Report 2
Sensors and actuators validation
INFO2055 - Embedded Systems Project

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

Academic Year 2020-2021

1 Introduction

As a reminder, the aim of our project is to collect data about the environment in a mushroom box to help the Pot'Ingé monitoring their mushrooms culture. This is done by using sensors that will measure temperature, air humidity, light level and CO_2 concentration. The data are then stored on an external flash memory before being retrieved by a smartphone thanks to a Bluetooth module.

For now, the humidity and the temperature sensors have been tested independently from the whole system (see section 2 for details) to make sure they operate correctly and to understand how to use them. For the other components, we mainly checked their principles through their datasheets. We also tried to program the PIC micro-controller through the circuit of the first Lab. The photodiode could not be tested because it has not been received yet but the principle is very similar to the temperature and the humidity sensors. The gas sensor will be added at the end because it is not available until at least the 15th March and we preferred to focus on the other parts of our circuits first before looking for another components available sooner.

The flash memory requires imperatively a 3.3 V voltage regulator that has not arrived yet and not the 5V voltage regulator that is currently used. However, we made research about it to understand how to correctly connect it and how to use it. The Bluetooth module was not tested yet because it can only be inserted in the circuit by soldering pins and its working principle is also complex and need additional research to be fully understood before being actually tested in practice.

Some components have not been physically tested but we do not considered this as an issue as all the sensors used will interact in the same way with the micro-controller, except for the gas sensor. Therefore, if we are able to retrieve data from the temperature sensor and humidity sensor independently, retrieving data from the photo-diode will not be any more difficult. Once the data from the different sensors are collected, they will be stored in the flash memory. If we are able to correctly write and read a predefined byte into and from the memory, replacing this predefined data by the actual sensors data should not be difficult. In the same way, if we can transfer one byte through Bluetooth, we should be able to transfer anything. Therefore, we think that putting all components together will be less challenging than actually making them work.

Finally, all remarks given as feedback for the previous report have been taken into account. The hardware schematic has been therefore modified (see section 3) and additional remarks are given to answer those comments (see section 4).

2 First tests

2.1 Laboratory circuit and supply system

As the supply system of our circuit is almost the same as the one of the first laboratory, we started by reproducing this circuit at the R100 lab. Then, we programmed the micro-controller with the example code to see whether the led is correctly flashing. The circuit is represented in FIGURE 1.

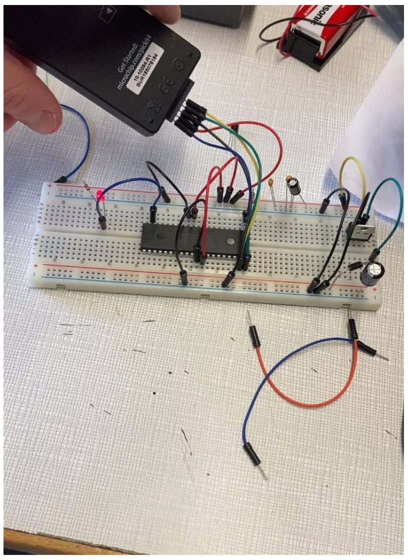


Figure 1: First laboratory session circuit

A voltage regulator with a 5V output is used in this case but for our project we need a 3.3V voltage regulator that has not been received yet. A PICkit4 has been used to program the microcontroller. We tested this circuit in two configurations, one with a 9V battery and one with the PIC powered by the computer itself.

2.2 Temperature sensor

In order to test the temperature and the humidity sensors, we reused the supply system to have a regulated voltage of 5V as input. Then, instead of retrieving a value on the microcontroller itself, we used a multimetre to directly measure the output voltage of the component. In order to test the temperature sensor, we reproduced the circuit given in the datasheet. An actual picture is given in FIGURE 2 while the mentioned circuit is shown in FIGURE 3.

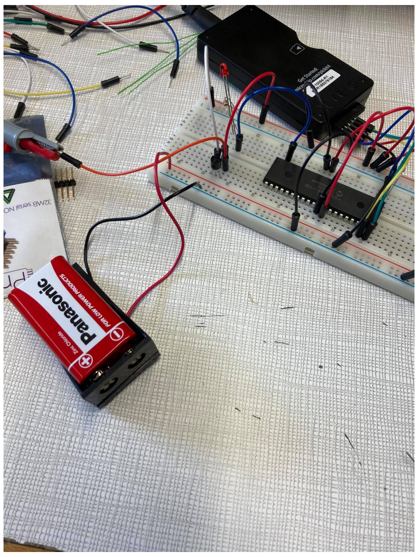


Figure 2: Test of the temperature sensor using the voltmeter

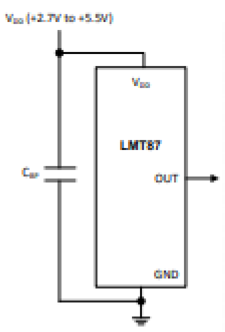


Figure 3: Circuit schematic of the temperature sensor

In our case, we got an output voltage of 2.330 V. To obtain the corresponding value in degree the following formula given in the datasheet was used:

$$V_{TEMP}(mV) = 2230.8mV - \left[13.582\frac{mV}{°C}\left(T - 30°C\right)\right] - \left[0.00433\frac{mV}{°C^2}\left(T - 30°C\right)^2\right]$$

In our future implementation, we will probably neglect the last term because it is very small and will be included in the error margin. In this case, the value in degree is around 22.5 degree Celsius with a margin error of ± 2 degrees.

2.3 Humidity sensor

The humidity sensor could not be placed directly on a breadboard because its pins were too small and aimed to be soldered. Therefore, we add a small wire to each pin in order to be able to use it. The actual soldering part was done by Professor Pascal Harmeling. The final result is shown in FIGURE 4.

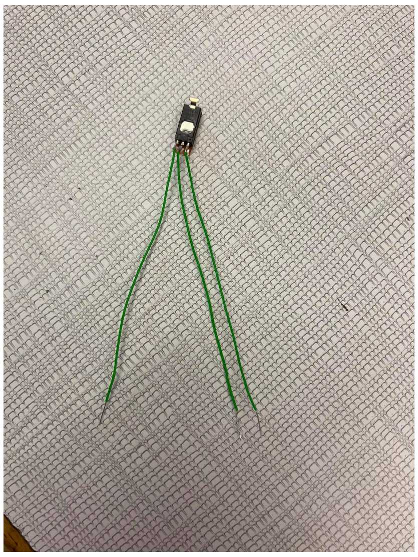


Figure 4: Humidity sensor with soldered wires

The difference in potential was measured again with a voltmeter between the ground and the output pin. We reproduced the circuit of the datasheet represented in FIGURE 5. In our case, we did not have a resistor of 65 kΩ. Therefore, we used one of 100 kΩ but this was not a problem as 65k was the load minimal value.

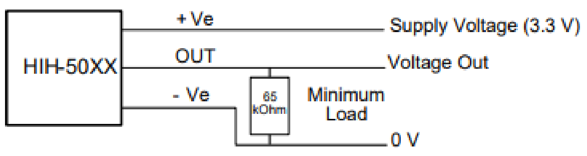


Figure 5: Circuit schematic of the humidity sensor

As the supply voltage of this component was supposed to be 3.3V, we used a voltage divider to convert our initial 5V input into a 3.3V one. To do so, we simply took three resistors with the same resistance and we took the voltage between the first and the second one like in FIGURE 6. Note that as resistors are not exactly the same, we did not have exactly 3.3 V but we had some closer to the components requirements and we could therefore test it.

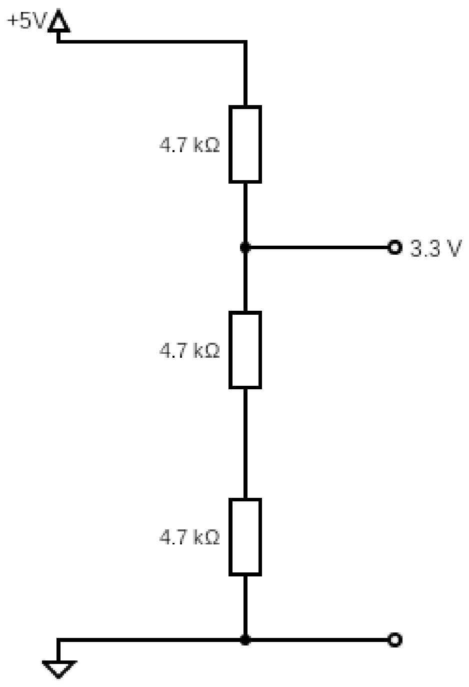


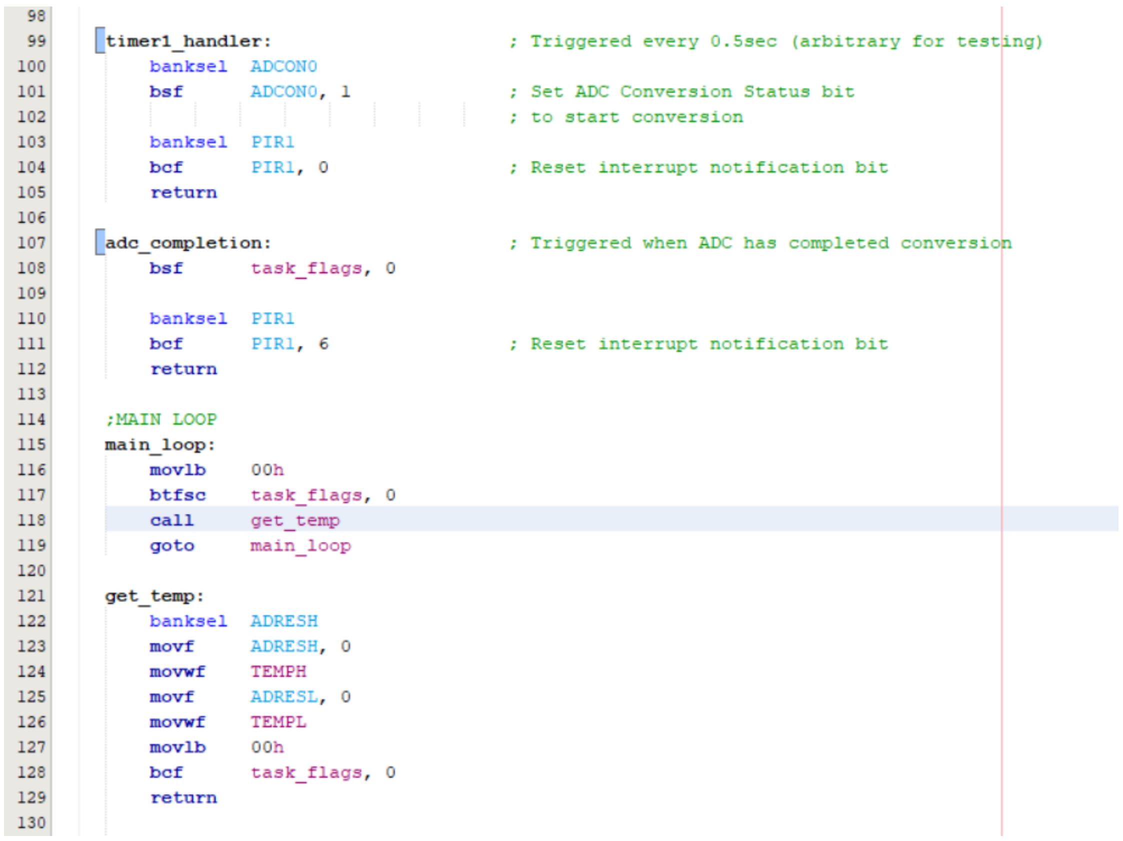
Figure 6: Voltage divider 5 to 3.3 V

Finally, like the temperature sensor, the output voltage can be converted thanks to a formula given in the datasheet into the corresponding percentage of air humidity at 25°C.

$$V_{\text{out}} = (V_{\text{supply}})(0.00636(\text{ sensor RH}) + 0.1515)$$

2.4 Test code for ADC

A first code, which is shown in Figure 7, has been written to test ADC function of the PIC. This code has been tested using the debugger tools from the MPLAB IDE in order to verify if everything worked as it should (i.e all pins are well configured and all registers are set to the correct values). The results obtained were the one we expected meaning that we managed to make the ADC module work correctly.



```
timer1_handler:                   ; Triggered every 0.5sec (arbitrary for testing)
    banksel  ADCON0
    bsf      ADCON0, 1            ; Set ADC Conversion Status bit
                                  ; to start conversion
    banksel  PIR1
    bcf      PIR1, 0              ; Reset interrupt notification bit
    return

adc_completion:                   ; Triggered when ADC has completed conversion
    bsf      task_flags, 0

    banksel  PIR1
    bcf      PIR1, 6              ; Reset interrupt notification bit
    return

;MAIN LOOP
main_loop:
    movlb    00h
    btfsc    task_flags, 0
    call     get_temp
    goto     main_loop

get_temp:
    banksel  ADRESH
    movf     ADRESH, 0
    movwf    TEMPH
    movf     ADRESL, 0
    movwf    TEMPL
    movlb    00h
    bcf      task_flags, 0
    return
```

Figure 7: Test code for ADC module

However, this code will not be sufficient to achieve what we need to do for the project. Indeed, we will need to use the ADC for different input pins in order to gather data from different sensors and this will require some adjustments that are detailed in subsection 4.4. Note that configuration of the right pin in input analog mode and of the ADCON0, ADCON1 and ADCON2 registers were also included in the code but the corresponding instructions are not shown in Figure 7.

3 Circuit

3.1 Schematics

In Figure 8, 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 ones selected here were just for drawing convenience.

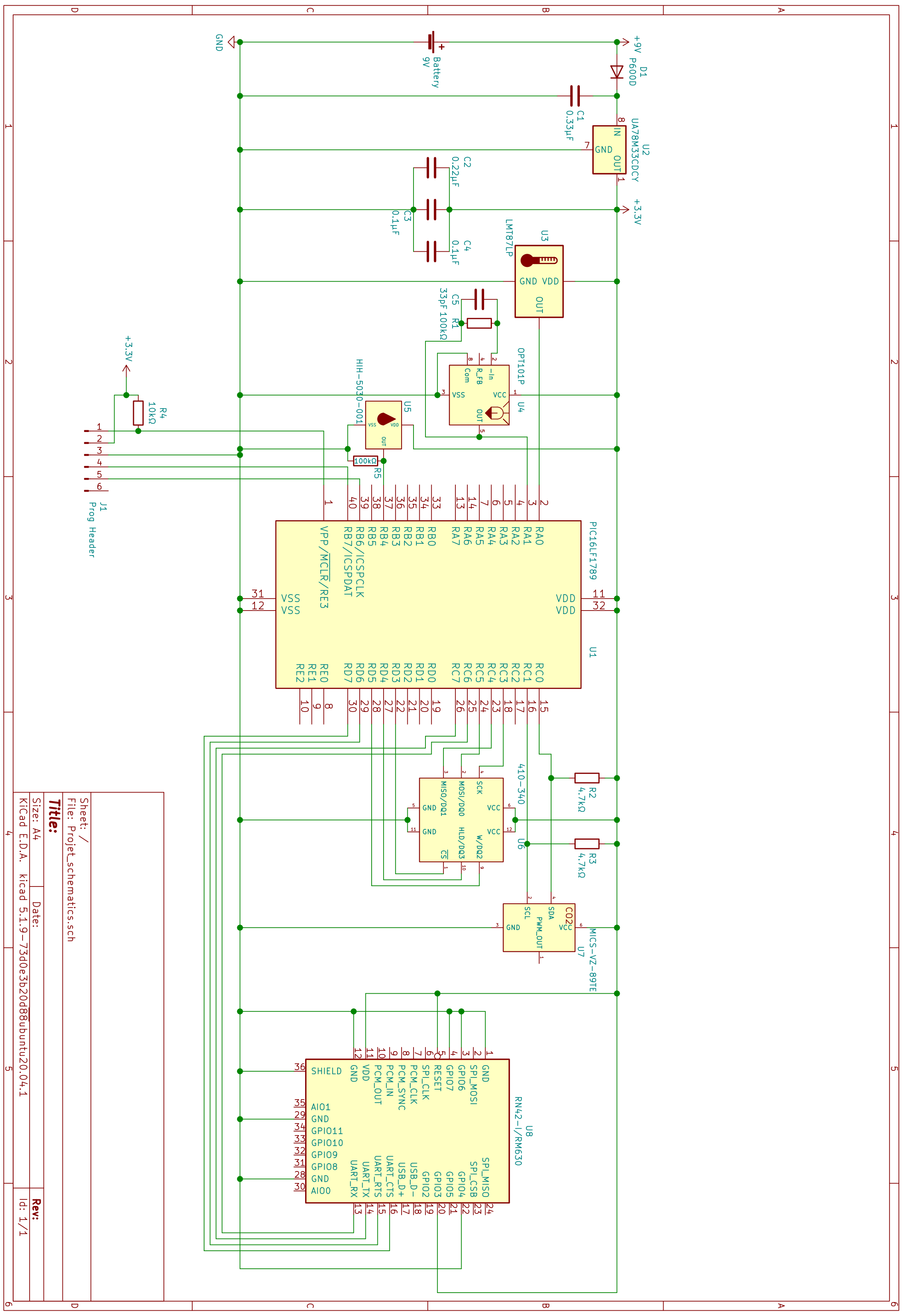


Figure 8: Hardware schematic

3.2 Changes

First, we forgot to add the programming header to our schematics. This is now done and we based our pin choice on the laboratory of Embedded Systems course. Then, we did not mentioned it but the output of the different sensors were connected to pins that can be selected as input for the Analog-to-Digital Converter (ADC) module so that we are able to convert an analog voltage to a corresponding binary representation.

As we did not fully understand how the flash module was working, we basically connected its pins to general input/output pins of the micro-controller. However, after some readings, we understood that it should be connected through an SPI bus. Therefore, MISO, MOSI and SCLK pins have been chosen to match SPI pins on the PIC micro-controller. Note that there was another possibility but those pins were already used by the programming header and we prefer to keep those pins separated from our circuit. Moreover, we previously put the Chip Select (CS) pin to the ground because we have only one slave and we therefore did not need to connect that to the PIC. However, we saw that by setting it to 1, and therefore 0 as input for the flash memory because the pin is inverted, we could put the flash memory in sleep mode. Therefore, we decided to connect it to a general I/O port of the micro-controller.

In the same way as the flash memory that was not connected to pins that manage SPI, the Bluetooth module was not correctly connected to pins that manage UART. Therefore, the UART_RX pin is connected to a TX pin of the PIC while the UART_TX pin is connected to a RX pin. Note that on the PIC those pins are defined for EUSART which is an advanced version of general UART.

Finally, we were asked to add some decoupling capacitor around our sensors. This was partially done. However we had gathered all decoupling capacitors into a single one ($C2$). $C2$ has 2 times the capacity advised for the PIC, the Bluetooth module, the flash memory and the temperature sensor. We decided to add two capacitors with the capacity advised for each component ($C3$ and $C4$). This should keep the supply voltage stable even if all the component are active at the same time.

4 Additional remarks

4.1 Flash memory module

The Flash memory module is interfaced through a SPI bus that works similarly to an I2C bus. Indeed, like in I2C, we have a Master and a Slave where the Master initiates the transaction and defines the transmission rate. Note that like in I2C, this does not mean that the Master is the sender. Then, we have a clock line SCLK that is used by the master to give the rate of the transmission in the same way the SCL line does it in I2C. One difference with I2C is that we can configure the clock polarity and phase meaning that we can choose whether data should be sampled on a rising or a falling edge (clock phase) and whether the clock is a pulse of 0 or a pulse of 1 (polarity). In our case, the flash module is configured such that it always samples data on a rising edge of the clock. Therefore, we will configure (CPOL, CPHA) to (0, 0) or (1, 1) on the PIC micro-controller. Then, we have two data lines that corresponds to the two directions in which data may flow. The MOSI line is used by the Master to send data to the Slave (Master-Out-Slave-In) while the

MISO line is used by the Master to receive data from the Slave (Master-In-Slave-Out). In our case, this means that we have one line (MOSI) to write data into the external memory and one line to read from it (MISO). At the PIC side, the MOSI line is the SDO line (SPI Data Output) while the MISO line is the SDI line (SPI Data Input). Finally, the addressing in SPI is simpler than in I2C. Indeed, each slave has its own control line with the master and is enabled when this line is set to High. In other words, the master have to output a control bit for each slave to select it. This is the Slave Select (SS) or the Chip Select line (CS). In our case, we have only one slave connected to the bus so this may be fixed to 1 but we will still connect it to a general output pin of the PIC so that we may add slaves in the future and we may also put the module in sleep mode when it is not used. An example of single-slave connection is given in figure 9 here below. Note that the module we have may be in QUAD MODE meaning that all exchanges are exchanges of four bits but this add complexity to our circuit without leading to an interesting improvement.

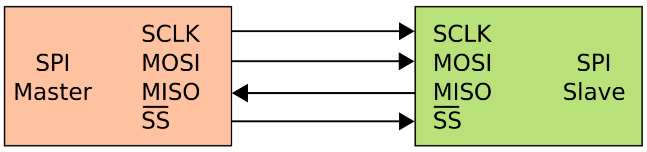


Figure 9: Basic SPI bus example

On our first software design, we were inspired by examples given in course where the module can trigger an interrupt when the write operation is finished. However, our module does not provide this feature. Therefore, we had to adapt our architecture. The idea of the interrupt was to be able to perform operation during the transmission. In our case, the system may take its time to respond because the application is not so time-sensitive as measured variables do not vary quickly over time. So, we may have a longer read/write operations that prevent other operations to be performed. Moreover, we can split data to exchange into small chunks so that operations can be performed between two chunks. We did not have time to make a working code to actually write or read something from the memory module but this is a point that will be very shortly addressed.

4.2 Timestamping measurements

In last feedback, we were suggested to timestamp our measurements. This is indeed a very interesting idea, in particular because it allows a better data manipulation after collection as we may know at what exact time some measurements were performed. However, we did not plan to have an external clock allowing us to know the current time. Nevertheless, we thought about a solution. The PIC micro-controller provides an internal clock signal that can be used like in the first lab to generate a clock signal of smaller frequency. Using this, we can count the number of cycles of a low-frequency clock so that we can add the current number of cycles to a measurements as timestamp. This may allow us to have a relative timestamp meaning that we will be able to know which measurement has been performed first and what time elapsed between two measurements. The final goal is to have an absolute timestamp so that we know at what exact time the measurement has been performed. To solve that, we simply need one starting point from our relative timestamp.

Indeed, it is sufficient to know at what time correspond the timestamp 0 to then create a mapping between timestamp and exact time as we know the time elapsed between two timestamp. An idea to get this is to reset the timestamp during a Bluetooth exchange. Indeed, the smartphone retrieving data knows the exact time and can therefore send it to an external server where all data is stored so that it knows at what time corresponds the timestamp 0 of the next batch of data. At first, we will not timestamp our measurements for the sake of simplicity of a first design. We will then see whether the proposed solution can be applied or not. If not, we may consider another software solution and if none is found, we may add an external component providing an absolute timestamp but all of this still needs additional research.

4.3 Sampling rate

The exact sampling rate has not been fully fixed yet but will probably be around a minute as temperature, light exposure, air humidity and CO2 concentration shouldn't vary quickly over time. However, this may be shorter if the client needs it (around 15 seconds) or longer if the practice shows us that those variables are very stable. This also might be something to discuss with our client and a mushroom expert and to test in practice.

4.4 ADC

In our project, we will need to use the ADC module with different input pins meaning that it will be necessary to change the channel selected as input for the ADC. Therefore, several adjustments will be made to the test code of Figure 7 to support our need. The skeleton of the program part that is concerned with the tasks that use the ADC module to collect data from the sensors is provided here under, along with the different explanations and remarks about this pseudo-code :

- At the moment, we only built code to gather data about temperature and humidity because we have not yet received the photo-diode required for our light level measurements. Nevertheless, the code structure we have chosen to measure temperature and humidity can be adapted easily to support collection of data from an additional sensor.
- When Timer1 overflows and triggers an interrupt it will trigger the first measurement task. Afterwards, tasks are triggered one at a time by the previous one. This mechanism is used to ease knowledge of which task has triggered the interrupt from the ADC.
- The job of each task is first to select the correct channel as input of the ADC. Then, it needs to wait for a given time that is called the "acquisition time" so that the charge holding capacitor of the ADC can be fully charged to the input channel voltage level. We decided to set this time to $5\mu s$ in order to be 100% sure that the charge holding capacitor is fully charged in any case. Finally, it will set the GO bit of ADCON0 register to make the ADC start the conversion.
- Currently, we store directly the value that the ADC outputs into some GPRs, without translating it into a voltage. This conversion will be more than likely implemented very soon. However, we do not know yet if we will save those voltages as they are into the flash memory module and further process them later (e.g in a smartphone

dedicated application after having retrieve the data via the bluetooth module) or if we try to translate those voltages into the right measurement unit (i.e degrees, % of humidity, etc.) before saving them.

```
interrupt void timer1() { -> Triggered by timer
   task_flags[0] = 1;
}

interrupt void adc_completion() { -> Triggered by ADC
   if (task_flags[1] == 1) {
      !! Store temperature data
   }
   else {
      !! Store humidity data
   }
}

void main() {

   for( ; ; ) {

      // adc_ready = GO/~DONE bit of ADCON0 register
      if (task_flags[0] == 1 && adc_ready) {
         !! Select ADC input of temp sensor
         !! Wait for acquisition time
         !! Set GO bit of ADCON0 register
         task_flags[1] = 1;
         task_flags[0] = 0;
      }

      // adc_ready = GO/~DONE bit of ADCON0 register
      if (task_flags[1] == 1 && adc_ready) {
         !! Select ADC input of hum sensor
         !! Wait for acquisition time
         !! Set GO bit of ADCON0 register
         task_flags[1] = 0;
      }
   }
}
```