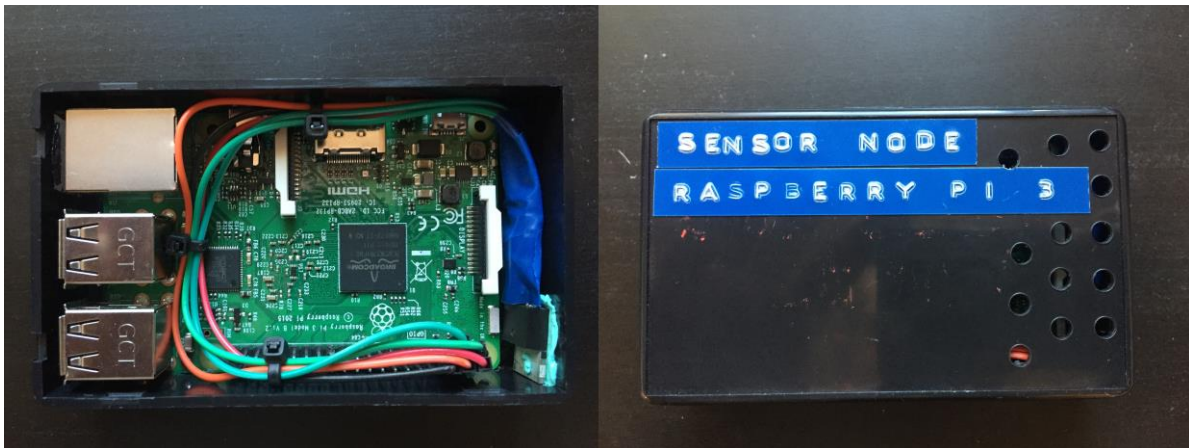
Figure 9-15 Raspberry Pi 3 sensor node prototype
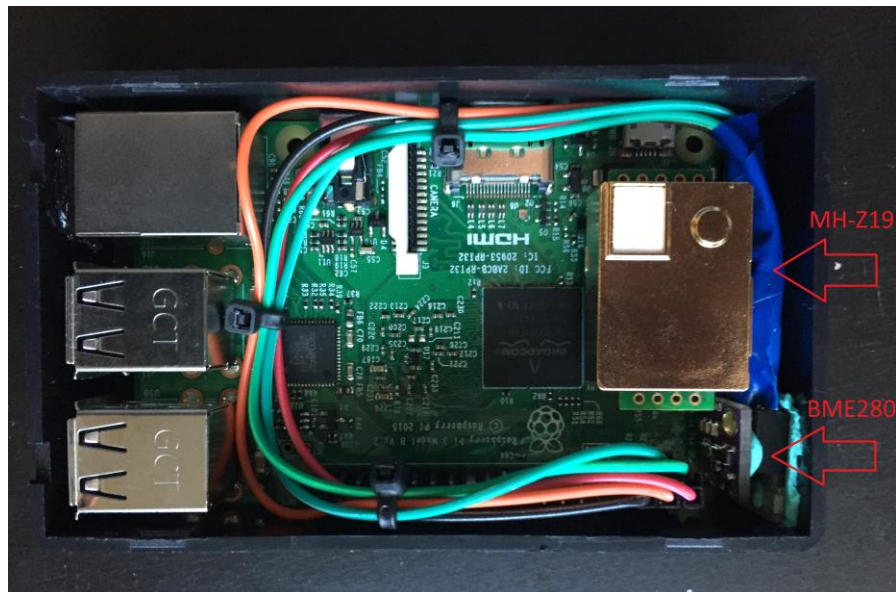
Figure 9-16 Original layout for the Raspberry Pi 3 sensor node prototype

The Raspberry Pi 3 data hub prototype, shown in Figure 9-17, was created using a commercial Raspberry Pi 3 enclosure box made for Raspberry Pi modules. No wires was connected since this was only a data hub for receiving and transmitting measurement data using the built-in Wi-Fi and Bluetooth module, and it is therefore not created any components overview as for the sensor nodes.

Figure 9-17 Raspberry Pi 3 data hub prototype

The Intel NUC data hub prototype was final product constructed and delivered by Intel Corporation. No wires was connected since this, in the same way as for the Raspberry Pi 3 data hub, only receives and transmits measurement data using the built-in Wi-Fi and Bluetooth module, and no components overview was therefore created.

10 Testing

This chapter describes the testing phase for the measurement and monitoring system created using the developed prototypes described in subchapter 9.2. The first two subchapters will describe the testing phase for the Intel NUC and Raspberry Pi 3 data hub using the MKR1000 and Raspberry Pi 3 sensor node. The next subchapter will show the results from the measured power consumption tests and the final subchapter will described the test environment created for testing the measurement and monitoring system over a longer period.

10.1 Intel NUC data hub with MKR1000 and Raspberry Pi 3 sensor node

For testing the Intel NUC data hub with the MKR1000 and Raspberry Pi 3 sensor node, both sensors was connected to the same wireless network and the bluetooth connection paired/trusted. The Intel NUC data hub was started using the LabVIEW run-time application, with 10 seconds loop interval, and the test was preformed in a normal house environment for a period of one hour while testing all features during this period. The computer and iPad used for the testing the Dashboard/HTML page was connected to the same wireless network.

The HTML features was tested for displaying real-time data in a simple HTML page using Web Services, giving the result shown in Figure 10-1. This data is updated every 10 seconds and are not showing the measurement data from the local SQL database.

URL: /WebService/HTTP_HTML

Output Terminals

Terminal Name	Terminal Value
1. Tag	MKR1000
2. Communcation Protocol	TCP
3. Temperature	31.14
4. Humidity	31.55
5. Pressure	986.78
6. Co2	37.05
7. DateAndTime	16.11.2016 14.57.57

Figure 10-1 Test of HTML page for the Intel NUC data hub

The option of changing sensor and communication protocol was done using URL commands, shown in Table 10-1, inserted into the web browser.

Table 10-1 URL commands for HTML page

URL:	Option:
192.168.2.148:8001/WebServices/Sensor_Select?value=0	MKR1000
192.168.2.148:8001/WebServices/Sensor_Select?value=1	Raspberry Pi 3
192.168.2.148:8001/WebServices/Communication_Protocol?value=0	TCP
192.168.2.148:8001/WebServices/Communication_Protocol?value=1	Bluetooth

Next, the LabVIEW Dashboard application was tested on an iPad for displaying the real-time measurement data in a GUI as shown in Figure 10-2. This application has the options of selecting sensor and communication protocol using a pushbutton instead of URL commands, and shows a more detailed overview with charts updating every iteration. The application uses "touch" mechanics for swiping between screens, zooming in on graphs and clicking on pushbuttons, resulting in a simple user interface.

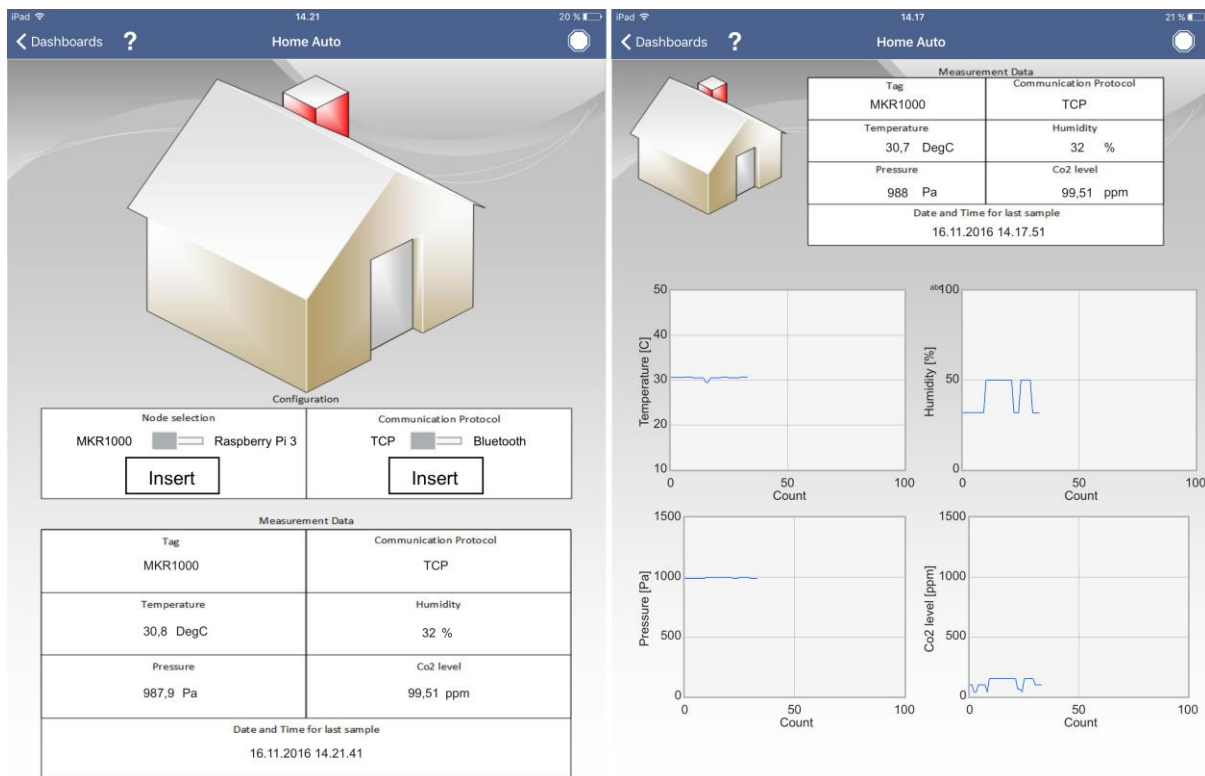Figure 10-2 LabVIEW Dashboard application showing overview and detailed overview

The application is built-up by two main screens, one showing a simple overview and the other a more detailed overview with graphs. In the simple overview, the user can select (insert) sensor node and communication protocol and get the measurement data from the selected sensor node. The measurement data displayed are Tag, Communication protocol,

temperature, humidity, pressure, Co2 level and Data/time for last sample. In the more detailed overview, the same measurement date are displayed, but with the additional graphs for the temperature, humidity, pressure and Co2 level. For connecting the application to the Intel NUC data hub the same IP address and Port number as for the HTML page is used, and the application can be used by any Apple iPad running LabVIEW Dashboard as long as the user is connected to the same network.

Last test was to confirm that the local SQL server received the measurement data every 10 seconds from the data hub and that this was inserted into a table called Measurement_data, contained the columns:

- Tag
- Communcations_Protocol
- Temperature
- Humidity
- Pressure
- Co2
- DateAndTime

During this test the sensor nodes was influenced by human interaction for creating variations in temperature, humidity, pressure and co2 level, giving the result seen in Figure 10-3. Different sensor nodes and communication protocol was also selected during this test period and the result shows that the data hub application is transferring the values correctly.

Figure 10-3 Test of local SQL server with 10 seconds iterations

10.2 Raspberry Pi 3 data hub with MKR1000 and Raspberry Pi 3 sensor node

For testing the Raspberry Pi 3 data hub with the MKR1000 and Raspberry Pi 3 sensor node, both sensors was connected to the same wireless network. The Raspberry Pi 3 data hub started the LabVIEW run-time application automatically at startup (embedded) with loop iteration set to 10 seconds. The test was performed in same environment as for the Intel NUC data hub, for a period of one hour, while testing all features during this period. The computer and iPad used for testing the Dashboard/HTML page was connected to the same wireless network, and the test preformed the same way as for the Intel NUC data hub.

First the HTML features was tested for displaying real-time data in a simple HTML page using Web Services, same as for the Intel NUC but with a slightly different IP address, and the result is shown in Figure 10-4.

Figure 10-4 Test of HTML page for the Raspberry Pi 3 data hub

Option for changing sensor and communication protocol was done using the same URL commands as shown in Table 10-1, but with the following IP address and Port number: 192.168.2.78:8001.

Next, the LabVIEW Dashboard application was tested using exactly the same application as for the Intel NUC data hub, but with different IP address and Port number, giving the same results as presented in subchapter 10.1 and is therefore not addressed any further.

The local SQL server was never implemented on the Raspberry Pi 3 data hub, due to lack of support for SQL software on Linux OS, and was therefore not tested.

10.3 Measured power consumption

The power consumption for the sensor nodes and the Raspberry Pi 3 data hub was measured using a mini OLED meter connected between the power outlet and the sensor node/data hub. The mini OLED meter displays voltage, current and power, giving a measurement on how much the uploaded application effected the total power consumption. The Intel NUC data hub however, was not measured since there was already power consumption data available from TPCDB as mentioned in subchapter 7.3.3.

The results from the measured power consumption for the MKR1000 sensor node, shown in Figure 10-5, gave the results:
U = 4.81V, I = 0.27A and P = 1.36W.

Figure 10-5 Power consumption for the MKR1000 sensor node

From subchapter 7.3.1 the estimations gave the results: U = 5V, I = 0.07A and P = 0.35W. This clearly indicates that the current and power consumption is higher than estimated. With a battery capacity of 1200mAh and consumption of 270mAh, the battery should last about 4.5 hours. Options like "sleep" mode between iterations and the use of capacitors can reduce the power consumption further; making the 1200mAh battery lasts for days.

The results from the measured power consumption for the Raspberry Pi 3 sensor node, shown in Figure 10-6, gave the results: U = 4.81V, I = 0.27A and P = 1.34W.

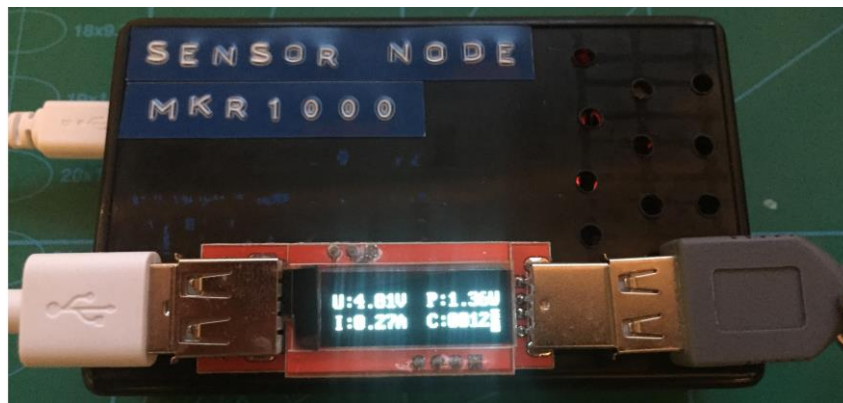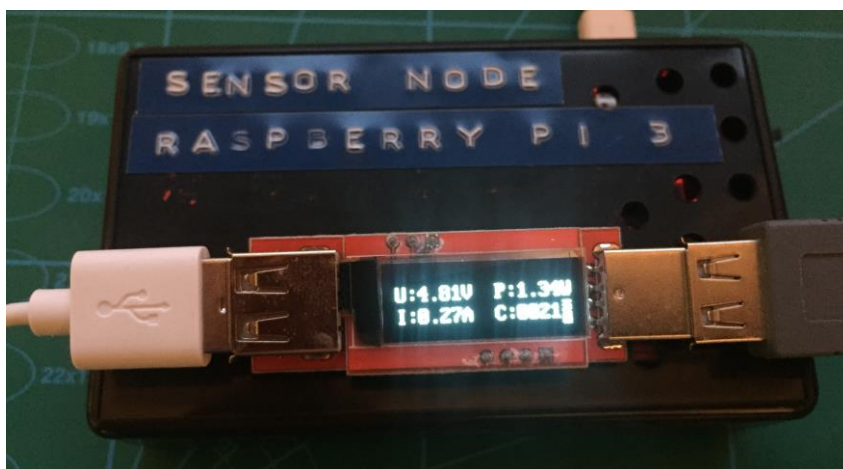

Figure 10-6 Power consumption for the Raspberry Pi 3 sensor node

From subchapter 7.3.2, the estimations gave the results: U = 5V, I = 0.34A and P = 1.7W. Comparing these results clearly states a difference between the measured and the estimated

power consumption. With a battery expansion board of 3800mAh and a consumption of 270mAh, same as the MKR1000, the battery should last for approximately 14 hours. The power consumption can also here be reduced by using capacitors and "sleep" mode between iterations, reducing the overall power consumption.

The measured power consumption for the Raspberry Pi 3 data hub was the same as for the Raspberry Pi 3 sensor node, giving the same results when measured. Since the Raspberry Pi 3 data hub was not intended to be used with a battery expansion board, further reduction in the power consumption using "sleep" mode or capacitors are considered unnecessary.

10.4 Test environment

The test environment for testing the sensor nodes and data hubs over a longer period are installed in the sensor lab at Telemark University College as shown in Figure 10-7. The Intel NUC data hub is connected with the MKR1000 and Raspberry Pi 3 sensor node, running the same application as mention in subchapter 10.1, but with 10 minutes loop iterations. For showing the values, as they are sampled, a small screen is connected to the data hub.
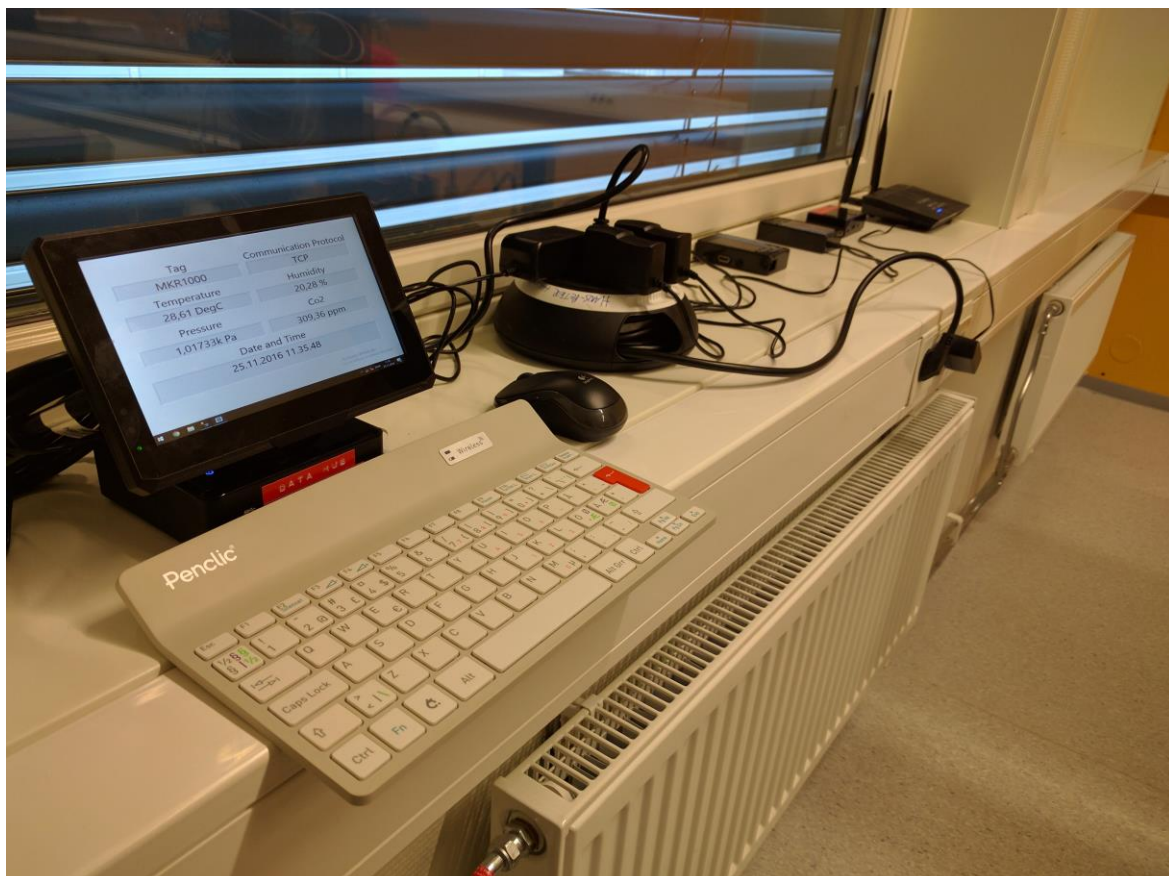
Figure 10-7 Overview of the test environment for the measurement and monitoring system installed in the sensor lab at Telemark University College

The sampled data are stored in the local SQL server every 10 minutes and will later be used for determining the stability and accuracy of the complete system over a longer period. The Raspberry Pi 3 data hub is not installed as part of the testing environment, since it does not store any values over time in the local SQL server. However, the application used on the Raspberry Pi 3 data hub is identical except for the SQL part, so testing one of the data hubs

was therefore considered sufficient to determine the stability and accuracy over time. The result after four days of sampling from 21.11.2016 to 25.11.2016 gave the results shown in Figure 10-8 and Figure 10-9, indicating that the system was stable but with inaccurate readings due to poor calibration.

	Tag	Communication_Protocol	Temperature	Humidity	Pressure	Co2	DateAndTime
1	MKR1000	TCP	22	30	1003	283	21.11.2016 11.05.48
2	MKR1000	TCP	23	28	1004	337	21.11.2016 11.15.48
3	MKR1000	TCP	24	27	1004	309	21.11.2016 11.25.48
4	MKR1000	TCP	25	25	1004	309	21.11.2016 11.35.48
5	MKR1000	TCP	26	25	1004	309	21.11.2016 11.45.48
6	MKR1000	TCP	26	24	1004	283	21.11.2016 11.55.48
7	MKR1000	TCP	26	24	1004	283	21.11.2016 12.05.48
8	MKR1000	TCP	27	24	1004	309	21.11.2016 12.15.48
9	MKR1000	TCP	27	23	1004	337	21.11.2016 12.25.48
10	MKR1000	TCP	27	23	1004	283	21.11.2016 12.35.48
11	MKR1000	TCP	27	23	1004	283	21.11.2016 12.45.48
12	MKR1000	TCP	27	23	1004	309	21.11.2016 12.55.48
13	MKR1000	TCP	27	23	1005	309	21.11.2016 13.05.48
14	MKR1000	TCP	27	23	1004	309	21.11.2016 13.15.48
15	MKR1000	TCP	27	23	1005	309	21.11.2016 13.25.48
16	MKR1000	TCP	27	23	1005	337	21.11.2016 13.35.48
17	MKR1000	TCP	27	23	1005	366	21.11.2016 13.45.48
18	MKR1000	TCP	27	23	1005	309	21.11.2016 13.55.48
19	MKR1000	TCP	27	23	1005	337	21.11.2016 14.05.48
20	MKR1000	TCP	27	22	1005	337	21.11.2016 14.15.48
21	MKR1000	TCP	27	22	1005	309	21.11.2016 14.25.48
22	MKR1000	TCP	27	22	1005	309	21.11.2016 14.35.48
23	MKR1000	TCP	27	22	1005	337	21.11.2016 14.45.48
24	MKR1000	TCP	27	22	1005	309	21.11.2016 14.55.48

Figure 10-8 Local SQL server Measurement_data table at the 21.11.2016

	Tag	Communication_Protocol	Temperature	Humidity	Pressure	Co2	DateAndTime
555	MKR1000	TCP	27	20	1019	309	25.11.2016 07.25.48
556	MKR1000	TCP	26	20	1018	283	25.11.2016 07.35.48
557	MKR1000	TCP	26	20	1018	283	25.11.2016 07.45.48
558	MKR1000	TCP	26	20	1018	337	25.11.2016 07.55.48
559	MKR1000	TCP	26	20	1018	309	25.11.2016 08.05.48
560	MKR1000	TCP	27	20	1018	283	25.11.2016 08.15.48
561	MKR1000	TCP	27	19	1018	309	25.11.2016 08.25.48
562	MKR1000	TCP	27	19	1018	337	25.11.2016 08.35.48
563	MKR1000	TCP	27	19	1018	309	25.11.2016 08.45.48
564	MKR1000	TCP	27	19	1018	309	25.11.2016 08.55.48
565	MKR1000	TCP	27	19	1018	283	25.11.2016 09.05.48
566	MKR1000	TCP	27	19	1018	309	25.11.2016 09.15.48
567	MKR1000	TCP	28	19	1018	258	25.11.2016 09.25.48
568	MKR1000	TCP	28	19	1018	309	25.11.2016 09.35.48
569	MKR1000	TCP	28	19	1018	309	25.11.2016 09.45.48
570	MKR1000	TCP	28	19	1018	337	25.11.2016 09.55.48
571	MKR1000	TCP	28	19	1018	309	25.11.2016 10.05.48
572	MKR1000	TCP	28	19	1018	258	25.11.2016 10.15.48
573	MKR1000	TCP	28	19	1018	309	25.11.2016 10.25.48
574	MKR1000	TCP	28	19	1017	309	25.11.2016 10.35.48
575	MKR1000	TCP	28	19	1017	283	25.11.2016 10.45.48
576	MKR1000	TCP	28	19	1017	309	25.11.2016 10.55.48
577	MKR1000	TCP	28	19	1017	283	25.11.2016 11.05.48

Figure 10-9 Local SQL server Measurement_data table at the 25.11.2016

Discussion

In this thesis, a monitoring system with sensor nodes and data hubs was developed and installed in a test environment at Telemark University College. The system is considered complete, but suggested improvements and further development are as following:

- Recalibrate the implemented sensors so that the temperature, humidity, pressure and Co2 level are more accurate.
- Add units in the HTML page for temperature, humidity, pressure and Co2 level.
- Implementing the remaining humidity, pressure and Co2 level sensors to the Raspberry Pi 3 sensor node.
- Implement more sensors to each sensor node for measuring units such as motion, gas/smoke, noise and ambient light.
- Create "control nodes" for controlling/steering devices such as heaters, doors, ventilation, etc.
- Create a webpage for displaying the measurement data from the Database.

Conclusion

Developed monitoring system worked as intended, but calibration is required for making the measurement data more accurate. Stability cannot be determined before a six-month period has passed. However, based on a five days trial, the stability of the system works well.

Intel NUC data hub running a LabVIEW run-time application, offers a simple GUI with the option of displaying the measurement data using either a HTML page or LabVIEW Data Dashboard. The data hub worked as intended and the measurement data was stored in a Local SQL database.

The Raspberry Pi 3 data hub, running same application as for the Intel NUC data hub, has no GUI, but offers the same option for displaying the measurement data using either the HTML page or LabVIEW Data Dashboard. The data hub worked as intended and is a suitable alternative to the Intel NUC data hub, but without bluetooth support.

MKR1000 sensor node running an Arduino IDE application, offers support for both Wi-Fi and Bluetooth for transferring the temperature, humidity, pressure and Co2 level. The sensor node can operate on a battery supply making it more flexible and it worked as intended with stable readings.

The Raspberry Pi 3 sensor node running a LabVIEW run-time application offers only temperature reading, but can easily be implemented with the remaining sensors for measuring humidity, pressure and Co2 level. The sensor node worked as intended with stable readings, but without bluetooth support.

The prototypes for the sensor nodes and data hubs are all developed with regards to simplicity, appearance and further development potentials, and can be produced in bigger quantities for commercial use.

Security regarding the developed monitoring system and its appurtenant sensor nodes and data hubs are considered high, in an scenario where an intruder specifically want to access the system. Since there are no security updates, some of the hardware used in the sensor nodes are vulnerable for security breaches. However, in a general scenario where intruders do not specifically target the system, the security is considered good enough.

References

1. Wikipedia contributors. *Internet of things*. 2016 [cited 2016 22.09]; Available from: https://en.wikipedia.org/wiki/Internet_of_things.

2. Techcrunch. *Why IoT Security Is So Critical*. 2015 [cited 2016 22.11]; Available from: https://techcrunch.com/2015/10/24/why-iot-security-is-so-critical.

3. Wikipedia contributors. *Bluetooth*. 2016 [cited 2016 22.11]; Available from: https://en.wikipedia.org/wiki/Bluetooth.

4. Wikipedia contributors. *Wi-Fi*. 2016 [cited 2016 22.11]; Available from: https://en.wikipedia.org/wiki/Wi-Fi.

5. Wikipedia contributors. *Serial Peripheral Interface Bus*. 2016 [cited 2016 22.11]; Available from: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus.

6. Wikipedia contributors. *I²C*. 2016 [cited 2016 22.11]; Available from: https://en.wikipedia.org/wiki/I%C2%B2C.

7. Wikipedia contributors. *LabVIEW*. 2016 [cited 2016 22.11]; Available from: https://en.wikipedia.org/wiki/LabVIEW.

8. National Instruments. *Getting Started with Data Dashboard for LabVIEW*. 2014 [cited 2016 22.11]; Available from: http://www.ni.com/tutorial/13757/en/.

9. National Instruments. *Enhancements to the Data Dashboard for LabVIEW app*. 2014 [cited 2016 22.11]; Available from: http://www.ni.com/white-paper/14033/en/.

10. Wikipedia contributors. *Arduino*. 2016 [cited 2016 22.11]; Available from: https://en.wikipedia.org/wiki/Arduino.

11. Wikipedia contributors. *SQL*. 2016 [cited 2016 22.11]; Available from: https://en.wikipedia.org/wiki/SQL.

12. Datarealm. *Five Advantages & Disadvantages Of MySQL*. 2014 [cited 2016 22.11]; Available from: https://www.datarealm.com/blog/five-advantages-disadvantages-of-mysql/.

13. ActiveDBSoft. *FlySpeed SQL Query*. [cited 2016 22.11]; Available from: http://www.activedbsoft.com/screenshots-querytool.html?screen=1.

14. Microchip Technology Inc. *Low-Power Linear Active Thermistor™ ICs*. 2007 [cited 2016 18.11]; Available from: http://ww1.microchip.com/downloads/en/DeviceDoc/20001942G.pdf.

15. Microchip Technology Inc. *MCP9700A*. [cited 2016 18.11]; Available from: http://www.microchip.com/wwwproducts/en/MCP9700A.

16. Amethem Inc. *What Is An NTC Thermistor*. [cited 2016 18.11]; Available from: http://www.ametherm.com/thermistor/what-is-an-ntc-thermistor.

17. Elcodis Company. *NTCLE100E3103JB0*. [cited 2016 18.11]; Available from: http://elcodis.com/parts/2242563/NTCLE100E3103JB0.html.

18. Wikipedia contributors. *Steinhart–Hart equation*. 2016 [cited 2016 18.11]; Available from: https://en.wikipedia.org/wiki/Steinhart%E2%80%93Hart_equation.

19. Vishay. *NTC Thermistors, Radial Leaded, Standard Precision*. 2016 [cited 2016 18.11]; Available from: http://www.vishay.com/docs/29049/ntcle100.pdf.

20. Microchip Technology Inc. *TC74*. [cited 2016 18.11]; Available from: http://www.microchip.com/wwwproducts/en/TC74.

21. Newark element14. *TC74A0-5.0VAT*. [cited 2016 18.11]; Available from: http://www.newark.com/microchip/tc74a0-5-0vat/thermal-sensor-2c-to220-5/dp/92C6554.

22. Microchip Technology Inc. *Tiny Serial Digital Thermal Sensor*. 2012 [cited 2016 18.11]; Available from: http://ww1.microchip.com/downloads/en/DeviceDoc/21462D.pdf.

23. Arduino. *MQ Gas sensors*. [cited 2016 22.11]; Available from: http://playground.arduino.cc/Main/MQGasSensors.

24. Lampa Tronics. *MQ-135 (Gas Sensor)*. [cited 2016 06.12]; Available from: http://lampatronics.com/product/mq-135-gas-sensor.

25. Olimex. *TECHNICAL DATA MQ-135 GAS SENSOR*. [cited 2016 22.11]; Available from: https://www.olimex.com/Products/Components/Sensors/SNS-MQ135/resources/SNS-MQ135.pdf.

26. Winsen. *MH-Z19 NDIR CO2 SENSOR*. [cited 2016 22.11]; Available from: http://www.winsen-sensor.com/products/ndir-co2-sensor/mh-z19.html.

27. Winsen. *Intelligent Infrared CO2 Module (Model: MH-Z19)*. 2015 [cited 2016 22.11]; Available from: http://www.winsen-sensor.com/d/files/PDF/Infrared%20Gas%20Sensor/NDIR%20CO2%20SENSOR/MH-Z19%20CO2%20Ver1.0.pdf.

28. adafruit. *Adafruit BME280 Humidity + Barometric Pressure + Temperature Sensor Breakout*. 2015 [cited 2016 22.11]; Available from: https://learn.adafruit.com/adafruit-bme280-humidity-barometric-pressure-temperature-sensor-breakout/overview.

29. eBay Inc. *Breakout BME280 Module Digital Temperature/Humidity/Barometric Pressure Sensor*. 2016 [cited 2016 22.11]; Available from: http://www.ebay.com.au/itm/Breakout-BME280-Module-Digital-Temperature-Humidity-Barometric-Pressure-Sensor-/262384536825?hash=item3d17564cf9:g:d1QAAOSwMmBVyWJb.

30. Bosch Sensortec. *BME280 Combined humidity and pressure sensor*. 2015 [cited 2016 22.11]; Available from: https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BME280_DS001-11.pdf.

31. Arduino. *Arduino MKR1000 & Genuino MKR1000*. [cited 2016 22.11]; Available from: https://www.arduino.cc/en/Main/ArduinoMKR1000.

32. HackerBoards.com. *Arduino aims $35, WiFi-enabled hacker board at IoT apps*. 2016 [cited 2016 06.12]; Available from: http://hackerboards.com/arduino-aims-35-dollar-wifi-enabled-hacker-board-at-iot-apps.

33. Arduino. *Arduino Pro Mini*. [cited 2016 22.11]; Available from: https://www.arduino.cc/en/Main/ArduinoBoardProMini.

34.   Sparkfun Electronics. *Arduino Pro Mini 328 - 5V/16MHz*. [cited 2016 22.11];
        Available from: https://www.sparkfun.com/products/11113.

35.   Wikipedia contributors. *Raspberry Pi*. 2016 [cited 2016 22.11]; Available from:
        https://en.wikipedia.org/wiki/Raspberry_Pi.

36.   The MagPi. *RASPBERRY PI 3 IS OUT NOW! SPECS, BENCHMARKS & MORE*.
        2016 [cited 2016 22.11]; Available from:
        https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/.

37.   Wikipedia contributors. *Next Unit of Computing*. 2016 [cited 2016 22.11]; Available
        from: https://en.wikipedia.org/wiki/Next_Unit_of_Computing.

38.   Team-MediaPortal. *Intel NUC DC3217IYE / DC3217BY*. [cited 2016 22.11];
        Available from: http://wiki.team-
        mediaportal.com/4_HTPC/HTPC_Hardware/Mini_HTPCs_(%22All_in_One%22)/Int
        el_NUC_DC3217IYE_%2F%2F_DC3217BY.

39.   Intel Corporation. *Mini PC: Intel® NUC Kit DC53427HYE and Board D53427RKE*.
        [cited 2016 22.11]; Available from:
        http://www.intel.com/content/www/us/en/nuc/nuc-kit-dc53427hye-board-
        d53427rke.html.

40.   wikispaces.com. *BlueTooth-HC05-Modules-How-To*. [cited 2016 06.12]; Available
        from: https://arduino-info.wikispaces.com/BlueTooth-HC05-HC06-Modules-How-
        To.

41.   Cambridge Silicon Radio. *BC417143B-DS-001Pg*. 2005 [cited 2016 06.12];
        Available from: http://yourduino.com/docs/CSR-BC417-datasheet.pdf.

42.   sparkfun Electronics. *WiFi Module - ESP8266*. [cited 2016 06.12]; Available from:
        https://www.sparkfun.com/products/13678.

43.   Espressif Inc. *ESP8266EX Datasheet*. 2015 [cited 2016 06.12]; Available from:
        http://download.arduino.org/products/UNOWIFI/0A-ESP8266-Datasheet-EN-
        v4.3.pdf.

44.   Wikipedia contributors. *iPad*. 2016 [cited 2016 22.11]; Available from:
        https://en.wikipedia.org/wiki/IPad.

45.   Apple Inc. *iPad 2 - Technical Specifications*. 2013 [cited 2016 22.11]; Available
        from: https://support.apple.com/kb/sp622?locale=en_US.

46.   Arduino. *MKR1000 Battery Life*. [cited 2016 22.11]; Available from:
        https://www.arduino.cc/en/Tutorial/MKR1000BatteryLife.

47.   Raspberry Pi Wikipedia contributors. *RPI Lithium Battery Expansion Board
        SKU:435230*. 2016 [cited 2016 22.11]; Available from:
        http://www.raspberrypiwiki.com/index.php/RPI_Lithium_Battery_Expansion_Board_
        SKU:435230.

48.   HackerBoards.com. *Mini UPS targets NUCs and other mini-PCs*. 2015 [cited 2016
        22.11]; Available from: http://hackerboards.com/mini-ups-targets-nucs-and-other-
        mini-pcs/.

49.		RasPi.TV. *How Much Power Does Raspberry Pi3B Use? How Fast Is It Compared To Pi2B?*. 2016 [cited 2016 22.11]; Available from: http://raspi.tv/2016/how-much-power-does-raspberry-pi3b-use-how-fast-is-it-compared-to-pi2b.

50.		The Power Consumption Database. *Intel NUC DC53427HYE*. 2013 [cited 2016 22.11]; Available from: http://www.tpcdb.com/product.php?id=1788.

51.		Microchip Technology Inc. *2.7V Dual Channel 10-Bit A/D Converter with SPI Serial Interface*. 2011 [cited 2016 23.11]; Available from: http://ww1.microchip.com/downloads/en/DeviceDoc/21294E.pdf.

FMH606 Master's Thesis

Title: Data Cloud Platform for Data Management, Logging, Control and Monitoring

TUC Supervisor: Hans-Petter Halvorsen, Nils-Olav Skeie

External Partner: National Instruments

Task Description:

Analyze, Design and Development of a Platform for Data Management, Logging, Control and Monitoring. The Platform will have many Applications, such as Smart House/Home Automation solutions, Process Control, SCADA Systems, etc.

The main goal is to Design and Implement an Open Platform for Data Management, Logging, Control and Monitoring.

Programming Languages should be LabVIEW, Arduino Programming Language, C#, MATLAB.

The Platform should be easy to deploy and manage. Web Programming like HTML, JavaScript, ASP.NET, etc. are also of interest.

Data Cloud Platform for Data Management, Logging, Control and Monitoring Overview:
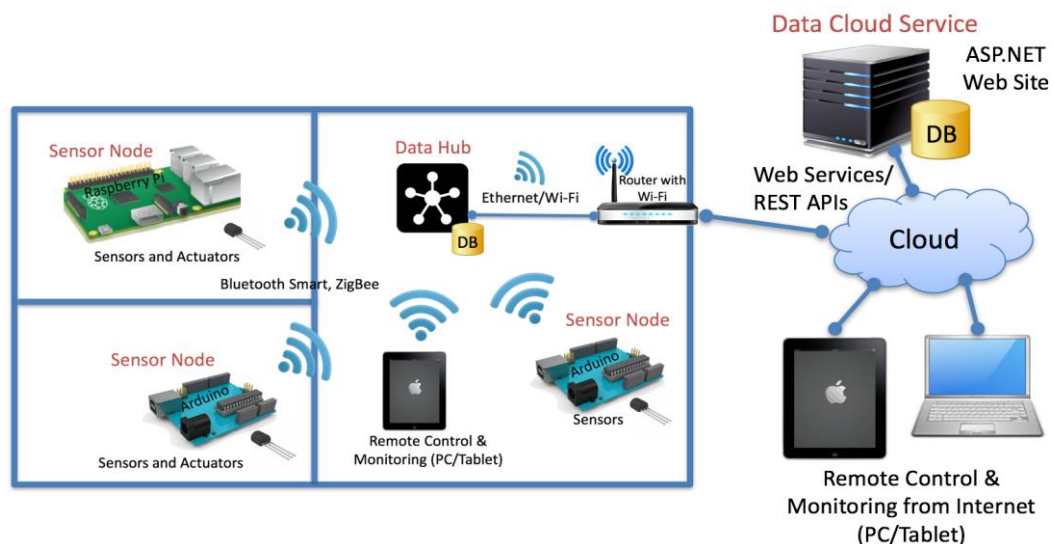
Some of the following topics should be investigated in this project:

1. General Overview of existing Solutions and explore **IoT Protocols** suitable (open APIs) for Home Automation Systems, e.g., IFTTT, X10, ZigBee, Z-wave, KNX, Bluetooth Smart, etc.
2. A "**Data Cloud Service**" for Data Management, Data Storage, Data Analysis and Data Collection shall be developed. A Database Server should be part of the Data Cloud Service for Management, Storage, Analysis and Monitoring of Data from multiple local "Data Hubs". The Data Cloud Service prototype should be installed on a server located at USN, in addition to a Microsoft Azure Server, but it should also be possible to deploy it other places.
   The development of the Data Cloud Service could consist of the following parts:
     o SQL Server Database for data storage.
     o Development of **Web Services/REST APIs,** both C# or LabVIEW can be used.
     o Development of a **ASP.NET Web Site** for Data Management and Monitoring.
3. A "**Data Hub**" for local collection of Data from multiple **"Sensor Nodes" (IoT Devices)** shall be developed. The Data Hub should also include a local Database for storage of Data. The "Data Hub" should support different open Protocols for communication with different "Sensor Nodes" (IoT Devices). The "Data Hub" can e.g., be an **Intel NUC** Computer, a **myRIO** unit from National instruments or a **Raspberry Pi 3** device. The "Data Hub" should be communicating with the "Data Cloud Service" using Web Services/REST APIs or other forms of standard APIs. The Hub should support different Wireless standards.
4. Explore **Sensors** (Indoor) that can be used with the Sensor Nodes, such as Temperature (°C), Humidity (%), CO_2 level (ppm), Noise level (dB), Ambient light (cd), Barometric Pressure ($mbar$), Motion sensor, Gas/Smoke sensor. Outdoor Sensor Nodes could also be considered.
5. Development of different <u>embedded</u> "**Sensor Nodes" (IoT Devices)** for collection data from Sensors and Control of Actuators. The Sensor Nodes shall send and receive Data from the Data Hub using Wireless Communication. The Data Hub will then communicate with the Data Cloud Service through the Internet Router. The "Sensor Nodes" should have a "professional" packaging with a protected case. It should also be battery powered. Battery, Sensors, etc. should preferably be inside the case. The Sensor Nodes can either connect to the "Data Hub" or directly to the "Data Cloud Service", depending of what kind of wireless protocols the "Sensor Nodes" are using. It's important that the "Sensor Nodes" are "embedded", meaning they are able to work without a PC connected to them.
6. Datalogging and Control from different **IoT Devices** ("Sensor Nodes") such as Arduino and **Raspberry Pi 3**, National Instruments myRIO, and USB based DAQ devices, etc. E.g., LabVIEW, LabVIEW LINX, standard Arduino Programming or Windows 10 IoT Core/C# can be used to program these devices.
     o **Especially, Raspberry Pi 3 and LabVIEW LINX should be explored** due to its embedded features.
     o **The new Bluetooth and Wi-Fi support for Raspberry Pi 3 should be explored.**
     o **The new Windows 10 IoT Core Anniversary Update should be explored**

7. Look into **Battery requirements** for the nodes and explore use of rechargeable batteries. Communication needs to be Low-Power since the "IoT Devices" should be placed around wirelessly with the use of only Battery Power. Explore new Wi-Fi and Bluetooth standards.
8. Explore **Wireless Communication**, such as Wi-Fi HaLow (IEEE 802.11ah), Bluetooth Smart (including next generation), XBee/ZigBee, together with e.g., Raspberry Pi, Arduino myRIO for Logging Sensor Data and Control of suitable Actuators in Smart Houses. The new Bluetooth and Wi-Fi support for Raspberry Pi 3 should be explored.
9. Use of **Data Dashboard for LabVIEW** App (https://www.ni.com/mobile/) for Monitoring and Control. Management and Monitoring should also be considered created from scratch using HTML, JavaScript, etc., which are connecting to the same Web Services.
10. Look into different **Security aspects** with IoT, Home Automation, etc.
11. A **Prototype** should be built into a **Suitcase** (2 units), including A Data Hub, pre-bundled Sensor Nodes with Sensors, such as Temperature (°C), Humidity (%), CO_2 level (ppm), Noise level (dB), Ambient light (cd), Barometric Pressure ($mbar$), Motion sensor, Gas/Smoke sensor. The packaging inside the suitcase should be and look "professional". A Sensor Node should also be installed in the Supervisors Office for surveillance of the Air Quality.
12. Integration of Weather Data from Weather Stations such as, e.g., Netatmo (https://www.netatmo.com) and general Web Services/REST APIs like "yr.no", etc.

Task Background: Data Logging and Monitoring is important in many aspects from Process Control, Automation Systems, Industry 4.0, Smart House and Home Automation Solutions, Internet of Things (IoT) Smart Cities, Smart Grid, Sensors (Cloud Computing and networks of data-gathering sensors), Bug Data, etc.

Student Category: IIA students

Practical Arrangements: None

Signatures:

Supervisor (date and signature):

Students (date and signature):

Appendix B – Price estimation for all components

The two tables shows the price estimation for the components from both eBay and Norwegian suppliers, with some additional components as alternatives to the ones used for the prototypes.

Components from eBay

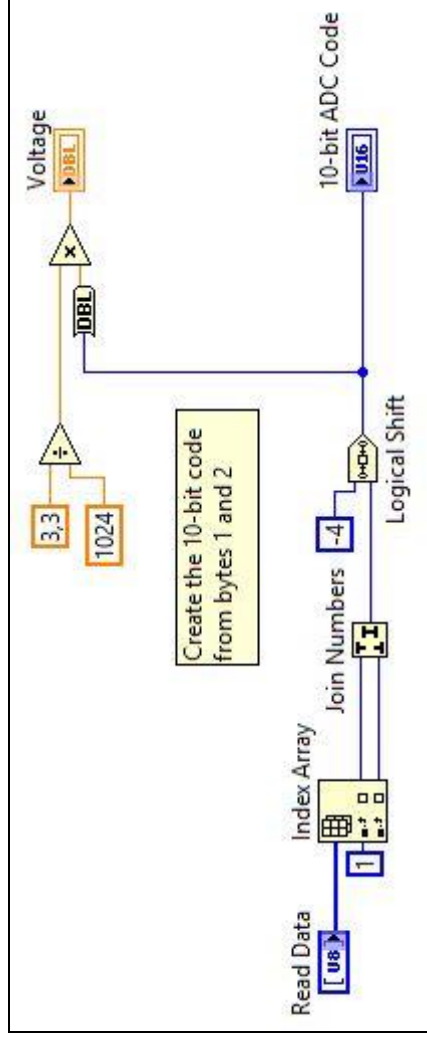| Item	Element#	Supplier	Unit price (incl. VAT)	Shipping	Cited date
Enclosure Box (129x41x67mm)	252460607715	eBay.com	kr 54,38	Free	25.09.2016
DHT11	151766751268	eBay.com	kr 8,17	Free	25.09.2016
BMP180	252411043336	eBay.com	kr 16,75	Free	25.09.2016
Arduino Uno R3	182169283852	eBay.com	kr 25,45	Free	25.09.2016
9V Battery Holder Box	141999152924	eBay.com	kr 10,42	Free	25.09.2016
Arduino Pro Mini	152110232869	eBay.com	kr 15,84	Free	25.09.2016
USB to TTL Serial Adapter	172241269495	eBay.com	kr 16,09	Free	25.09.2016
ESP8266	221909856153	eBay.com	kr 17,08	Free	25.09.2016
Enclosure Box (100x60x25mm)	262394740653	eBay.com	kr 8,17	Free	25.09.2016
MG811	172014508858	eBay.com	kr 327,41	Free	25.09.2016
HC05	251886755046	eBay.com	kr 27,12	Free	25.09.2016
Raspberry Pi 3 Model B	272344301346	eBay.com	kr 247,37	Free	25.09.2016
Genuino MKR1000	252385580189	eBay.com	kr 453,84	kr 62,18	25.09.2016
Li-polymer battery (3,7V 1200mAh)	291847202339	eBay.com	kr 36,68	Free	25.09.2016
Micro SD 16GB (Class 10)	252394455711	eBay.com	kr 41,18	Free	25.09.2016
4AA Battery Holder box	281008033227	eBay.com	kr 28,33	Free	25.09.2016
3AA Battery Holder Box	331583401812	eBay.com	kr 6,12	Free	25.09.2016
Micro USB Wall charger (EU plug)	152199717485	eBay.com	kr 44,00	Free	23.11.2016
Barebone Intel NUC (NUC5CPYH)	291874727049	eBay.com	kr 1 841,00	Free	25.09.2016
Micro USB charging cable (10 cm)	371680406059	eBay.com	kr 6,19	Free	24.09.2016
Battery Expansion Board	112053437983	eBay.com	kr 173,19	Free	25.09.2016
MH-Z19	152259567650	eBay.com	kr 244,13	Free	05.10.2016
MQ-135	201446441473	eBay.com	kr 28,61	Free	05.10.2016
BME280	281745479989	eBay.com	kr 54,54	free	23.11.2016
Enclosure Box (Raspberry Pi 3)	301697165657	eBay.com	kr 16,65	Free	23.11.2016

Components from Norwegian suppliers

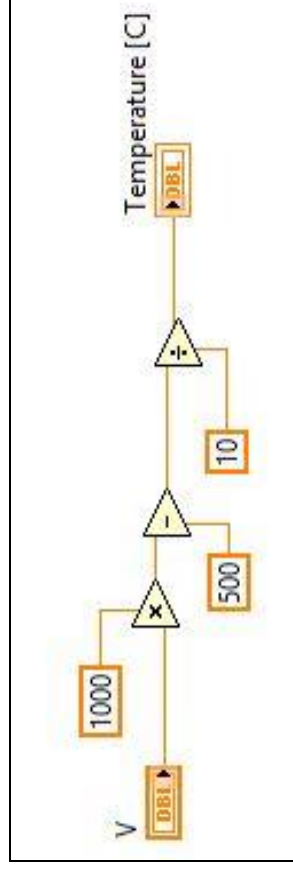| Item	Art number#	Supplier	Unit price (inc. VAT)	Shipping	Cited date
Enclosure Box (123x72x39mm)	89004	kjell.com	kr 49,90	kr 79,00	27.09.2016
DHT11	87877	kjell.com	kr 59,90	kr 79,00	27.09.2016
BMP180	NaN	kultogbillig.no	kr 106,00	kr 39,00	27.09.2016
Arduino Uno R3	87960	kjell.com	kr 159,90	kr 79,00	27.09.2016
9V Battery Holder Box	39781	kjell.com	kr 49,90	kr 79,00	27.09.2016
Arduino Pro Mini	87964	kjell.com	kr 129,90	kr 79,00	27.09.2016
USB to TTL Serial Adapter	87872	kjell.com	kr 149,90	kr 79,00	27.09.2016
ESP8266	87947	kjell.com	kr 109,90	kr 79,00	27.09.2016
Enclosure Box (101x60x26mm)	89002	kjell.com	kr 39,90	kr 79,00	27.09.2016
MG811	**Not available from Norwegian suppliers**				27.09.2016
HC05	NaN	kultogbillig.no	kr 160,00	kr 79,00	27.09.2016
Raspberry Pi 3 Model B	88000	kjell.com	kr 499,00	kr 79,00	27.09.2016
Genuino MKR1000	**Not available from Norwegian suppliers**				27.09.2016
Li-polymer battery (3,7V 1200 mAh)	87923	kjell.com	kr 149,90	kr 79,00	27.09.2016
Micro SD 16GB (Class 10)	92956	kjell.com	kr 199,90	kr 79,00	27.09.2016
4AA Battery Holder box	39780	kjell.com	kr 49,90	kr 79,00	27.09.2016
3AA Battery Holder Box	39744	kjell.com	kr 29,90	kr 79,00	27.09.2016
Micro USB Wall charger (EU plug)	95719	kjell.com	kr 199,90	kr 79,00	27.09.2016
Barebone Intel NUC (NUC5CPYH)	862882	komplett.no	kr 1 328,00	Free	27.09.2016
Micro USB charging cable (10 cm)	38-5875	clasohlson.com	kr 69,90	kr 49,00	27.09.2016
Battery Expansion Board	**Not available from Norwegian suppliers**				27.09.2016
MH-Z19	**Not available from Norwegian suppliers**			0	05.10.2016
MQ-135	NaN	kultogbillig.no	kr 70,00	kr 39,00	05.10.2016
BME280	138191	digitalimpuls.no	kr 250,00	kr 79,00	23.11.2016
Enclosure Box (Raspberry Pi 3)	87206	kjell.com	kr 99,90	kr 79,00	23.112016

Appendix C – LabVIEW G-code for the simple version sensor node

The LabVIEW G-code for the simple version sensor node consists of four sections:
Main VI, GenBytes SubVI, ConvertADCVoltage SubVI and VoltToDegrees SubVI

Main VI

GenBytes SubVI

ConvertADCVoltage SubVI

Voltage

Read Data

Index Array

Join Numbers

Logical Shift

10-bit ADC Code

3,3

1024

Create the 10-bit code
from bytes 1 and 2

-4

VoltToDegrees SubVI

Temperature [C]

V

1000

500

10
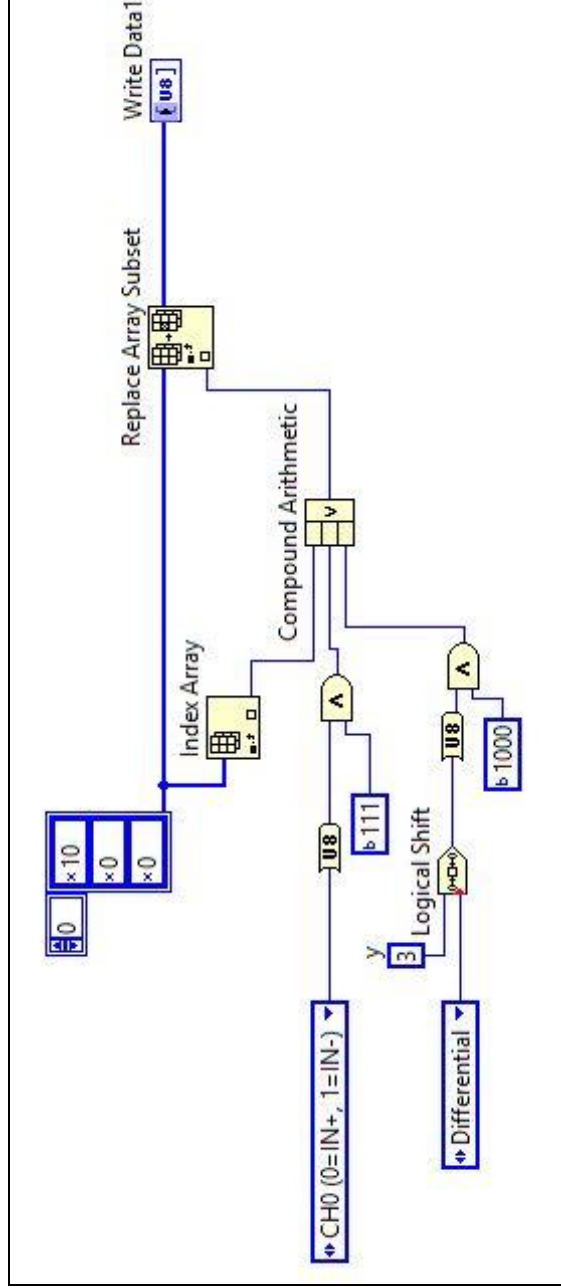
Appendix D – Arduino IDE (C++) code for MKR1000 sensor node prototype

The Arduino IDE code for the MKR1000 sensor node prototype consists of three sections: Main, WiFiStatus and SerialMonitor.

Main

```
/*

Code for MKR1000 Sensor node.

Created 2016 by S.Schulstock.

Libraries used: SPI.h / WiFi101.h / BME280.h / MQ135.h

*/

/* ---- Include ---- */

#include <SPI.h>

#include <WiFi101.h>

#include <BME280.h>

#include "MQ135.h"

/* ---- END Include ---- */
```

```
/* ---- Global Variables ---- */

float temp, hum, pres, ppm;

//BME280 (Temperature, Humidity, Pressure):

BME280 bme;                              // BME280 library declared (bme).

uint8_t pressureUnit(1);                 // Set pressure unit (0 = Pa, 1 = hPa, 2 = Hg, 3 = atm).

bool metric = true;                      // Set metric or imperial system.

//MQ135 (Co2 level):

MQ135 gasSensor = MQ135(6);              // Analog input set to "6" for the gasSensor function.

//WiFi:

char ssid[] = "IoT";                     // Network SSID.

char pass[] = "HOMEAUTO";                // Network password.

int keyIndex = 0;                        // Network key Index number (needed for WEP).

int status = WL_IDLE_STATUS;

WiFiServer server(80);                   // TCP/IP Port number set to 80

boolean alreadyConnected = false;        // Check whether if client was connected previously

/* ---- END Global Variables ---- */
```

```
/* ---- Setup ---- */
void setup()
{
    Serial1.begin(9600);                    // Set Serial1(Bluetooth) baud rate to 9600.

    while ( status != WL_CONNECTED)         // Try to connect to Wifi network.
    {
        status = WiFi.begin(ssid, pass);    // Send ssid and pass key.
        delay(3000);                        // Wait/delay 3 seconds for connection to be established.
    }
    server.begin();                         // Start the WiFi server.
    bme.begin();                            // Enable(begin) the bme function.
    //printWifiStatus();                    // "Optional" : Print WiFi status (ssds, IP and RSSI).
}
/* ---- END Setup ---- */
```

```
/* ---- Loop ---- */

void loop()

{

// Get measurement data (from sensors):

bme.ReadData(pres, temp, hum, metric, pressureUnit);      // ReadData function (Pressure, Temperature, Humidity).

ppm = gasSensor.getPPM();                                  // getPPM function (Co2 level).

//Send measurement data using Bluetooth (if available):

if(Serial1.available()>0)                                  //If Bluetooth connection established.

{

Serial1.println(String(temp));                            // Write temperature to LabVIEW as String.

Serial1.println(String(hum));                             // Write humidity to LabVIEW as String.

Serial1.println(String(pres));                            // Write pressure to LabVIEW as String.

Serial1.println(String(ppm));                             // Write Co2 concentration to LabVIEW as String.

Serial1.println("MKR1000      ");                         // Write Sensor Tag to LabVIEW as String.

}
```

```
//Send measurment data using TCP/IP (if available):

WiFiClient client = server.available();            // Wait for a new available client.

if (client)

{

    if (!alreadyConnected)

    {

        client.flush();                            // Clear out the input buffer.

        alreadyConnected = true;                   // Set "alreadyConnected to "True" if already Connected.

    }

    if (client.available() > 0)

    {

        server.println(String(temp));              // Write temperature to LabVIEW as String.

        server.println(String(hum));               // Write humidity to LabVIEW as String.

        server.println(String(pres));              // Write pressure to LabVIEW as String.

        server.println(String(ppm));               // Write Co2 concentration to LabVIEW as String.

        server.println("MKR1000        ");         // Write Sensor Tag to LabVIEW as String.

    }

}
```

```
//SerialMonitor();           // "Optional" : Print sensor data to Serial Monitor.

delay(2000);                 // Wait 2000ms (Response time for BME250 min 2000ms).

}

/* --- END Loop --- */
```

WiFiStatus

```
/*
Code for printing WifiStatus (SSID, IP and RSSI) for MKR1000 Sensor node.

Created 2016 by S.Schulstock.

Based on example from WiFi101.h Library.
*/

/* --- Function --- */
void printWifiStatus()
{

// Print the SSID for attached network:
Serial.print("SSID: ");
```

```
Serial.println(WiFi.SSID());

IPAddress ip = WiFi.localIP();

// Print the WiFi shield's IP address:

Serial.print("IP Address: ");

Serial.println(ip);

// Print the received signal strength:

long rssi = WiFi.RSSI();

Serial.print("signal strength (RSSI):");

Serial.print(rssi);

Serial.println(" dBm\n");

}

/* ---- END Function ---- */
```

```
/*

    Code for printing sensor data to Serial Monitor (Temperature, Humidity,Pressure, Co2).

    Created 2016 by S.Schulstock.

*/

/* ---- Function ---- */
void SerialMonitor()

{

//Print the Temperature, Humidity, Pressure, Co2 and Tag to the SerialMonitor:

Serial.println(temp);

Serial.println(hum);

Serial.println(pres);

Serial.println(ppm);

Serial.println("MKR1000");

Serial.println();

}

/* ---- END Function ---- */
```

Appendix E – LabVIEW G-code for the Raspberry Pi 3 sensor node prototype

The LabVIEW G-code for the Raspberry Pi 3 sensor node prototype consists of five sections:
Main VI, I2C SubVI, Bit_To_Temp SubVI, TCP SubVI and Bluetooth SubVI.

Main VI

I2C SubVI

SubVI for reading "Bit" values from TC74 using I2C communcation Protocol
----
Created by S.Schulstock

Temperature (C)

Bit_to_Temp SubVI

LINX Resource

error out

I2C Read.vi

I2C Channel (1)

I2C Slave Address (hex) 48

Bytes To Read 2

LINX Resource in

error in (no error)

Timeout (1000 ms)

1000
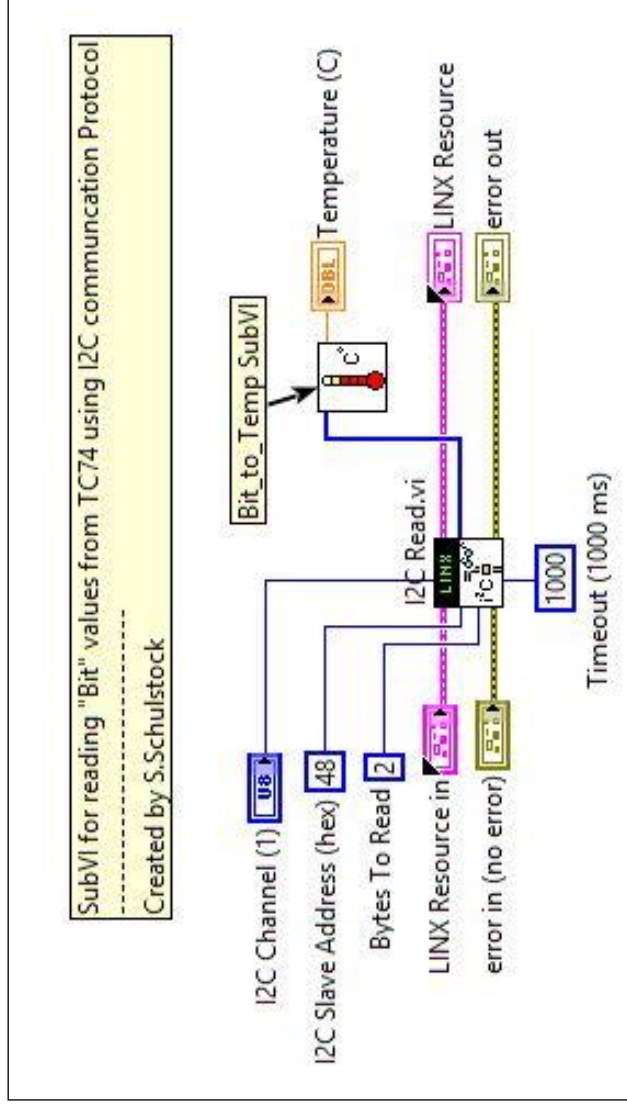
Bit_To_Temp SubVI

SubVI for converting "Bit" value to "Temperature" value
- - - - - - - - - - - - - - - -
Created by S.Schulstock

Temperature (DegC)

U16 Two's Compliment To Decimal.vi

Boolean Array To Number

Resoultion (set to 0.5)

Temperature adjustment

Array Subset

Number To Boolean Array

Data Input(bit) from TC74

Index Array

Join Numbers

To Unsigned Word Integer

Index and Length set for the 0.5 resolution

TCP SubVI

SubVI for sending measurement data using "TCP"

Created by S.Schulstock

Measurement data formated into string

format string
%2.2f, %2.2f, %2.2f, %2.2f, %s

Temperature
Humidity
Pressure
Co2

Tag   Raspberry Pi 3

Format Into String

TCP Write

TCP Close Connection

Writes/Sends data over TCP

Close the connection after measurement data has been sent

TCP Listen.vi

Port

Creates a listener and waits for an accepted TCP network connection at specified port (80)

Bluetooth SubVI

SubVI for sending measurement data using "Bluetooth"

Created by S.Schulstock

Wait(delay) 2000ms)

2000

Close the connection after measurement data has been sent

Bluetooth Close Connection

Bluetooth Close Connection

Writes/Sends data over Bluetooth

Close connection if there are no requests

Bluetooth Close Connection

Bluetooth Write

Wait for connection request

Bluetooth Wait On Listener

uuid B62C4E8D-62CC-404b-BBBF-BF3E3BBB1374

Bluetooth Create Listener

Creates a listener/connection with specified uuid

%2.2f, %2.2f, %2.2f, %2.2f, %s

Temperature

Humidity

Pressure

Co2

Tag Raspberry Pi 3

Format Into String

Measurement data formated into string

Appendix F – LabVIEW G-code for the Intel NUC data hub prototype

The LabVIEW G-code for the Intel NUC data hub prototype consists of five sections:
Main VI, Bluetooth SubVI, TCP SubVI, Convert_Data SubVI and SQL SubVI.

Main VI
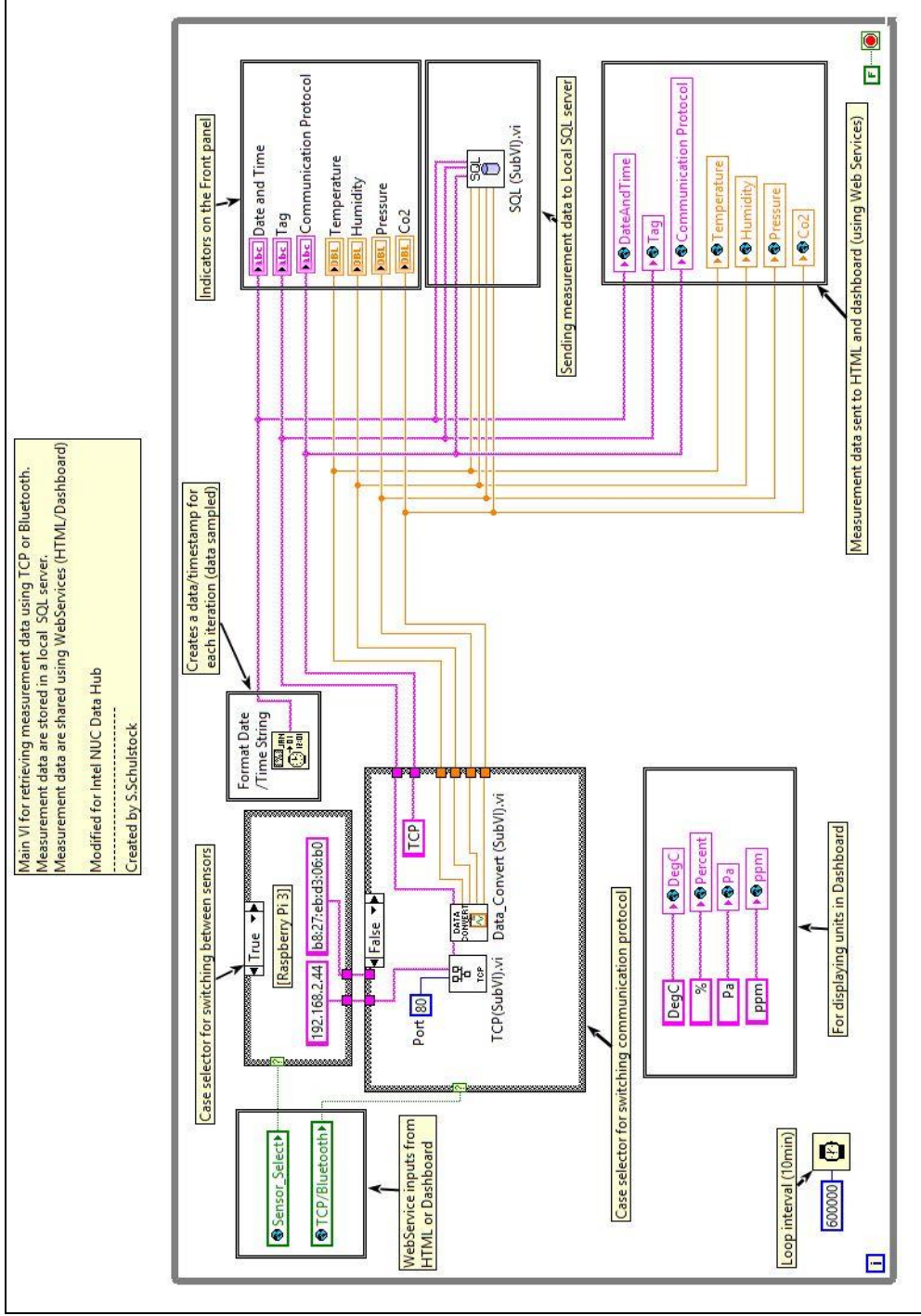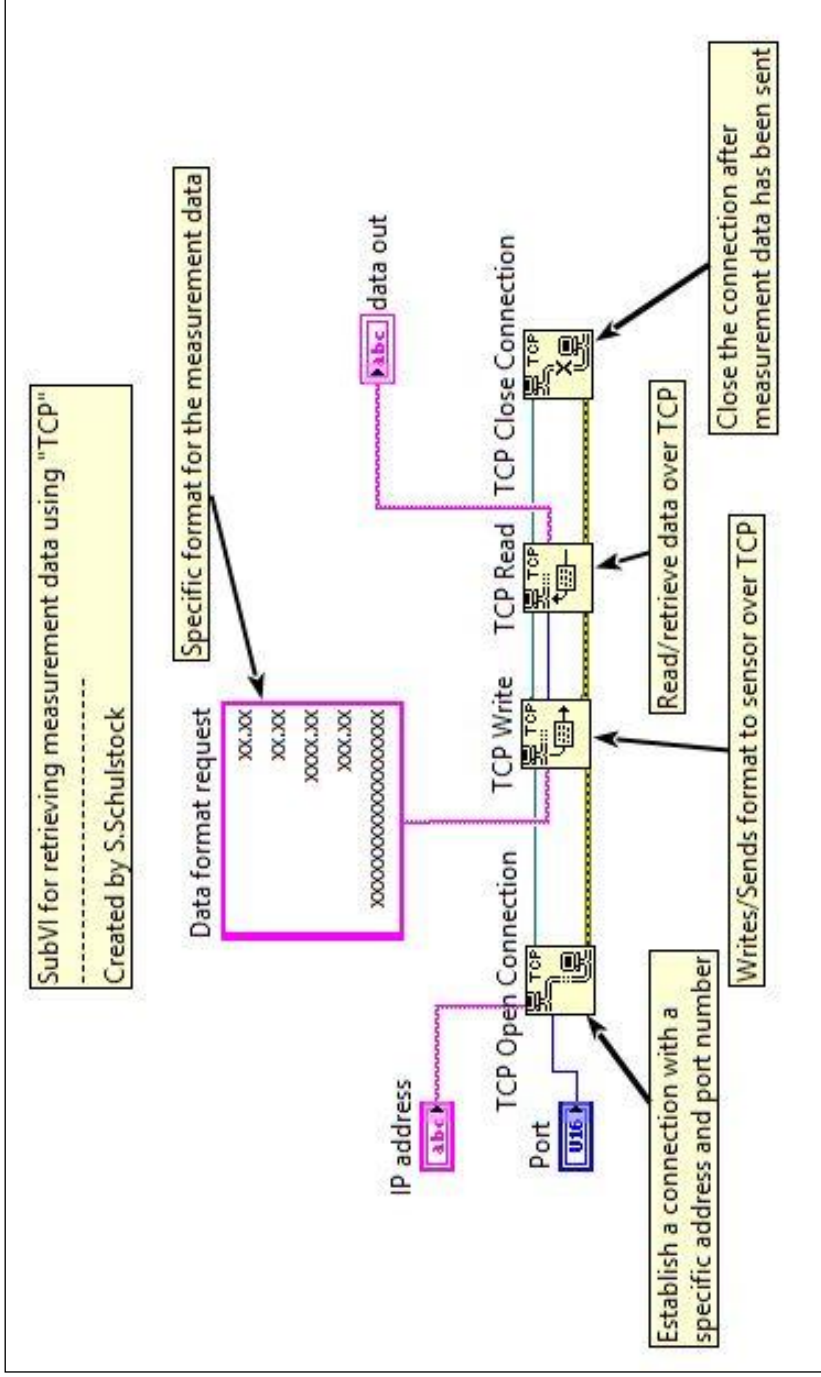
Bluetooth SubVI

SubVI for retrieving measurement data using "Bluetooth"
- - - - - - - - - - - - - -
Created by S.Schulstock

Specific format for the measurement data

Data format request

| xx.xx |
| xx.xx |
| xxxx.xx |
| xxx.xx |
| xxxxxxxxxxxxxxxxxx |

Address

Bluetooth Open Connection   Bluetooth Write   Bluetooth Read   Bluetooth Close Connection

data out

channel (0)

1

Establish a connection with a specific address and channel

Writes/Sends format to sensor over Bluetooth

Read/retrieve data using bluetooth

Close the connection after measurement data has been sent

TCP SubVI

SubVI for retrieving measurement data using "TCP"
- - - - - - - - - -
Created by S.Schulstock

Specific format for the measurement data

Data format request

xx.xx
xx.xx
xxxx.xx
xxx.xx
xxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxx

data out

IP address

TCP Open Connection

Port

TCP Write

TCP Read

TCP Close Connection
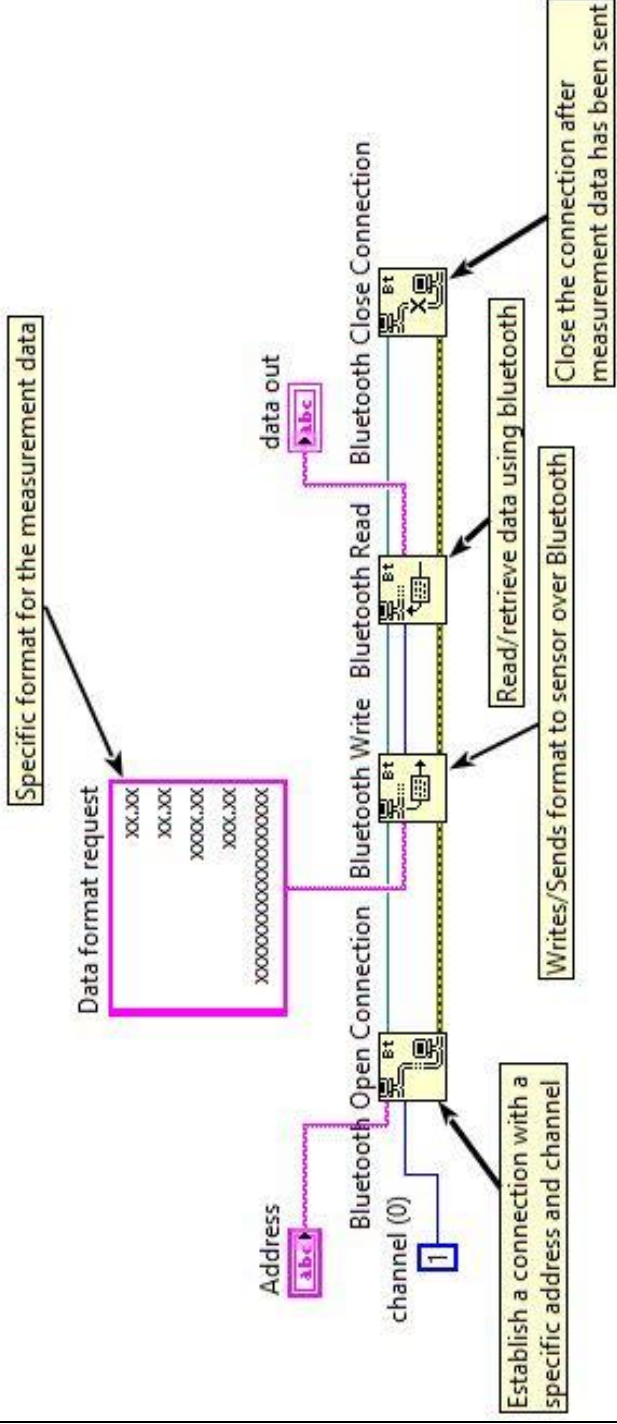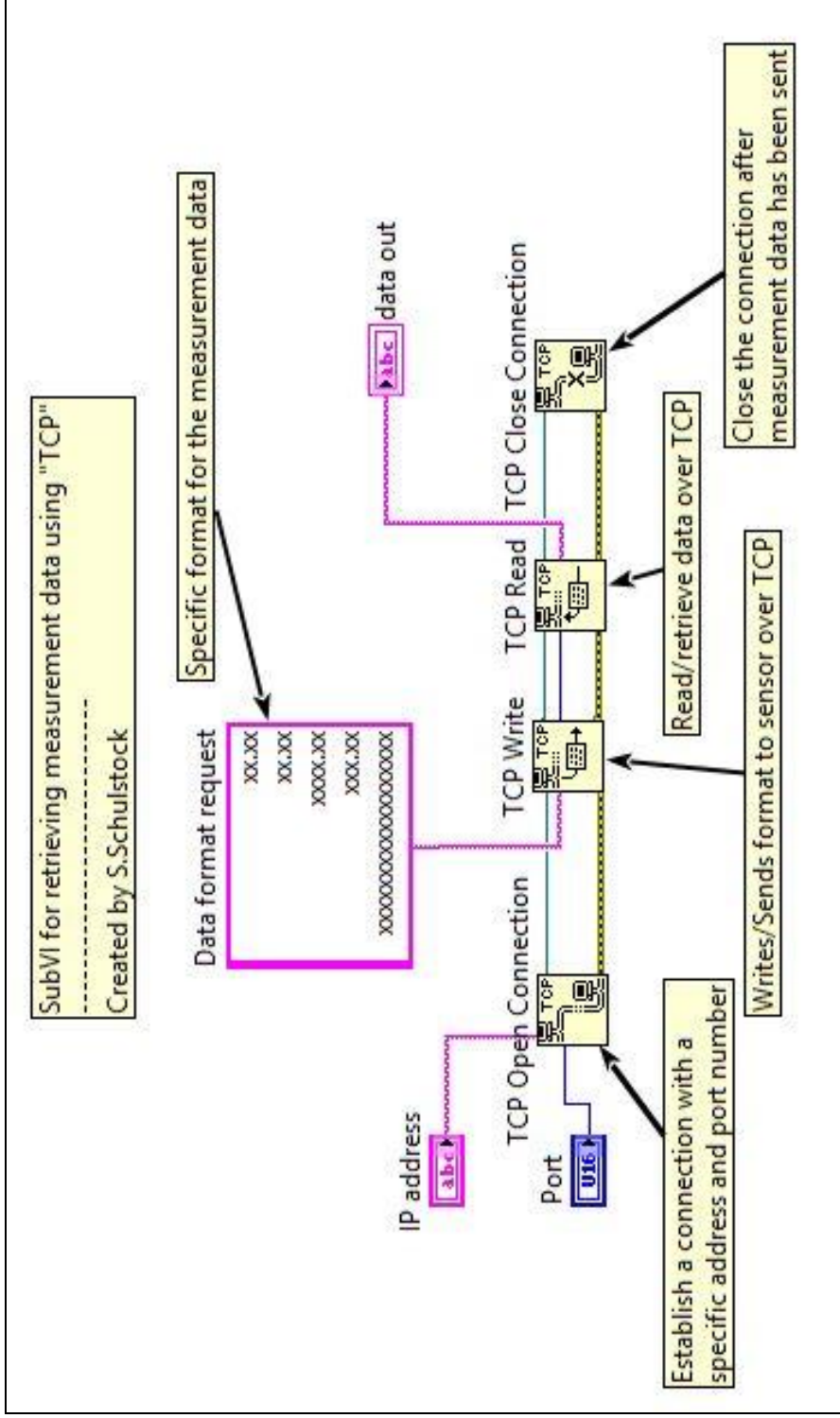
Establish a connection with a specific address and port number

Writes/Sends format to sensor over TCP

Read/retrieve data over TCP

Close the connection after measurement data has been sent

Convert_Data SubVI

SubVI for splitting/converting "data in" (string) to numbers and string
----
Created by S.Schulstock

Creates a substring (based on a offset)

Interprets string (based on a offset) to floating-point numbers

Use system decimal point

String Subset

Fract/Exp String To Number

Tag

Temperature

Humidity

Pressure

Co2

offset 30

offset 7

offset 14

offset 23

data in

SQL SubVI

SubVI for inserting measurement data into a Local SQL server
- - - - - - - - - - - - - - - - - -
Created by S.Schulstock

Opens a connection to SQL server

Inserts data into SQL server

Closes the connection to the SQL server

SQL Close.vi

SQL Execute.vi

SQL Open.vi

Name of the database

Data_Hub_Database

Format Into String

Formats the measurement data into
a String based on the the SQL query

Tag

Communication Protocol

Temperature

Humidity

Pressure

Co2

DateAndTime

SQL query for inserting the measurement data
in correct format (based on the tables in SQL)

SQL Query %;; insert into Measurement_Data (Tag, Communication_Protocol, Temperature, Humidity, Pressure, Co2, DateAndTime) values ('%s', '%s', %f, %f, %f, %f, '%s')

Appendix G – LabVIEW G-code for the Raspberry Pi 3 data hub prototype

The LabVIEW G-code for the Raspberry Pi 3 data hub prototype consists of four sections:
Main VI, Bluetooth SubVI, TCP SubVI and Convert_Data SubVI.

Main VI

Bluetooth SubVI

SubVI for retrieving measurement data using "Bluetooth"

- - - - - - - - - - - - - - -

Created by S.Schulstock

Specific format for the measurement data

Data format request

```
         XX.XX
         XX.XX
        XXXX.XX
         XXX.XX
  XXXXXXXXXXXXXXXXXXXX
```

data out

Address

channel (0)

1

Bluetooth Open Connection   Bluetooth Write   Bluetooth Read   Bluetooth Close Connection

Establish a connection with a specific address and channel

Writes/Sends format to sensor over Bluetooth

Read/retrieve data using bluetooth

Close the connection after measurement data has been sent

TCP SubVI

SubVI for retrieving measurement data using "TCP"

Created by S.Schulstock

Specific format for the measurement data

data out

Data format request

xx.xx
xx.xx
xxxx.xx
xx.xx
xxxxxxxxxxxxxxxxxxxxx

TCP Write

TCP Read

TCP Close Connection

Close the connection after measurement data has been sent

Read/retrieve data over TCP

Writes/Sends format to sensor over TCP

TCP Open Connection

IP address

Port

Establish a connection with a specific address and port number

Convert_Data SubVI

SubVI for splitting/converting "data in" (string) to numbers and string

- - - - - - - - - - - - - - -

Created by S.Schulstock

Creates a substring (based on a offset)

Use system decimal point

String Subset

Tag

Fract/Exp String To Number

Temperature

Humidity

Pressure

Co2

Interprets string (based on a offset) to floating-point numbers

offset 30

offset 7

offset 14

offset 23

data in

Appendix H – Systematic procedure (user manual)

Systematic procedure (user manual) for developing an embedded system using Raspberry Pi 3 hardware.

Raspberry Pi embedded system with LabVIEW LINX

LabVIEW
MakerHub

LINX

Contents:

❖ Information

❖ Software installation

❖ Raspberry Pi setup

❖ LabVIEW Run-Time installation on Raspberry Pi

❖ Connecting to Raspberry Pi

❖ Uploading «Blink» example to Raspberry Pi

❖ Running «Blink» example

❖ Temp_test example using TMP36

❖ Uploading «Temp_test» example to Raspberry Pi

❖ Running «Temp_test» example

LINX

❖ Information

LINX

❖ Information

Important:

An OS (operating system) must be installed on the
Raspberry Pi before LabVIEW can be uploaded!

Recommended OS:

➤ Raspbian
(https://www.raspberrypi.org/downloads/raspbian/)

Alternative OS:

➤ NOOBS (New Out Of the Box Software)
(https://www.raspberrypi.org/downloads/noobs/)

LabVIEW
MakerHub

LINX

❖ Information

Software:

➤ LabVIEW 2014 SP1
(http://www.ni.com/download/labview-for-education-2014-sp1/5281/en/)

➤ LabVIEW 2015/2016 *
(http://www.ni.com/download/labview-development-system-2016/6046/en/)

➤ VI Package Manager (LINX)
(http://ftp.ni.com/evaluation/labview/lvtn/toolkits/jki/vipm-windows.exe)

* Can not be used for uploading VI's to Raspberry PI

LINX

❖ Information

Hardware:
➤ Raspberry Pi
➤ Ethernet cable
➤ Mini usb cable
➤ Router/Hub

Retailers:
➤ (http://no.rs-online.com/web/)
➤ (http://www.ebay.com/) (Recommended)

LabVIEW
MakerHub

LINX

❖ Software installation

LINX

Software installation

Installation order:
- LabVIEW 2014
- Real-Time-2014
- VISA
- Linx

LINX

❖ Software installation

1. Mount LV4E2014SP1-Win.iso
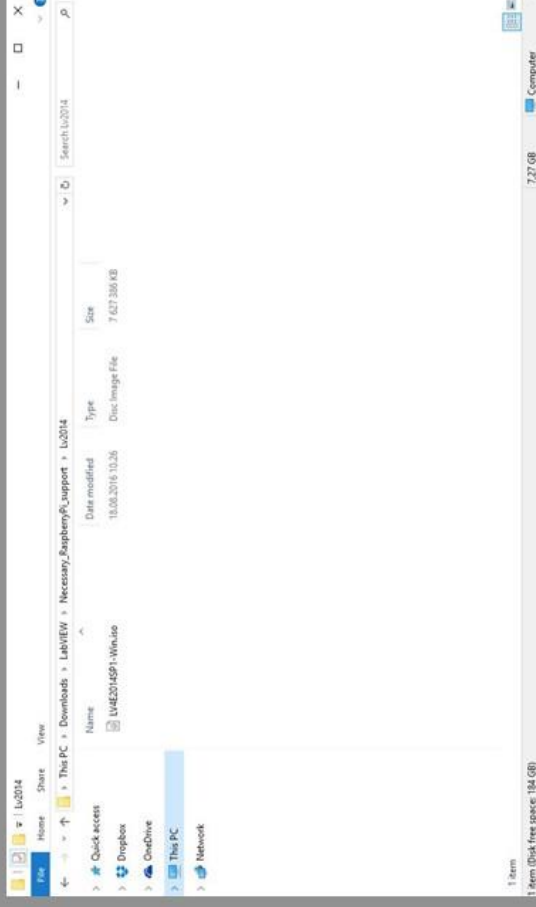
❖ Software installation

2. Navigate to LabVIEW 2014 folder
3. Run setup
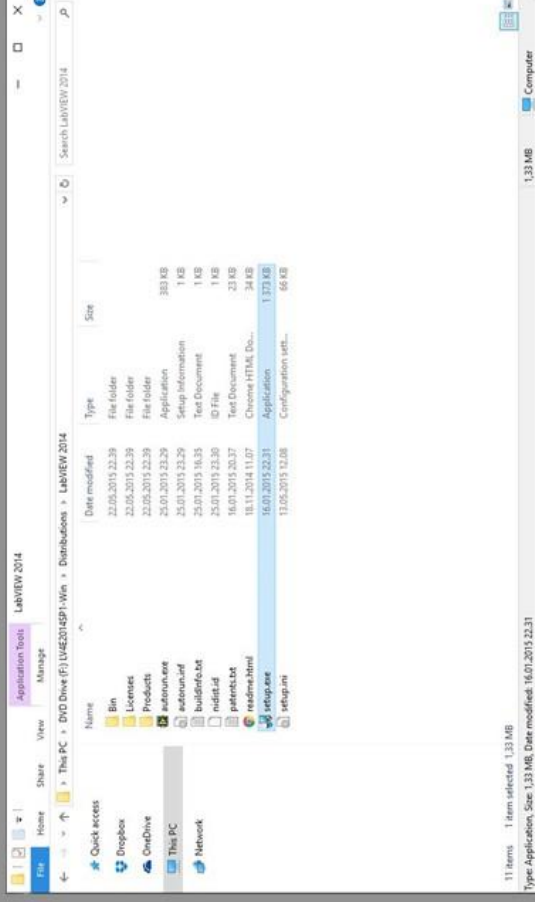
LINX

❖ Software installation

LINX

1. Navigate to LabVIEW 2016 folder *
2. Run setup

LabVIEW MakerHub

* Optional

❖ Software installation

1. Install VI Package Manager
2. Open VI Package Manager and search for «Linx»

LINX

LabVIEW
MakerHub

LINX

❖ Software installation

2. Choose to install «Digilent Linx» on LabVIEW 2014

3. Install then for LabVIEW 2016 *

LINX

❖ Raspberry Pi setup

LabVIEW
MakerHub
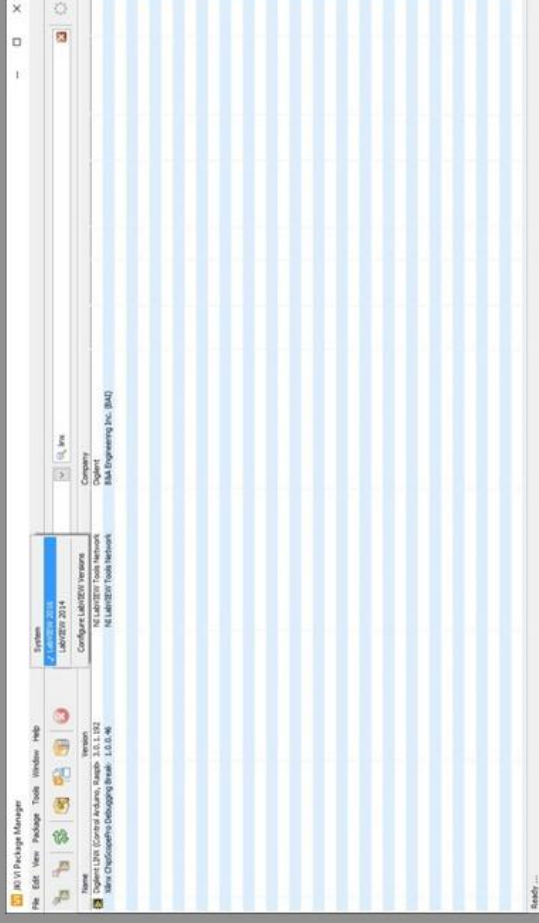
❖ Raspberry Pi setup

LINX

MakerHub (LabVIEW)

Alternativ 1: Using a router (recommended)

1. Connect the Raspberry Pi to a router using an Ethernet cable
2. Power up the Raspberry Pi using the mini-usb cable

Alternativ 2: Directly to PC

1. Follow the instructions on this page:
   (http://www.instructables.com/id/Direct-Network-Connection-between-Windows-PC-and-R/)

LabVIEW
MakerHub

LINX

❖ LabVIEW Run-Time installasion on
Raspberry Pi

LabVIEW
MakerHub

LINX

❖ LabVIEW Run-Time installasion on Raspberry Pi

1. Create a new «Blank Project» in LabVIEW 2014 (Not 2015/2016)

LabVIEW MakerHub
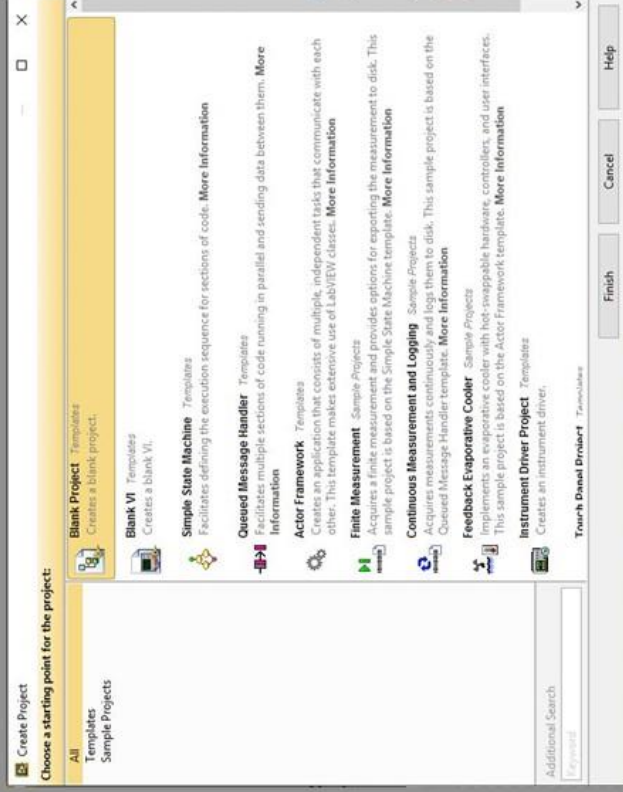
LINX

❖ LabVIEW Run-Time installasion on Raspberry Pi

2. Configurate Linx target (Tools -> MakerHub -> Linx -> Linx Target Configuration...)
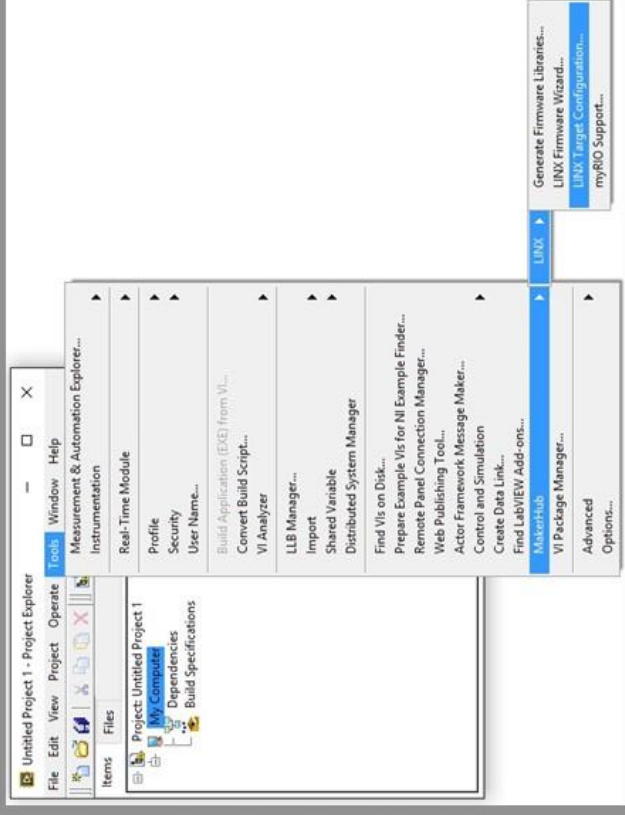
LabVIEW
MakerHub

LINX

❖ LabVIEW Run-Time installasion on Raspberry Pi

3. Connect to Target (Enter IP address, username and password)

Note;
If you dont know the IP address follow these instructions:
(https://www.raspberrypi.org/documentatio n/remote-access/ip-address.md)

LINX

❖ LabVIEW Run-Time installasion on
Raspberry Pi

4. Click «Install software» and «Install»
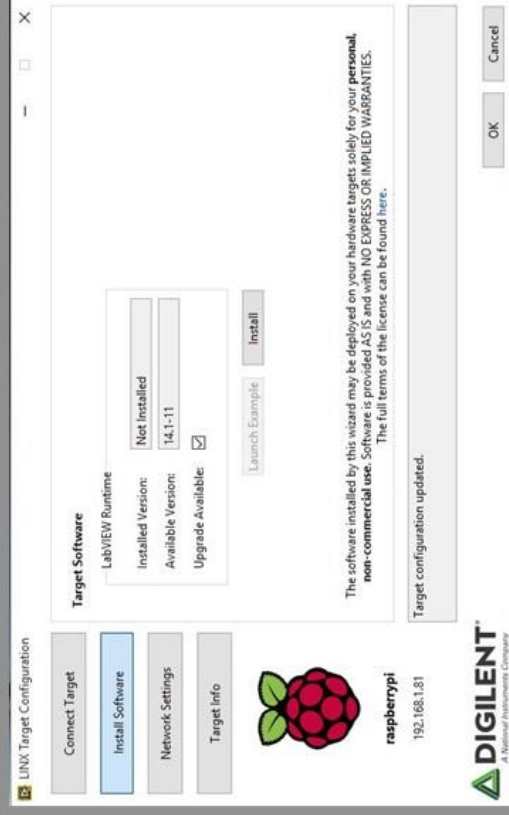
LabVIEW
MakerHub

LabVIEW
MakerHub

LINX

❖ Connecting to Raspberry Pi

LabVIEW
MakerHub
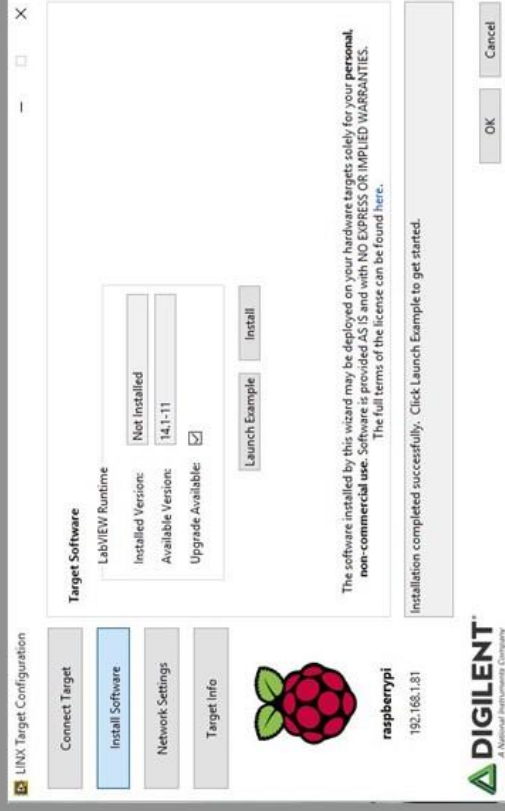
LINX

❖ Connecting to Raspberry Pi

1. Add a new «Target and Devices»
   (Right Click Project -> New -> Target and Devices)

LabVIEW MakerHub
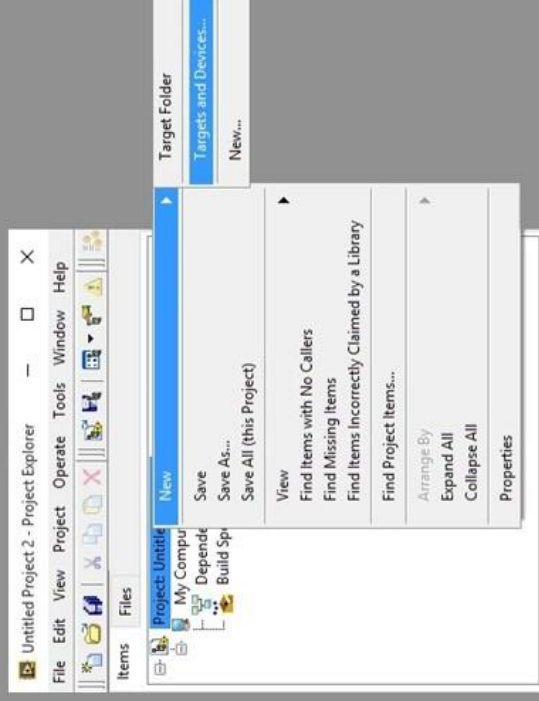
LINX

❖ Connecting to Raspberry Pi

2. Select «New target or device» and Linx «Raspberry Pi»

Add Targets and Devices on Untitled Project 2

Targets and Devices

○ Existing target or device
  ◉ Discover an existing target(s) or device(s)
  ○ Specify a target or device by IP address

◉ New target or device

Targets and Device Types
  ⊞ 🗁 Compact Vision System
  ⊞ 🗁 Embedded Vision System
  ⊟ 🗁 LINX
    📄 BeagleBone Black
    🔴 Raspberry Pi 2 B
  🖳 Real-Time Desktop
  🖳 Real-Time Industrial Controller
  🖳 Real-Time PXI
  ⊞ 🗁 Smart Camera
  ⊞ 🗁 Touch Panel

Refresh   OK   Cancel   Help

Note:
If LINX folder is not available(not showing), go to;
«Program Files (x86)\National Instruments\LabVIEW 2014\vi.lib\MakerHub\LINX\Resources» and extract «LinxProvider2014.zip» into «Program Files (x86)\National Instruments\LabVIEW 2014» folder.

Overwrite files if asked for

❖ Connecting to Raspberry Pi

3. Right click «Raspberry Pi 2B» and select «Properties»

LINX

LabVIEW
MakerHub

LINX
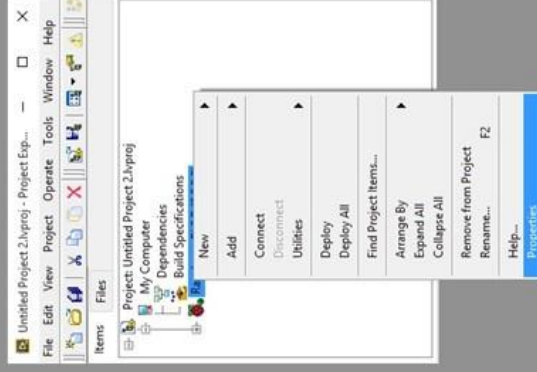
❖ Connecting to Raspberry Pi

4. Insert the «IP address» (and change «Name» if wanted)

Connecting to Raspberry Pi

LINX
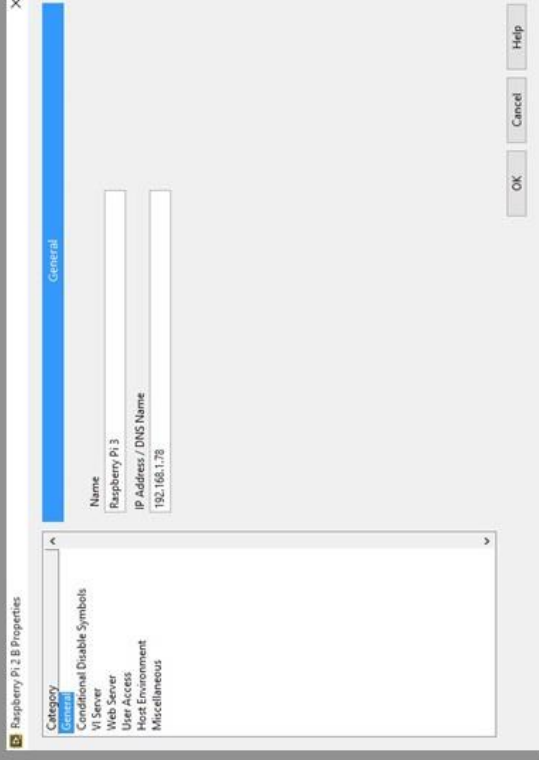
5. Right click «Raspberry Pi» and then «Connect»
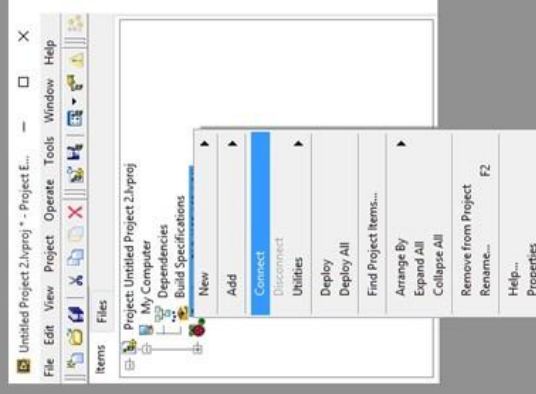
Connecting to Raspberry Pi

6. There should now be a green light(indicator) showing on the right side of the «Raspberry Pi» icon

LINX

LINX

❖ Uploading «Blink» example to Raspberry Pi

LabVIEW
MakerHub
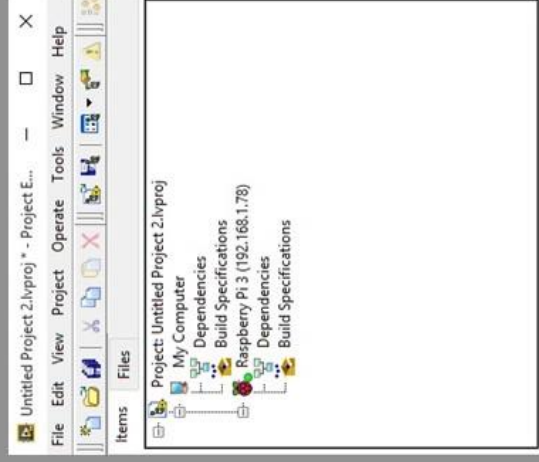
❖ Uploading «Blink» example to Raspberry Pi

1. Click on «Help» and «Find Example»

LabVIEW
MakerHub

LINX

❖ Uploading «Blink» example to Raspberry Pi

2. Search for «Linx» and select «Linx – Blink (Advanced).vi»

LINX

❖ Uploading «Blink» example to Raspberry Pi

3. In the block diagram; Remove «Serial port» (red cross) and change input from «Serial» to «Local I/O» (blue arrow)

LINX

MakerHub
LabVIEW
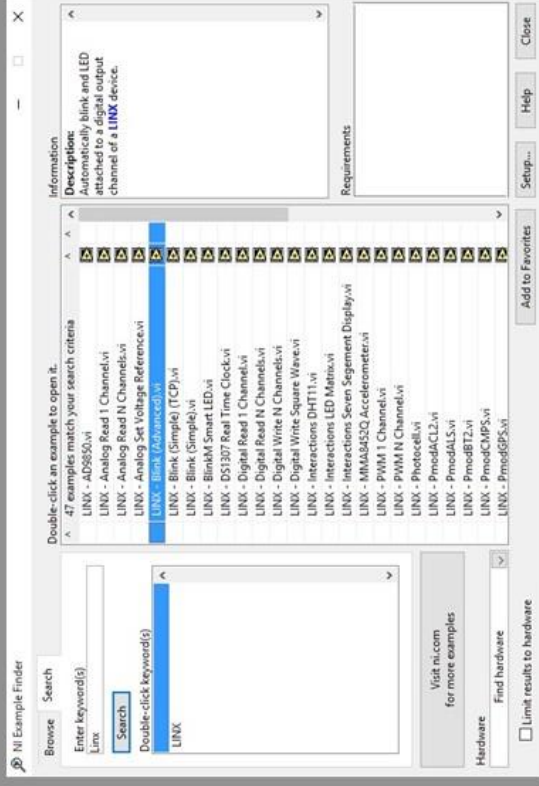
❖ Uploading «Blink» example to Raspberry Pi

4. In the front panel; Change input from «13» to «7»
5. Right click and select «Make current value default»
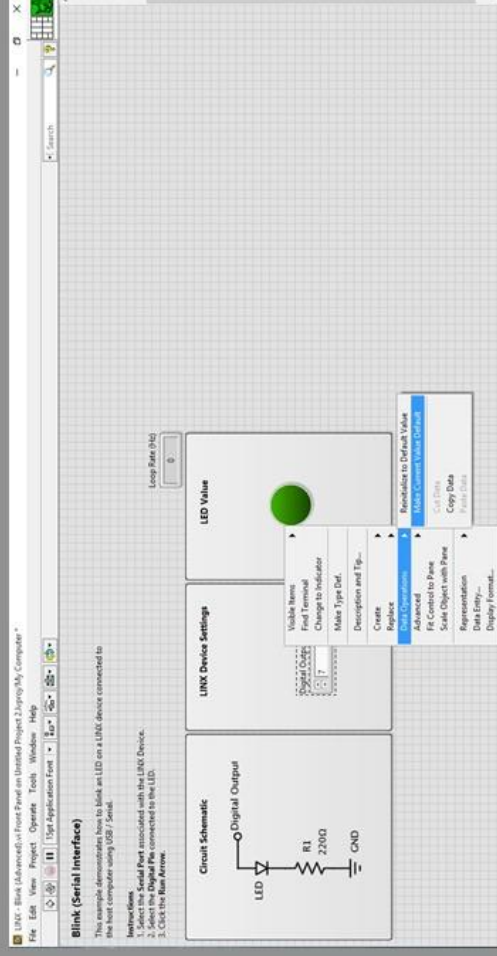
❖ Uploading «Blink» example to Raspberry Pi

6. Save the .VI and name it Blink_test
(File -> Save as -> Rename -> «Blink_test»)
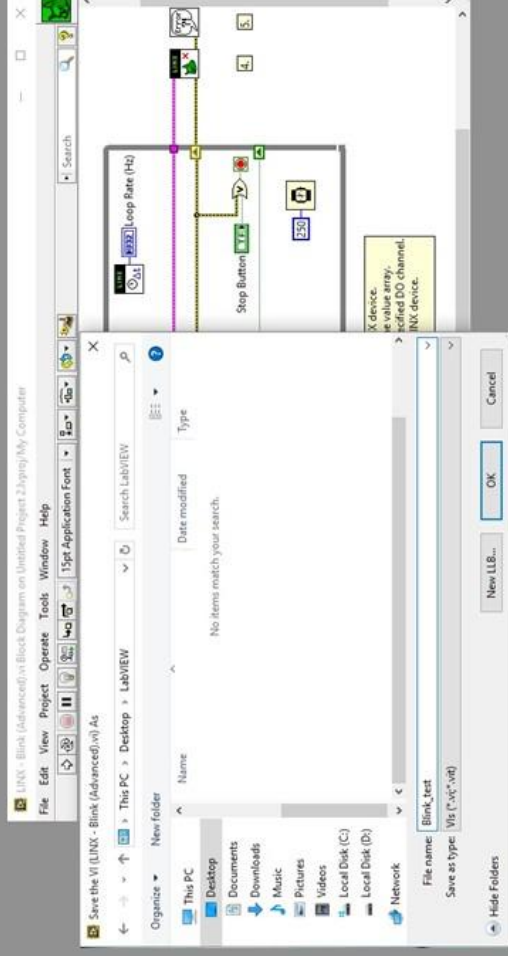
❖ Uploading «Blink» example to Raspberry Pi

7. Drag and drop the .VI into «Raspberry Pi 3»
8. Add new Real-Time Application
   (Right click «Build specifications» -> «New» -> «Real-Time Applications»)
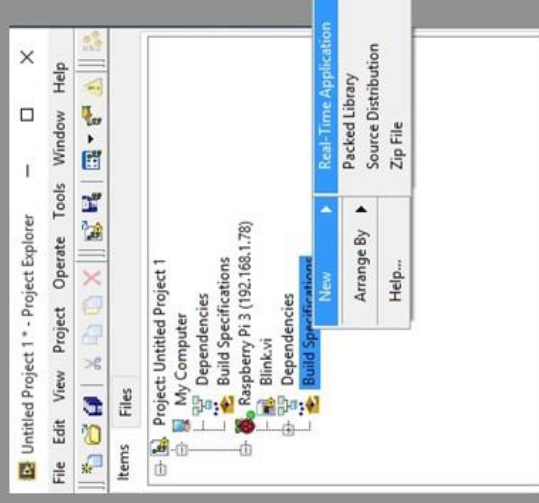
❖ Uploading «Blink» example to Raspberry Pi

9. Name the build specification «Blink»

LabVIEW
MakerHub
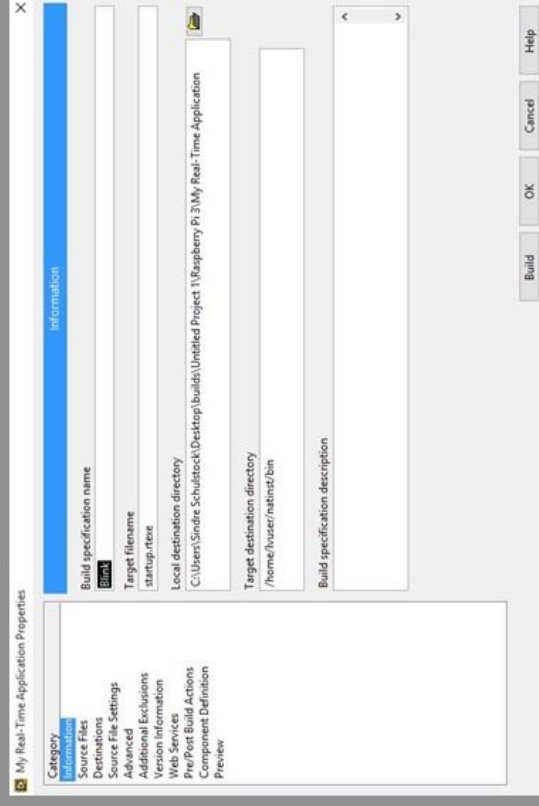
LINX

❖ Uploading «Blink» example to
Raspberry Pi

10. Set the .VI as «Startup Vi»
(Source files -> Select .VI and click blue arrow)

LabVIEW
MakerHub

LINX

❖ Uploading «Blink» example to
Raspberry Pi

11. Build the .VI
(Right click on .VI under «Build specifications» and click «Build»)
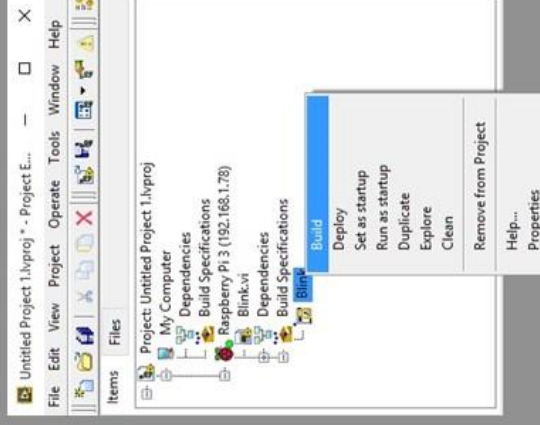
❖ Uploading «Blink» example to
Raspberry Pi

12. Set the built .VI as startup
(Right click on .VI under «Build specifications» and click «Run as startup»)

LINX

LabVIEW
MakerHub
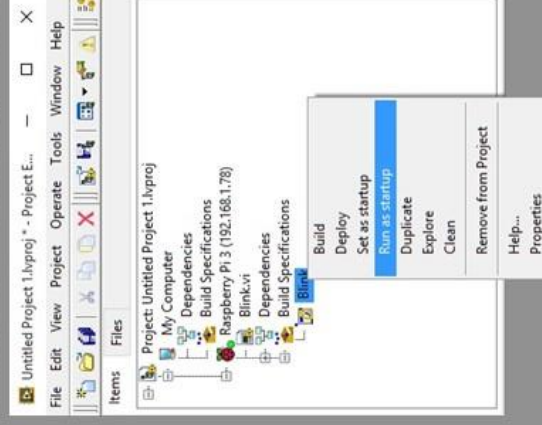
LINX

❖ Uploading «Blink» example to
Raspberry Pi

13. Reboot the target (Raspberry Pi)
(Click «Yes» when promoted to reboot the target)
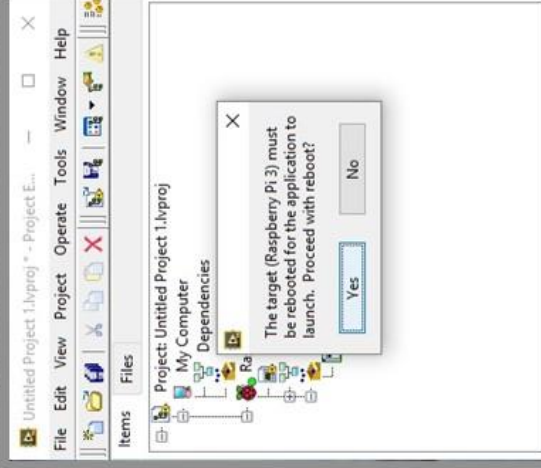
❖ Running the «Blink» example

LINX

LabVIEW
MakerHub

LINX

❖ Running the «blink» example

1. Remove the Ethernet cable
2. Connect/build as shown below
3. Power up the Raspberry Pi (mini usb)
4. Confirm that the LED is blinking!

LINX

MakerHub
LabVIEW
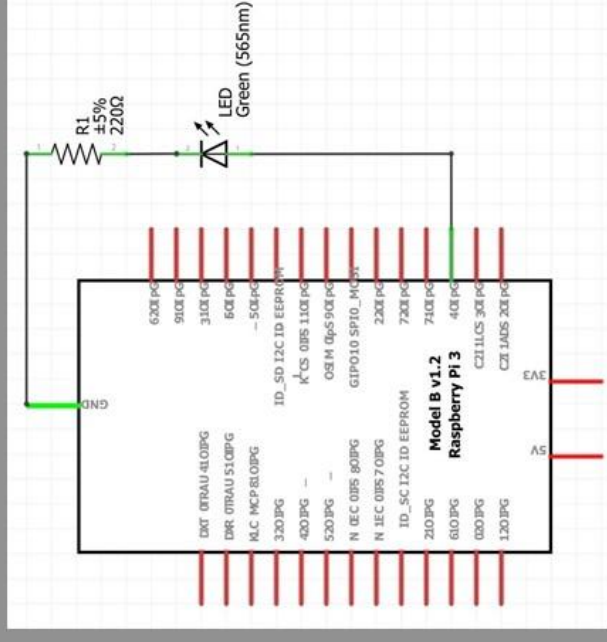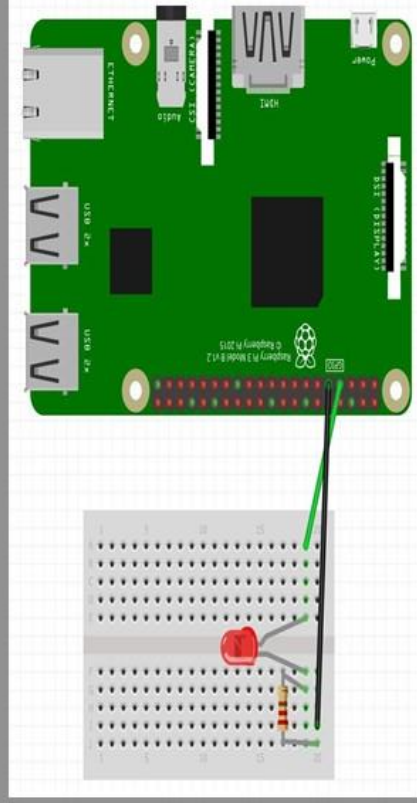
❖ «Temp_test» example using TMP36

LINX

❖ «Temp_test» example using TMP36

1. Build this .VI example

LINX

LabVIEW
MakerHub

❖ Uploading «Temp_test» example to Raspberry Pi

LabVIEW
MakerHub

LINX

❖ Uploading «Temp_test» example to Raspberry Pi

1. Drag and drop the Temp_test.VI(and its subVI's) into «Raspberry Pi 3»
2. Add new Real-Time Application
3. Set «Temp_test» as startup VI and the subVI's as <u>Always included</u>
4. Set the built .VI as startup and rebot the target (Raspberry Pi)
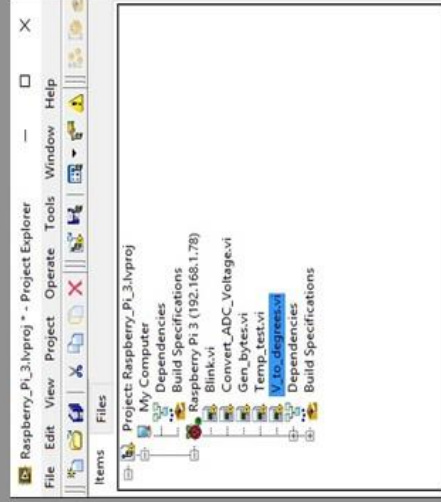
LINX

MakerHub

❖ Running the «Temp_test» example
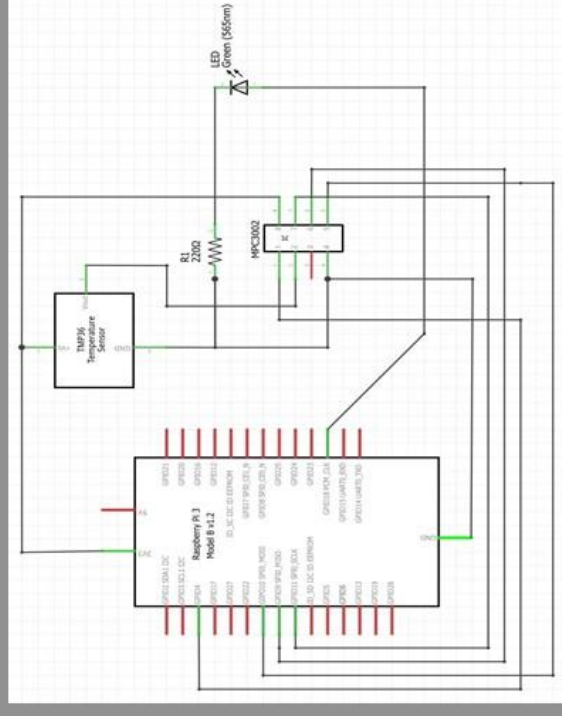
LabVIEW
MakerHub

LINX

❖ Running the «Temp_test» example

1. Remove the Ethernet cable
2. Connect/build as shown below
3. Power up the Raspberry Pi (mini usb)
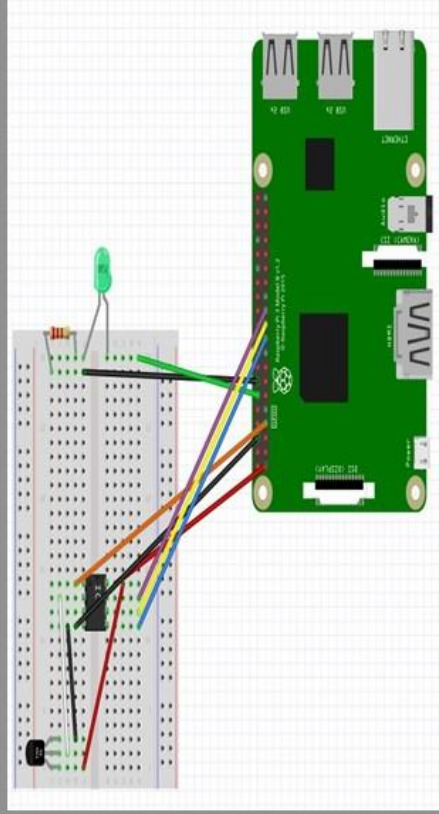
Datasheet for MCP 3002:
(http://www.ee.ic.ac.uk/pcheung/teaching/ee2_digital/MCP3002.pdf)

Datasheet for TMP36:
(http://(http://www.analog.com/media/en/technical-documentation/data-sheets/TMP35_36_37.pdf)

❖ Running the «Temp_test» example

4. Confirm that the temperature displayed in the front panel is correct (using a reference, for example a «multimeter»)
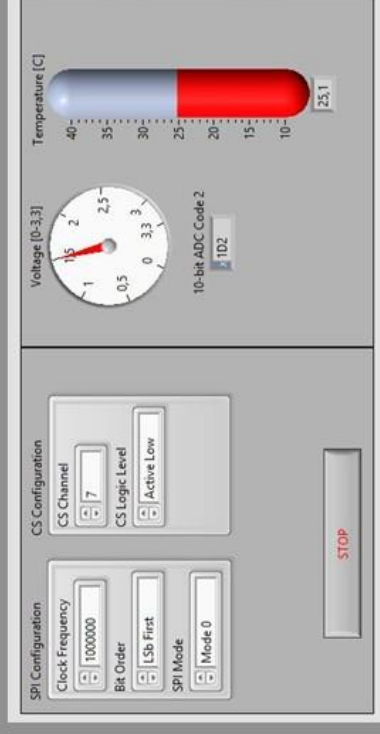
For more information:

LINX Tutorials:

➤ https://www.labviewmakerhub.com/doku.php?id=learn:tutorials:libraries:linx:start

Forums:

➤ https://www.labviewmakerhub.com/forums/viewforum.php?f=12

LINX