Report 1
Hardware and Software architecture
INFO2055 - Embedded Systems Project

Antoine MALHERBE	-	s164138
Chloé PREUD'HOMME	-	s154552
Jérôme BAYAUX	-	s161630
Tom PIRON	-	s161415

Academic Year 2020-2021

1 Project objectives

Our project objective consists in building a environmental measurements device for a mushroom box. This mushroom box is an idea of the Pot'Ingé, a group of students and PhD students building a vegetable garden at the University of Liège. They want to be able to harvest mushrooms and we need to help monitor their culture. This requires monitoring several environment variables such as temperature, air humidity, light level and CO2 concentration. These will be measured through different sensors. In our process, we will first concentrate on the three first mentioned ones and add the gas measurements once the previous ones works because the sensor used for this last variable is more difficult to handle and quite expensive. All of the schema however include this sensor as it is part of the design.

Once data is collected, it will be stored on an external flash memory because internal memory is not sufficient for us to keep an interesting amount of data. Eventually, our goal is to allow someone to retrieve this data on its smartphone through a Bluetooth transfer. Once data is on the smartphone, it will be easily available for the client as it can be stored on an external server and visible on a web interface. However, our idea is to go step by step in our process and implements features one by one in order to be able to deliver intermediate results to the client.

2 Hardware

For the hardware part, a list of components is required to implement the electronic circuit and to answer the objectives of the project. A tradeoff between our requirements, the ability to insert the component in the circuit and the price was done. This list of components with their purchase links and prices can be found in Table 1. The price for each component were calculated including 21% VAT to know if the order is within the budget of 35€. Some out-of-budget components will also be used. Their prices and links can be found in Table 1 below the total price of the initial order.

Components	**Prices with 21% VAT**
Voltage regulator	0.7018 €
1 capacitor of 0.33μF for voltage regulator	0.0554 €
Diode P600D	0.0693 €
Temperature sensor	0.7744 €
Light sensor	8.9177 €
1 resistor 100kΩ for light sensor	0.1391 €
1 capacitor 33 pF for light sensor	0.3134 €
Humidity sensor	11.374 €
1 capacitor 0.22μF for power stability	0.1355 €
Flash memory	11.6523 €
2 resistors 4.7kΩ for gas sensor[1]	0.1216 €
Total to order	**34.2578 €**
Battery connector	2.3716 €
Bluetooth module	16.8553 €
Gas sensor	23.0626 €
Total	**76.5497 €**

Table 1: Components list

2.1 Schematics

On Figure 1, you can see the schematic of our hardware and the connection between all components. It should be noted that unless specified otherwise in the next section all connection to the PIC are interchangeable to any other I/O pin. The one selected here where just for drawing convenience.

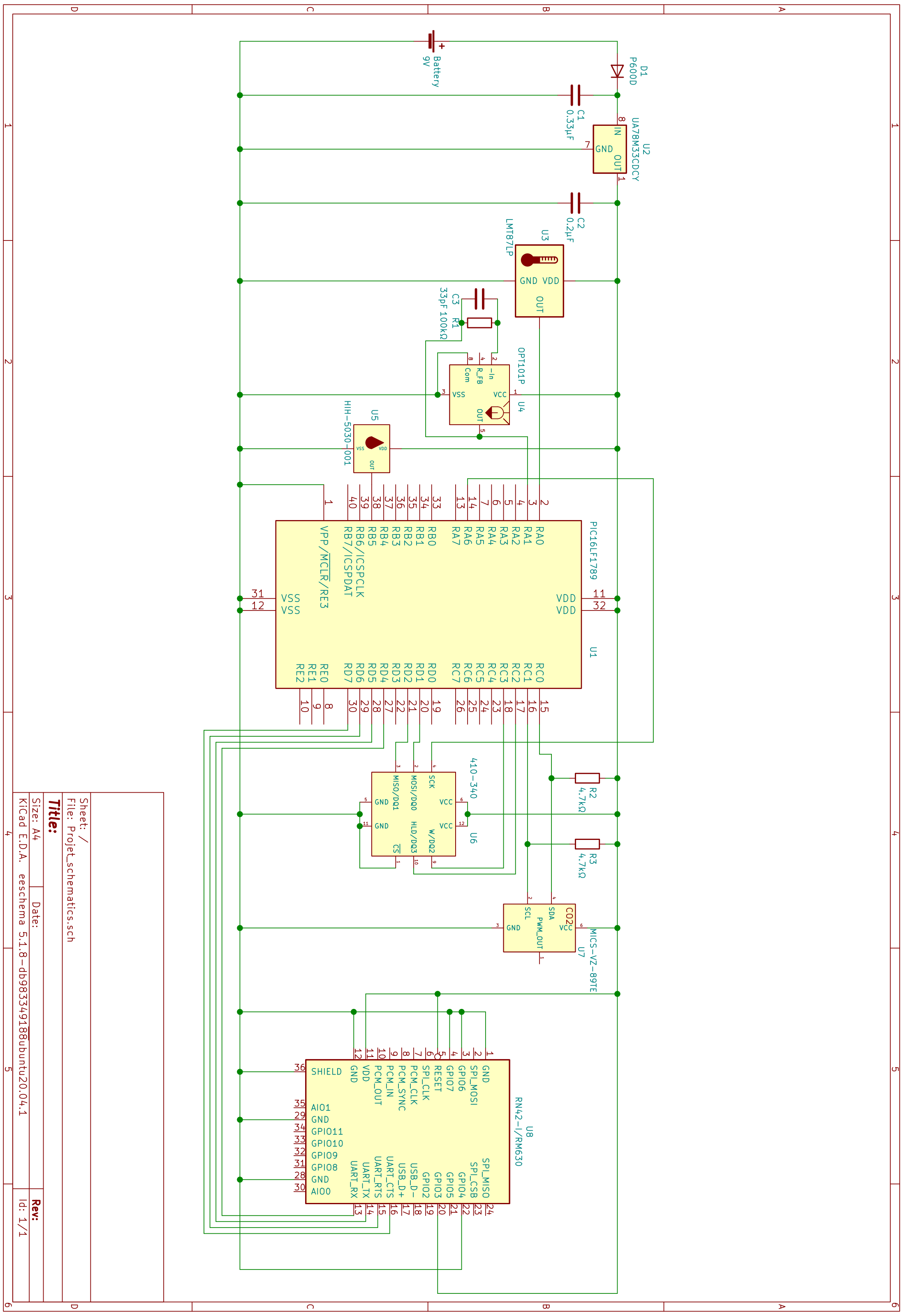


Figure 1: Hardware schematic

2.2 Description

The schematic of the hardware in Figure 1 is described from left to right.

- First of all we need a power supply. The energy source will be a 9V battery. A battery connector is necessary to connect the battery to the circuit.
  Then a diode `D1 P600D` is used to protect the circuit in case the polarity of the battery is reversed.
  Finally we use a linear voltage regulator `UA78M33CDCY` to bring to voltage to $3.3V$ which is the commonly accepted voltage by all the components. The choice of this particular component was driven by the fact that it has 3 pins (`IN`, `OUT`, `GND`) which make it easy to connect in the circuit. Two capacitors (C1=0.33μF and C2=0.2μF) are needed to use the component in the circuit. The first one is to stabilise the output of the battery and the second to absorb spikes in the power consumption primarily by the PIC, the Bluetooth module and the CO_2 sensor. Note that $C_2 = 0.2\mu F$ in the schematic, is actually two $0.1\mu F$ in parallel it was just easier to draw it as one capacity. This circuit can deliver an output current up to $500mA$, which is way above the expected consumption of the full system.

- To collect the temperature, a temperature sensor `LMT87LP` is used. It is an integrated circuit with an analog output voltage linearly and inversely proportional to the temperature. It has a precision range between −50 to 150°C. It can operate up to a 5.5 V power supply with a low power consumption of 5.4 μA. Moreover, its output is protected against short circuit. It has a gain of −13.6mV/°C. Its output pin is connected to a general I/O pin, `RA0`, of the PIC16LF1789 microcontroller.

- A circuit including a photodiode, `OPT101P` is used as a light sensor. The pin `-In`, connected to the negative inputs connected to the output to reduce the sensibility of the sensor. Those additional components allow to have a sensitivity of around $2.3\frac{V}{mW/cm^2}$ with a range between $0V$ and $2.15V$. This should allow us to measure light between total darkness and normal indoor artificial lightning. The output is the connected to a general I/O pin, `RA1`, of the PIC.

- To measure the level of humidity with an increased accuracy in the environment of the mushrooms, the humidity sensor `HIH-5030-001` depicted by a droplet on the schematic, was chosen because it can resist to multiple hazards like condensation, dust, dirt, oils and common environmental chemicals which is necessary in this particular application. Moreover, it has a near linear voltage output so it is possible to easily connect it to the input of a controller and it has a current draw of 200μA which is good for battery operated systems. In our circuit, its output is simply connected to a general I/O pin, `RB5`, of the PIC16LF1789 microcontroller.

- The PIC16LF1789 microcontroller is the central point of our circuit and will perform all software operation like reading values on sensors, storing them on an external memory module and eventually sending it via Bluetooth.

- The flash memory `410-340` is used to store collected data because the internal storage of the PIC is not sufficient for us to store an interesting amount of data. All of the communication pins are connected to the PIC via general I/O pins. The only pin that is not connected to the PIC, except for the power supply, is the $\overline{CS}$ which is connected to the ground, this pin is used to select a memory module when you have multiple ones. Has we only have one, we need it to be always selected. The signal *SCK* should be connected to an I/O output that can output a clock signal, if we want to transfer data as fast as possible.

- The gas sensor `MICS-VZ-89TE`. This component has I^2C outputs. Therefore, two of his outputs is connected to two lines `SDA` for data input and output and `SCL` for the I^2C clock of the I^2C bus. The lines of the bus are connected to the power supply with two pull up resistors of 4.7kΩ. The `SDA` and `SCL` lines of the bus are connected to Schmitt Trigger inputs with I^{2}C levels of the PIC16LF1789 microcontroller `RC3` and `RC4`. Moreover it is highly sensitive and can resist to shocks and vibrations.

- The bluetooth module `RN42/RN42N` is used to send data outside of our system. In particular, the idea is to send data on a smartphone that will be able to send it on an external server. It has a considerable series of pins that should be connected appropriately to the PIC16LF1789 microcontroller. In order to communicate with the microcontroller, the `UART RX` and `UART TX` should imperatively be connected to general purpose I/O pins of the microcontroller to receive and to transmit data. Other pins connections were chosen such as the `UART RTS` and `UART CTS` connected to general I/O pins of the microcontroller to add some transfer control possibilities. Moreover, the baud rate is set to low to be able to communicate data further. It can easily be set to high as well. Two other pins are used to set the bluetooth operation mode to slave mode, which mean that the module wait for a master device to connect to it.

The solution of using a Bluetooth module as buffer towards an external server is mainly driven by energy consumption and simplification of software. Indeed, Bluetooth may consume around 40 times less energy than WiFi[2] and the amount of data we need to transfer is not so big so, even if the transfer duration is longer with Bluetooth than with WiFi, it should not take hours to collect data. It is much simpler to use a bit of energy from the smartphone battery that can be refilled instead of changing the battery of our system more regularly. Considering simplification of software, it seems easier to be interrupted when a Bluetooth device is available and sends it data instead of being connected to a WiFi that needs to be configured and periodically sending data using it.

3 Software

3.1 Architecture

In order to choose which software architecture we will use, we simply looked for the simplest one that fits our needs in terms of latency. In our solution, we will have two types of tasks: collecting data and saving it.

To handle them, the very first possibility is to use a Round-Robin architecture. However, this model will not be sufficient for us because the time needed for our different tasks to operate is not similar at all. Indeed, retrieving data from sensors can be performed very quickly while writing in memory takes a lot of time. With Round-Robin architecture, we would have to wait for the write operation to be finished before making another measurement.

[2] `http://routeur-4g.com/consommation-denergie-wifi-vs-bluetooth/`

A better solution would be to be able to stop the writing to make some measurements. This can easily be achieved through interrupts. In particular, a Round-Robin architecture with interrupts perfectly suits our needs as latency is not a crucial point in our problem. Indeed, retrieving data from sensors do not need to be done right at the moment as what we measures do not vary a lot with time (temperature will not drop from 5° C in 1 second). In this architecture, the maximum response time is equal to the sum of execution time of other interrupts. As our different measurements will not take too long, this time is perfectly acceptable in our case. Moreover, as measured variables do not vary a lot over time, we would not need to perform measurements too often. This means that the writing task will have time to operate as it will not be interrupted too often. Above that, this interrupt system will also be very useful when we will add Bluetooth to our system as we will be able to interrupt our microcontroller to send data when a device is connected.

3.2 Tasks

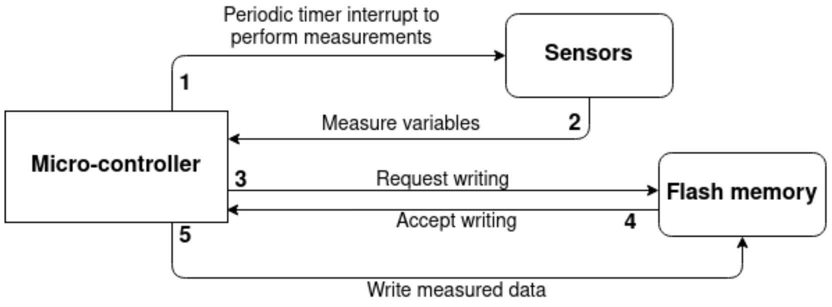
As said before, we basically have two big tasks to perform. First, we collect data by making some measurements through our sensors. Then, we will save this data on a flash memory or on an external device using Bluetooth. More precisely, our problem is divided into the following tasks.

- Collecting environmental data from sensors. This is a periodic task triggered by a timer. The exact value of this timer still need to be determined according to our client needs but we may for instance measure temperature every minute.
- Saving data to flash memory module. This task takes a long time to be fully executed but can be interrupted and continued after.
- Sending data to an external storage using Bluetooth. When a device is connected and requests a transaction, we have two subtasks: read data from external flash memory and send it through the Bluetooth module.

As explained previously in section 3.1, we use interrupts to react quickly to some important events. Here are the sources from where those can be triggered:

- Flash memory interrupts: those are used to know when the memory module is ready to operate, i.e. when it finished its current operation.
- Bluetooth module interrupts: those are used to deal with the Bluetooth feature. We have some interrupts similar to those in the flash memory as we need to send data to the external module but we also have special ones related to transactions (establishment, disconnection, ...)
- 1 Timer interrupt: this is used to trigger the execution of our single periodic task whose job is to collect data from sensors and store it in local memory.

The figure 2 here below is a simplified diagram explaining how modules interacts and how tasks will be performed in our solution.



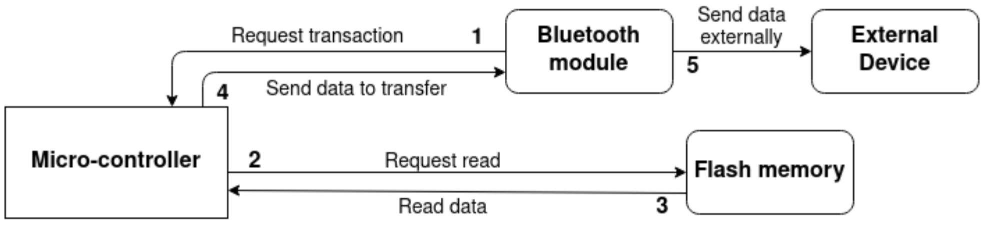


Figure 2: Interactions between modules

In this figure, we have two diagrams describing the two main processes in our solution.

The upper diagram describes how data is collected and written on the flash memory. This is triggered by a timer inside the micro-controller (1). Then, the micro-controller reads values on sensor output (2). After that, it checks whether the flash memory is ready to be used in write mode (i.e., if there is no concurrent read) (3, 4) and finally it sends data to the flash memory so that measurements are saved.

The lower diagram describes how data is transferred to an external device. First, the Bluetooth module detects that a device is connected and request a transaction (1). Then, the micro-controller tries to read data on the flash memory module (2, 3). When we have retrieved this data, we send it to the Bluetooth module (4) that will eventually send it to the external device.

3.3 Organisation

Here is the pseudo-code for our project software. Some notes about it can be found at te end of the section.

```
static volatile Bool: ready_to_collect = False, ready_2 = False,
                       flash_ready = True, bluetooth_transaction = False,
                       bluetooth_ready = True, bluetooth_in_transaction = False;

static volatile fifo: rx_flash;
```

```
!! Triggered by timer to collect data
interrupt void timer_collect() {
   ready_to_collect = True;
}

!! Triggered when flash memory finished its previous operation and
   becomes available again
interrupt void flash_ready() {
   flash_ready = True;
}

!! Triggered when flash memory has finished its read operation and
   requested data can be used
interrupt void flash_receive() {
   data = !! Read data from flash module;
   fifo_put(rx_flash, data);
}

!! Triggered when bluetooth module wants to start a transaction
interrupt void bluetooth_transaction() {
   bluetooth_transaction = True;
}

!! Triggered when bluetooth device is disconnected
interrupt void bluetooth_disconnect() {
   bluetooth_in_transaction = False;
}

!! Triggered when bluetooth module finished its previous operation and
   becomes available again
interrupt void bluetooth_ready() {
    bluetooth_ready = True;
}

void main() {

   !! Initialize interrupt vectors;
   static int request_data = 0;
   static fifo tx_flash;
   !! Initialize other global variables;

   for( ; ; ) {

      if(ready_to_collect) {
         !! Collect data from all sensors
         fifo_put(tx_flash, data);
         ready_to_collect = False;
      }

      if(fifo_content_size(tx_flash) > 0) {
         if(flash_ready) {
            data = fifo_get(tx_flash);
            !! Save data on flash memory (1)
            flash_ready = False;
         }
      }
      if(bluetooth_transaction and not bluetooth_in_transaction) { (2)
         !! Set up number of data wanted from flash module
         request_data += !! Number of data requested by bluetooth device;
         bluetooth_transaction = False;
         bluetooth_in_transaction = True;
      }

      if(request_data > 0) {
         if(flash_ready) {
            request_data -= 1;
            !! Read data from flash module
            flash_ready = False;
         }
      }

      if(bluetooth_in_transaction) {
         if(fifo_content_size(rx_flash) > 0) {
            if(bluetooth_ready) {
               !! Send data to bluetooth module
               bluetooth_ready = False;
            }
         }
         else {
            bluetooth_in_transaction = False;
         }
      }
   }
}

void fifo_put(fifo q, data)
{
    Bool interr_enabled;
    !! Do something;
    interr_enabled = disable(); (3)
    !! Critical section;
    if(interr_enabled)
        enable();
    !! Do something;
}
```

3.4 Organisation remarks

- (1): In our application, we do not need to disable interrupts when we send data to flash module as it is done with the serial example of the course. Indeed, even if we use the same registers to send data to the flash memory as the ones used to receive data from it because we will not try to retrieve and send data at the same time (e.g if we ask for some data, `flash_ready` will be set to `False` so we can not send anything until it switches to `True`, but then interrupt to read data will also be triggered so that we will save the data before attempting another operation).

- (2): As the operations executed by this task are very basic and quite short, we could be tempted to place them directly inside the interrupt that triggers the task and to remove this task from the main code. However, it is better to let this as is, because we should always avoid to put non-urgent operations inside interrupts.

- (3): We need to do a little trick here in order for `fifo_put` function to work correctly both inside interrupt code and inside main code. Here we assume that `disable()` returns true if interrupts were enabled in the first place. So, we will only execute `enable()` if they were enable before we disabled them, so if we are executing the function in the main code and not in an interrupt routine.
- This pseudo-code is inspired by the example about a serial filter that can be found in the theoretical course and also by some exercises from the practical lessons.
- This pseudo-code should be seen as an overview of our implementation, not as a complete and correct code that will be used as is. Indeed, some signals could be replaced by specific registers of modules, especially Bluetooth module that provides a pin that works like our `bluetooth_ready` signal. Moreover, we will have to deal with actual components that may have specific behaviour that may need additional signals.
- We assume all peering (establishment of a secured exchange channel) features related to Bluetooth are performed by the external Bluetooth module allowing us to simply know when a device request data, e.g. when we need to send data.