Février ,2021

TP synchronisation

SKIKER HICHAM

Rapport de TP

1-INTRODUCTION :

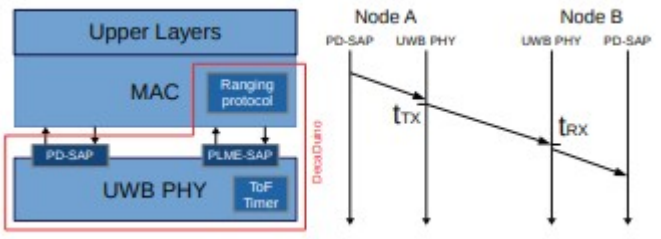
LE DECAWINO EST UNE CARTE BASÉE SUR LE PJRC TEENSY 3.1/3.2 ET LE MODULE DECAWAVE DWM1000, QUI EST UN ÉMETTEUR-RÉCEPTEUR À BANDE ULTRA-LARGE (UWB).

LE DECAWINO PERMET LA COMMUNICATION SANS FIL ET LES APPLICATIONS DE TÉLÉMÉTRIE, C'EST-À-DIRE L'ÉVALUATION DE LA DISTANCE ENTRE LES NŒUDS À L'AIDE DU SIGNAL ÉLECTROMAGNÉTIQUE.

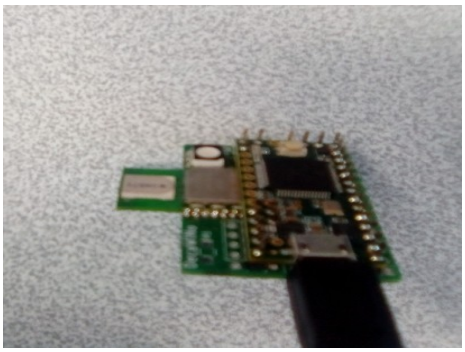
DECAWINO

LE DECAWINO A ÉTÉ CONÇU PAR LE LABORATOIRE DE L'IRIT À TOULOUSE, FRANCE (INSTITUT DE RECHERCHE EN INFORMATIQUE DE TOULOUSE).

EN EFFET EN S'INTÉRESSENT LA PARTIE DE SYNCHRONISATION ENTRE ÉMISSION ET RÉCEPTION C'EST 'A DIRE IMPLANTER UNE HORLOGE ENTRE NODE A ET B POUR FIABILISER LES DATA ENVOYER, ET AUSSI S'ASSURER LA BONNE RÉCEPTION. CELA VA GÉNÉRER UN RÉFÉRENTIEL DANS LA COMMUNICATION SOIT NOTRE INFORMATION ARRIVER AU CIBLE OU RETARDER PAR RAPPORT AU CIBLE .



DECAWINO : CARTE ARDUINO SHIELD MONTÉ AVEC UNE INNOVATION DES CHERCHEURS.



1. AVANT Commencez le TP ,on à utiliser DecaDuino avec des exemples

Ces exemples sont disponibles dans le menu de l'IDE Arduino :

Fichier > Exemples > DecaDuino

DecaDuinoSender

Ce code montre comment utiliser la bibliothèque DecaDuino pour envoyer des messages sur la radio UWB ultra wide band .

DecaDuinoReceiverSniffer

Ce code montre comment utiliser la bibliothèque DecaDuino pour recevoir des messages sur la radio UWB. On utilise le serial terminal pour visualiser les messages reçus.

Faire un expéditeur de données
Le croquis DecaDuinoSender illustre l'utilisation de la bibliothèque DecaDuino pour envoyer des messages sur la radio UWB.

Après l'initialisation de l'émetteur-récepteur, le croquis est réalisé périodiquement :

crée une trame de MAX_FRAME_LEN bytes (120 bytes),
l'envoie en appelant decaduino.pdDataRequest(),
attend que la trame ait été envoyée par l'émetteur-récepteur avec la mention "while (! decaduino.hasTxSucceded())" ;,
attend 1 seconde.

Après

Réalisez un récepteur de données/renifleur d'images
Le croquis DecaDuinoReceiverSniffer montre l'utilisation de la bibliothèque DecaDuino pour recevoir des messages sur la radio UWB.

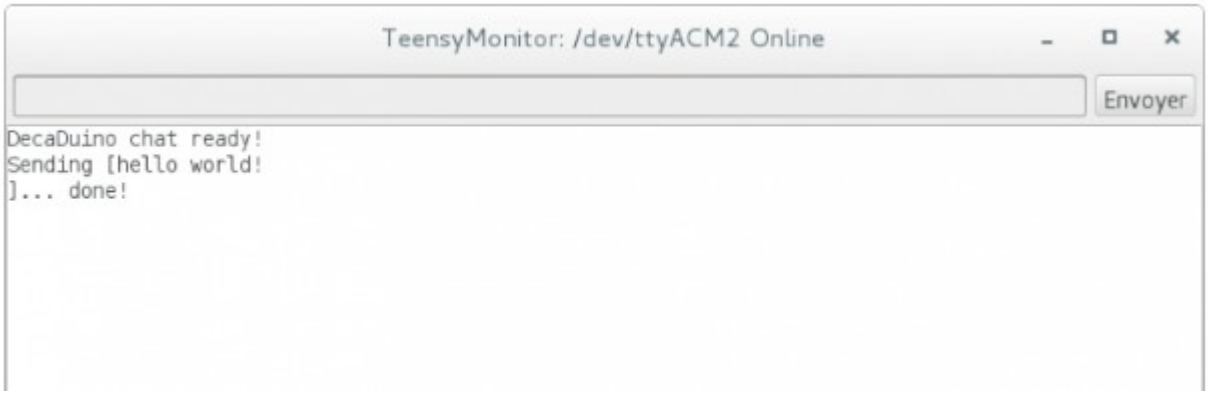
- Module DWM1000

Le DWM1000 est un module émetteur-récepteur sans fil conforme à la norme IEEE802.15.4-2011 UWB, basé sur le circuit intégré DW1000 de Decawave. Ce module permet la localisation d'objets dans des systèmes de localisation en temps réel (RTLS) avec une précision de 10 cm à l'intérieur, des communications à haut débit jusqu'à 6,8 Mbps, et une excellente portée de communication jusqu'à 300 m grâce à des techniques de réception cohérentes.

Prend en charge les débits de 110kbps, 850kbps et 6,8Mbps
Fréquence de 3,5GHz - 6,5GHz - canal 1, 2, 3, 4, 5, 7

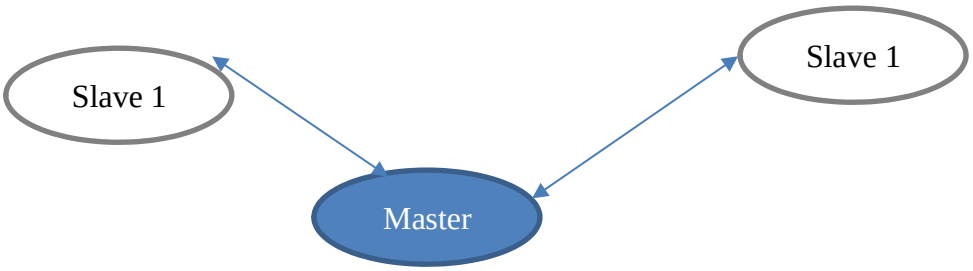
Exemple de code sender :

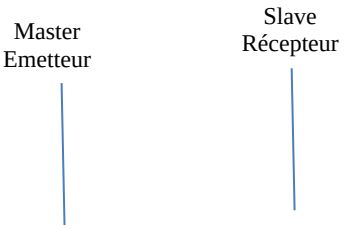
On a utilisé la pin 13 de la carte, en effet c'est une led pour visualiser la bonne émission ou l'inverse, dans la partie réception ; après on utilise baude rate 115200 pour synchroniser le serial terminal au même baude rate.
Cette carte possède des ports analogiques, digitale, on peut transmettre les data capteurs vers un autre capteurs, ces types de capteurs sont premièrement intelligent qui communiquent par des ondes électromagnétiques et suivant un algorithme il peut reçoivent les donnes ou émettre les donnes.
Interface de programmation IDE .



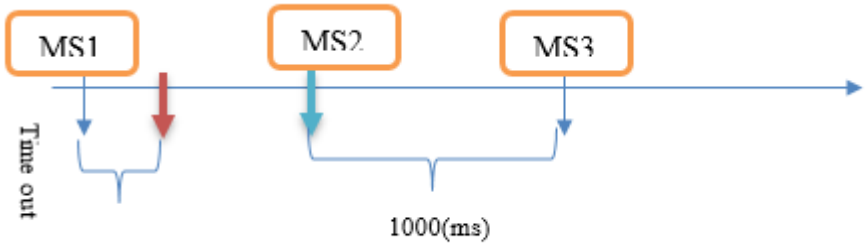
2. Choisir le protocole de communication
En effet dans cette partie j'ai envoyé toutes les trames sans aucune condition, mais dans la réception je peux mettre des conditions ou des filtres pour analyser les data que je veux contrôler.

3. Echanges des data
Durant ce TP, parfois il Ya de blocage au niveau soit en réception soit en émission, donc pour résoudre ce genre de problème, soit initialiser par le bouton qu'existe déjà dans la carte soit on déconnecte la carte et envoyé le programme de nouveau.





Dans cette partie, on va voir comment le master va imposer son horloge pour que le slave recoit data par l'émetteur.
Le temps de capture on utilise millis cette fonction permet de scanner le temps existant.
Dans le diagramme en bas entre deux messages on 1s, notre objectif c'est de savoir le time out dans la zone flèche marron, lorsque on atteind la position flèche bleu, justement objectif si on perd information on a une possibilité d'envoyer encore le message, mais une fois on arrive au MS2 (flèche bleu), il faut s'intéresser au trame message 2.

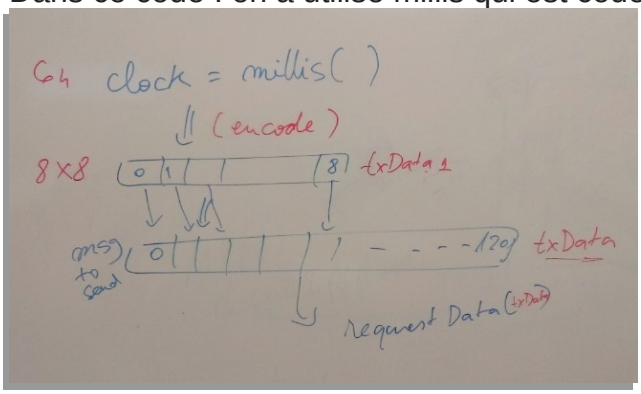


Partie 3 :
Dans le code pour émission :

```
/ DecaDuinoSender
// This sketch shows how to use the DecaDuino library to send messages over the UWB radio
// by Adrien van den Bossche <vandenbo@univ-tlse2.fr>
// This sketch is a part of the DecaDuino Project - please refer to the DecaDuino LICENCE file for licensing details
```

```
#include <SPI.h>
#include <DecaDuino.h>

#define MAX_FRAME_LEN 120
uint8_t txData[MAX_FRAME_LEN];
uint8_t txData1[8];
uint16_t txLen;
DecaDuino decaduino;
int rxFrames;
long temp=0;


void setup()
{
pinMode(13, OUTPUT); // Internal LED (pin 13 on DecaWiNo board)
Serial.begin(115200); // Init Serial port
SPI.setSCK(14); // Set SPI clock pin (pin 14 on DecaWiNo board)

// Init DecaDuino and blink if initialisation fails
if ( !decaduino.init() ) {
Serial.println("decaduino init failed");
while(1) { digitalWrite(13, HIGH); delay(50); digitalWrite(13, LOW); delay(50); }
}
}


void loop()
{
// make dummy data, send it and wait the end of the transmission.
digitalWrite(13, HIGH);
temp= millis();
decaduino.encodeUint64 (temp, txData1);
Serial.print(temp);
for (int i=0; i<MAX_FRAME_LEN; i++) {
txData[i] = i;
}
for (int j=0; j<8; j++) {

txData[j]=txData1[j];
decaduino.pdDataRequest(txData, MAX_FRAME_LEN);
while ( !decaduino.hasTxSucceeded() );
digitalWrite(13, LOW);

// wait 1 second
delay(1000);
```

Dans ce code : on a utilisé millis qui est coder 64 bit, ainsi on a utilisé la fonction encode



Car le msg envoyé est sur 120 bit. On a affecté temp pour calculer le temps. A travers une boucle on va lire les data frame jusqu'à max data frame. Après ça en à utiliser la fonction decaduino pour envoyer les trames au nouveau tableau.

Par contre le code pour réception :

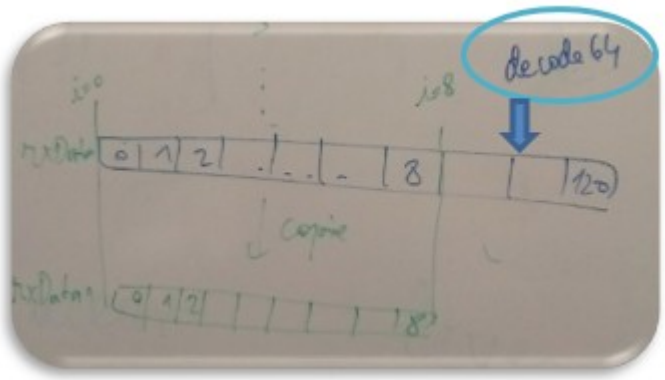
DecaDuinoReceiver
// This sketch shows how to use the DecaDuino library to receive messages over the UWB radio.
// The sketch prints the received bytes in HEX; it can be used as a frame sniffer.
// by Adrien van den Bossche <vandenbo@univ-tlse2.fr>
// This sketch is a part of the DecaDuino Project - please refer to the DecaDuino LICENCE file for licensing details

```
#include <SPI.h>
#include <DecaDuino.h>

#define MAX_FRAME_LEN 120
uint8_t rxData[MAX_FRAME_LEN];
uint8_t rxData1[8];
uint16_t rxLen;
DecaDuino decaduino;
int rxFrames;
long received=0;
long TIME =0;
uint64_t rxclock;


void setup()
{
pinMode(13, OUTPUT); // Internal LED (pin 13 on DecaWiNo board)
Serial.begin(115200); // Init Serial port
SPI.setSCK(14); // Set SPI clock pin (pin 14 on DecaWiNo board)

// Init DecaDuino and blink if initialisation fails
if ( !decaduino.init() ) {
Serial.println("decaduino init failed");
while(1) { digitalWrite(13, HIGH); delay(50); digitalWrite(13, LOW); delay(50); }
}

// Set RX buffer and enable RX
```

```
decaduino.setRxBuffer(rxData, &rxLen);
received=millis();
decaduino.plmeRxEnableRequest();
rxFrames = 0;
}
void loop()
{
// If a message has been received, print it and re-enable receiver
if ( decaduino.rxFrameAvailable() ) {
digitalWrite(13, HIGH);
Serial.print("#"); Serial.print(++rxFrames); Serial.print(" ");
Serial.print(rxLen);
Serial.print("bytes received: |");
for (int i=0; i<rxLen; i++) {
Serial.print(rxData[i], HEX);


Serial.print("|");
}
for (int j=0; j<8; j++) {
rxData1[j]=rxData[j];
rxclock= decodeUint64 ( *rxData1);
Serial.print(rxclock);
Serial.println();
decaduino.plmeRxEnableRequest(); // Always renable RX after a frame reception
TIME=received-clock;

digitalWrite(13, LOW);
}
```

On a inversé, utiliser decodeunit 64 pour 8 positions, de la même manière cette fois ici on a le tableaux rx Data qu'on va affecter les data frame.

En effet on a deux temps, temps local réception et temps d'émission, c'est dire on peut déduire le temps de différence pour juger en est en avance ou en retard.



4. Conclusion :

Malheureusement durant le TP, on n'a pas terminé la partie protocole SISP, On a compris la partie synchronisation constitue un élément clé pour la connectivité des objets. Durant ce Tp on a appris des nouveaux concepts dans le volet communication entre les capteurs.