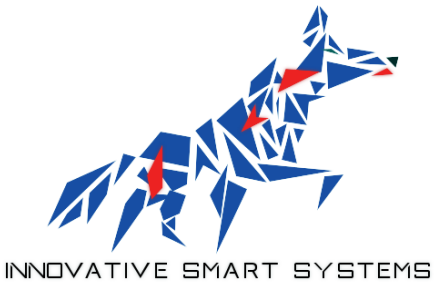
TP : SYNCHRONISATION



INSA TOULOUSE

Marc Kassé

2020/2021

I. Introduction

In this practical session, I did experiment the synchronisation in IOT thanks to a transceiver called DecaWave which uses a specific library in Arduino, Decaduino. This tranceiver was developed in a laboratory in Toulouse. I used the Decaduino library available in order to establish communication between the sender and receiver.

II. Getting familiar with Decaduino

The used primitives in this part are :

- ***decaduino.setRxBuffer :*** Sets the RX buffer for future frame reception. Received bytes will be stored at the beginning of the buffer.
- ***decaduino.plmeRxEnableRequest :*** Sets transceiver mode to receive mode.
- ***decaduino.rxFrameAvailable :*** Returns true if a frame has been received.
- ***decaduino.init :*** Initializes DecaDuino and DWM1000 without addressing fields filtering (Promiscuous mode)
- ***decaduino.pdDataRequest :*** Sends a len-byte frame from buf.
- ***decaduino.hasTxSucceeded :*** Returns true if the last transmission request has been succefully completed.

I modified the Arduino code from the two files (sender and receiver) in order to set up a logical isolation between pairs :

- Sender

The frame has a length of 120 bits maximum. In order to identify the sender, I set the first bit with a choosen identification number.

```
txData[0] = id ; (with id = 22 defined in DecaDuinoSender)
```

- Receiver

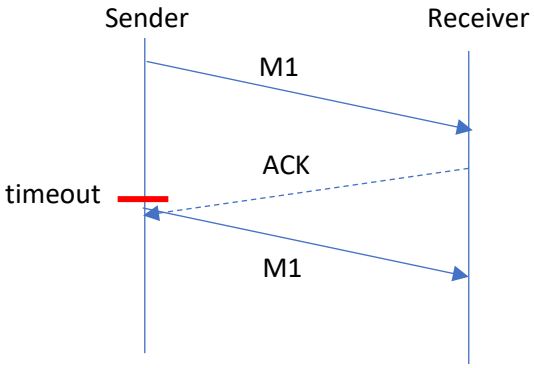
In order to receive the message from the corresponding sender, I added a filter on the first bit.

```
if (rxData[0]=22){
  for (int i=0; i<rxLen; i++) {
    rxData[i] = i;
    Serial.print(rxData[i], HEX);
    Serial.print("|");
  }
}
```

III. Design of a first simple protocol

Implementation of a simple DATA + ACK with configurable timeout and retries values.

The goal is to send a message to a receiver. Once the receiver receives the message, it sends the acknowledgment to the sender. If the acknowledgement is not received after a *timeout* period, the message is sent again. The number of retransmission of the message is determined by *retries* value.



- Void loop (sender):

The sender will send a message then it will switch to receiving mode in order to receive the acknowlegment. After the message is sent, the time it was sent is recorded in a variable.

At a reception of a message, I check if this message is the acknowlegement I am waiting for. I also calculate the time passed between emission and reception and compare it to the timeout. If this time exceed the timeout, the message must be resent until the number of retries defined is reached.

```
while ( decaduino.rxFrameAvailable() ) {
  timeReceived=millis();
  laps=timeSent-timeReceived;
  if (laps<=timeout){
    if (rxData[0]=22){
        Proceed lecture of the message ;
    else {
     print a text message ;
    }
    Exit the while loop ;
 else {
    if (number of retransmission is lower to retries){
      Resend message ;
      Increment retransmission variable ;
    else {
      Exit the while loop ;
    }
  }
}
```

- Void loop (receiver) :

The receiver, when a message is received, will return a message containing the id number as first bit (in our case 22). This is the choice I made to identify a specific message.

IV. Implementation of a star synchronization

The used primitives in this part are :

- ***encodeUint64(uint64_t value, address of the uint8_t buffer)*** : Formats an uint64 value as a list of uint8 values.
- ***decodeUint64(address of the uint8_t buffer)*** : Builds an uint64 value from two uint8 values.

- Void loop (sender) :

I entered the time in a variable. I encoded the time in a buffer thanks to the primitive function encodeUint64. Then I created the message to be sent with first bit equal the id number and the 8 next bits that match the 8 bits of the buffer.

```
timeT=millis();
decaduino.encodeUint64(timeT, buf);
txData[0] = 22; //id number
 for (int i=1; i<9; i++) {
   txData[i] = buf[i-1];
 }
 for (int i=9; i<MAX_FRAME_LEN; i++) {
   txData[i] = i;
 }
```

- Void loop (receiver) :

Once the message is received, I checked that this message is the one I am interested in thanks to the id number 22. Then, I read my buffer encoded in the 8 bits after the first bit with corresponds to the id number. Finally, I used decode function to transform my buffer into a 64 bits value which corresponds to the time sent by the master. Doing this allows to calculate the offset between master clock and slave clock.

```
timeS=millis();
   if (rxData[0]=22){
     for (int i=1; i<9; i++) {
       buf[i-1]=rxData[i];
     }
     timeM=decaduino.decodeUint64(buf);
     offsetcalc=timeS-timeM;
     timesS=offsetcalc+timeM;
     for (int i=9; i<rxLen; i++) {
       rxData[i] = i;
       Serial.print(rxData[i], HEX);
       Serial.print("|");
     }
```

V. Conclusion

Thanks to the decaduino, we learned how to synchronize a network of master/slave nodes by including the time of the master in the frame sent to the slaves. The slaves calibrate their clocks in accordance with the master clock.