Lab 6: Programming and Remote Control of a DISAL Arduino Xbee Sensor Node

This laboratory requires the following software (installed in the course computer rooms) and hardware (provided to each group for this exercise session only):

- Arduino IDE
- 2 Arduino Mega boards with Xbee shield
- 1 USB cable

The laboratory duration is approximately 3 hours. Although this laboratory is not graded, we encourage you to take your own personal notes as the final exam might leverage results acquired during this laboratory session. For any questions, please contact us at **sis-ta@groupes.epfl.ch**.

Information

In the following text you will find several exercises and questions.

- The notation **S** means that the question can be solved using only by uploading and executing code, possibly in combination with some other simple software or hardware manipulations
- The notation **Q** means that the question can be answered theoretically.
- The notation **I** means that the problem has to be solved by implementing, possibly debugging, and executing a piece of code and performing a simulation.
- The notation **B** means that the question is optional and should be answered if you have enough time at your disposal.

This lab uses the Arduino + Disal Shield kit, to revise basic concepts about this hardware tool, such as uploading code and using tools like serial plotter and monitor, please refer to Lab 5.

Note: If you experience issues with the connection of the hardware, resetting the connection by disconnecting and re-connecting components often helps.

Part 1: Control remote sensor

In this part of the lab, we will expand on some concepts we touched in Lab 5. You might remember that we set up a transmitter to send the data to a receiver, which in turn was sending the data to the computer. Today we want to control which type of data we receive from the transmitter. In this scenario, a sensor node, called base station, will be connected to the computer and ask another sensor node, referred to as remote sensor, to send different data depending on what the user is requesting. For this part, recall what we saw in the last lab about the `PAN ID` of the network and remember to **change the PAN ID by substituting the number 2019 with the number of your computer** in the computer room (e.g., if computer number=3, `Serial3.print("ATID 3\r");`). Do this for **both the base station and the remote sensor** code.

1. **(I):** Select one of the nodes to use as a base station, and one as a remote sensor. Open the code in *part1/basestation* and complete the C code to modify the variable *command* based on which button is pressed. Depending on which button is pressed, the user will ask for temperature, humidity or light data. Specifically:

    - If BTN_T is pressed, command= "T"
    - If BTN_H is pressed, command = "H"
    - If BTN_L is pressed, command = "L"

2. **(S):** Now look at the whole code. Can you explain exactly what is happening? When is the command for new data sent to the remote sensor? Upload the code to your selected base station board.

3. **(I):** Open the code in *part1/remotesensor,* understand it, and complete the C code to send back the remaining two types of data when requested by the receiver. Be careful about maintaining the format "LETTER DATA" (where letter is L, H, T and data is the respective data) for all the data you send back. Upload your code to the remote sensor node. *Hint: Use the functions getHumidity() and getTemperature().*

Test your code by connecting the base station board to the computer, turning on the remote sensor (by battery power), opening the Arduino Serial Monitor and checking that, when buttons are pressed, the type of data that gets sent changes accordingly.

4. **(S):** With the boards turned on, run the scripts *saveSerialData.py* and *cleanData.py* located in the *python* folder, similarly to what you did in the last lab (you might have to rerun the command *pip3 install pyserial*). Press the buttons while you save the data to collect different types of data.

5. **(I):** Open the *plotter.m* script contained in the *matlab* folder. Write MATLAB code to compute the average, min and max of the three quantities (light, humidity and temperature) using the data that you saved.

6. **(Q):** You just worked on a more complex data sharing network. Can you draw a schematic of the data sharing and the data processing that you just implemented?

Part 2: Interrupts

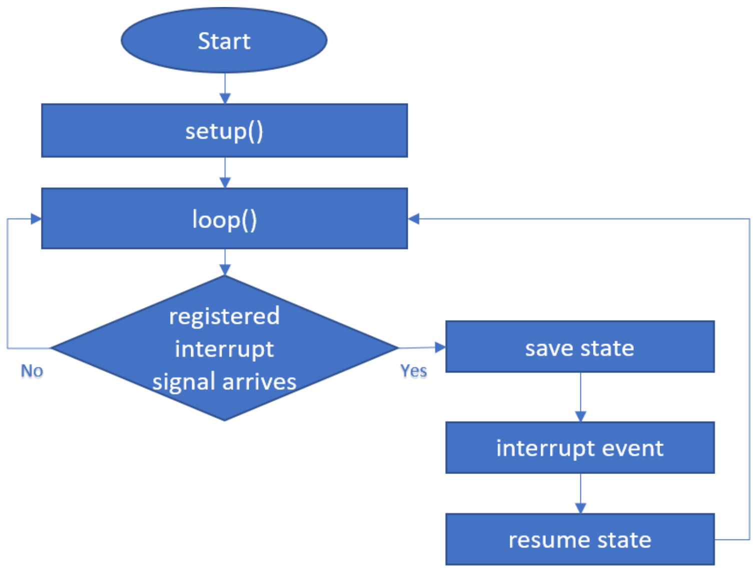
In this part of the lab, we will study interrupts. To understand what interrupts are and why they are useful, let's start by imagining a scenario where our base station node is deployed for field experiments, and it samples temperature and humidity data with a frequency of 0.1Hz. To save energy, the program updates the values of temperature and humidity on the screen every 10 seconds and waits in between samples. However, the field operators want to use the same device to record the number of falcons that they see in the field and they want to do so by pressing a button on the board every time they see a falcon. To help them, we will write code that increases the value of a variable to keep track of how many times the button was pressed.

7. **(I):** Go to the folder *part2/environmentalmonitoring* and look at the code. Complete the C code to keep track of the number of times the button is pressed. Upload it to the base station node. *Hint: Adding one to the value of the counting variable can be done efficiently with the line button_press++;*

8. **(S):** Test your code by pressing the button closer to the screen. What is happening? Why?

In this example, we can see that this code is not effective in keeping track of the number of times the button was pressed. The button press can only be captured when the event happens during the time slot every 10s when the system checks the button state, otherwise it is ignored. In real applications this is not acceptable since non-responsiveness when certain events happen could severely damage our equipment or put people in danger.

Interrupts are used to alert the processor that an event that needs attention occurred, so that a timely response can be activated. In our example, the interrupt condition (pressing a button) should alert the processor of an emergency event and trigger a response by executing a function called Interrupt Service Routine (ISR). ISRs are special functions initiated by interrupts. The body of these functions is usually kept very short to not halt the normal operations of the processor for too long. Before executing the ISR, the processor saves its state, ensuring that, once the ISR is executed, it can resume normal operation. The workflow of the system with an interrupt event is as follows:



To activate interrupts on a pin, we need to know which group (port) it belongs to and activate that group. The group and pin definition are different for different embedded platforms. Go to the folder *part2/interrupt* and look at the code. In the setup, we configure the interrupt on digital pin 13 by writing to some registers of the processor. Registers are temporary storage place inside the CPU and are used for a variety of purposes. In our case, we use the function `cli()` to clear all the register. We then indicate that we want digital pin 13 (which is connected to our "UP" button) to be configured as an interrupt. We first define the ports that will trigger interrupts by using the PCICR register and then set the pin that will trigger interrupts by using the PCMSK register. Afterwards we enable the interrupts by calling the function `sei()`. Understanding the specifics of how to set registers is outside the scope of this course.

9. **(I)**: In *part2/interrupt,* write C code to increase the value of the variable “value” in the allocated space. Upload it to the base station and test your code by opening the serial monitor and pressing the button closer to the shield’s screen to increase the value of the printed variable. How does the value increase? Why?

10. **(S):** Looking at the code, do you notice something unusual about the usage of the function `ISR(PCINT0_vect)`? Can you guess what is happening?

The function `ISR(PCINT0_vect)` contains the interrupt service routine. It is important to know that its name and arguments cannot be changed, since they refer exactly to the ISR for the interrupt we enabled according to the microprocessor code. Now we will improve the environmental monitoring code by using interrupts.

11. **(I):** Go to the folder *part2/monitoringwithinterrupts* and complete the C code to increase the value of the variable *`button_press`* using an interrupt. Check that your code is working by uploading it to the base station. When does the number of button presses get updated on the screen? Why? Does it correspond to the number of times you pressed the button? If not, can you explain why and can you fix your code to display exactly the number of times the button was pressed?

What we just programmed is usually referred to as a hardware interrupt, since the ISR is triggered when a condition related to the state of the hardware changes. Hardware interrupts are very commonly used in real embedded systems. You are using interrupts all the time! Whenever you move or click on your mouse, an interrupt is generated and the Central Processing Unit (CPU) of your computer knows that it has to deal with your request with high priority.

12. **(Q):** Reflect on the lab so far. How could the code in part1 be improved with the knowledge you have now on interrupts?

Part 3: Energy-aware communication

In this part of the lab, you will put together several elements that you have learnt so far to create a sensing station that is energy-aware. This means that your node will transmit information only in some occasions to efficiently use the power of the battery.

You are going to recreate this situation: your node is monitoring a greenhouse with several plants that need stable temperature to thrive. In normal operation mode, your system sends temperature data with a frequency of 1 Hz. However, when the temperature in greater than 28 degrees Celsius, you want to send data at 10 Hz to improve the sensing and have more information about this critical condition.

Open the file *energyawaretransmitter.ino* in the Part 3 folder. You will only need to use the base station node in this problem, as the wireless communication with the remote sensor is not needed.

13. **(I)**: Implement the scenario above by detecting when the temperature reaches a certain stage and changing the value of the frequency of data transmission accordingly. Upload your code on the node, then connect the base station to the computer and open the serial plotter. Test your implementation by placing a finger on the temperature sensor to increase its temperature.

IMPORTANT: What to do before handing the board back to the TAs

Before handing the boards back to a TA, go to the folder *maintenance* and upload the code you find inside on both boards. This code reads the battery voltage and prints it on the OLED screen. This way, we can quickly turn the boards on and see if they need to be recharged. Thank you for your help!