Practical sessions

The objective of this practical work is to implement synchronization protocols and ranging protocols using the properties of time stamping of UWB signals.

Table of Contents

1. First stage
   1.1. Resources
   1.2. Check the working environment
2. Getting started with the UWB transceiver
   2.1. Getting started with sample sketches
   2.2. Design of a first simple protocol
3. MAC synchronization
   3.1. Implementation of a star synchronization
   3.2. Implementation of a SISP synchronization
4. Ranging synchronization
   4.1. Sketches examples
   4.2. 2M-TWR implementation
   4.3. N-TWR implementation
   4.4. BB-TWR implementation

---

1. First stage

1.1. Resources

Table 1. Resources available

Resource	URL
DecaDuino library	https://www.irit.fr/~Adrien.Van-Den-Bossche/decaduino/
DecaWiNo's page	https://wino.cc/decawino/
SiSP protocol	https://www.irit.fr/~Adrien.Van-Den-Bossche/sisp.pdf

1.2. Check the working environment

- Hardware: each student has a PC, a DecaWiNo card, a USB cable, an oscilloscope and a connection to the oscilloscope.
- Software: Arduino environment is used for all of these exercises. On the workstation, check for the presence of:
  - Arduino Software
  - TeensyDuino *add-on*: `File > Examples > Teensy`
  - DecaDuino library: `File > Examples > DecaDuino`

If one of these points is not verified, proceed with the installation before continuing. The website indicated at the beginning of the document contains the information needed to proceed.

2. Getting started with the UWB transceiver

2.1. Getting started with sample sketches

The objective of this section is to get familiar with the PLME-SAP.

DecaDuino library includes several sample sketches available in `File > Examples > DecaDuino`.

1. Open the sketch `DecaDuino Sender` and the sketch `DecaDuino Receiver Sniffer`, list the used primitives and understand their purpose using the documentation of DecaDuino.

2. Note the lack of addressing in the messages, and modify the sketches in order to set up a logical isolation between pairs of nodes: each pair must only receive the messages that concern it.

2.2. Design of a first simple protocol

The objective of this part is to code and test a first simple protocol of type `DATA + ACK`*.*

1. Implement a simple DATA + ACK with configurable `timeout` and `retries` values.
2. View transmission times, time interval between frame, timeout, etc. using the oscilloscope.

To view the times, program the MCU to generate high / low states on one or two GPIOs and use an oscilloscope to view the signals.

Make sure to systematically disable reception before requesting the sending of a message.

3. MAC synchronization

3.1. Implementation of a star synchronization

The objective of this section is to use the PLME-SAP and the time stamp primitives to implement a first simple star synchronization protocol.

The DW1000 transceiver present on the DecaWiNo allows very precise stamping of messages. For this, DecaDuino provides the developer with several primitives. In this exercise, we will use them to implement a first star synchronization protocol, as illustrated in the following figure.

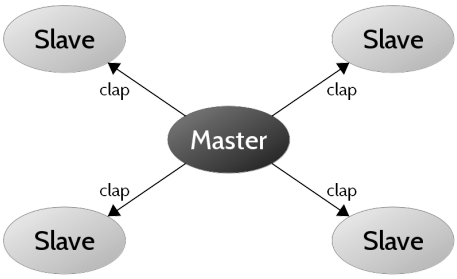

Figure 1. Star synchronization

Regularly, the *Master* node broadcasts a `clap` message indicating the time at which it sends the clap. The *Slave* nodes receive this message, calculate their *offset* with the master node and apply it on their local time.

1. Identify the needed primitives
2. Design the protocol :
   a. Propose the sequence of messages and their format,
   b. For each type of node (master, slave), specify the local algorithms,
   c. Propose an automata.
3. Implement and test

To view the quality of the synchronization, program the MCU to generate an edge on a GPIO and use an oscilloscope to view this difference on several nodes.

3.2. Implementation of a SISP synchronization

The objective of this section is to use the PLME-SAP and the time stamp primitives to implement the SiSP protocol.

In this exercise, we will implement a second synchronization protocol. This time, there is no longer any hierarchy between the participating nodes, as illustrated in the following figure.

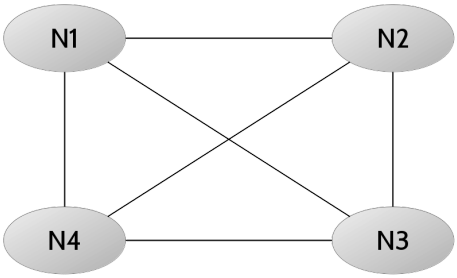

Figure 2. Mesh synchronization without hierarchy

Here, each node regularly broadcasts a `clap` message, but synchronization will be obtained by several consensus of the participating nodes, according to the principle of the SiSP protocol.

1. Identify the needed primitives
2. Design the protocol :
   a. Propose the sequence of messages and their format,
   b. For each type of node (master, slave), specify the local algorithms,
   c. Propose an automata.
3. Implement and test

Once again, to view the quality of the synchronization, program the MCU to generate an edge on a GPIO and use an oscilloscope to view this difference on several DecaWiNos. It is also possible to use the RGB LED available on the DecaWiNo to visualize the synchronization consensus.

4. Ranging synchronization

The objective of this section is to use PLME-SAP and time stamp primitives to implement ranging protocols.

Thanks to the very precise stamping of the messages on transmission and reception (15 ps), it is possible to use the transmitted and received messages with the DWM1000 to measure the _ radio time of flight_ and thus obtain a relatively precise evaluation of the distance separating the two nodes. This is the *ranging* process.

4.1. Sketches examples

The DecaDuino library includes several sketches of examples implementing in *ranging* protocols. They are available in `File> Examples> DecaDuino`.

- Using the documentation available online (Documentation of DecaDuino and tutorial entitled "*Using DecaWiNo and ranging protocols*"), get started with the different example sketches and understand the functioning of the Two-Way Ranging (TWR) and Symmetric protocols Double Sided Two-Way Ranging (SDS-TWR).
- Do some tests. Note the lack of addressing in the messages and modify the sketches such that logically isolate the communication between pairs of `DecaWiNos`.
- Observing in practice the effect of clock drift (*skew*) on the precision of *ranging* in TWR.

4.2. 2M-TWR implementation

The objective of this section is to propose an implementation of the Two-Messages TWR protocol (2M-TWR).

The 2M-TWR protocol (see course) is an improvement of the TWR protocol with a reduction to two messages. Stamps t2 and t3 are sent in the `ACK` message, which saves the last message as illustrated in the following figure.

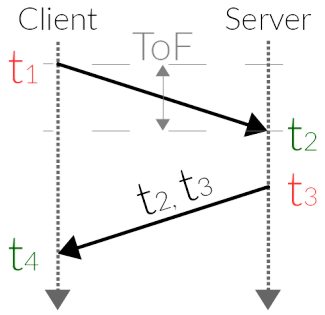

Figure 3. TWR and 2M-TWR sequence diagrams

Proceed to the design, implementation and testing of the protocol.

1. Identify the needed primitives
2. Design the protocol :
   a. Propose the sequence of messages and their format,
   b. For each type of node (master, slave), specify the local algorithms,
   c. Propose an automata.
3. Implement and test

4.3. N-TWR implementation

The objective of this section is to propose an implementation of the N-TWR protocol.

The N-TWR protocol (see course) is an improvement of the 2M-TWR protocol where several anchors (servers) can be requested by the mobile (client) with a single message. Stamps t2 and t3 of each anchor are sent in the `ACK` messages.

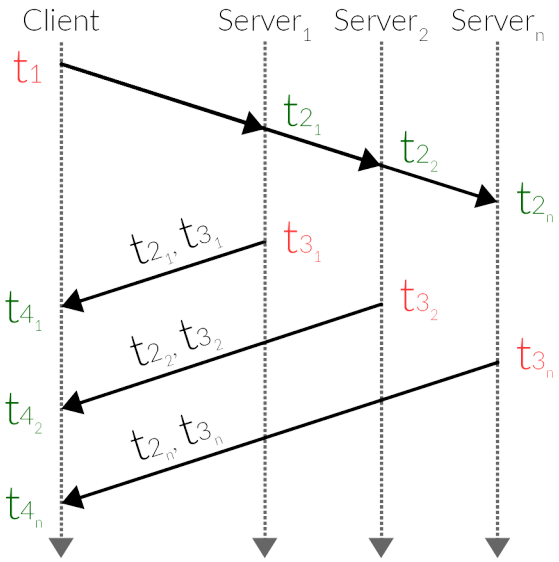

Figure 4. BB-TWR sequence diagrams for three triangle nodes

Proceed to the design, implementation and testing of the protocol.

1. Identify the needed primitives
2. Design the protocol :
   a. Propose the sequence of messages and their format,
   b. For each type of node (master, slave), specify the local algorithms,
   c. Propose an automata.
3. Implement and test

4.4. BB-TWR implementation

The objective of this section is to propose an implementation of the Broadcast-Based TWR protocol (BB-TWR).

The BB-TWR protocol (see course) is a broadcast version (*broadcast*) and generalized to *n* nodes of the 2M-TWR protocol, as illustrated in the following figure.

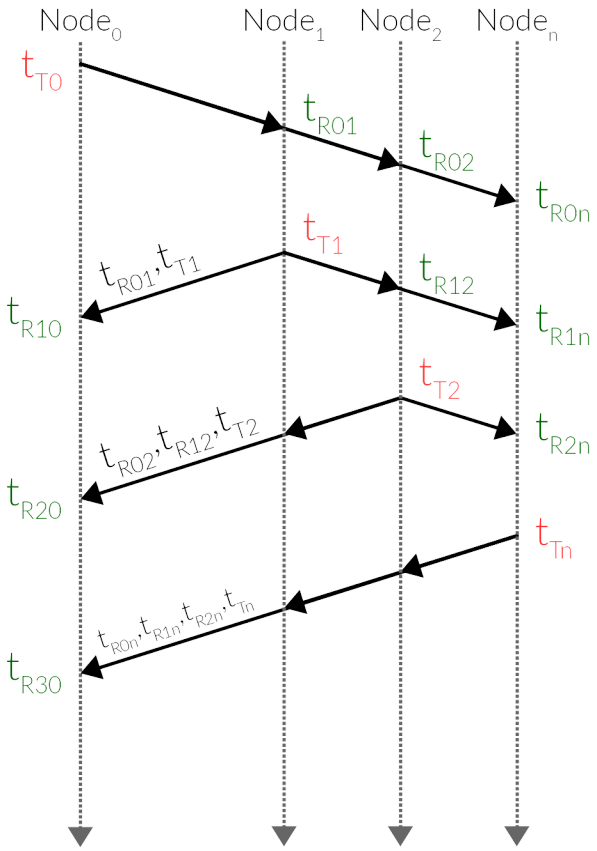

Figure 5. BB-TWR sequence diagrams for three triangle nodes

Proceed to the design, implementation and testing of the protocol.

1. Identify the needed primitives
2. Design the protocol :
    a. Propose the sequence of messages and their format,
    b. For each type of node (master, slave), specify the local algorithms,
    c. Propose an automata.
3. Implement and test