Lab 1: Transmitting information

1 Introduction to the Arduino labs

When we discuss data communication technology, our base is the layered OSI reference model. The lowest layer (L1) describes the physical characteristics of a link, while the link layer (L2) handles framing, addressing, error detection and in some cases error correction e.g. Automatic Repeat Request (ARQ). The objective of the four labs is to study some of the aspects of these two layers in a wireless environment. The communication protocol devised for the labs is based on Pure ALOHA[1].

Another, equally important, objective is to give an insight on the importance of understanding and following definitions and specifications given for a project. That implementation of for example a data communication protocol is crucial for co-operation with other implementations of the same protocol. In our example, you have the full control of the implementation of one of the nodes, which of course gives you the freedom of implementation of the inner workings of the node. But for this node to co-operate with the other nodes in the lab network, you must follow the specifications.

The reason for these two nodes to communicate with each other is to be able to from one node remotely set which Light Emitting Diode (LED) to illuminate on the other node. The *Master Node* is the node whose LEDs are controlled, while the *Development Node* acts as a remote control. Both nodes have three different-coloured LEDs and a button. While the button on the *Development Node* is pressed, the LEDs light in sequence. When the button is released, the ID of the then lit LED should be transmitted to the *Master Node*, where the same coloured LED should illuminate. Upon successful transmission, the *Development Node* should return to waiting for a response from the *Master Node*. Similarly, after addressing the instructions in the received frame and sending an ACK frame, the *Master Node* returns to waiting for another frame.

The nodes have limited memory, computational, and Input/Output (I/O) resources. This imposes constraints on the speed of communication, redundancy, and complexity of the application. The nodes are for example single threaded. It is therefore non-trivial to run concurrent processes such as simultaneous transmission and reception on the device. These constraints have to be dealt with in your implementation.

The environment is described in the *Reference Manual*[2]. The *Reference Manual*[2] contains communication protocol specifications, technical information regarding the hardware and the Arduino software used in the nodes, library documentation and more.

In the Arduino context, a program is known as a *sketch*. Therefore we will use that word instead of program in the documentation.

2 Learning outcomes

After performing the first of the four labs, you should have gained understanding of

- protocol definitions and state transition graphs, and how to implement them.
- how to compose a frame and how to transmit it taking into consideration bit rate and sample time.

3 On the use of Generative Artificial Intelligence (GAI)

At LTH, students are expected to learn how to use and evaluate the results of GAI tools relevant to their education. If you or your lab group decides to use GAI in any form, this should be specified. Direct copying of GAI results is considered quoting. If GAI results are used as inspiration to formulate your own code, it is treated as a reference. In both cases, the source must be stated, and the question or prompt given to the GAI tool should be included in the citation. Rules for copying and referencing (even when getting help from a friend) still apply.

4 Lab tasks

The first two labs are about the implementation of the necessary Physical Layer (L1) mechanisms to enable two units to communicate with each other over an Infra-red (IR) link. In this lab you will

- Implement the necessary functionality in the *Development Node* so that it is able to transmit a frame to the static and predictable *Master Node*.
- Convey an action from the *Development Node*, the remote control, to the *Master Node*. The *Master Node* should then actuate the action.

These two objectives have been broken down into tasks which you should complete during the first lab. To your help, a sketch skeleton and a library of variables and methods are prepared for you. The skeleton is built around a state machine, which is implemented as a `switch` statement and associated `case` clauses, see the listing of `skeleton5.ino` in the *Reference Manual*. The states and the state transitions described in the following, refers to this state machine implementation.

5 Some tips before you start

The solutions to all three tasks have a well defined start and end. But all tasks are implemented in the `loop()` function of the sketch. To halt the execution after a final state, go to the state HALT.

The protocol states that the frame is 32 bits which fits nicely into a `long` integer. Also, the Start Frame Delimiter (SFD) and the Cyclic Redundancy Check (CRC) field are both 8 bits, which is the size of a `byte`. Bit manipulation in C/C++ is a bit tricky so study bit operation section in the *Reference Manual*. The bit manipulation needed in these labs are mainly shifting in a bit in a buffer and reading the most or least significant bit in a byte or integer.

In all tasks, the payload field is the only field of the Data Link Layer (L2) frame sent by the *Development Node* that is relevant. All other frame fields can be set to 0 as they will not be taken into account.

There is often a need to print out a frame's content. The `print_frame()` method is available in both the `Transmit` and `Receive` classes, see the *Reference Manual*.

Use the layer interfaces: As interface between the application and L2 layer, you are adviced to use the library array `message[]`. As interface between the L2 and L1 layers the library array `frame[]` should be used. These arrays are declared in the classes `Transmit` and `Receive`, see the *Reference Manual* for interface specifications.

Constants vs. variables: Constants can be used when assigning a value to a variable, but not the other way round. If the compiler expects a variable where it sees a constant, it will issue a message of warning. The compiled code will perform as expected.

Declare functions: From the layered reference model for data communication, we learn that "divide and conquer" is a valuable modus. Breaking down a task into smaller, well-defined subtasks with the objective of solving one part of the bigger problem eases the process. It is also easier to read the code and understand what is done where. Each non trivial subtask that is identified is easier to construct and manage if it is declared as a function or a sub-routine. Write the functions at the end of the sketch.

Use the switch/case construction wisely in combination with functions! Call functions, that do the job, from the case clauses. Do not write your full code inside the case clauses. Do not declare variables inside the switch/case construction.

Constants as parameters in function calls: If you for example declare a function for the L1 transmitter like

```
void l1_send(byte l2frame[], int framelen)
```

and call that function with `l1_send(test_frame, LEN_FRAME);` you will get a warning message. This is because `test_frame[]` is a constant. But if you change the declaration to

```
void l1_send(const byte l2frame[], int framelen)
```

there will be no warning, independent of the parameter is a constant array or a normal variable array.

6 Task 1

The first task is to allow for an operator to select one of the three user LED's to later be used as the message to send to the *Master Node*.

6.1 Hints

- This is a part of the application, in which state should this therefore be done?
- The user LEDs are a part of the shield, can you find a suitable method in the shield class to use?
- Print the selected LED number on the *Serial Monitor*. Check that the print out on the *Serial Monitor* is the expected when compared to Table 3.3 in the *Reference Manual*.
- Make the HALT state the next state.

7 Task 2

In this task you should send the content of a test frame to the *Master Node*. One of the *Master Node*'s LEDs should light up.

7.1 Hints

- It is suggested to write the code for the 'sender' as a function outside of the state machine. Declare the function for example as `void l1_send(unsigned long frame, int framelen)` and call it with `l1_send(tx.frame, LEN_FRAME);`. It is now easy to change the frame to send from `testframe` to your own built frame in the next task.
- Sending bits as pulses is an L1 task. In which state should this therefore be done?
- Remember what the sample time is, use the library constant when referring to it in the code.
- The task is to send the preamble, the SFD and finally the frame bits in one sequence of pulses using the transmit IR LED, defined as pin `PIN_TX`. Check the global constants in the *Reference Manual* for the preamble and SFD and use these when sending.

- Read up on bitwise operations in the *Reference Manual*, the operations and examples described here will be useful.
- Check the *Reference Manual* for the predefined `testframe`, which is a fully functional L2 frame. This long integer should be seen as the interface between the layers L2 and L1. It is declared as `const unsigned long`, i.e. it does not include the preamble and SFD.

8 Task 3

You should now be ready to combine the functionality from task 1 and task 2. This task is to first select a LED, put the corresponding message into a frame and then send that frame to the *Master Node*. The selected LED should light up on the *Master Node*.

8.1 Hints

- Task 3 is about filling an L2 frame with appropriate data before sending it. If you have created code for task 2 wisely, this task is limited to just filling the L2 frame and use that as the interface buffer instead of the test frame.
- First, you need to put your selected led from task 1 in the interface variable between the application layer and L2. As this task is about transmitting, you might find a useful variable in this class.
- At L2, you need to create your frame before sending it to the *Master Node*, in which state should you do this?
- In this state, fill the interface between L2 and L1, i.e. the frame, with the individual L2 frame variables. Do this by first setting the individual frame fields to the correct values and then generate the frame. Variables and methods in the Frame and Transmit classes might come in handy for this task. Remember that the Transmit class inherits from the Frame class.
- Use your generated frame as the interface between L2 and L1 and transmit this frame instead of the test frame from task 2.

References

[1] Alohanet, pure aloha. `https://en.wikipedia.org/wiki/ALOHAnet#Pure_ALOHA`. Last visited 2023-01-09.

[2] Data Communication Lab: Reference Manual, 2023, 2024.