KTH | Kungliga Tekniska Högskolan

IL2213 Design Project

Final Report

Contiki OS on ATmega128RFA1

Group 5

Prodromos Mekikis mekikis@kth.se

Guodong Guo guodong@kth.se

Stefano Vignati vignati@kth.se

Saad Fakher fakher@kth.se

Ibrahim Kazi ibrahimk@kth.se

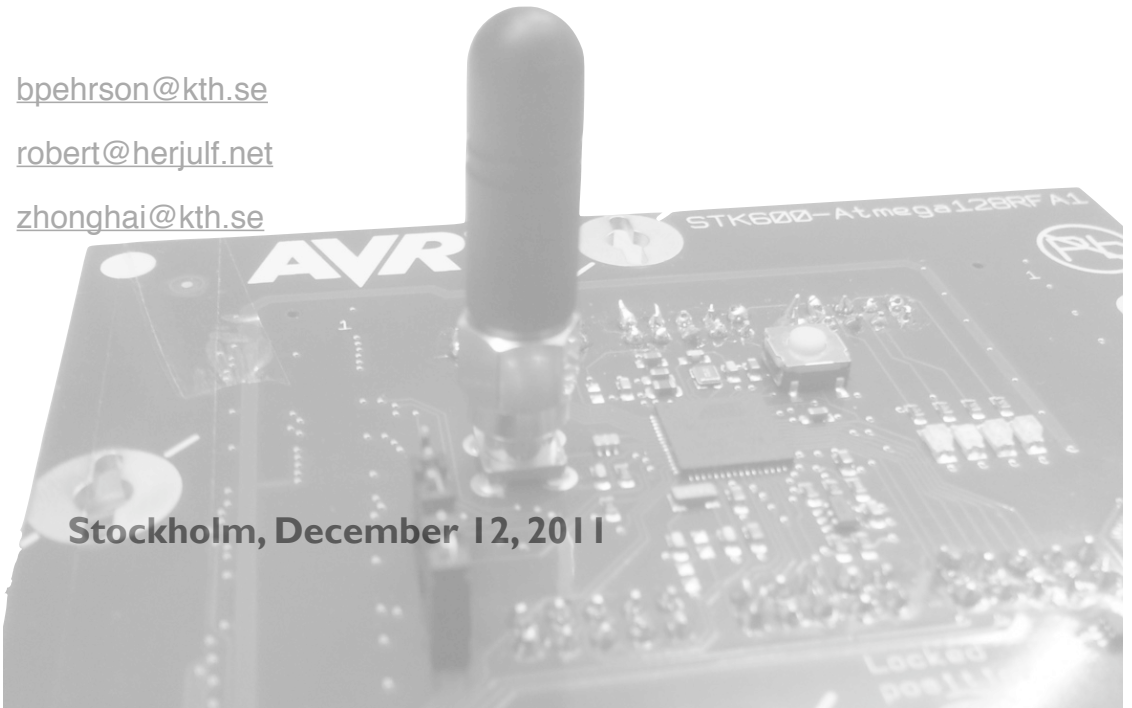
Mussie Tesfaye mtesfaye@kth.se

Advisors

Bjorn Pehrson bpehrson@kth.se

Robert Olsson robert@herjulf.net

Zhonghai Lu zhonghai@kth.se



Stockholm, December 12, 2011

Table of Contents

I. Technical Report

1. Introduction

1.1 Background

The Internet is mutating constantly since the last twenty-five years. It is progressively evolving from a network of interconnected computers to a network of interconnected objects and thus creating an 'Internet of things'. These objects will sometimes have their own Internet Protocol addresses, be embedded in complex systems and use sensors to obtain information from their environment and use actuators to interact with it [1].

The Internet of things is the revolution that will lead to connect to the web all the object that are surrounding us. The design of objects connected to the Internet is by now across the board in all sectors, and especially in the media. Estimates suggest that in 5 to 10 years there will be 100 billion devices connected to the Internet [2].

1.2 Motivations

The importance of this project is very clear in this context. The project is meant to design an experimental wireless sensor network that can monitor a spectrum of processes and environmental parameters. Thanks to the characteristic to be connected on Internet, all these parameters and processes can be accessed by everywhere in the world. This is really useful in several industrial fields such as tracking and monitoring the status of the goods inside a container shipped around the globe.

1.3 Related work

The aim of the project is to run an already developed OS on a micro-controller. The OS is the Contiki OS. This is a small, open source, highly portable multitasking computer operating system developed for use on a number of memory-constrained networked systems ranging from 8-bit computers to embedded systems on micro-controllers, including sensor network motes. Contiki is developed by a group of developers from industry and academia lead by Adam Dunkels from the Swedish Institute of Computer Science [3].

Moreover, the system will be powered by a ultra-capacitor cell, charged via solar cell, based on the work of the GreeNet project [4].

1.4 Technical Report Structure

The whole developed system consist in a network array of sensing motes that send data to a server mote, which will communicate with a gateway able to display all the information regarding the different motes.

The system has two types of motes. A server, which is connected with the gateway, and the clients which collect measurements and send them to the server mote. A client

mote is equipped with a sensors and power supply board. This board provides power to the client mote (+3.3V) and to the sensors conditioning circuit (+5V). It is self-powered and requires 4xAA batteries. The sensors sense the environment to get the value of pressure, humidity, light and voltage of the UPS which they monitor.

All these values are then processed in buffer stage in order to have an output compatible with the input of the analog to digital converter and also reject any noise than can damage it. The AVR-ATmega128RFA1 board also integrates an on chip temperature sensor which is also used in this project. The internal MUX is then selecting the signal source and the ADC is converting the analog value in its digital correspondents every seconds. After every conversion the client mote is sending UDP packets via 802.15.4 radio standard, using the IPv6 network layer protocol. Between one operation and the other the mote is going into sleeping mode to preserve power. The packets security is granted by a XOR protocol implemented in all the client motes and in the server. A unique key preserve the data to be decrypted.

The server mote is also equipped with a sensor and power supply board, but in addition it handles also a serial RS232 communication with the gateway. The sensing procedure works the same as described for the client mote. The servers listen to the data from the motes in the area around and at the same time collect the values from its ADC. The server is also decrypting the packets via a reverse XOR procedure. The successive operation is to forward all the measurements via serial messages to the gateway. A serial-to-USB converter is used for this goal.

Finally the gateway is used to receive, store and display the data. The gateway is opening a USB port to read the messages received by the server mote. A C program is then collecting the data, divide them according to the type and the source mote address and finally displaying them throughout an HTML page.

The technical report consists of 7 chapters. Chapter 1 is the introduction, which gives the background, motivation, related work of this project. The last part of it is the technical report structure, which is presenting here. In the following figure, the reader can see the system structure and the corresponding sections of each part of the structure in this report.

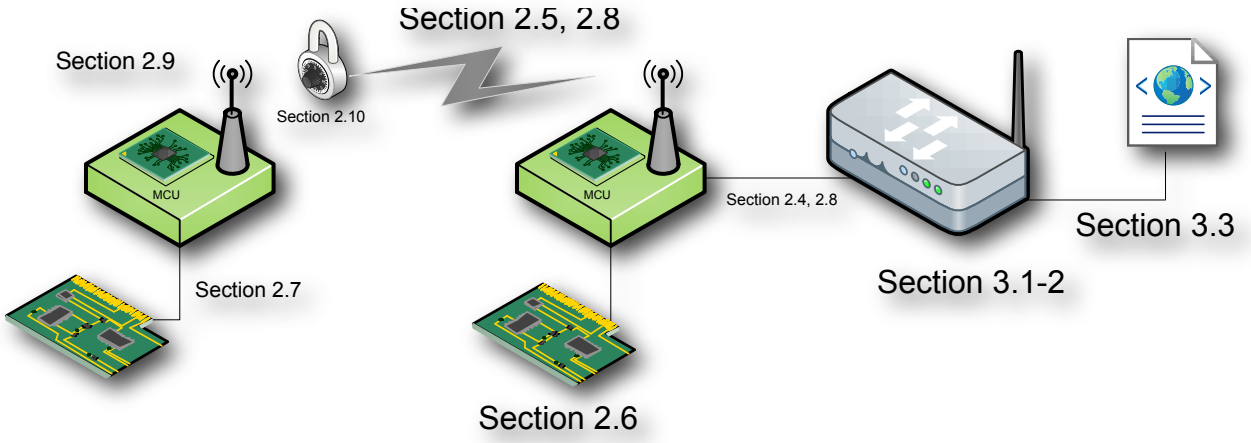


Figure 1.1: *System structure and the equivalent sections*

Chapter 2 deals with running Contiki OS on ATmega128RFA1 and implementing all the features that should be included in a sensor node. It contains 10 sections. The first

section introduces the hardware and software tools needed to setup the developing environment for sensor nodes.

Section 2.2 describes in detail the pins for hardware connections of ATmega128RFA1 board to the AVRISP mkII programmer and to the power Sparkfun ProtoBoard v1 Series, which is providing power and serial communication.

Section 2.3 is about running the blind led program on the ATmega128RFA1 board as a start.

Section 2.4 talks about the RS232 communication protocol which is used for the communication between the server board and the gateway, and for showing all the messages the MCU is printing to the terminal.

Section 2.5 presents setting the UDP-IPv6 communication between the server mote and the client mote.

Section 2.6 explains the sensors used in this project and the conditioning circuits for scaling the outputs of the sensors to fit the input range of the ADC.

Section 2.7 is about the ADC programming for sampling and converting of the sensors' outputs.

Section 2.8 introduces the data format for communication.

Section 2.9 presents the power reduction schemes.

Section 2.10 explains the data security added to the communication.

Chapter 3 addresses the construction of the gateway including the hardware, the operating system running on it and the RS232 serial communication between the server sensor node and the gateway. Besides, the HTML pages are designed to present the data sent by all the motes.

Chapter 4 presents a statistical analysis for the measurements by the ADC from all the sensors.

Chapter 5 is an noise analysis of all the components which will cause deviation of the output of ADC conversion from the actual value of sensors.

Chapter 6 addresses the performance and power consumption of the whole system and the specification of all the components.

Finally, **chapter 7** presents the critiques of the system and looks into the future work.

2. Contiki OS and the µController unit

Contiki OS [7] is a lightweight open source operating system developed by Adam Dunkels. It is used in embedded systems and wireless sensor networks and it is designed for micro-controllers like MSP430, AVR and ARM.

Its main features are:
• Multitasking kernel
• Protothreads
• TCP/IP networking including IPv6
• Communication: uIP and RIME
• Programming model
• Power profiling to measure consumption at network scale

Contiki is programmed in C and it is the state of the art among the different operating systems used for wireless sensor networking. In the following sections there is the description of the work that the project team did to prepare, port and operate Contiki OS on the ATmega128RFA1 MCU.

2.1 Tools

In this project, there are two type of tools the team used. Hardware and software tools. The hardware tools are:

- The AVR ATmega128RFA1 Evaluation Kit 1, including the two boards plus the antennas.
- The AVRISP mkII to program the AVR boards.
- A sparkfun prototype board series 1 for providing regulated power to the boards and serial communication interface, thanks to the FTDI chip which is on this board.
- Several electrical components used for connection and soldering.
- All the required sensors
  - Humidity Sensor
  - Temperature Sensor
  - Pressure Sensor
  - Light Sensor

The software tools that the team uses are:

- AVR studio for the initial tests
- AVRdude for ISP programming
- AVRgcc is for compiling for the AVR chipsets
- AVRlibc which is a set of libraries for programming AVR devices in C
- Instant Contiki 2.5 which is the complete Contiki OS environment in an Ubuntu Linux virtual machine
- Terminal program (Cutecom), to receive and send messages throughout the serial port.
- Coda, for the development of the Javascript code of the website
- G++ compiler, to compile the program that reads the data from the serial port

2.2 Preparing the hardware

To debug and test the system it was necessary to connect the AVR-ATmega128RFA1 board to the AVRISP mkII programmer and to the power Sparkfun Proto Board v1 Series, which is providing power and serial communication. The team decided for the moment to use these discrete boards instead of using an integrated solution, due to the early phase of the project.
The final result looks like this:

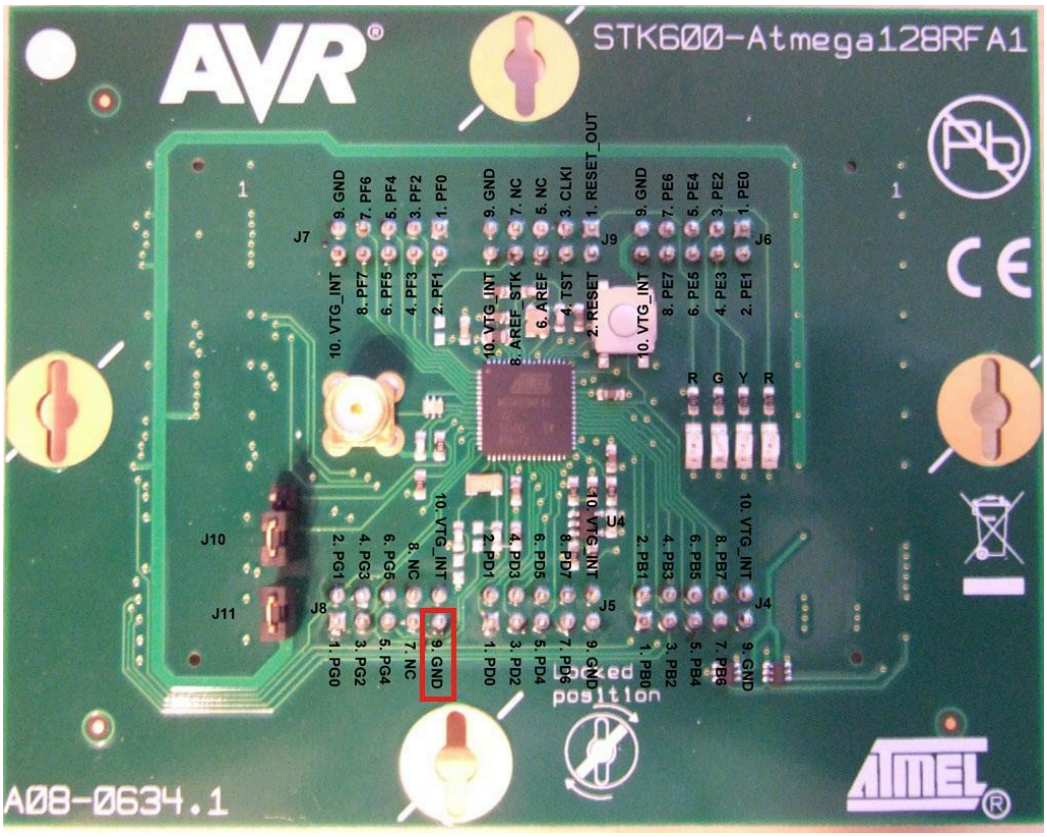


Figure 2.1: *The MCU with the connections*

For these reasons, it was necessary to solder pins and create connection between the boards. The AVR programmer's connector looks like this:

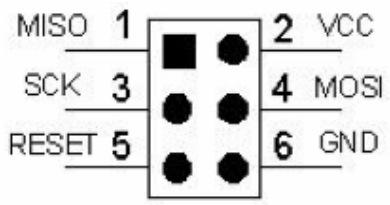


Figure 2.2: *AVR programmer's connector*

However the respective pins on the AVR-ATmega128RFA1 board, are not placed according to the same pattern, but they are spitted between to connector headers (the team does not understand the reason of this choice from AVR). Thanks to some wires, the problem was easily overcame, but it is really tricky to connect and disconnect all the

cables, when is necessary to program two boards. That is why the team chose to leave the wire connect to both boards and connect only the final 6-holes header to the AVRISP mkII programmer.

Finally the team soldered the pins for 3.3V and RS232 TX/RX on the sparkfun board and connected to the corresponding pins on the AVR-ATmega128RFA1 board: VTG_INT and GND for power and UASART1 TXD and RXD for the serial communication.

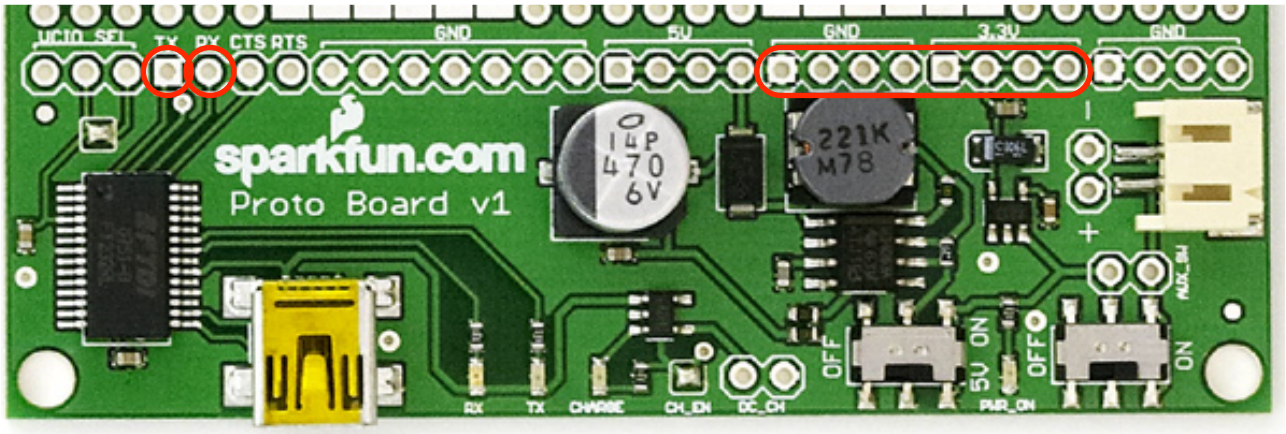


Figure 2.3: *The proto-board with the FTDI chip*

2.3 Starting with a Blink Led program

The easiest program to implement on a board to try if it is working or not is to try to light some leds. In the Contiki OS there is already defined a function, which is doing this automatically:

```
leds_blink();
```

The problem, with the function that the leds are not defined for the AVR-ATmega128RFA1 board used in this project. Consequently the team decided to implement this algorithm using the AVR commands.

First they wrote the code in AVR Studio 5, compiled and executed successfully. After they implemented the same code on Contiki OS main file to make it executing.

The code algorithm is very simple. First it initialize the port E, where the leds are connected by setting the direction to output:

```
DDRE = 0b11111111;
```

After, the on/off functionalities are written inside a while loop. The leds are turned ON and OFF by reversing biasing the output, so with HIGH level they are OFF and vice-versa. Finally they change their state every 100 ms.

```
while(1){
    PORTE = 0xFF;    // Set the LEDs OFF
    _delay_ms(100); // delay 100mS
    PORTE = 0x00;    // set the LEDs ON
    _delay_ms(100); // delay 100mS
}
```

2.4 Communication RS232

RS-232 is serial communication standard and is commonly uses in computer serial ports. The RS232 port of the computer is used to communicate with peripheral devices like mouse, keyboard etc.

There is a need to communicate between the Gateway and the Mote so the team set the RS232 port as the communication protocol. They have tested the RS232 by sending and receiving data from the MCU. Since modern computers don't have RS232 port, Serial to USB converter is used in between MCU and PC. This serial to USB converter FT232R is a plug and play device in Linux and need no external drivers. Cutecom software is used to monitor data on the PC.

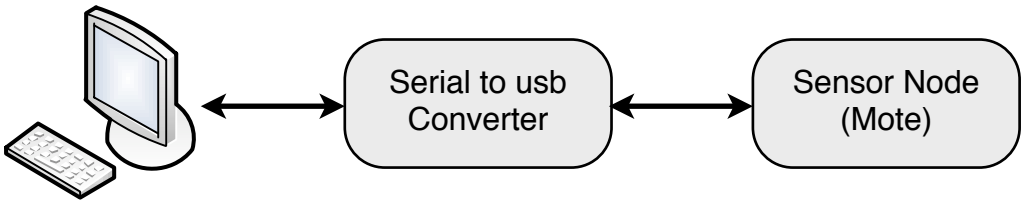


Figure 2.4: *Serial communication*

Sensor Node (mote): the UART is configured to receive and send the data. The receive interrupt is enabled so that when it receives data it immediately process them and do some other tasks, otherwise instead of polling the status bits of receive. This improves greatly the efficiency of the mote.

UART Configuration:

- Baud rate: 38400 bits per second
- Data bits: 8
- Parity: None
- Stop bits: 1
- Flow Control: None

The baudrate was selected 38400 bits per second in order to minimize the bit error rate (0.2%) for the operating frequency of the MCU (8 MHz).

UART Initialization:

```
 rs232_init(RS232_PORT_1, USART_BAUD_38400, USART_PARITY_NONE |
USART_STOP_BITS_1 | USART_DATA_BITS_8 | USART_RECEIVER_ENABLE |
USART_INTERRUPT_RX_COMPLETE);
```

UART Transmit:

```
rs232_print(RS232_PORT_1,ptr_string);  // send string of character
rs232_send(RS232_PORT_1, 'c');   // sent one character
```

UART Receive:

```
// UART is initialized and interrupt handler
rs232_set_input(RS232_PORT_1,uart_rx_handler);
// function address is passed
```

```
// this function is called when ever data is received by UART
int uart_rx_handler (unsigned char uart_rx232_in) { … }
```

Serial to USB Converter

It's a plug and play device in Linux but in Window one has to install the drivers and make the virtual serial port. It uses FT232R IC that is single IC that handles all the operation of conversion. It is CMOS and TTL compatible and is very power efficient.

Computer

On the computer side we are actually sending and receiving the data, we have tried communication with both Linux and Windows OS. In Linux we have used Cutecom software to send and receive the data and no drivers are required in this case. The Cutecom is GUI and configured as follows:

- Device: /dev/ ttyUSB
- Baud rate: 38400 bits per second
- Data bits: 8
- Parity: None
- Stop bits: 1
- Flow Control: None

Where as in windows, HyperTerminal can be used to send and receive the data. Drivers of FT232R need to be installed that are available on the IC's website after installation creates a virtual serial port and works in same way as ordinary serial port. Then the COM port no is noted from device manager and HyperTerminal is configured in same way as Cutecom.

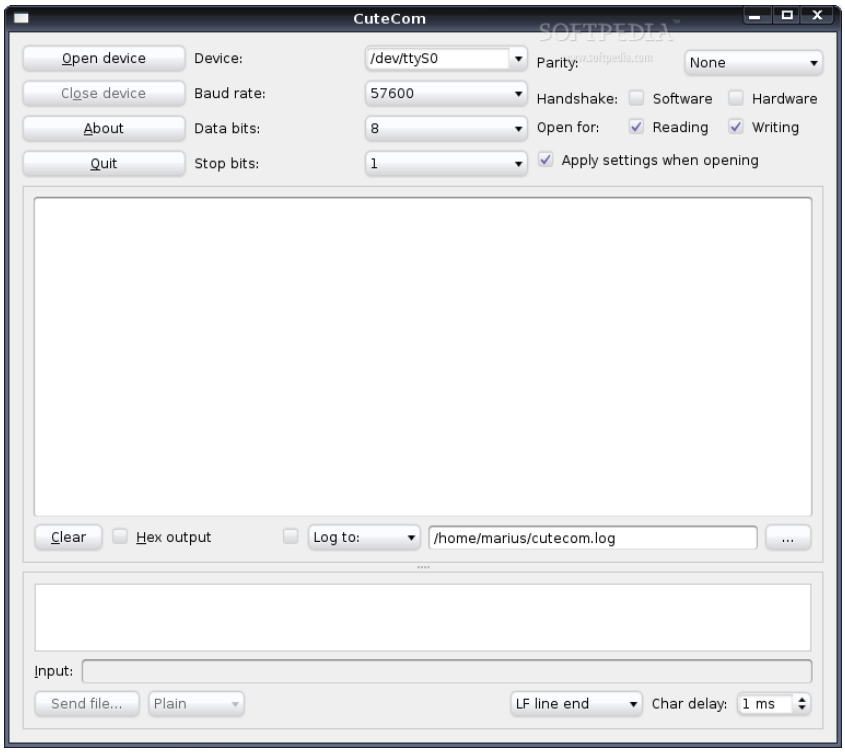


Figure 2.5: *The interface of the Cutecom program*

2.5 UDP-IPv6 communication

In this section we describe the procedure to test the example program udp-ipv6 in contiki on ATmega128RFA1 board, which sends UDP packet from client node to server node. The network layer protocol is IPv6. Contiki uses the MAC address of the ATmega128RFA1, which is defined in the EEPROM, to create and assign node IPv6 address.

The MAC address is defined in *platform/avr-atmega128rfa1/contiki-main.c* file

```
/* Put default MAC address in EEPROM */
uint8_t mac_address[8] EEMEM = {0x02, 0, 0, 0xff, 0xfe, 0,
0, 0x01};
```

The MAC needs to be changed when the program is loaded onto the client node, in order to give it a different IPv6 address for client-server communication. For example,

```
/* Put default MAC address in EEPROM */
uint8_t mac_address[8] EEMEM = {0x02,0,0,0xff,0xfe,0,0,0x11};
```

Server Program

To generate binaries for platform ATmega128RFA1, udp-ipv6 program is compiled as

```
make TARGET=avr-atmega128rfa1
```

This generates two binaries `udp-server.avr-atmega128rfa1` and `udp-client.avr-atmega128rfa1`. These are ELF files, which cannot be programmed by avrdude. ELF file contains the program to be written in flash and the MAC address to be written in EEPROM. They need to be extracted as two separate hex files.

This can be done by issuing the following command :

```
avr-objcopy -j .eeprom --set-section-
flags=.eeprom="alloc,load" --change-section-lma .eeprom=0 -
O ihex udp-server.avr-atmega128rfa1 server.eep

and

avr-objcopy -R .eeprom -R .fuse -R .signature -O ihex udp-
server.avr-atmega128rfa1 server.hex
```

This would generate server.hex file that has the program payload and server.eep file that has MAC address of the node to be written in EEPROM.

Now atmega128rfa1 should be burned through the following commands (in order specified here) :

```
sudo avrdude -p m128rfa1 -c avrispmkII -P usb -U flash:w:server.hex
sudo avrdude -p m128rfa1 -c avrispmkII -P usb -U eeprom:w:server.eep
```

This would set up the server code. From Hyper terminal/ Cutecom, which is connected to UART through USB to UART converter chip like FT232, the information of server booting up is showed in the following figure:

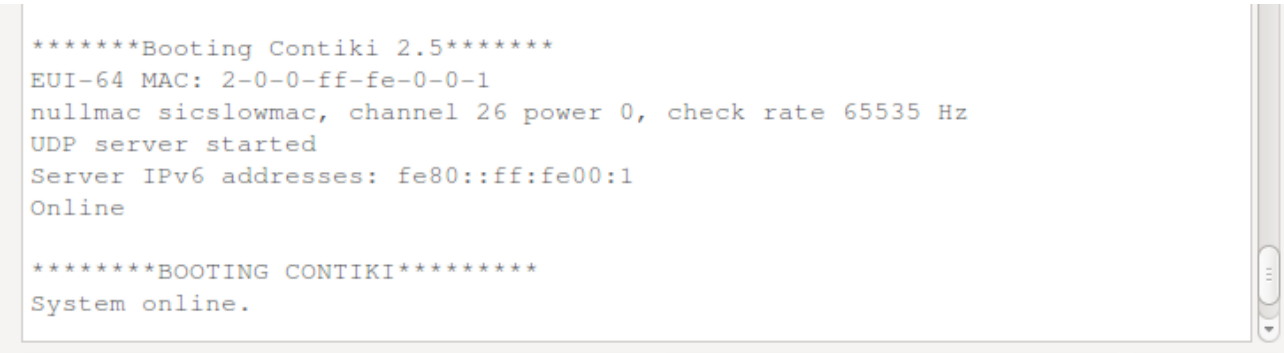


Figure 2.6: *Server booting*

Client Program

Before modifying client code, the MAC address defined in contiki-main.c needs to be changed, as discussed earlier. Because this MAC address is used to form IPv6 address, and the IPv6 addresses for the client and server should be different.

After changing the MAC address, the client program should be changed to include the address of the Server atmega128rfa1, which is printed on the Cutecom, so that the client can send message to the server.

The code for the set-connection_address function in udp-client should be changes as follows:

```
static void
set_connection_address(uip_ipaddr_t *ipaddr)
{
   uip_ip6addr(ipaddr,0xfe80,0,0,0,0xff,0xfe00,0x1);
}
```

The program should be programmed again to generate binaries with updated MAC address and server IP address. This can be done by issuing the similar commands as described in Server section, as well as burning code onto ATmega128RFA1 board.

When we have programmed client and server code with the above modifications, connect both boards to a computer and open two different Cutecoms (one for each ATmega128RFA1) to see their outputs.

We can find the client sending UDP messages to the server ATmega128rRFA1 periodically, the server also replies with an UDP message once it receives the message from the client.

```
*******Booting Contiki 2.5*******
EUI-64 MAC: 2-0-0-ff-fe-0-0-11
nullmac sicslowmac, channel 26 power 0, check rate 65535 Hz
UDP client process started
Client IPv6 addresses: fe80::ff:fe00:11
Created a connection with the server fe80::ff:fe00:1 local/remote port 3001/
3000
Online

********BOOTING CONTIKI*********
System online.
```

Figure 2.7: *Booting*

```
Client sending to: fe80::ff:fe00:1 (msg: Hello 1 from the client)
Response from the server: 'Hello from the server! (1)'
Client sending to: fe80::ff:fe00:1 (msg: Hello 2 from the client)
Response from the server: 'Hello from the server! (2)'
```

Figure 2.8: *Client sends and Server responds*

```
Server received: 'Hello 1 from the client' from fe80::ff:fe00:11
Responding with message: Hello from the server! (1)
Server received: 'Hello 2 from the client' from fe80::ff:fe00:11
Responding with message: Hello from the server! (2)
```

Figure 2.9: *Server receives and responds*

Procedure Simplification

The procedure above has been simplified by us using a script. First, put the following code in a file named 'elfx', and put this file in the udp-ipv6 folder.

```
# to make executable type: $ chmod +x elfx
# to run it: $ ./elfx filename
# by Stefano Vignati
make TARGET=avr-atmega128rfa1
avr-objcopy -R .eeprom -R .fuse -R .signature -O ihex $1.avr-
atmega128rfa1 application.hex
avr-objcopy -j .eeprom --set-section-flags=.eeprom="alloc,load" --
change-section-lma .eeprom=0 -O ihex $1.avr-atmega128rfa1
application.eep
sudo avrdude -p m128rfa1 -c avrispmkII -P usb -U
eeprom:w:application.eep -U flash:w:application.hex -F
```

Second, type the command: `$ chmod +x elfx`
Third, check the MAC address in the contiki-main.c file, which in this case should be

```
/* Put default MAC address in EEPROM */
uint8_t mac_address[8] EEMEM = {0x02,0,0,0xff,0xfe,0,0,0x01};
```

Then type the following command in the terminal, which will compile the files and

programme the server board: `$ ./elfx udp-server`

Last, change the MAC address in the contiki-main.c file, which in this case should be

```
/* Put default MAC address in EEPROM */
uint8_t mac_address[8] EEMEM = {0x02,0,0,0xff,0xfe,0,0,0x11};
```

Then type the following command, which will compile the files and programme the client board: `$ ./elfx udp-client`

Note that you still need to change the set-connection_address function in the client program for the first time.

2.6 Sensors

Five sensors are used at each node. They measure atmospheric pressure, temperature, battery voltage, humidity and light intensity. Sensor for pressure, light and humidity were ordered online along with the op-amps for the signal conditioning circuits.

For the temperature, there is a built in sensor on the ATmega128RFA1 board. The battery voltage doesn't require a sensor, just a conditioning circuit. All the sensors along with the circuits were soldered onto two separate veroboards for each node. A series of 1.5V AA batteries are used to power the sensor boards.

In the figure 2.10, the reader can see a picture of the sensor board. Of course it would be much better if the team could make a PCB instead of using a veroboard, but due to the limited time they didn't manage to do it. However, it is in the future improvements as one of the most important tasks.

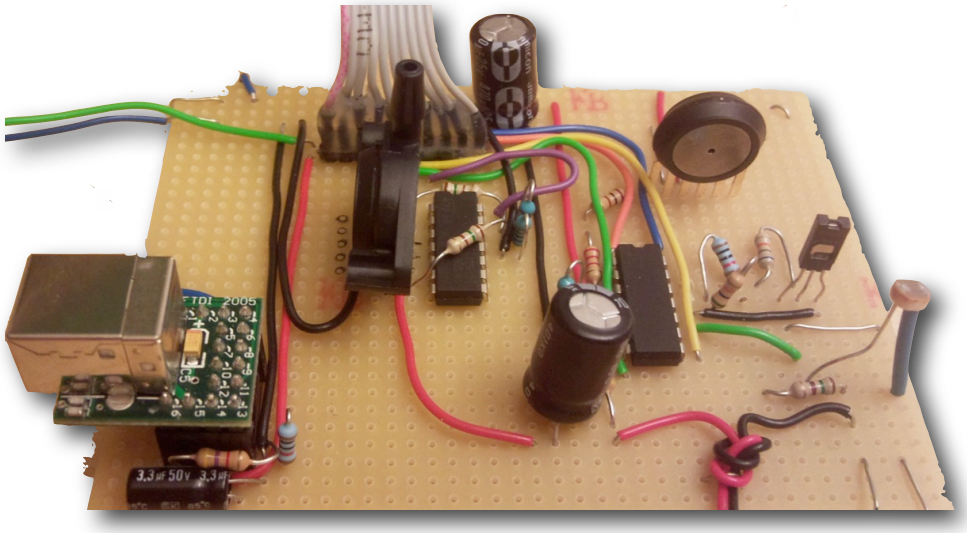


Figure 2.10: *The sensors board*

Pressure Sensor

The pressure sensor used, MPX2200, is an absolute pressure sensor to measure the atmospheric pressure. The MPX2200 series devices are silicon piezo-resistive pressure sensor providing a highly accurate and linear voltage output directly proportional

to the applied pressure. The sensor is a single monolithic silicon diaphragm with the strain gauge and a thin-film resistor network integrated on-chip. There can be an offset of 1mV which corresponds to approximately 5 kPa in pressure. This can be accounted inside the controller. The sensor along with its transfer curve is shown in the figure below.

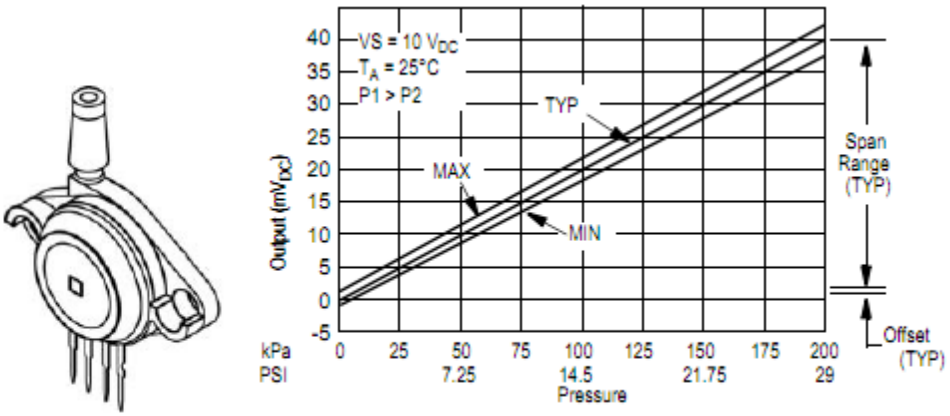


Figure 2.11: *Graphical representation of the MPX2200 sensor and the output characteristic graph*

The first stage is a voltage follower to avoid the loading of the sensor. All of the op-amps are provided with a supply voltage of +12V. The second stage gives a voltage gain of around 40. This gain can be adjusted by setting the ratio of the resistors (gain = R3/R1). Where R4= R3 and R2 = R1. The output of this circuit can be used with the ADC. The sensor is also provided with a supply voltage of +12V. The circuit is shown below.

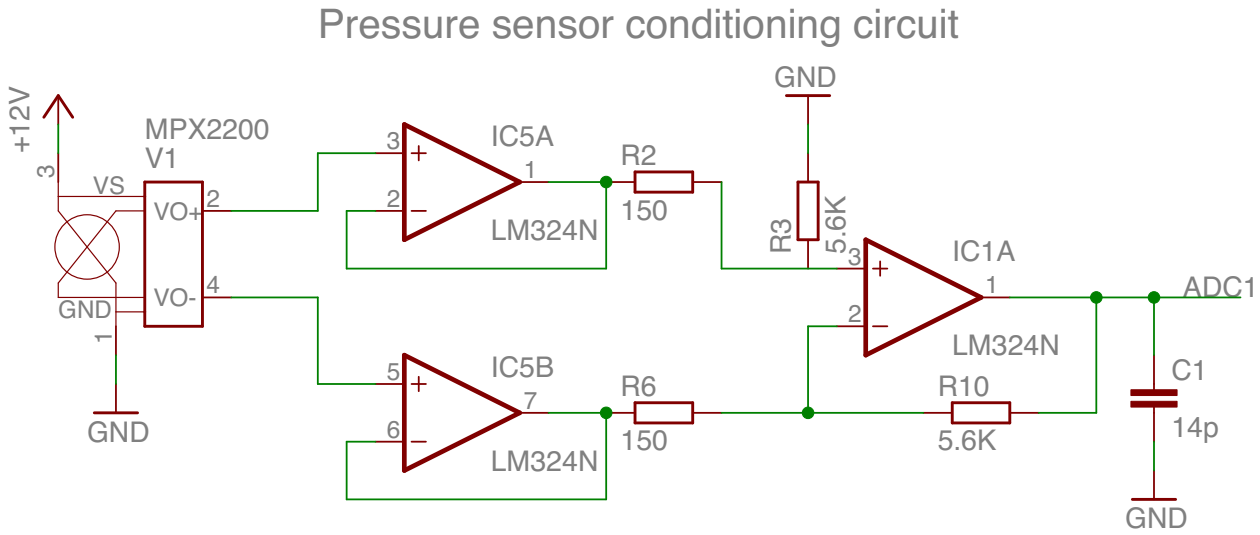


Figure 2.12: *MPX2200 conditioning circuit*

This circuit converts the 0 to +40mV voltage to 0 to +1.6V thus using the full scale of the ADC. The ADC input relates to the pressure sensor output as:

$$V_{ADCin} = \frac{R_{10}}{R_6}\left(V^{+}_{pressure} - V^{-}_{pressure}\right), \text{ also } R_{10} = R_3 \text{ and } R_6 = R_2$$

Voltage Measurement

The sensor board has also been equipped with a voltage divider circuit to measure the voltage of a source which varies from +12V to 0V. But the voltage divider values have been calculated by taking +18V as the maximum input voltage, to be on the safe side. The voltage divider circuit is shown in the figure below.

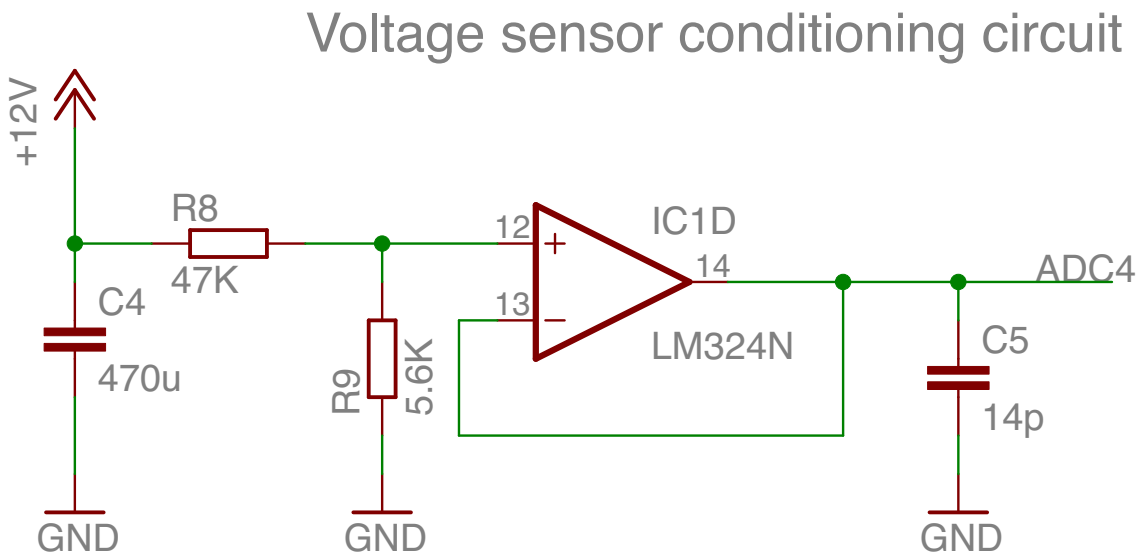


Figure 2.13: *Voltage conditioning circuit*

The voltage at the non-inverting terminal of the op-amp is:

$$V^{+} = \frac{R_9}{R_9 + R_8} V_{+12V}$$

Humidity Sensor

The humidity sensor used is Honeywell's HIH-4000-001. The linear output enables direct connection with the controller but the maximum range of ADC is up to +1.6V so the sensor o/p needs to be scaled before connecting to ADC. The scaling is achieved by a resistive divider and then a buffer to detach the circuit from the controller. The I/O graph of the sensor is shown below which shows a linear relationship. The supply voltage for the sensor is +5V.

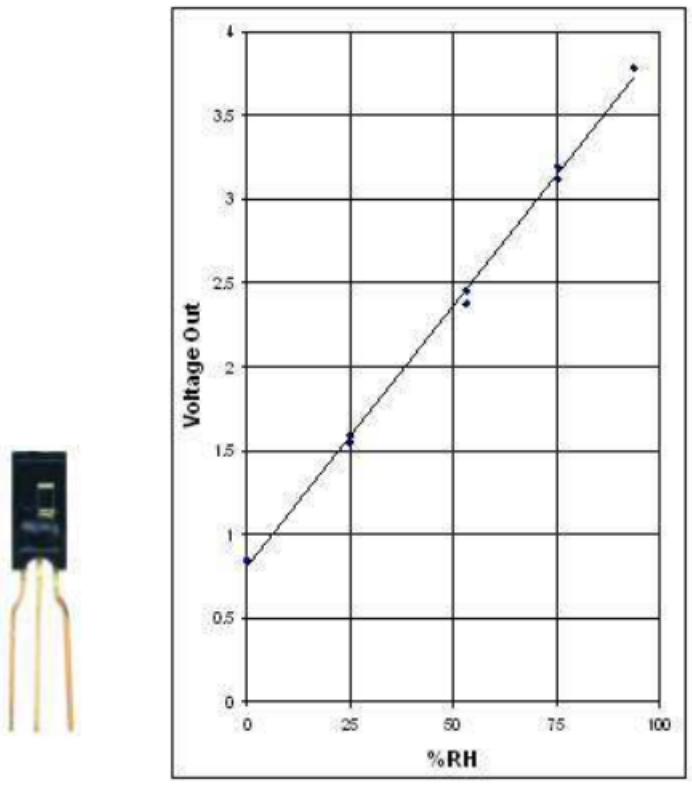


Figure 2.14: *Picture of of the HIH-4000-001 sensor and the output characteristic graph*

The maximum output voltage at 98% humidity is 3.8V which is scaled to the full scale voltage of the ADC (+1.6V). The circuit below is basically a voltage divider followed by a buffer. The buffer is needed so that the input impedance of the micro-controller pin doesn't load the voltage divider. The output of the circuit is grounded using a capacitor to filter AC noise. The scaling factor is approximately 0.41 which will correlate the maximum output of the sensor to the full range input of the ADC.

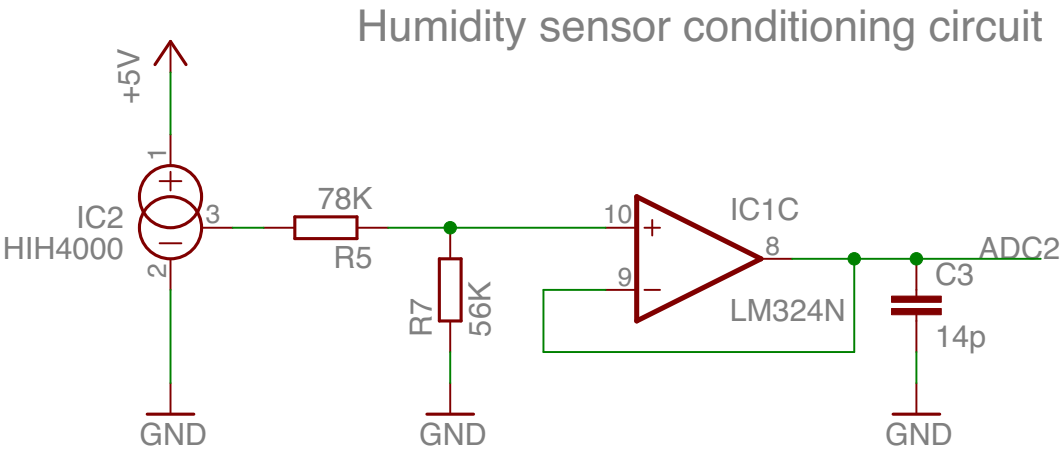


Figure 2.15: *HIH-4000-001 conditioning circuit*

The relation for sensor output and ADC input is: $V_{ADCin} = V_{outsensor}\left(\frac{R_7}{R_5 + R_6 + R_7}\right)$

Light Sensor

The light sensor used is a simple Light Dependent Resistor (LDR) giving a resistance of 2.5 Kohm for the brightest light and 500 Kohm at dark. A simple voltage divider scales the maximum voltage at 2.5 Kohm to +1.6 V.
The circuit used is as follows:

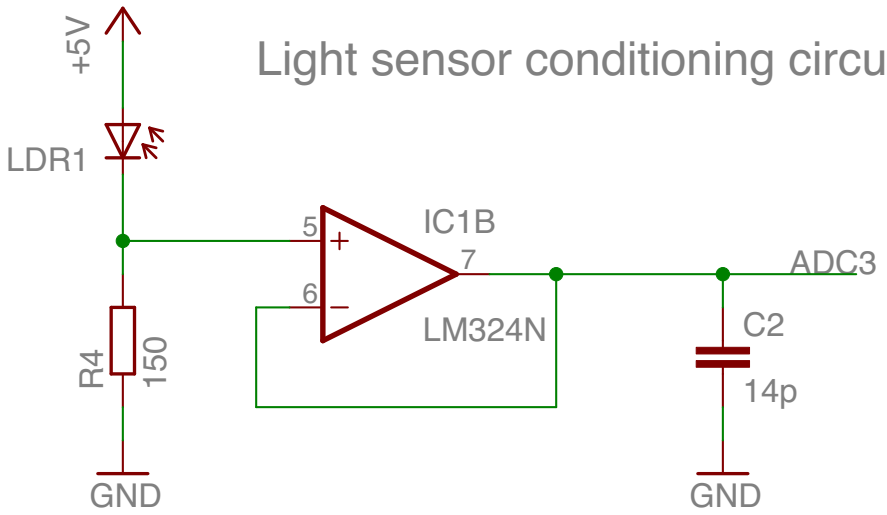


Figure 2.16: *LDR conditioning circuit*

The LDR is use in a voltage divider configuration and the voltage across the 150 ohm resistor gives the reading about light intensity. The buffer is used to avoid any loading, and hence the consequent changing of the divider value.

The transfer equation is :

$$V_{ADCin} = +5V \cdot \left(\frac{R_4}{R_{LDR} + R_4}\right)$$

Temperature Sensor

The on-chip temperature can be measured using a special setup of the A/D converter inputs. The integrated temperature sensor provides a linear, medium-accurate voltage proportional to the absolute temperature (in Kelvin). This voltage is first amplified with the programmable gain amplifier and then processed with the A/D converter. The temperature resolution is 1K.

The ideal result can be calculated when using the internal +1.6V reference voltage according to the following equation:

$$ADC_{TEMP} = 241.4 + 0.885 \cdot \theta / °C$$

The temperature sensor is connected to a differential input channel with a gain of 10.

Power Supply

To operate the sensor boards and the controller as standalone, a combination of +1.5V AA batteries and voltage regulators is used to supply 12V, 5V and +3.3V for the system. +12V are for the pressure sensor and the op-amps, +5V is for the light sensor, and 3.3V for the microcontroller board.

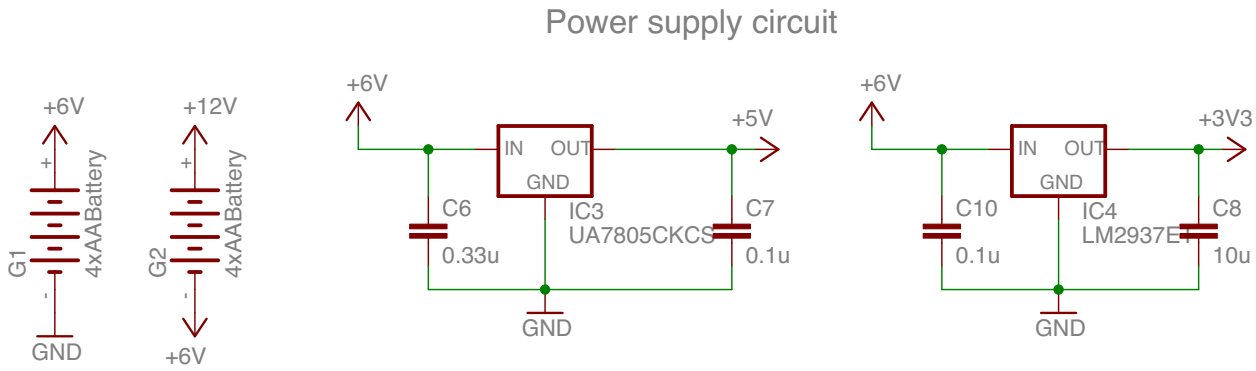


Figure 2.17: *Power supply circuit*

A series combination of 8 +1.5V AA batteries is used to generate +12V and a series combination of 4 +1.5V AA batteries is employed to generate +6V. The +12V generated can be directly used with the op-amps and the pressure sensor, whereas the +6V is supplied to the voltage regulator (UA7805CKCS) to generate +5V and to the voltage regulator (LM2937ET) to generate +3.3V.

2.7 Analog to Digital Converter

To use the Analog to Digital Converter (ADC), adc.h and adc.c files were built to implement the ADC operation on the ATmega128RFA1 board. The adc.h and adc.c files are adapted from the same files for another avr platform “avr-ravenlcd”, which is existing in contiki 2.5 folder. Since the pins name and register bits the ADC of different avr chips are unified by <avr/io.h>, only minor changes need to be made in order to run the code on the ATmega128RFA1. For example, the Power Reduction Register is PPR0 for ATmega128RFA1 instead of PPR.

Three functions are defined for ADC operation. They are listed below:

- `int adc_init(adc_chan_t chan, adc_mux5 mux5, adc_trig_t trig, adc_ref_t ref, adc_ps_t prescale);`

- `void adc_deinit(void);`

- `static int16_t doAdc(adc_chan_t chan, adc_mux5 mux5, int8_t avg);`

In *adc_init* function, first the ADC enable bit PRADC in the Power Reduction Register needs to be set to zero. Then the ADC clock frequency should be set by set the prescaler bits. In this project ADC_PS_128 is always chosen. Because the clock frequency of ATmega128RFA1 is 8 MHz, so the ADC clock frequency is 8MHz / 128 = 62.5 kHz. The reference voltage 1.6 V is chosen. The ADC enable bit ADEN is set to one. The ADC interrupt enable bit ADIE is set to one as well here to enable ADC noise canceller. Another important thing is to choose input channel for the ADC, which is determined by the register bit MUX5 in the ADCSRB register and MUX 4:0 in the ADMUX register. The channels for different sensors are shown in Table 2.1. Note that the MUX5 should be set before MUX4 to MUX0. The *adc_init* should be called every time the CPU wakes up and a new round of ADC conversion should start.

Sensors	MUX 5:0
Internal temperature	101001
Humidity	000110
Light	000100
Pressure	000111
Battery voltage	000101

Table 2.1: *The input channels for different sensors*

In this project, every time after the ADC finishes reading and converting all the sensors' output on the board once and the data are sent as UDP package, the micro-controller will be put in power saving sleep mode. Before entering power reduction operation, the adc_denit function should be always called first, which is used to disable the ADEN and the PRADC.

The doAdc function is used to get ADC conversion result for a specific sensor. The ADC conversion is operated in sleep mode, as the ADC features a noise canceller that enables conversion during sleep mode to reduce noise induced from the CPU core and other I/O peripherals. The noise canceller is used with ADC Noise Reduction mode SLEEP_MODE_ADC here. Every time a new ADC conversion starts, the CPU goes to sleep. When the ADC conversion finishes, the ADC Conversion Complete Interrupt will wake up the CPU. The CPU will jump to the corresponding ISR - EMPTY_INTERRUPT (ADC_vect), which is empty here, and then goes back to the program after the sleep_cup(). To make the data from a sensor more accurate, an average of the results of

multiple ADC conversions are enabled. Figure 2.18 shows the process of the ADC loop operation.

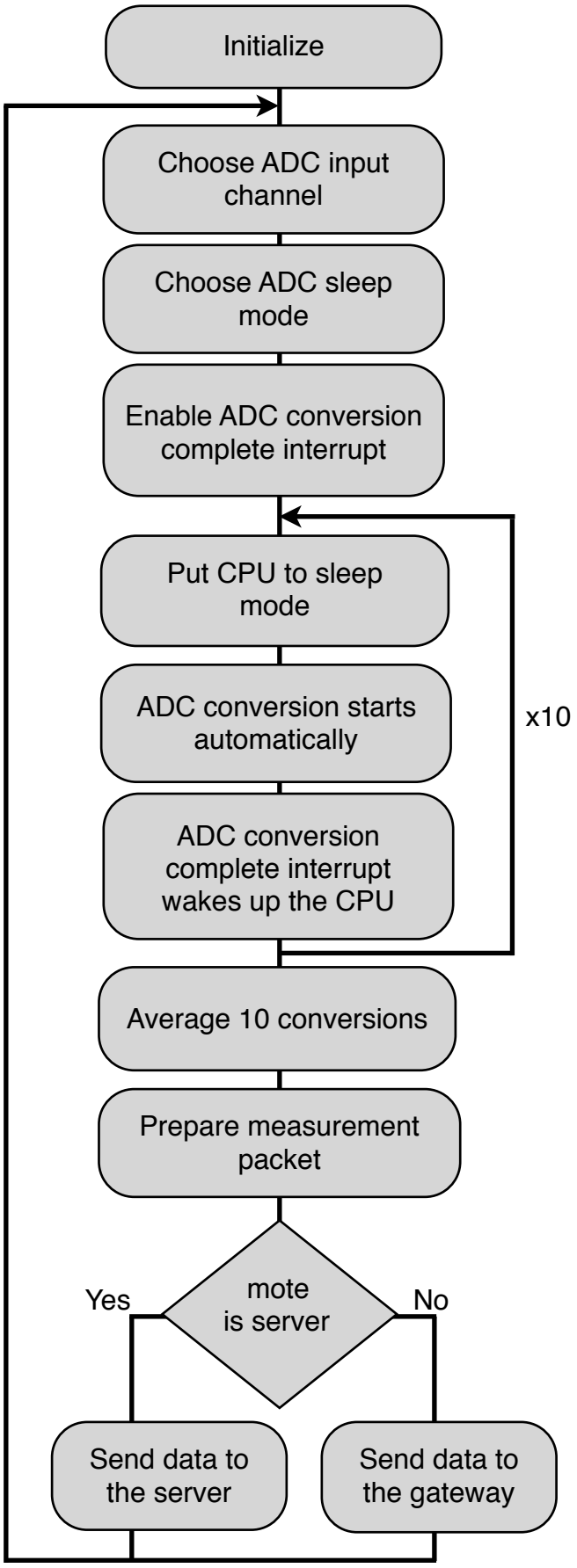


Figure 2.18: *The ADC loop operation*

Formulas for extracting the sensor data from the A/D conversion result

Apart from the formula for the internal temperature sensor, which is from the data sheet of atmega128rfa1, all the other formulas are deduced and estimated from the characteristic curves shown in the sensors' data sheet. As a result, they are not very accurate.

- Internal temperature sensor

$$\theta = \left(1.13 \cdot ADC - 272.8\right)\ ^{o}C$$

- Humidity sensor

$$H = \left(0.119 \cdot ADC - 25.532\right)\%$$

- Light sensor

$$L = \left(\frac{10^{5.44}}{\left(\frac{480000}{ADC}\right) - 150} \right) lx$$

- Pressure sensor

$$P = \left(0.1953 \cdot ADC\right) kPa$$

- Battery voltage sensor

$$V = \left(0.01465 \cdot ADC\right) V$$

2.8 Data format convention

Data format convention for the client mote to the server

After connecting all the sensors to the ADC ports on the ATmega128RFA1 board, the analog output of the sensors will be converted to digital number between 0 and 1023. Then the client need to send the data to the server by radio, while the server need to send the data from the client and itself to the gateway by the UART.

The client sends the data according to the following rules: Once there is a data from the ADC, the client will send it. The format is three continuous 16 bits integers.

The first integer is the identifier for the board. For example, “1” stands for the server, and “2” stands for the client. More identifiers could be assigned when more sensor nodes are added in the network.

The second integer is the identifier for the specific sensor on the board. In this project, “0” stands for the internal temperature sensor in the ATmega128RFA1 chip, “1” is the humidity sensor, “2” is the light sensor, “3” is the pressure sensor, and “4” is the battery voltage sensor.

The third integer is the data calibrated from the output of the ADC for different sensors, which is the meaningful measurement for the sensors. For example, the data for the internal temperature can be “25” in room temperature, which stands for “25 °C”. The

humidity reading can be “31”, which means the humidity percentage is 31%. The reading “219” for the light sensor means the light intensity is 219 lx. The reading “98” for the pressure sensor means the atmospheric pressure is 98 kPa. The reading “12.3” for the battery voltage sensor means the voltage is 12.3 V.

Every time the data for a specific sensor on a specific board is stored in an array int16_t buf[3]. Then this array is send by the UDP package.

Data format convention for the server node to the gateway

Every time the server gets a result from its ADC or a UDP package from the client, it will print the three integers out using the same format. The gateway will then get this message from the server.
The command for printing the data from the server itself is:

```
printf("{%01d%01d%04d}", buf[0],buf[1],buf[2]);
```

The command for printing the data from the client is:

```
int16_t * pt = (int16_t*)uip_appdata;
printf("client{%01d%01d%04d}", *pt,*(pt+1),*(pt+2) );
```

For example, {120261} denotes that the data from the No.2 sensor (Light sensor in this project) of the server board is 0261.

2.9 Power Management

The ATmega128RFA1 supports six software selectable power management mechanisms. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or hardware reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC, to minimize switching noise during ADC conversions.

In Standby mode, the RC oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption. In Extended Standby mode, both the main RC oscillator and the asynchronous timer continue to run. In Deep Sleep mode all major digital blocks with no data retention requirements are disconnected from main supply providing a very small leakage current. Watchdog timer, MAC symbol counter and 32.768kHz oscillator can be configured to continue to run [6].

A typical example of the power consumption of the radio is shown below where the transmitter is running at maximum power and the CPU running at 16MHZ.

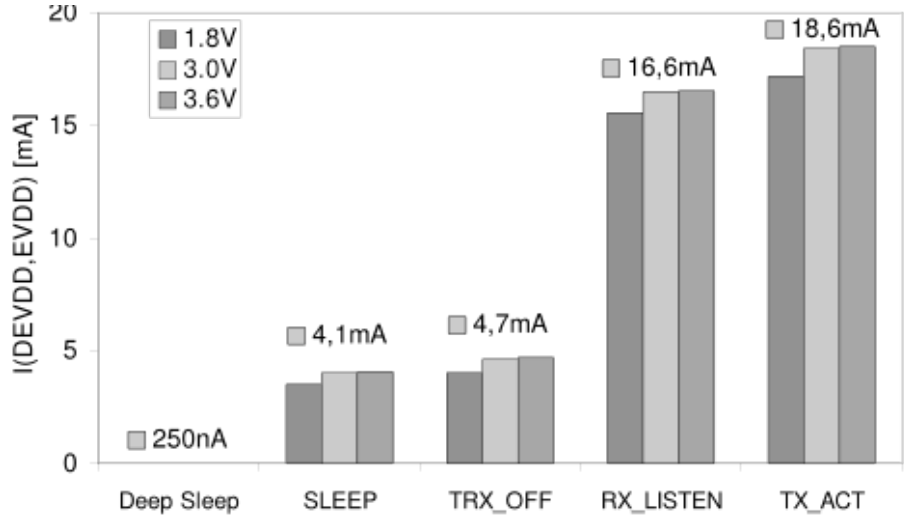


Figure 2.19: *Radio Transceiver state [6]*

The power management implemented in this project is the sleep mode, which is enough to save power and easy to implement with the limited time. The sleep is implemented in the sleep_now(int time) function which this function uses timer 2 and its interrupt to sleep the CPU for duration passed. Timer 2 is a PWM and is mostly used to perform PWM control. Timer 2 has a PWM counter, which is an eight-bit register. The PWM generator is configured to generate a pulse with a duty ratio of 50% and these pulses are counted. When the PWM counter register overflow there will be an over flow interrupt and this interrupt is used to wake the CPU.

In the sleep_now() function the PWM was configured to produce a normal pulse with a 50% duty and the frequency of the pulse was adjusted using this formula .

$$f_{pwm} = \frac{f_{clk}}{(prescale \cdot 256)}$$

Where f_{pwm} is the frequency of the pulses and f_{clk} is the frequency of the CPU. In our case we used f_{clk} 8MHZ and a pre-scale of 1024 giving us a f_{pwm} equal to 31.5HZ. The frequency is multiplied by 30 to make the interrupt every 1 second. Further reading can be found on [6] and the code is also well commented.

The power management is not implemented in the Contiki OS but in avr/sleep.h we find useful functions like the sleep mode select and CPU sleep.

The pseudo code of the *sleep now* function is:

```
Sleep_now(){

     Enable Timer 2 overflow interrupt;
     Set Timer 2 operation mode;
     Set Timer 2 frequency;
     Select sleep mode;
     While (not interrupted by the Timer 2 overflow)
     CPU sleep;// actual command cpu_sleep();
}
```

2.10 Security

Data security is the means of ensuring that data access is suitably controlled. Thus data security helps to ensure privacy. Data security is part of Information security. There are many methods to secure the data from external influence. One way is to hide the data by encrypting them. Popular Encrypting algorithms today that are considered to be very secure are AES, DES, RSA, TDES and many other. Some of them apply on block of data and some on stream of data.

In the project the team used simple block encryption that hides the sensors' data over the wireless interface. It takes a block of data and XOR it with a key.

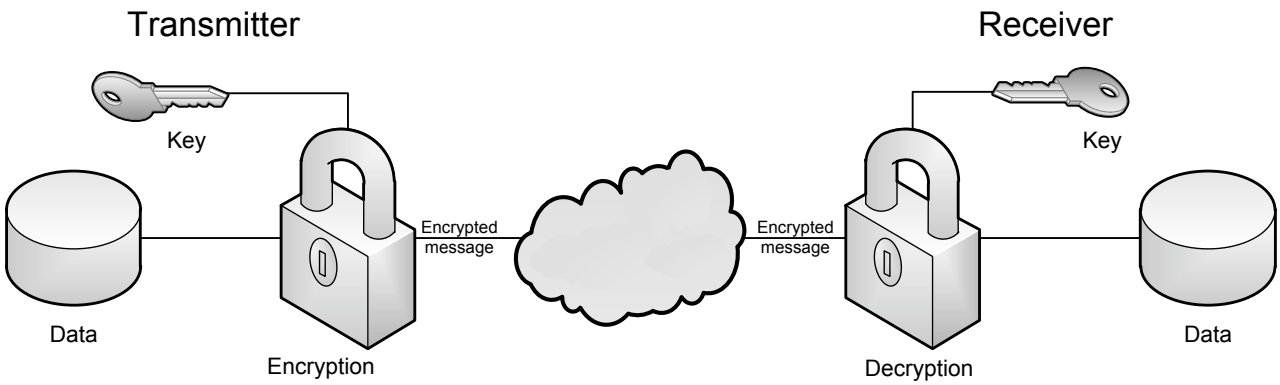


Figure 2.20: *Security mechanism*

On the other side, the receiver XOR the message with the same key in order to decrypt the message. It looks very simple but it is secure and may take days to break the encryption. Also, it needs less power to encrypt and decrypt the data.

The motivation of choosing such an encryption is:

- Because the sensors data don't required high security
- In order to save the power as the number of operations on data is directly propositional to power. Which means more operations in MCU more power will be used. Note the wireless node is battery operated so it must be power efficient in order to operate for long time.

Even though the team implemented the above algorithm, ATmega128RFA1 has AES hardware accelerator which is a very secure encryption. The motivation is the same as the hardware accelerator takes much less power and time compared to software so the team did a literature study on this algorithm and tried to implement this algorithm on the project. The results at the time that this report was being written are very positive, since the MCU encrypts the data, but there are some issues that could be fixed in the future with the decryption.

Advance Encryption Standard (AES)

AES is a cryptography technique, which is also known as Rijndael, and is a block cipher algorithm used as an encryption standard by the U.S. Government. AES has a fixed block sizes of 128 bits and three values of key size 128, 192, or 256 bits.

To implement this algorithm in software is time and power consuming, because of the complexity, but the hardware accelerator of AES in ATmega128RFA1 can be used for

encryption of the data. So far the encryption of data is working and the team is still working on the decryption. They have used the ECB mode for encrypting the data. Some details of these standard is given below. There are four registers in ATmega128RFA1

Configuration Registers:

- **AES_STATUS:** This register is used to give the status of the encryption/decryption if the operations are done or not or if there is any error or not during the operations.
- **AES_CTRL:** This register is used to set the mode either CBC or ECB and to start the operations
- **AES_KEY:** This register is used to write the 16 byte key. And also used to read the key to decrypt the data.
- **AES_STATE:** It used to write the 16 byte of data to encrypt or read 16 bytes of decrypt data.

Steps to do Encryption and Decryption:

Key Setup: Write encryption or decryption key
1. AES configuration:
- Select AES mode ECB or CBC
- Select encryption or decryption
- Enable the AES Encryption Ready Interrupt

2. Write Data: Write plain text or cipher text to DATA buffer

3. Start operation: Start AES operation
4. Wait for AES finished: Wait until AES encryption/decryption is finished successfully. Any one of the given operations can be done.
- AES_READY IRQ or
- polling AES_DONE bit
- wait for 24 μs

5. Read Data: Read cipher text or plain text from DATA buffer

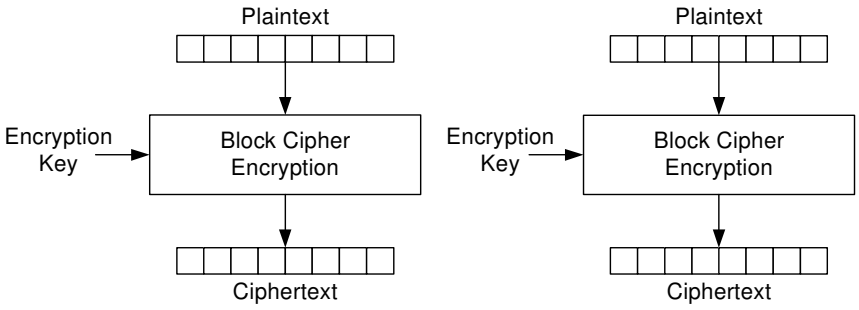


Figure 2.21: *ECB Mode - Encryption*

3. The Gateway

3.1 Alix2d13

Alix2d13 is the gateway that the team uses for receiving, storaging and displaying the measurements of the motes. It is a system board that runs a linux distribution (Bifrost) which is explained in the next section. Alix2d13 has two USB ports, one of which we use for the serial communication between the final mote and the gateway.

Furthermore, it has 3 Ethernet ports for the connection to the internet and a WiFi module for connection with other devices. Unfortunately, it is not possible to use the WiFi module for the connection with the motes, because the MCU do not support the WiFi protocol but only the ISO 802.15.4. In the following figure we can see the Alix2d13 board that we use for the purposes of the project

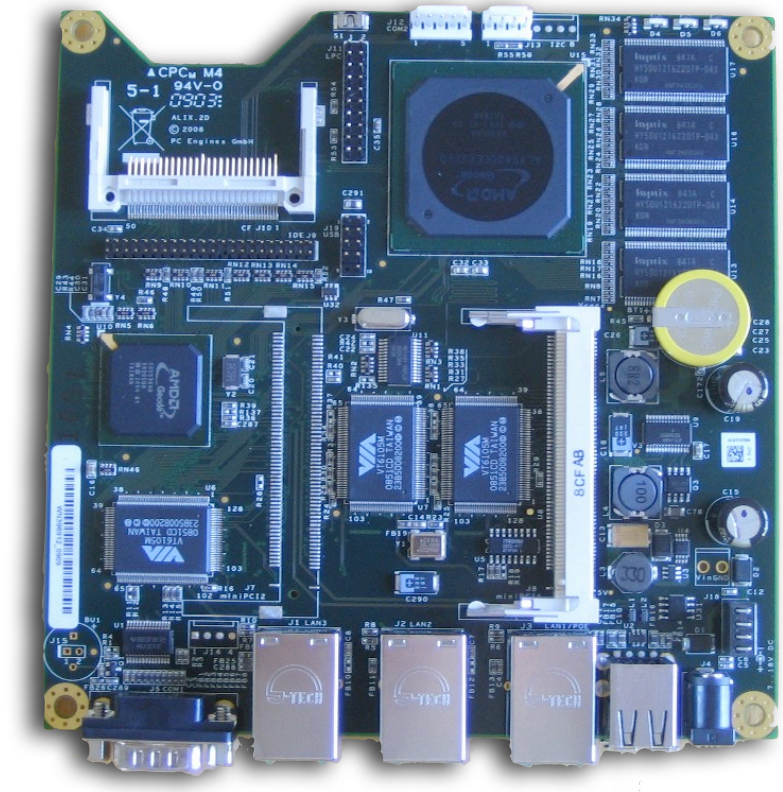


Figure 3.1: *The Alix2d13 board*

The board has a compact flash with a capacity of 8GB which is more than enough for the operating system (Bifrost is only ~100MB) and the storaging of the measurements. Also, it has a 500 MHz CPU (AMD Geode LX800) and a 256 MB DDR DRAM.

Finally, this board is perfect for our purposes because it is very low power. It consumes around 5W, which is by far more energy efficient that a common PC.

3.2 Bifrost OS

Bifrost is the operating system that is installed on the gateway. It is a small Linux distribution for USB media, flash disks etc. It's mainly targeted for production and infrastructure networking and routing.

Its main characteristics are:

- •Linux for infrastructure
- •Stability
- •Network performance
- •Simplicity
- •Dependency reduction

There is a standard method that the team used for building and installing packages into the OS. It is a predictable and repeatable building environment for Bifrost binaries which consists of a uclibc mini-native chroot environment and a package of management and build scripts [5]. For the purposes of the project, the team built and installed several packages to update or add new features to the gateway. Since Bifrost OS is a very lightweight distribution, many important features of a typical OS were missing.

In the beginning of the project the team was trying to install a C compiler as long as many other packages like PHP and MySQL in order to keep a history of the measurements in a database. However, this option was rejected since we desire to have a very power efficient gateway without overloads. Instead the compiling of the c-programs that we use is being done in another linux machine by setting the CFLAGS of the g++ compiler to CFLAGS=-march=i586 and LDFLAGS=-static in order to make an executable that has inside all the needed libraries.

Finally, in order to keep history we could just write on a txt file with the C-program or find another more elegant way. It depends on the user requirements and the sponsors opinion. However the team chose to keep a history log in a text file and to create XML files which then are read by the website and create diagrams as it will be explained in the following sections.

3.3 Serial Communication

The serial communication between the gateway and the MCU is being done through the USB port of the gateway. The Server-MCU is not connected to the Sensors board only in order to get the required power and to measure the sensors, but also in order to use the FTDI chip for the serial communication. In that way the communication is easier between the gateway and the MCU.

For reading the data from the MCU, the team developed a C-program that opens the USB port and reads the sent data. In order to run the program, the user has just to type on the command line inside the Bifrost the command `$ sca.out`

In that way the program `sca.out` will open the `ttyUSB0` port with a baudrate of 38400 bps, 8 data bits per packet, one stop bit and none parity bit.

When the MCU has sent the measurements to the gateway, the C-program will receive them and it will parse them character by character and, according to the predefined format explained in section 2.7, when it receives the closing bracket “}” it will do the following:

• Store the latest measurement to an array

• Parse the first two characters so as to understand who sent the data and what is the unit

• According to the result of the previous step

i. Store the values in a descriptive format to a txt file (log)
The format is: “Time Mote Unit Value”
e.g. 11:20 Server Temp 24°C

ii. Change the value for the monitor operation of the HTML page

iii. Write in XML format the value for the history operation of the HTML page
The format of each XML entry is:
“<WSN><M>*unit*</M><V1>*value1*</V1><V2>*value2*</V2></WSN>”

3.4 HTML page

The measured values from the sensors will be sent over the air from one MCU to another and the sensor node which will be connected to the gateway, will transfer through the serial port the measurements to the gateway. The C-program will then receive the results and print them to the html page for the monitoring operation and write an entry to the XML file for the history operation. In the figure below we can see the output screen. The html page will be updated according to the needs of the customers and how frequently they want to update the results.

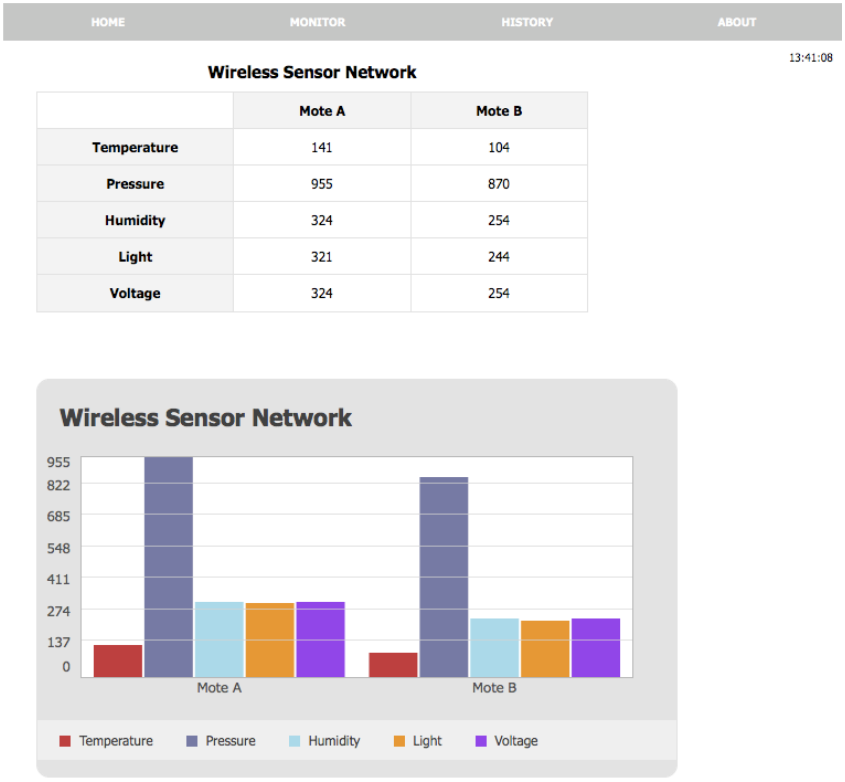


Figure 3.3: *The HTML page that monitors the measurements*

In the following figure we can see the history function of the web page. In this we first have to define the time period which can be:

- Last minute
- Last ten minutes
- Last hour
- Last day

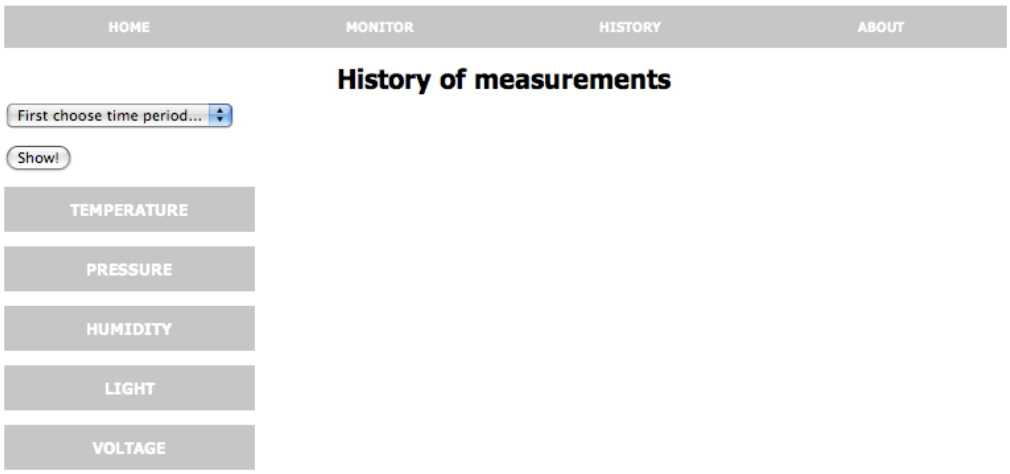


Figure 3.4: *History of measurements*

Of course the time period is depended on the number of the results that the system has recorded. Potentially, the time period could be much longer, for example weeks, months or even years. The only restriction is the capacity of the gateway which currently is 8 GB.

When we choose the time period that we want to see the measurements for, we can open whichever of the unit tabs we want among Temperature, Humidity etc.

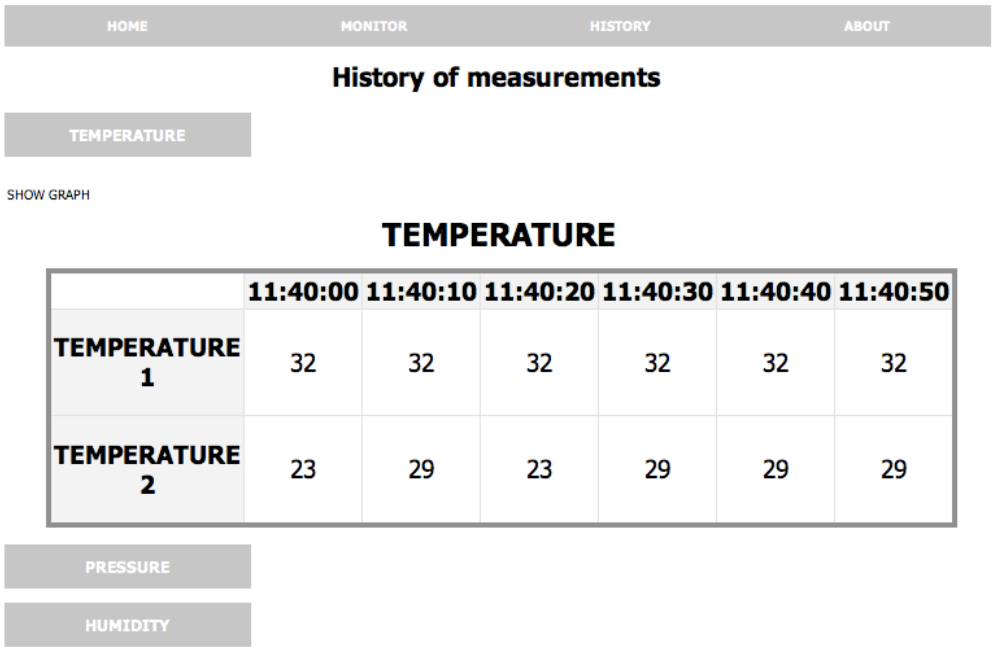


Figure 3.5: *The table with the measurements in the history*

In the figure above we can see an array with the measurements for the last minute. We can click the button SHOW GRAPH, in order to see the measurements in a graphical format. In the following figure we can see the graph which shows in a more clear way the differences in the measurements between the motes.

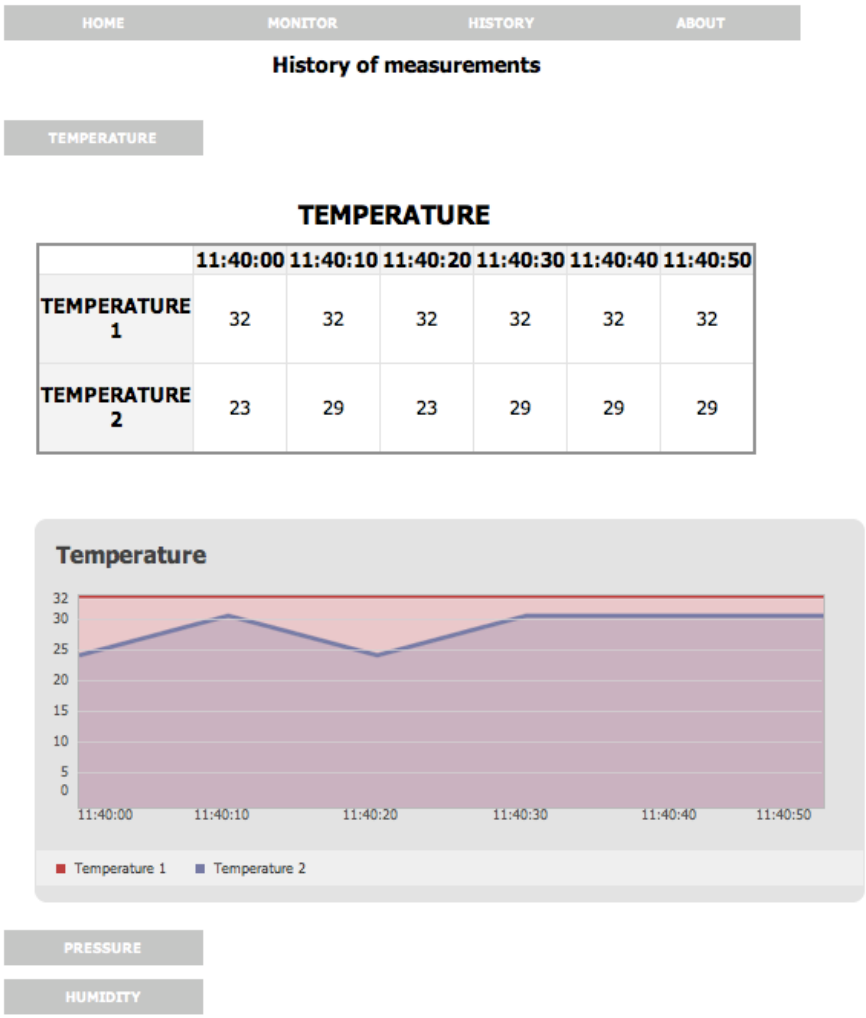


Figure 3.6: *The HTML page that presents the history of measurements.*

The whole programming of the site was made in HTML and Javascript. Since we cannot install MySQL and PHP on our server (Alix board), it is not possible to keep a database and load the data from it. However, we can parse the XML files that are created by the serial communication program with javascript, build the table and with jQuery's help design the schematic on the computer of the client.

4. Statistical Analysis

In this section the statistical analysis of the measurements is presented. In order to test the precision and the difference of the sensors on the two boards, the conditions are given as follows: The voltage supply for the two ATmega128RFA1 boards is 3.3V; the voltage supply for the humidity and light sensors is 5.0V; the voltage supply for the pressure and battery voltage sensors is 12.0V.

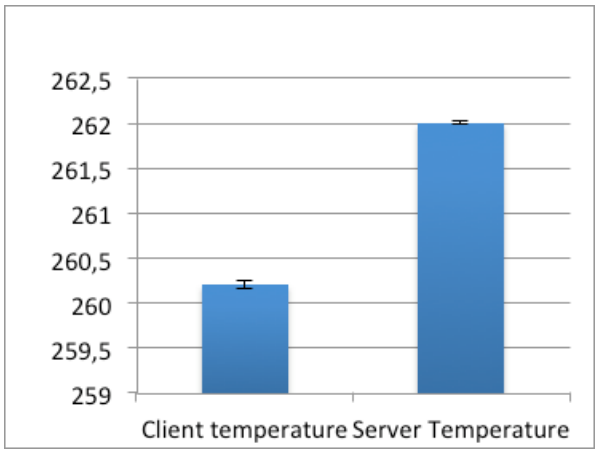
	Client Temp	Server Temp	Client Humid	Server Humid	Client Lumin flux	Server Lumin flux	Client Press	Server Press	Client Batt Volt	Server Batt Volt
Average	260,2	262	431,4	435,3	186,8	220,2	569,9	556	828,3	826,5
Standard error	0,043	0,017	0,352	0,272	0,044	0,066	0,157	0,072	0,047	0,064
Median	260	262	432	436	187	220	570	556	828	827
Standard deviation	0,426	0,173	3,519	2,725	0,44	0,656	1,566	0,718	0,469	0,642
Sample variance	0,181	0,03	12,384	7,428	0,193	0,43	2,454	0,515	0,22	0,412
Interval	2	2	31	19	2	4	14	3	1	4
Minimum	259	261	405	420	186	218	562	555	828	824
Maximum	261	263	436	439	188	222	576	558	829	828
Counting	100	100	100	100	100	100	100	100	100	100
Confidence level (95%)	0,084	0,0345	0,6983	0,5408	0,087	0,1301	0,311	0,1424	0,093	0,1274

Table 4.1: *Descriptive analysis*

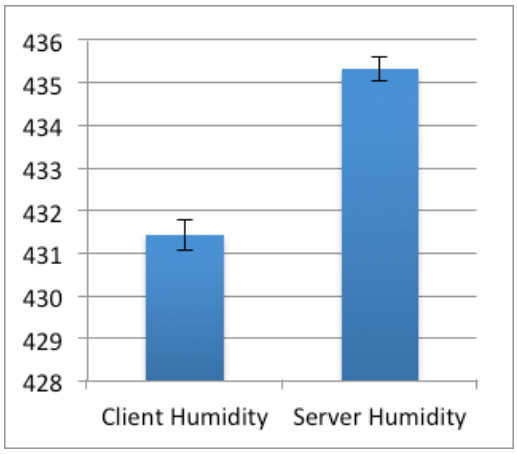
For every sensor, a sample of 100 is get from 100 ADC conversions, with one second's time interval between two samples. From Table 4.1, there are two observations.

First, the outputs of the same kind of sensors on different boards have deviations. And the deviation for light sensors and pressure sensors are quite large. Even the deviation of the average outputs of internal temperature sensors is 2. Possible explanations could be the offset of the ADCs on two boards are different, and the sensors scaling circuits on two boards have brought some errors, such as the deviations of the resistances from its nominal values.

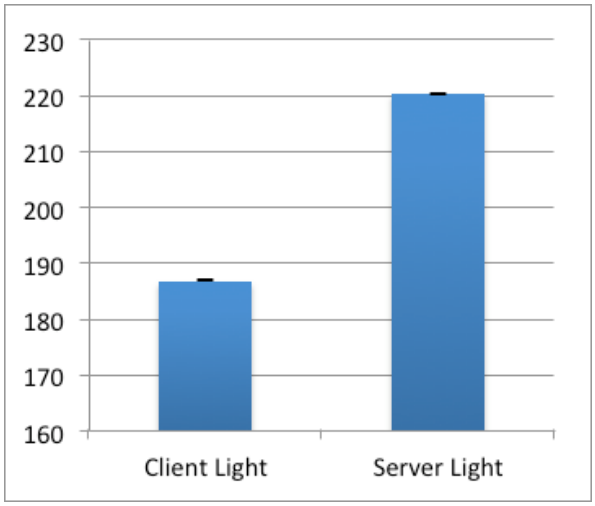
Second, the stability of the outputs of all the sensors is quite good, except the humidity sensors. From Table 4.1, the humidity ranges from 22.8% to 26.5%. While from the datasheet of the humidity sensor, the accuracy of it is 3.5% at room temperature. So the result is still acceptable.



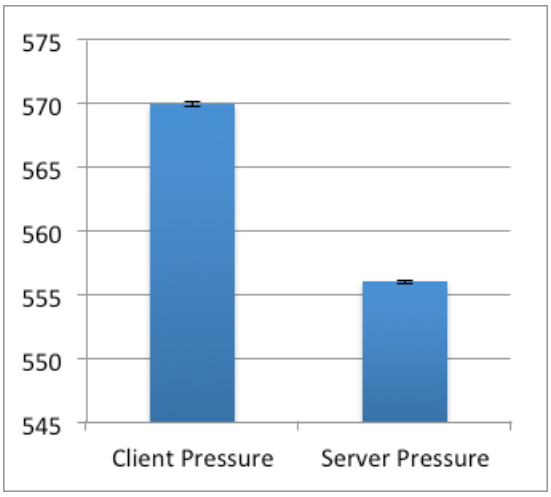
a) Internal Temperature Sensors



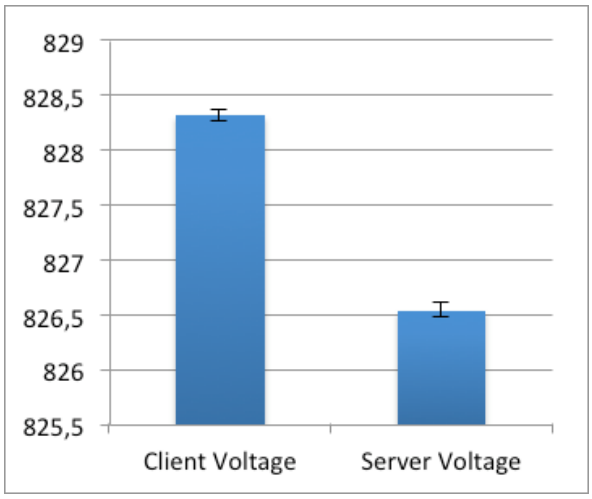
b) Humidity Sensors



c) Light Sensors



d) Pressure Sensors



e) Battery Sensors

Figure 4.1: *The average and 95% confidence level of different sensors' ADC conversion output*

5. Noise Analysis

The signal generated by the sensors is conditioned in the following steps: first amplified by 1 by the operational amplifier LP324N, then inside the ATmega128RFA1 board there is another analog buffer, and after the signal is digitalized by an analog to digital converter and finally processed.

All this stages introduce noise. In this paragraph it is analyzed the sources and amplitude to verify how many bit will be lost due to noise. [11]

The following diagram is representing the functional blocks of the conditioning system for one sensor.

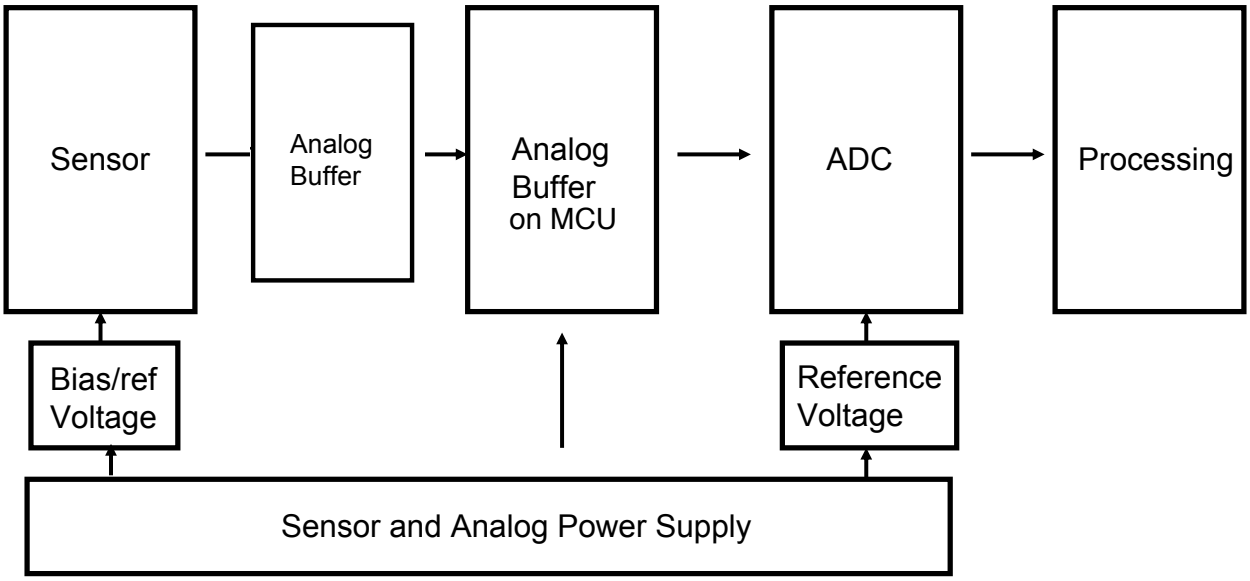


Figure 5.1: *Sensing circuit representation*

The assumptions are that everything will be operating over a range of ambient temperatures from 15 to 35 degrees C. Average room temperature is assumed to be 25 degrees C. The common components for all the sensors are: the ADC and the amplifier.

5.1 ADC noise

To convert the analog signal to digital format the AVR-ATmega128rfa1 use the internal 10-bit analog to digital converter. It has an internal reference of 1.6V. The resolution of the ADC is:

$$V_{LSB} = 1.6V / 2^{10} = 1.56mV$$

From the MCU datasheet we consider the value for the worst case: [6]

- Input offset error: $2.34mV = 1.5LSB$
- Gain error: $2.34mV = 1.5LSB$
- Differential Non-linearity (DNL): $1.56mV = 1LSB$

Total lost LSBs due to ADC noise is: $4LSB$

5.2 Sensors operational amplifier

The datasheet of the LP234N [8] gives all the informations about the noise parameters. As already explained the amplifiers all used in buffer configuration so the gain will be always equal to 1. The input offset voltage and the input bias current can be calibrate out so we will not consider them.

- Input resistance: Rin is extremely high $10^{12}\Omega$ so it is not significant.

- Output resistance: Rout is the output resistance of the Op Amp. It forms a voltage divider with the input resistance of the ADC, where the current is the leakage current.

$$200\Omega \bullet 1\mu A = 0.2mV \text{ in LSBs: } 0.128LSB$$

- Voltage noise: assuming noise is a narrowband:

$$V_N = 80nV/\sqrt{Hz} \bullet 1 = 80nV$$

$$80nV/1.56V = 5.13 \bullet 10^{-5} LSB$$

this value is really small so we can neglect it in the analysis.

- Current noise: assuming noise is a narrowband and the equivalent input impedance of 19.9k ohm:

$$V_{IN} = 0.6fA/\sqrt{Hz} \bullet 19.9k\Omega \bullet 1 = 11.9pV$$

$$11.9pV/1.56mV = 7.65 \bullet 10^{-9} LSB$$

this value is really small so we can neglect it in the analysis.

- Input offset voltage temperature drift:

$$TCV_{OS} = 10\mu V/°C \bullet 20°C \bullet 1 = 0.2mV = 0.13LSB$$

Total lost LSBs due to amplifier noise is 0.26 LSB, it is a low number because the gain is 1.

5.3 Sensors noise

The sensors used are: pressure sensor, humidity sensor, light sensor and voltage sensor. The following paragraphs will illustrate how the noise will affect the sensors.

5.3.1 Pressure sensor

From the sensor datasheet (MPX2200) [9], in the output characteristics it is shown a deviation of the output voltage from the linear curve. The worst case is when the pressure is maximum (200kPa) in this case the deviation is:

$$5mV = 3.2LSB$$

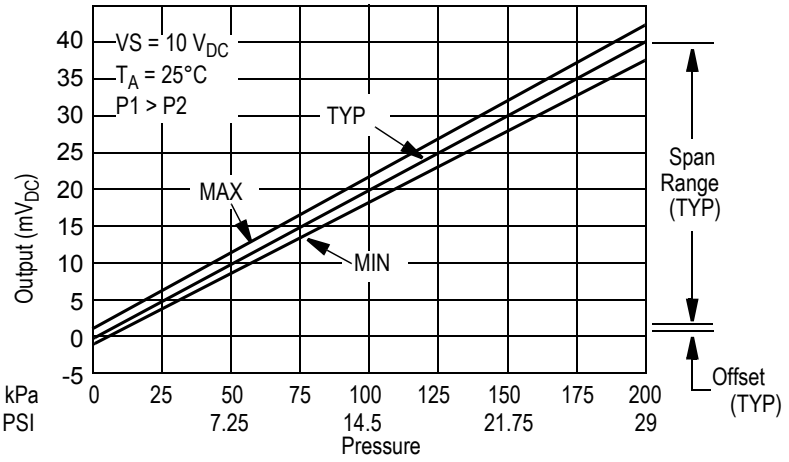


Figure 5.2: *MPX2200 pressure vs. output voltage characteristic*

5.3.2 Humidity sensor

The humidity sensor [11] has a maximum inaccuracy of +-3.5% when measuring the value of RH so this means that there will be a difference of 3.5% of voltage mismatch. On 1.6V reference this will be:

$$3.5\% \cdot 1.6V = 56mV = 36LSB$$

5.3.3 Light sensor

The LDR shows in the test to present an internal resistance of 2k ohm when exposed to high light.
Error sources are analyzed:

- LDR device tolerance: 5%;
- R resistor tolerance: 0.1%;
- LDR temperature coefficient: +-2% drift over 20 degrees C;
- R temperature coefficient: negligible over 20 degrees C.

The device tolerance for both LDR and R will be adjusted out in the buffer stage. They are fixed errors that don't vary for the same part. The temperature drift for LDR is significant and cannot be adjusted out. Calculate resulting voltage output drift for worst case.

The worst case is at full illumination:

$$2k\Omega + 2\% \rightarrow 4.74V \rightarrow 5V - 4.74V = 0.26V$$
$$2k\Omega - 2\% \rightarrow 4.55V \rightarrow 5V - 4.55V \rightarrow 0.45V$$

the difference is: 190mV, this number represent the thermal drift of the sensor over 20°C.

$$190mV = 122LSB$$

5.3.4 Voltage sensor

The sensing circuit for the 12V battery voltage is a simple voltage divider and it is designed to tolerate variations up to 18V. In any case the noise here is only due to resistor tolerance which is 0.1% and is fixed one, so it can be calibrated out.

5.4 Total noise

Summing everything up:

Noisy components	Error in mV	Error in LSBs
Pressure + Amp + ADC	11.63	7.46
Humidity + Amp + ADC	62.8	40.26
Light + Amp + ADC	190	126.26
Voltage + Amp + ADC	6.64	4.26

Table 5.1: *Error in mV and LSBs for each component*

So from the above chart the resulting values in bit are:

$$presure = \frac{\log(1024 - 7.46)}{\log 2} = 9.98bit$$

$$humidity = \frac{\log(1024 - 40.26)}{\log 2} = 9.94bit$$

$$light = \frac{\log(1024 - 126.26)}{\log 2} = 9.8bit$$

$$voltage = \frac{\log(1024 - 4.26)}{\log 2} = 9.99bit$$

To sum up, for all the sensors there is no full 10 bit precision but from the analysis it is shown that 9 bit are correct. The light sensor is the most noisy.

6. Design Metrics

6.1 Performance and specifications

These are the specification of Wireless node (mote). This list includes only the features that are used in the project.

No	Specification	Values
1	ZigBee	2.4 GHz IEEE 802.15.4
2	Transmission Output Power	up to 3.5 dBm
3	Data Rates	Up to 2 Mb/s
4	Power Consumption	75 mW (typical power consumed by each mote)
5	Pressure Range	0 - 200 kPa
6	Humidity Range	0 - 98%
7	Temperature range	-40^{o}C to 125 ^{o}C
8	Light Intensity	~10^{-4} to 1000 Lux
9	ADC Resolution	10 Bit
10	Quantization Error	0.75 mV
11	Maximum Sampling rate	3 kSamples per Sec
12	Data security and integrity	AES 128 bit
13	Battery operation Time	160- 300 hours

Table 6.1: *Specification chart*

6.2 Power estimation

The power consumed by each mote is calculated by taking the typical power consumed by each mote and the sensors. The voltage and the current consumed by each component is taken from their data-sheets. Note that not all the devices are operates on same voltage and also the leakage currents are not considered in this calculation so the actual power consumed by the mote will be definitely more then what is calculated. These are only the rough estimates.

No	Components	Current (mA)	Voltage (V)	Power (mW)
1	ATmega128RFA1	2.5	3.3	8.25
2	Pressure Sensor	8	12	96
3	Humidity Sensor	0.5	5	2.5
4	Light Sensor	0.02	5	0.1
5	Operational Amplifier	0.7	5	3.5
6	Operational Amplifier	0.7	12	8.4
	Total	12.42		118.75

Table 6.2: *Power estimation chart*

- Current = 12.42 mA
- Power = Voltage x Current = 118.75 mW

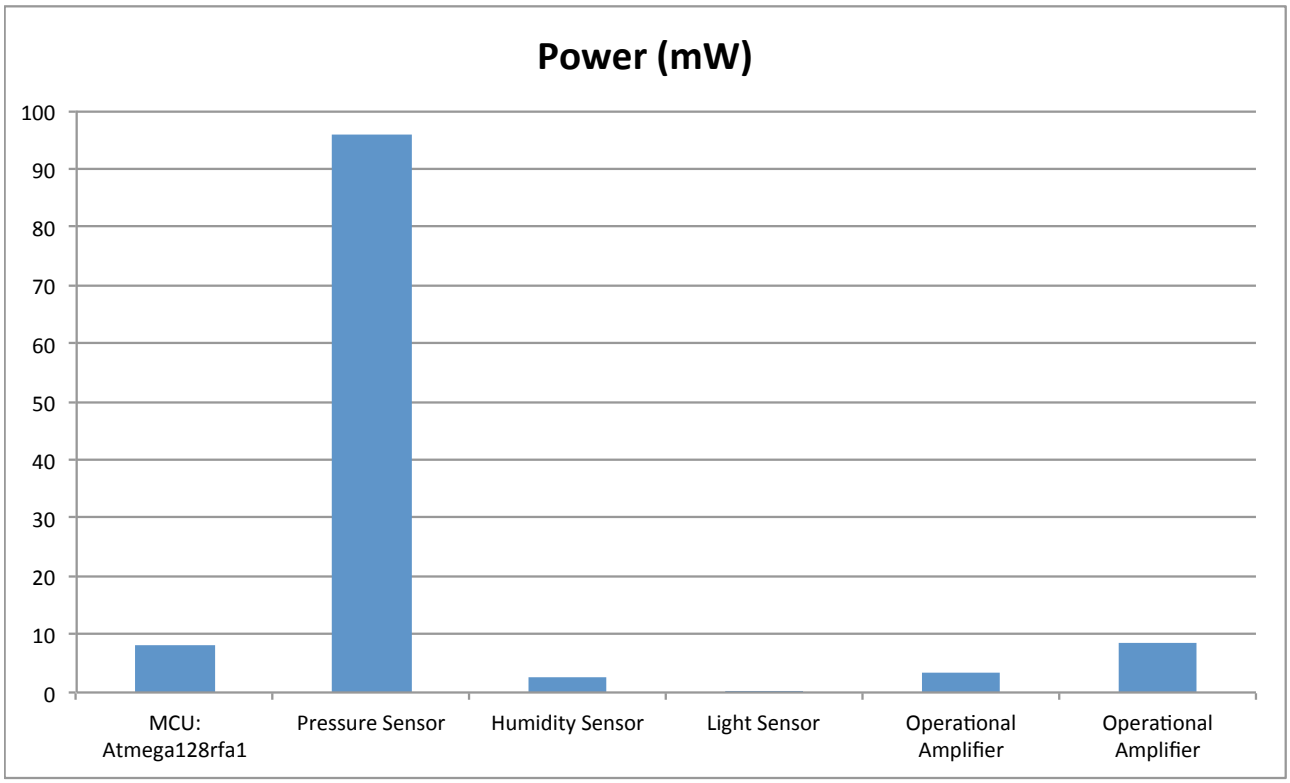


Figure 6.1: *Components power consumption*

Actual Power Consumption

The actual power is measured by using the digital multi-meter (DMM) and measure the voltage and current. Note there some extra components on the board that we are not using and also there are some leakage currents from resistors which increase the power consumption of mote and there is a big difference between the theoretical and actual power.

Server

The Server is working all the time receiving data from its clients and sending its own sensors data to gateway.

Components	Voltage (V)	Current (mA)	Power (mW)
Pressure Sensor, Operational Amplifier	12	10	120
MCU, EEROM, Antenna etc	3.3	20.2	66.6
Humidity, LDR, OpAmp	5	8.89	44.45
Total			**223.45**

Table 6.3: *Server power consumption*

Client

The client sleeps for 5 seconds and wakes up after that sample all the sensors data and send it to the server in 5 seconds. The sleep function is implemented to save the power and increase the battery operational time.

Components	Voltage (V)	Current (mA)	Power (mW)
Pressure Sensor, OpAmp	12	10	120
MCU, EEROM, Antenna etc	3.3	15	49.5
Humidity, LDR, OpAmp	5	8.89	44.45
Total			**213.95**

Table 6.4: *Client power consumption*

In this diagram 'W' is the time required for the MCU to wake up and 'S' is the time required to go to sleep. Sx (x= 1,2,3,4,5) is the time that the MCU takes to sample and process the data of each sensor and send them to server.

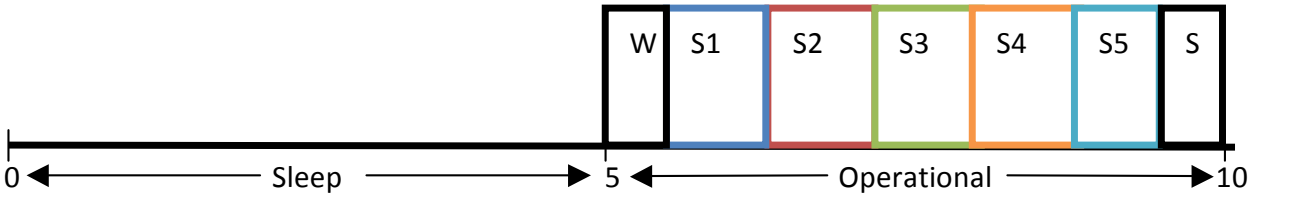


Figure 6.2: *Operations time division in milliseconds*

Operation Time Calculations

The system is powered by 1.5V batteries that are connected in series to reach 6V and 12V supplies. The charge of each battery is 2500mAh. The operational time of battery is given by:

- Battery charge = 2500 mAh
- Mote Current Consumptions = 12.42 mA
- Operation time = Battery charge / Mote Current = 201 hours

Note: these are rough estimates and the actual time can only be calculated by doing several experiments.

6.3 Cost Analysis

In this project were used both hardware and software in this project.

Software

All the software used are distributed under GNU public license for free software:

- Contiki-OS = Freeware
- AVR Studio and Tool chain = Freeware
- AVR dude = Freeware

Hardware

The hardware employed comprehend as main components: two wireless nodes and one gateway. The price of each components are given below:

- Mote(each) = 934.5 SEK
- Gateway = 735 SEK
- Total Cost of the Project = 2 x 934.5(Motes) + 735(Gateway) = 2604 SEK

The cost details of one mote is given in the following table:

No	Components	Part No.	Units	Cost (SEK)
1	Development board	ATmega128RFA1	1	365
2	Pressure Sensor	MPX2200	1	78
3	Light Sensor	LDR	1	3.34
4	Humidity Sensor	HIH-4000	1	208
5	Operational Amplifier	LM324	2	16
6	Resistor	5.6kΩ,56kΩ,78kΩ,150Ω	9	52
7	Capacitors	470μF, 10μF	7	4
8	Batteries	1.5v, AAA	8	6.6
9	Veroboard		1	62.5
10	Battery Case	Four AAA cell holder	2	29
11	Serial to USB converter	MM232R	1	110
	Total (each Mote)			**934.5**

Table 6.5: *Cost analysis chart*

7. Critique and future improvement

7.1 Critique

Wireless sensor network (WSN) is an active research area with application ranging from simple water treatment control to a distributed industrial control system. For all this application finding a suitable operating system (OS) that is power efficient and able to support the entire system requirement is a critical issue. The Contiki OS on the ATmega128RFA1 MCU have been able to address most of these design requirements. The Contiki OS was found to be easy to use, implement and verify and yet small enough to implement on MCU [7]. The ATmega128RFA1 board has all the basic hardware (HW) support necessary to support WSN, such as wireless support, ADC, and enough memory with Fast Processor [6]. The members of this project team proudly suggest and recommend the adoption of the Contiki OS and the ATmega128RFA1 MCU as a standard WSN development environment or for any WSN problem. This being positive critique negative critics regarding the work done by the project team and the Contiki OS are:

- Power efficient and optimal sensor design. The chosen sensors and their implementation design could/should be optimized for better power utilization and uniform power distribution.
- Most sensor reading is not accurate and not properly calibrated. Especially the light sensor and pressure sensor readings are inaccurate and differ from one mote to another.
- The internal sensor embedded in the two test motes showed variation and this variation might be critical if the system require high accuracy.
- Some sensors exhibited a non-linearity, which forced a high approximation to implement on the MCU.
- Some components chosen in the design require different voltage (the MCU run at 3.3V, few sensors run on 5V and few run on 12V), this is not a good design causing further system instability and power loss.
- The choice of light and pressure sensors were very bad because they were not power efficient. If the system is to be implemented on battery the two sensors should be changed, otherwise the system power will be wasted.
- The ADC value conversion equations are designed for particular voltage and small change as 0.5V in the supply voltage could have a big change on the value seen by the MCU after the ADC conversion.
- The power supply on the boards is not well regulated. Power regulators were requested but couldn’t reach on time. The system might suffer significantly from variation in the supply voltage and this might be critical.
- The power saving implemented on the MCU doesn’t provide much power saving as it was intended.
- There is no power saving mechanism implemented on the sensor boards. This might be unacceptable if the system is run on battery.
- Communication scheme chosen might be inefficient with regards to the number of motes and the motes working environment.
- The security system implemented is just a simple XOR which might not be enough and it would have been nice if the system could take advantage of the key-encryption security available on the MCU.

- Communication with the gateway is one way and it cannot be used to configure the motes. This might put the system a bit short handed with regards to central configuration and upgrading the system.
- The html display used is not generic, so testing the system with more than two motes is not easy. Plus if there is a change in the one sensor, i.e one more sensor is added or removed, the whole system might need to be edited and changed.
- The system is implemented on test board with simple wires, this might cause the system to be unstable and simple wiring error could burn the MCU.

7.2 Future Work and improvements

WSN being a vast topic and the fact that the project was implemented with constraints of time and resource there are always room for improvement and change. Although there are many things that need to be done to standardize the WSN, here are few things the members of this project suggest to start with. The contents are grouped in sub-topics just for the sake of organizations.

Contiki OS

The development of the OS could benefit more from standardization. That is making the files and functions of the OS more general and less specific. Most implementation is MCU specific and modifying the OS for MCU that is not supported is not easy. Plus finding what is supported for which MCU and what is not is not easy. This is a major drawback and if the OS is to become standardize it has to address these issue and make the OS easy to implement on any MCU.

The same is to documentation most of the documentations available are MCU specific and having a support for each MCU is tedious and might end up overlooking important issues. Most documentation is MCU specific and shallow when it comes to details. Improving the documentation could increase the likely hood of the OS becoming a standard for WSN.

Integrated development environments (IDE) help users take full advantage of the OS and its support. Plus IDEs facilitate development and prototyping. Implementing the OS on Eclipse or some other IDE and making it available to all uses is far better solution than having a virtual machine with text editor. Currently the best development environment is the Instant-Contiki, which is a 4GB virtual machine image and coding, and debugging is done using standard text editors, which is less efficient as it is impossible to do in system debugging. Putting the development environment on IDE is far better solution than having a virtual machine.

The simulators and testing environments available for the OS suffer from similar problem as mentioned above, they are very MCU specific. The user cannot take advantage of these resources unless the MCU is supported. Plus the simulator could use better user interface and improved status visualization. Improving these simulators and making them more general could farther increase the fame of the OS.

ATmega128RFA1

Though there are no improvements regarding the Atmega128RFa1 MCU, there are lots of works regarding PCB design and standard WSN board. To make the MCU a standard WSN board there should be a farther work regarding on board power supply,

USART (serial) to USB conversion, USART to COM or RJ45, fixed connection for programing, debugging and testing and enough area to mount sensors and signal transceivers.

WSN on ATmega128RFA1 using Contiki OS

This design project implements the Contiki OS on the Atmega128RFa1 for a WSN. The major categories of the project are the communication, the ADC and The Sensors. Communication is about communication between the different motes and the communication to the gate way through which the Internet. The ADC is where the data acquisition, scaling and interpretation are done. The sensors are responsible for gathering of information and transduce it and pass it to the MCU.

Communication

Most improvements here have to do with standardization and betterment of quality. Such improvements could be using the serial port and the USART for serial communication instead of the current implementation where the USART is converted to USB and read through the USB port. Even farther improvements could be converting the USART outputs to RJ45 instead of serial COM or USB and using it on standard network devices.

Another improvement area could be standardizing the wireless communication between the mote and designing a suitable protocol that can easily be implemented for reliable and secure data communication between motes. Plus this protocol should address the issues of addressing and dynamic rout configuration with out violating the design requirements and limitations.

ADC

Though there is not much improvement in these section future works could be improving data range for better resolution and event triggered sampling instead of pooled sampling could be interesting investigation areas. Plus the team has implemented the ADC driver for Atmega128RFa1 testing these drivers, verifying and documenting the code and finally contributing it to the Contiki OS should be done after the final presentation of the project.

Sensors

Regarding sensor much work could be done starting with sensor board PCB design. The sensor board could be part of or a separate part from the WSN board, which host the MCU and the IOs. Sensors are very sensitive to noise so a proper module should be designed to reduce these problems. The PCB should also address the need for steady and noiseless power.

Further works could be in improving the sensitivity of the sensors and designing them to consume as low power as possible. Plus using getting to improve sensors power consumption by turning them off when not needed are some of the future work.

In sum the WSN is very interesting topic and there are much work to be done especially regarding the area of standardizing and the members of this project believe this could be the next big thing after the Internet.

8. References

[1] Commision of the European Communities, "*Internet of Things — An action plan for Europe*", June 2009, Available: http://ec.europa.eu/information_society/policy/rfid/documents/commiot2009.pdf

[2] Davide Casaleggio, "*Internet of things*", Focus, Feb 2011, Available: http://www.casaleggio.it/pubblicazioni/Focus_internet_of_things_v1.81 - eng.pdf

[3] "*Contiki OS Webpage*", Available: http://www.contiki-os.org/

[4] "*Green Networks*", CSD-KTH, Available: http://csd.xen.ssvl.kth.se/csdlive/content/green-networks

[5] *"Bifrost-build"*, jellas, Available: https://github.com/jelaas/bifrost-build

[6] Atmel Corporation. (2011). ATmega128RFA1 Datasheet . *8-bit Microcontroller with Low Power 2.4GHz Transceiver for ZigBee and IEEE 802.15.4*. 1 (1), 1-560. Available: http://www.atmel.com/dyn/resources/prod_documents/doc8266.pdf

[7] Adam Dunkels. (2007). Contiki 2.5 Reference Manual. *The Contiki Operating System 2.5*. 1 (1), 1-330.

[8] LP324N datasheet,"*Quad Operational amplifier*", Available:www.ti.com/lit/ds/symlink/lp2902.pdf

[9] MPX2200 datasheet,"*Pressure sensor*", Available: http://cache.freescale.com/files/sensors/doc/data_sheet/MPX2200.pdf?fsrch=1&sr=1

[10] HIH-4000-001 datasheet,"Humidity sensor", Available: http://sensing.honeywell.com/index.php?ci_id=49852&la_id=1

[11] Smith, M.T. , Sensor Based Systems, "*Signal Characteristics and Conditioning*", Lecture 3

[12] LDR datasheet, "*Photoconductive cell*", Available: http://www.farnell.com/datasheets/612931.pdf

[13] UA7805CKCS datasheet, "*Voltage regulator*", Available: http://www.ti.com/litv/pdf/slvs056m

[14] LM2937ET datasheet, "*500 mA Low Dropout Regulator*", Available: http://www.national.com/profile/snip.cgi/openDS=LM2937

II. Appendices

Appendix A: Project Management Report

1. General Summary

In general the project was successful. All the primary goals have been reached and all the assignments were delivered on time. The project team followed the suggestions of the advisors and did the most they could during the short period they had to accomplish the project. All the work packages were finished before their deadline and in some occasions the tasks were started even before their start time.

The most important factor that led to the project success was the best practice cooperation among the group members whose lesson learned after this project was that project management techniques in conjunction with well established communication is a key to make your work easier, faster and enjoyable. In the following paragraphs the authors provide a descriptive analysis on the different aspects of the project.

2. Follow-up of objectives

In the project plan the primary goal of the project was to design and implement a wireless sensor network with motes that are controlled by the AVR ATmega128RFA1 microprocessors and operated by the Contiki OS. The motes had to be connected wirelessly and the final mote should be connected to a gateway that displays the measurements of each mote in a graphical format through a HTTP server. All those primary goals were achieved by the team according to the project plan.

However, there was one more primary goal that was not accomplished due to the dependence on another external team, which was not ready yet with their project. The purpose of this goal was the connection of each mote with ultra-capacitors in order to get power from them. Although the team had enough time in order to accomplish also this goal, the advisors suggested to work on the optimization and better integration of the project and to skip this task so as not to lose time.

Nevertheless, the team worked on the power management of the micro-controller units and managed to make the whole system power aware, since the motes are put to sleep mode every time they don't take measurements from the sensors.

Furthermore, the team managed to accomplish also the secondary goal of the project plan which was the data security, which provides security to the data while they are transfered over the air.

To sum up, the project plan was followed in almost all the aspects of the project. There were some deviations on it, but the reasons of the deviations were beyond the project team.

3. Experiences and suggested improvements

During the project, the team members had many positive and negative experiences. Some of them are described in the following table. The experiences listed are reflecting the opinion of all the team members.

Positive experiences ✔	Negative Experiences ✘
Gained knowledge about Contiki OS and Wireless Sensor Networks	Bad working environment. The working room was too small for 12 people.
Learnt the difficulties to work with many people and to depend on the work of others.	The team couldn't access the floor where the room was located.
Working with people from different technical and cultural backgrounds gives a sense of diversity.	In the beginning there was lack of communication among the team members.
Learnt the importance of proper scheduling your tasks and meeting deadlines.	Sometimes work load not properly divided among team members.
Reached the final goal successfully!	Lack of hardware resources.

Communication

The most important positive experience is one that started to be negative. In the beginning there was a difficulty in the communication among the team members. However, after a few meetings the situation became much better and since then the project progressed much faster.

Hardware Resources

One problem that frustrated a lot the team members was the lack of hardware resources. It is reasonable that the sponsors could not provide with more hardware resources that the initials, but that caused a problem in the different tasks inside the team, since some tasks were in parallel with others. There were many times that the board was needed by more than one team members which caused some delay in the progress of the project. However, the best practice communication among the team members helped to solve this problem whenever it appeared.

Working Environment

The sponsors provided to the team a shared room with another group. Unfortunately, the room was most of the time overcrowded. However the team decided to work on another lab whenever it was available. Furthermore, the team had no access to the floor where the room was located, although the sponsors asked for permission from the IT support of the building.

These two problems, although seem not important, constrained the project progress in some occasions because the team couldn't access its resources. It would be better if the sponsors and the advisors could arrange those trivial things before the beginning of the project in order to make it easier for the project team.

4. Summary of time and resource plans

During the execution time of the project, all the tasks were dealt according to the project plan. In some occasions, like the security and integration parts, the tasks were done before their scheduled time which shows that the team was working in the next tasks whenever there was free time.

However, in the beginning of the project there were many problems due to the lack of experience from the team members. The team spent too much time trying to understand and do the initial steps. Fortunately, the initial steps were scheduled in the project plan to take more than the normal time, so the plan was followed even in this situation.

Another interesting thing regarding the resource allocation is that although the plan was followed, most of the members of the team worked more than the initial plan. This was of course due to their inexperience on the different tasks they had to accomplish. However, after the “kick start” was given, the team worked more efficiently and with better results. All this data are visible to the resource allocation plans of each team member in the end of this final report.

In the following table we can see the hours that each member spent in each work package. It is visible that the team members followed the plan and in some cases spent more time which is probably due to their lack of experience in some tasks.

Member	PM	Contiki	Network	Sensors	Power	Security	Finalization	Planned Hours	Actual Hours
Guodong Guo	10h	81h	10h	28h			36h	163h	165h
Ibrahim Kazi	10h	50h		49h	12h	3h	12h	136h	160h
Mussie Tesfaye	14h	61h	22h		24h		56h	167h	177h
Prodromos Mekikis	66h		70h			2h	50h	188h	190h
Saad Fakher	39h	77h	6h	26h		9h	26h	148h	142h
Stefano Vignati	22h	71h	6h	18h	1h		17h	160h	135h
							Total	**962h**	**969h**

Table 4.1: *Resource allocation*

In total the team spent only 7 hours more than the plan which corresponds to 0.7% more work. In the midterm report this number was 11% and it was written that the time budget will not increase more in the future unless some new problems arise. Fortunately, no problems have arisen and the team spent less time in the end of the project which compromised the additional working hours of the beginning of the project.

5. Final comment and lessons learned

The lessons learned in this project according to the team members are:

- Keep good communication with your partners
- Stick to the schedule and monitor it regularly
- Organize better meetings with other groups
- If you can do a task sooner, do it!

It is recommended to follow those four tips for the future projects in order to have more possibilities for a successful ending of the project.

6. Gantt Chart

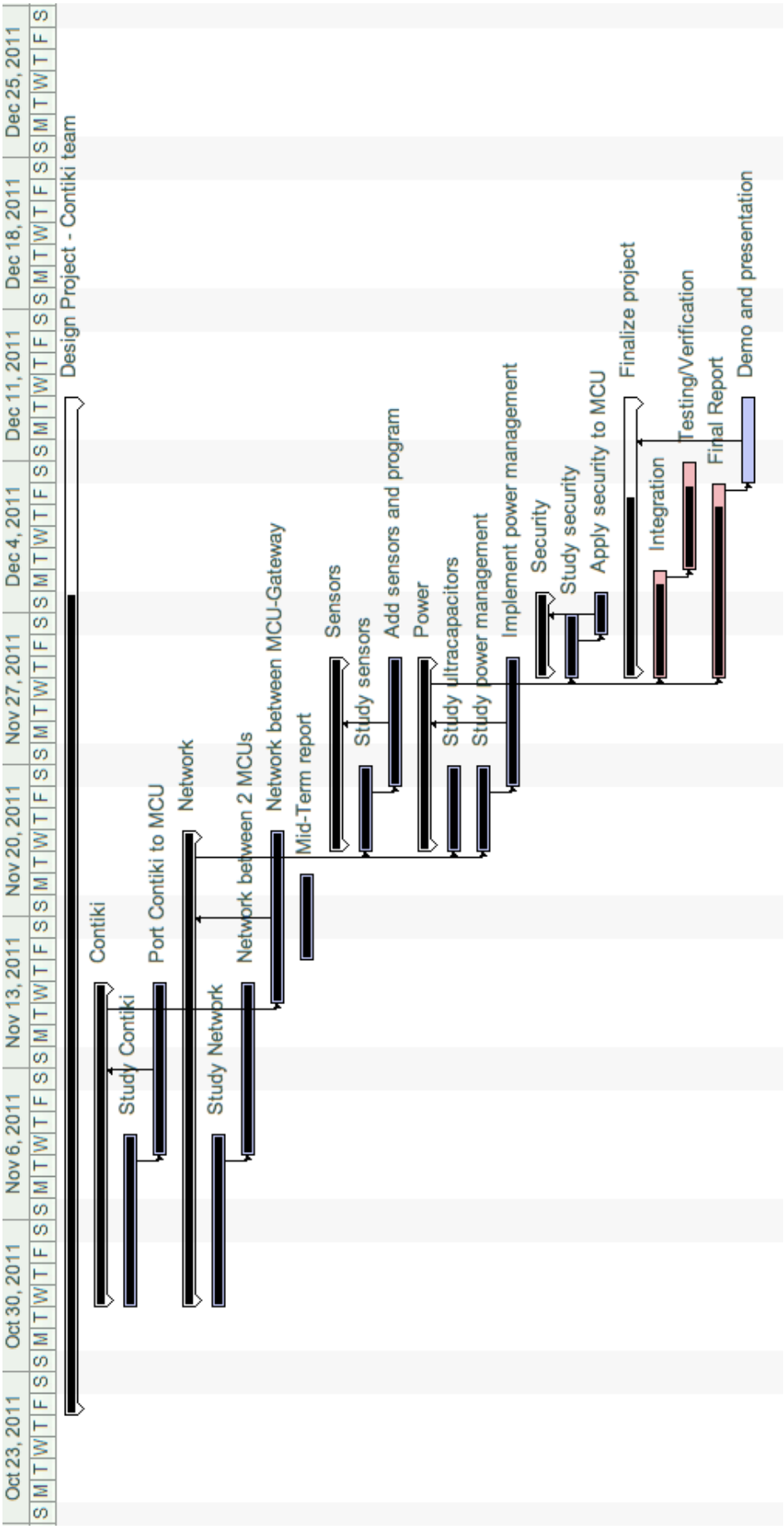


Figure 6.1: The Gantt Chart

7. Resource Allocation Plans

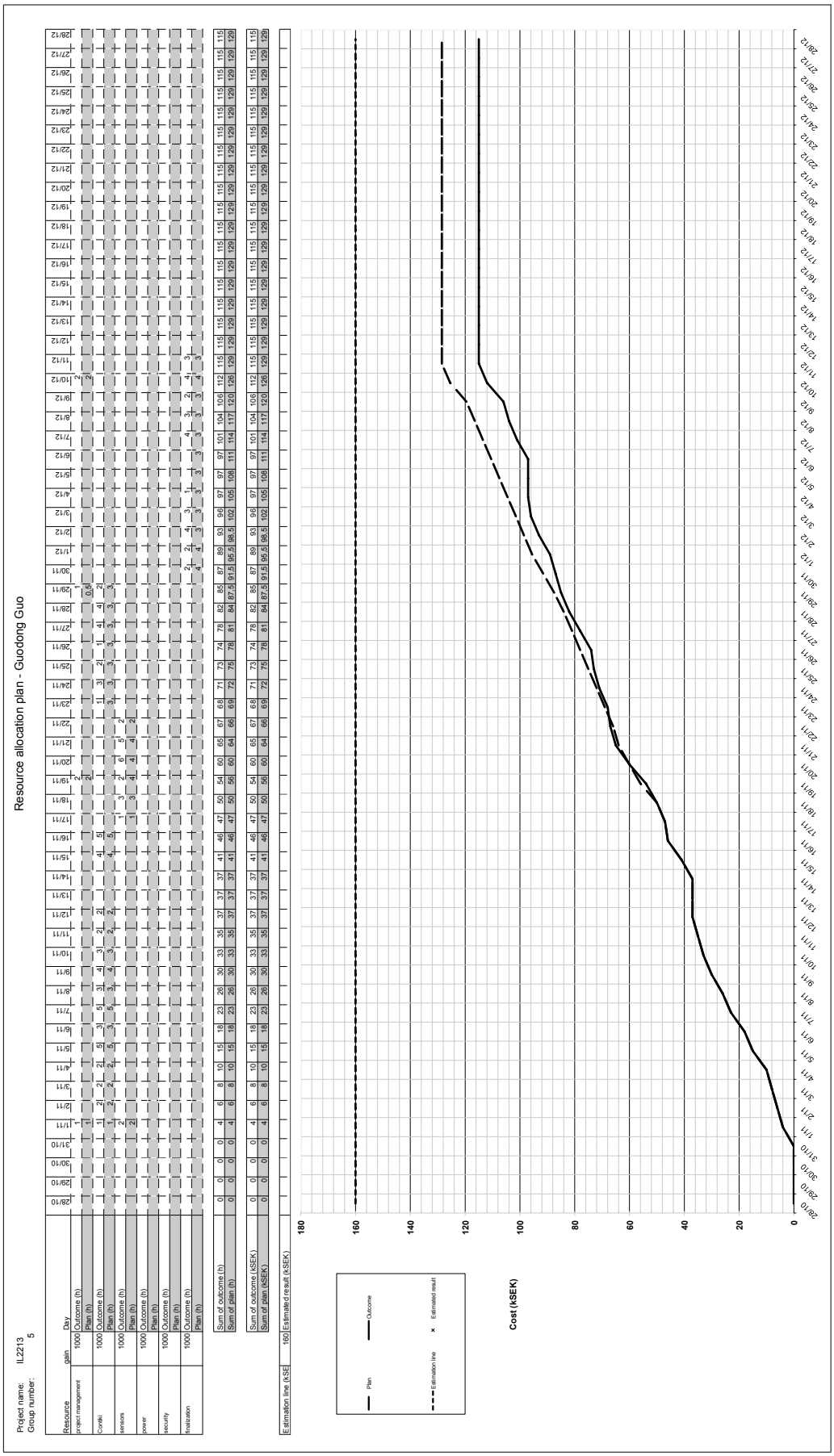


Figure 7.1: Guo's resource allocation plan

Figure 7.2: Vignati's resource allocation plan

Project name: IL2213
Group number: 5

Resource allocation plan - Mussie Tesfaye

Resource	gain	Day	28/10	29/10	30/10	31/10	1/11	2/11	3/11	4/11	5/11	6/11	7/11	8/11	9/11	10/11	11/11	12/11	13/11	14/11	15/11	16/11
project management	1000	Outcome (h)	8	4				1					0,5	1								
		Plan (h)	4	4				1	0	0	0	0	1	1	0	0	0	0	1	1	0	0
network	1000	Outcome (h)			3	3	3	3	2	3	3	2										
		Plan (h)			2	2	2	2	2	2	2	0	2	2	2	2	2	2	0	2	2	2
Contiki	1000	Outcome (h)			3	3	3	4	4	4	4	4	4		4	2	2	2		2	2	2
		Plan (h)			2	2	2	2	2	2	2	0	2	2	2	2	2	2	0	2	2	2
power	1000	Outcome (h)														2	2	2		2	2	2
		Plan (h)																				
integration	1000	Outcome (h)																				
		Plan (h)																				
finalization	1000	Outcome (h)																				
		Plan (h)																				
		Sum of outcome (h)	8	12	18	24	30	38	44	51	58	64	68,5	69,5	73,5	77,5	81,5	85,5	85,5	89,5	93,5	97,5
		Sum of plan (h)	4	8	12	16	20	25	29	33	37	37	42	47	51	55	59	63	64	69	73	77
		Sum of outcome (kSEK)	8	12	18	24	30	38	44	51	58	64	68,5	69,5	73,5	77,5	81,5	85,5	85,5	89,5	93,5	97,5
		Sum of plan (kSEK)	4	8	12	16	20	25	29	33	37	37	42	47	51	55	59	63	64	69	73	77
Estimation line (kSE	160	Estimated result (kSEK)																				

Resource	gain	Day	17/11	18/11	19/11	20/11	21/11	22/11	23/11	24/11	25/11	26/11	27/11	28/11	29/11	30/11	1/12	2/12	3/12	4/12	5/12	6/12
project management	1000	Outcome (h)																				
		Plan (h)	0	1	1	1	1	1	1													
network	1000	Outcome (h)																				
		Plan (h)	2	2	2	0	2	2	2													
Contiki	1000	Outcome (h)	2	2	2		2	2	2													
		Plan (h)	2	2	2	0	2	2	2													
power	1000	Outcome (h)	2	2	2		2	2	2													
		Plan (h)							2	4	0	4	4	4								
integration	1000	Outcome (h)							2	4		4	4	4								
		Plan (h)													4	4	4	0	4	4		
finalization	1000	Outcome (h)													4	4	4	4	4	4	4	4
		Plan (h)															0	0	0	0	2	2
		Sum of outcome (h)	102	106	110	110	114	118	124	128	128	132	136	140	144	148	152	156	160	164	168	172
		Sum of plan (h)	81	86	91	92	97	102	109	113	113	117	121	125	129	133	137	137	141	145	147	149
		Sum of outcome (kSEK)	102	106	110	110	114	118	124	128	128	132	136	140	144	148	152	156	160	164	168	172
		Sum of plan (kSEK)	81	86	91	92	97	102	109	113	113	117	121	125	129	133	137	137	141	145	147	149
Estimation line (kSE	160	Estimated result (kSEK)																				

Resource	gain	Day	7/12	8/12	9/12	10/12	11/12	12/12	13/12	14/12	15/12	16/12	17/12	18/12	19/12	20/12	21/12	22/12	23/12	24/12	25/12	26/12	27/12	28/12
project management	1000	Outcome (h)																						
		Plan (h)																						
network	1000	Outcome (h)																						
		Plan (h)																						
Contiki	1000	Outcome (h)																						
		Plan (h)																						
power	1000	Outcome (h)																						
		Plan (h)																						
integration	1000	Outcome (h)																						
		Plan (h)																						
finalization	1000	Outcome (h)	4	4	4	4	4	4																
		Plan (h)	2	2	0	4	4	6																
		Sum of outcome (h)	176	180	184	188	192	196	196	196	196	196	196	196	196	196	196	196	196	196	196	196	196	196
		Sum of plan (h)	151	153	153	157	161	167	167	167	167	167	167	167	167	167	167	167	167	167	167	167	167	167
		Sum of outcome (kSEK)	176	180	184	188	192	196	196	196	196	196	196	196	196	196	196	196	196	196	196	196	196	196
		Sum of plan (kSEK)	151	153	153	157	161	167	167	167	167	167	167	167	167	167	167	167	167	167	167	167	167	167
Estimation line (kSE	160	Estimated result (kSEK)																						

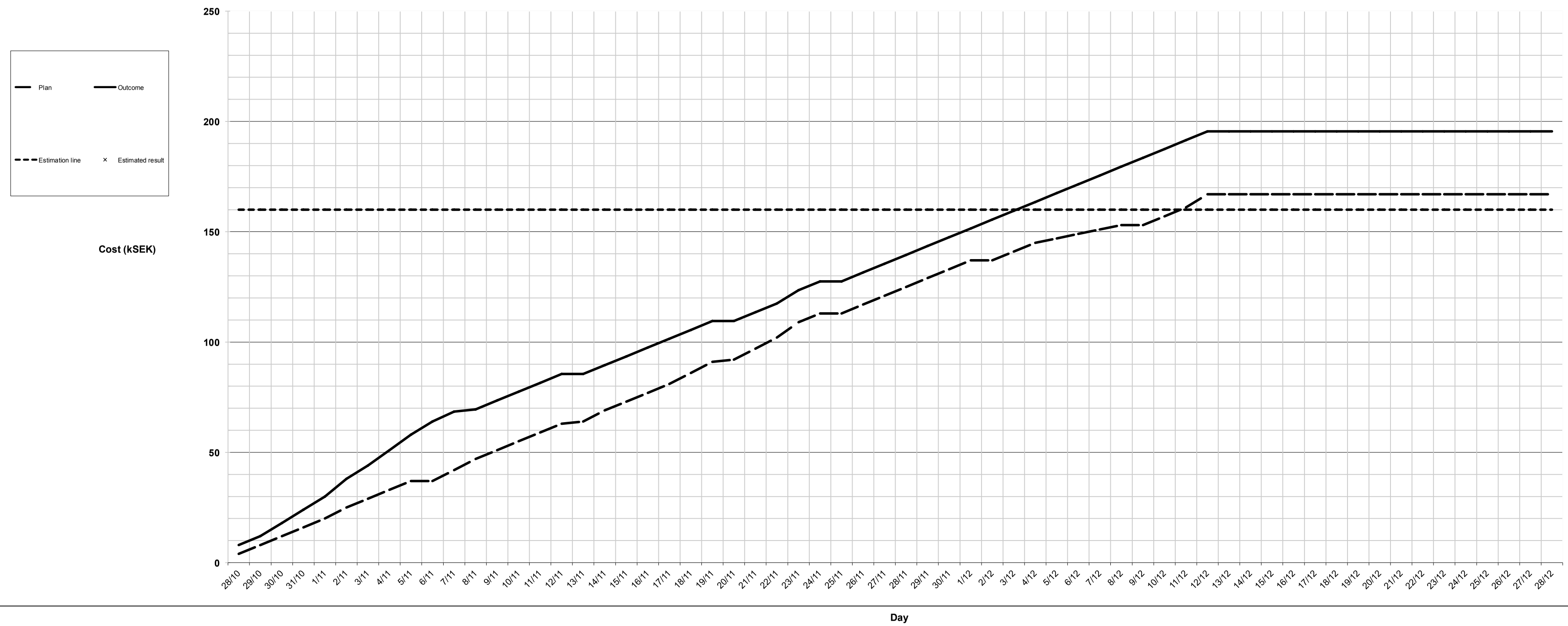


Figure 7.3: Tesfaye's resource allocation plan

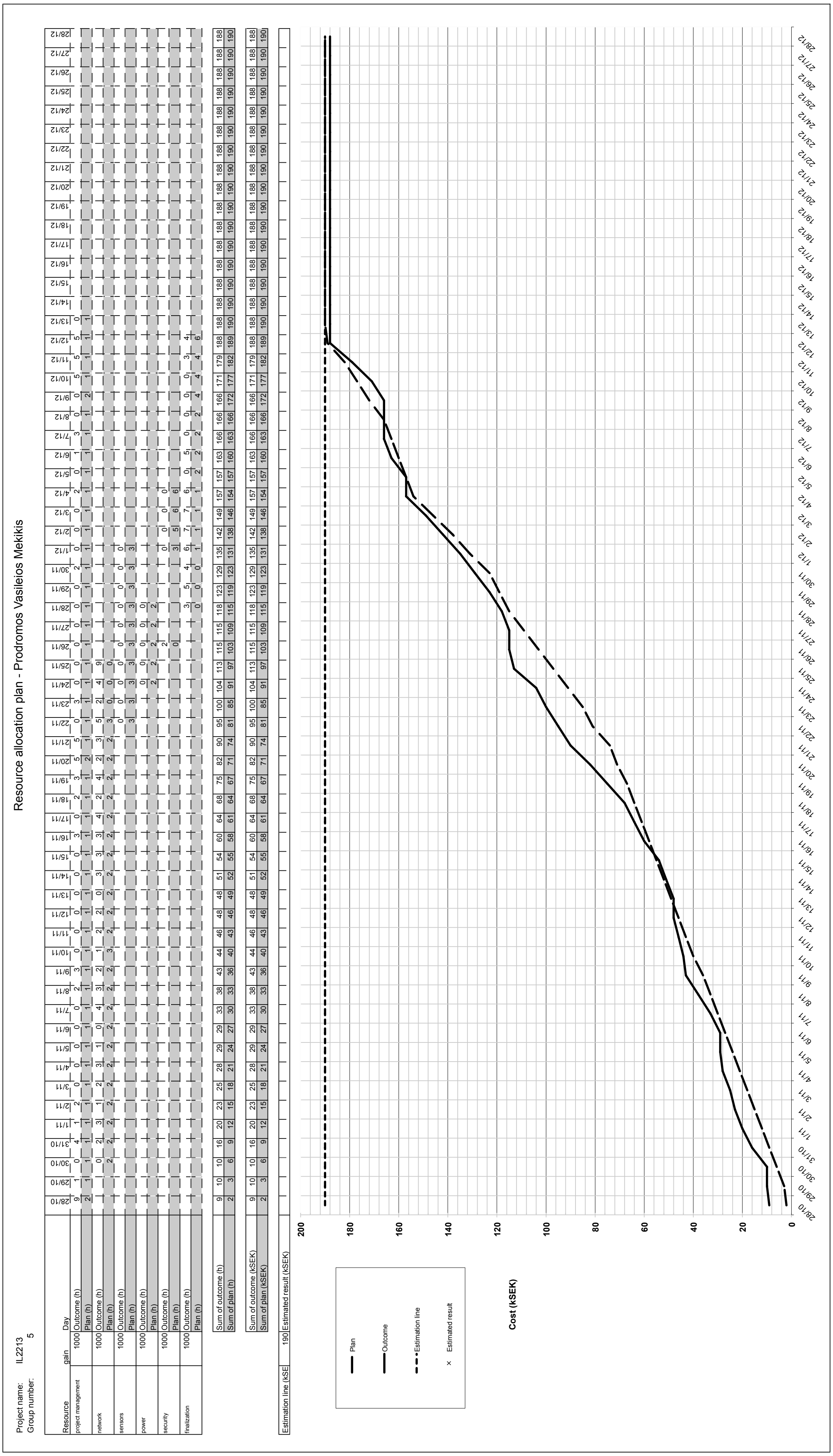


Figure 7.4: Mekikis's resource allocation plan

Industriella informations- och styrsystem, KTH
www.ics.kth.se

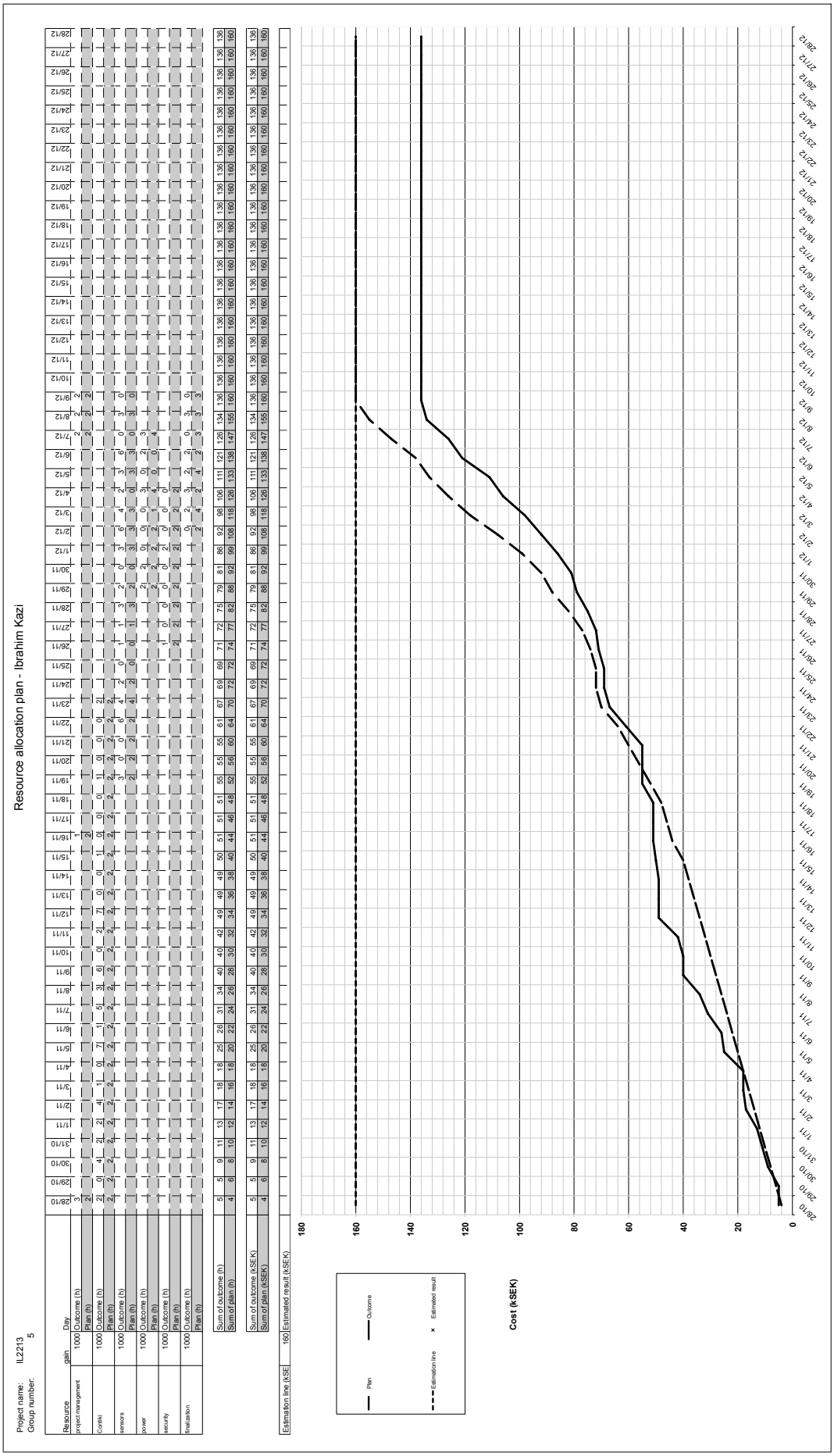


Figure 7.5: Kazi's resource allocation plan

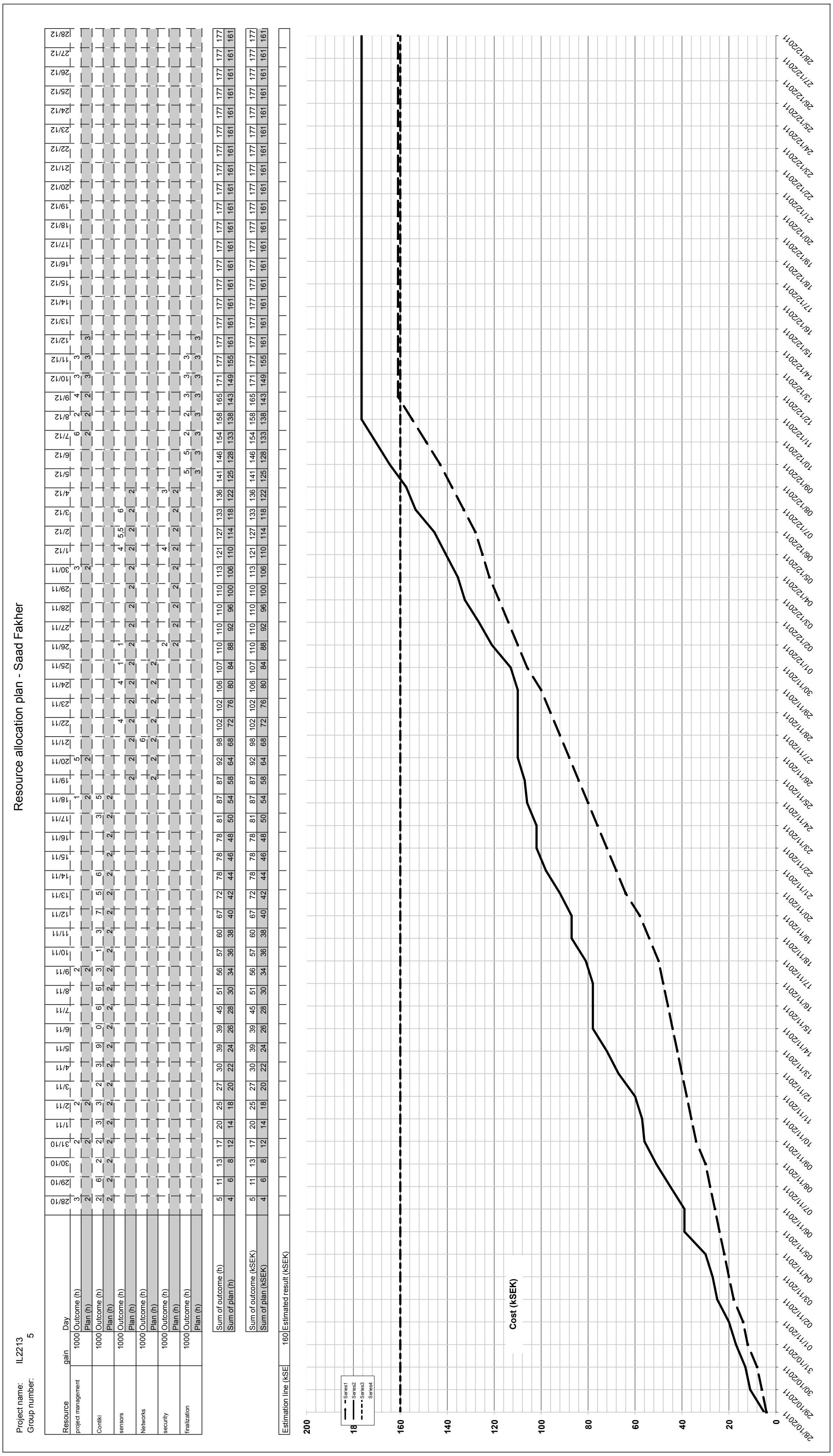
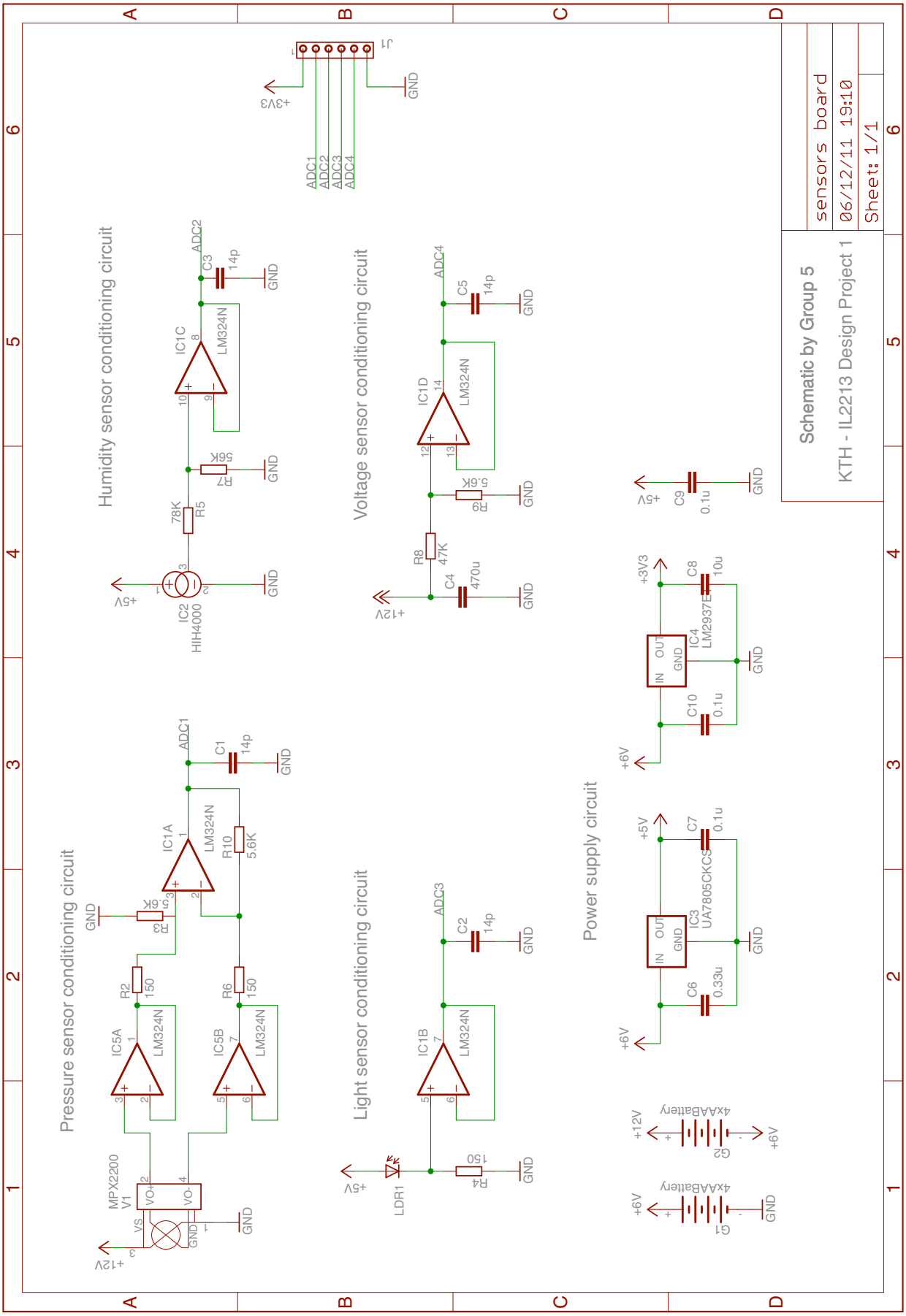


Figure 7.6: Fakher's resource allocation plan

Appendix B: Sensors Board Schematic



Appendix C: Source code

The source code for all the parts of the project is available at the following link:

http://code.google.com/p/project5-contiki-os/