Meteorological station

Documentation & report

Date: 15.04.2019

Information Technology

Monday 10:15, group 9

IFE, 4th semester

# Team & task assignment

|  |  |
| --- | --- |
| Piotr Kaźmierski (leader) | * Bluetooth + UART |
| * temperature & humidity sensor |
| Piotr Kocik | * button + interrupts |
| Michał Kuśmidrowicz | * LCD screen + I2C |
| Kacper Kubicki | * pressure sensor + SPI |
| * LED + PWM |

# Devices

* Arduino Uno Rev3
* DHT11 temperature & humidity sensor
* BMP280 temperature, pressure & altitude sensor
* Green LCD 2x16 with LED backlight through a I2C converter
* HC-05 Bluetooth module
* RGB LED with common anode
* Tact Switch 6x6mm / 4,3mm THT

# Project description

## General description

The aim of the project was to create a meteorological station which could display the information relating to the weather conditions using an LCD screen with the ability to change currently shown information via a button or remotely, via Bluetooth. Additionally we wanted to add a persistent indicator of a specific weather condition in the form of an RGB LED diode.

## Information to be displayed

Each reading/information is displayed separately, centred on the LCD screen with a heading that informs which mode is currently used, e.g.:

**TEMPERATURE**

**28°C**

Possible information to be displayed are:

* Temperature
* Heat index (perceived temperature)
* Atmospheric pressure

## How to use it?

The currently displayed information can be changed via interrupt driven button (tactile switch) or remotely via Bluetooth using for example a smartphone with an appropriate app installed. The button allows to loop through all the screens one by one in a forward manner. Bluetooth allows to go back and forth between the screens as well as read the data remotely repeatedly or on demand.

### Control using Bluetooth

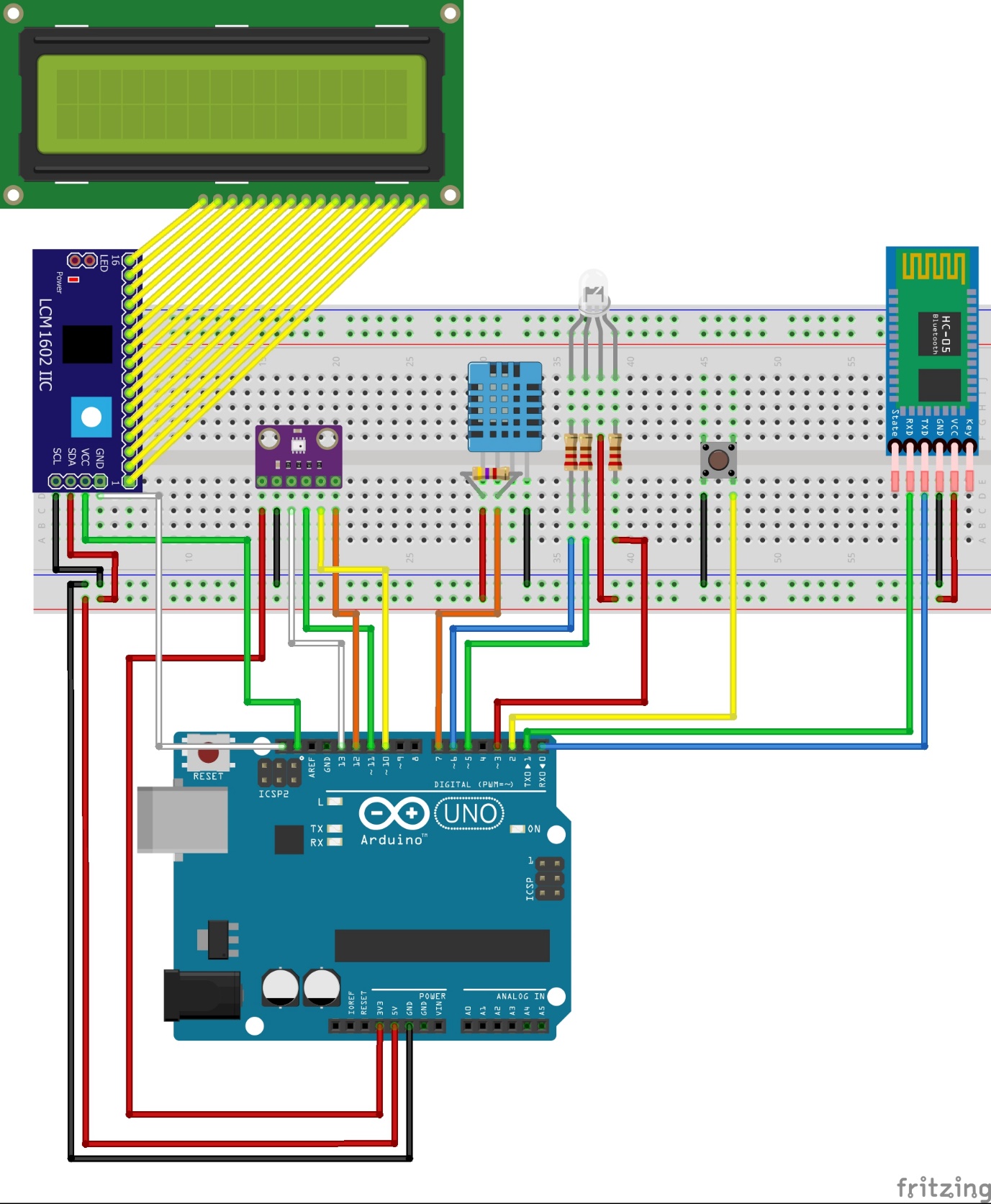
To use the meteorological station via Bluetooth one must have a Bluetooth enabled device with an app that can perform serial communication via Bluetooth. In our case we’re using “Bluetooth Electronics” by keuwlsoft which is available for Android devices via Google Play. It allows users to send commands via a serial terminal emulator and assign specific terminal commands to buttons and sliders, which makes controlling the device remotely even easier.

**//TUTAJ SCREENSHOTY Z APLIKACJI I JAKIŚ DOKŁADNIEJSZY OPIS**

# Peripherals and interface configuration

## GPIO

Below the GPIO connection diagram is presented:



The table below presents the pinout:

|  |  |  |  |
| --- | --- | --- | --- |
| Pin | Alias | PWM | Assigned to |
| 0 | RX | No | Bluetooth module TX |
| 1 | TX | No | Bluetooth module RX |
| 2 | IRQ0 | No | Tactile switch (interrupt enabled) |
| 3 | IRQ1 | Yes | RBG LED red component |
| 4 | - | No | - |
| 5 | - | Yes | RGB LED green component |
| 6 | - | Yes | RGB LED blue component |
| 7 | - | No | DHT11 data pin |
| 8 | - | No | - |
| 9 | - | Yes | - |
| 10 | SS | Yes | BMP280 CSB pin |
| 11 | MOSI | Yes | BMP280 SDA pin |
| 12 | MISO | No | BMP280 SDO pin |
| 13 | SCK | No | BMP280 SCL pin |
| 14 | A0 | N/A | - |
| 15 | A1 | N/A | - |
| 16 | A2 | N/A | - |
| 17 | A3 | N/A | - |
| 18 | A4, SDA | N/A | I2C LCD converter SDA pin |
| 19 | A5, SCL | N/A | I2C LCD converter SCL pin |

Pins 0 to 13 are digital pins. Some of them are PWM.

Pins A0 to A5 are analogue pins.

Pins 0 and 1 are used for UART communication via the Bluetooth module.

Pin 2, which is assigned to interrupt 0 (IRQ0) is used for the tactile switch.

Pins 3, 5, and 6 are used for red, green and blue components of the RGB diode respectively.

Pin 7 is used for the DHT11 sensor data send & receive.

Pins 10 through 13 are used for SPI communication with the BMP280 sensor.

Pins A4 and A5 are used as SDA and SCL in I2C protocol to communicate with the I2C LCD converter

Some of the pins are configured by the library functions, such as SPI pins, I2C pins and DHT11 pin.

### DHT11

DHT11 is configured using the library functions:

//meteo.ino - our code

#define DHTPIN 7

#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE); //DHT type object allows us to interface with the sensor via it’s class methods

void setup() {

//...

dht.begin();

//...

}

//DHT.cpp - DHT11 library

void DHT::begin(uint8\_t usec) { //the declaration has a default value for usec set

//...

pinMode(\_pin, INPUT\_PULLUP); //\_pin is a variable set by the DHT constructor that corresponds to the first constructor arugment (in our case DHTPIN that equals 7)

//...

}

An equivalent direct way of doing it is:

pinMode(DHTPIN, INPUT\_PULLUP);

Doing it via directly setting registers:

DDRD |= B10000000; //sets pin 7 to INPUT

PORTD |= B10000000; //sets pin 7 to HIGH, enabling pullup resistors

INPUT\_PULLUP sets a pin into an input mode with a pullup resistor attached. This reverses the way input should be read, since now high voltage = 0, and low voltage = 1.

### USART

USART pins don’t have to be configured.

## I2C and screen

In our project the I2C protocol is used for communication with the LCD screen. This is realised by using the LiquidCrystal library. Firstly, one has to create object representing the LCD screen and configure it via constructor and method begin():

LiquidCrystal\_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);

void setup() {

//...

lcd.begin(16,2);

//...

}

These numbers refer to EN (enabling pin), RW (pin writing/reading to/from - LCD), RS (pin selects registers between Instruction Register and Data Register), data pins (4,5,6,7), the back light control pin (3) and POSITIVE is the back light polarity. 16 and 2 stand for size of the screen, respectively its width and height. I2C uses two wires for communication- SDA (Serial Data) and SCL (Serial Clock). On Atmel ATmega328P I2C is also called TWI (two wire interface).

Under the hood, the I2C connection is established in the following way – firstly, the register bit TWINT (TWI Interrupt Flag) is reset by setting its value to 1. Then, the interface TWI is unlocked by setting a value of bit called TWEN (TWI Enable bit). After that, TWSTA bit (TWI Start Condition bit) is set to 1, which means that the transmission will be performed in ‘Master’ mode. Then, a while loop is waiting for re-setting flag TWINT, which will confirm that the hardware chip correctly performed the start sequence of the transmission.

void I2C\_start() {

// Assigning a value to control register - sending START condition

TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTA);

// Wait for TWINT flag set. This indicates that the START

// condition has been transmitted

while (! (TWCR & (1 << TWINT)));

}

## SPI

## USART

USART is configured via library function call:

const int DEFAULT\_BAUD\_RATE = 9600;

void setup() {

Serial.begin(DEFAULT\_BAUD\_RATE);

//...

}

To configure USART via register manipulation we have to enable the receiver and transmitter, as well as set the baud rate.

UCSR0B = (1<<RXEN0)|(1<<TXEN0); //enables the receiver and transmitter

UBRR0L = 103; //sets the baud rate to 9600

The rest of the settings are kept at default values and the most important of those are:

* Mode of operation: asynchronous
* Parity: disabled
* Stop bits: 1
* Character size: 8 bits

To send data via UART we have to first check if the USART Data Register Empty (UDRE0) bit is set to 0. If so, we can load the data into the USART transmit data buffer register. In case of 9-bit characters the highest bit has to be manually put into the Transmit Data Bit 8 (TXB80). Example:

void USART\_Transmit(unsigned int data) {

/\* Wait for empty transmit buffer \*/

while (!(UCSR0A & (1 << UDRE0))))

;

/\* Copy 9th bit to TXB8 \*/

UCSR0B &= ~(1 << TXB8);

if (data & 0x0100)

UCSR0B |= (1 << TXB80);

/\* Put data into buffer, sends the data \*/

UDR0 = data;

}

To receive data via UART we have to first check if there have been no errors during transmission. To do that, we have to check that USART Parity Error (UPE0) and Frame Error (FE0) bits are set to 0. If so, we can check if there’s data available for us by determining wheter USART Receive Complete is set to 1. If that’s true we can read data from USART receive data buffer register (UDR0) and optionally the Receive Data Bit 8 (RXB80), if we want to receive 9 bits of data. Example:

unsigned int USART\_Receive(void) {

unsigned char status, resh, resl;

/\* Wait for data to be received \*/

while (!(UCSRnA & (1 << RXCn)))

;

/\* Get status and 9th bit, then data \*/

/\* from buffer \*/

status = UCSRnA;

resh = UCSRnB;

resl = UDRn;

/\* If error, return -1 \*/

if (status & (1 << FEn) | (1 << DORn) | (1 << UPEn))

return -1;

/\* Filter the 9th bit, then return \*/

resh = (resh >> 1) & 0x01;

return ((resh << 8) | resl);

}

## Interrupts

## PWM

## BMP280 temperature, pressure & altitude sensor

## DHT11 temperatue & humidity sensor

## HC-05 Bluetooth module

HC-05 Bluetooth modules default UART settings are the same as Arduino’s default UART settings, therefore it requires no additional setup. It also doesn’t require any special actions to be interfaced with, since it works like a standard serial connection. The only required thing to use it is a serial monitor app installed on the Bluetooth client device.

## LCD Screen with an I2C converter

# Failure Mode and Effect Analysis

Some components of the device are of the highest importance – without them, the device cannot fulfil its basic function. However, failure of some other components does not render the whole device unusable. Some components’ importance is context-dependent. The table below presents items or functionalities together with their importance for the project’s operation.

|  |  |
| --- | --- |
| **Item/function** | **Importance** |
| Microcontroller | critical |
| Power supply | critical |
| Screen & I2C converter | low / critical |
| Button | low / critical |
| Bluetooth module | low / critical |
| Temperature and humidity sensor | high |
| Pressure sensor | high |
| LED | low |

The microcontroller and power supply are essential to device’s functioning. The microcontroller is responsible for the program logic and controlling the devices, therefore it cannot fail. Because the device does not have a battery power supply must be constantly available.

The low / critical importance stems from the fact that we can read data and change the currently displayed information in a few ways:

* Changing
  + Using a button
  + via Bluetooth using an app
* Displaying
  + On the LCD screen
  + via Bluetooth using an app

Therefore, one way of changing the displayed information can fail, as well as one way of changing the displayed information. However, if both fail, the device wouldn’t be able to function properly: worst case scenario is that nothing is displayed (Bluetooth & screen fail) or we’re stuck on a given mode (button and Bluetooth fail).

The sensors are of high importance since they provide all the data to be displayed. Without them, the data would be either unavailable or “frozen”, that is the reading would always be the same.

LED is of the lowest importance – it only serves as an indicator of one value and it does not affect other components in any way. Moreover, it can be replaced by just adding another mode that would display the reading normally indicated by the LED.

Power supply failure can be detected in a number of ways, for example the LED or screen may flicker. Power‐related faults may also appear when a short is caused by making improper electrical connections, such as connecting Vin pin to the ground. This may happen either by putting cables in the incorrect places, or dropping a piece of metal on the board.

A microcontroller failure would probably be easy to detect, as most likely it would be a complete failure of the chip.

Screen’s or I2C converter’s failure would be easily noticeable, since it would either have problems with backlight or displaying the content properly.

Button’s failure might result in fake presses or no reaction when pressed. That would be easy to notice since the screen would change by itself (fake presses) or stay the same even if the button was pressed.

Bluetooth module’s failure could have multiple symptoms, for example the device could disconnect spontaneously, the communication could stop working and so on.

Sensors could stop responding altogether or start giving wrong readings. The 2nd symptom could be hard to realise if the error would be small, unless we had other, reference sensors.

LED’s failure might not be so obvious, since it could also be a power supply failure. Nevertheless, if the LED started shining more dimly or stopped shining altogether it could be sign that it has been damaged.

All of the components could be easily replaced by just swapping them, since they’re connected to the breadboard. The microcontroller is socketed, so it too could be easily replaced. In case of power supply failure the whole component would have to be replaced or if the problem is with the board, the quickest way would be to replace it whole.

# References

* ATmega328P Datasheet (<http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega328PB-Automotive-Data-Sheet-40001980B.pdf>)
* DHT11 Datasheet (<https://www.mouser.com/ds/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf>)
* BMP280 Datasheet (<https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMP280-DS001.pdf>)
* HC-05 Datasheet (<http://www.electronicaestudio.com/docs/istd016A.pdf>)
* HD44780 Datasheet (<https://www.sparkfun.com/datasheets/LCD/HD44780.pdf>)
* Arduino reference materials:
  + Port manipulation: <https://www.arduino.cc/en/Reference/PortManipulation>
  + Digital pins use guide: <https://www.arduino.cc/en/Tutorial/DigitalPins>

# Other information

Due to the fact that the screen and Bluetooth module use a lot of energy, the screen may sometimes flicker. Moreover, the Bluetooth app may sometimes display wrong data. It has been confirmed that is a problem with the app itself, since it does not occur with other apps or when using USB connection,