


TP DOMOTIQUE

Table des matières

TP1 a : Mise en œuvre d'un capteur de température I2C.....	2
1. Project n°1 : led maquette	2
2. Project n°2 : LCD maquette	3
3. Project n°3 : lcd_ds1631maquette	4
4. Projet n°4 : lcd_ds1631maquette en autonomie	7
TP1 b Envoie d'informations à distance à l'aide d'émetteur /récepteur XBEE	11
1. Projet n°5 : Faire converser 2 modules Xbee.....	11
2. Project n°6 : bonjourxbee1.....	11
3. Projet n°7 : Projet complet cad : envoi à distance de l'info température	12
4. Projet n°9: mise en veille de l'émetteur Xbee.....	1
Annexe :	2

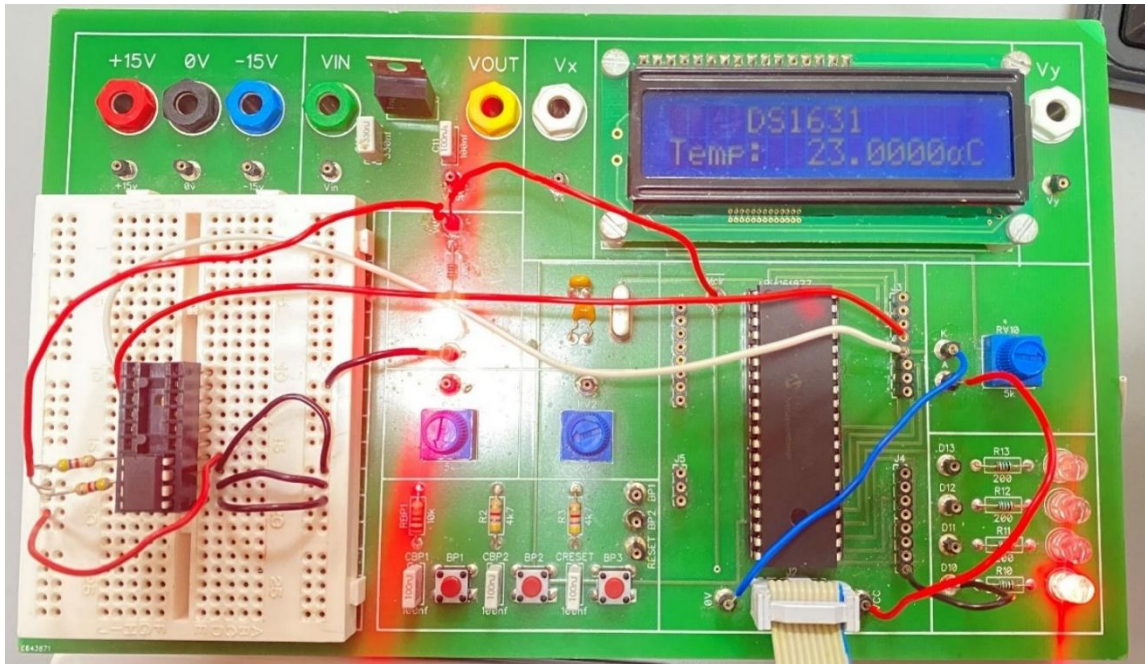
DBIBIH Oussama	Signature : 12.10.2025
MARTI Matthis	Signature : 12.10.2025
Date	11/10/2025

Responsables TP :

Thierry PERISSE	thierry.perisse@univ-tlse3.fr	
Hélène LEYMARIE	helene.leymarie@univ-tlse3.fr	

TP1 a : Mise en œuvre d'un capteur de température I2C.

1. Project n°1 : led maquette



Code :

```
while(1) {

    // --- Affichage sur l'écran LCD ---
    // Affiche le texte contenu dans la variable 'text=OUSSAMA' sur la première ligne du LCD
    Lcd_Out(1, 1, text);          // Ligne 1, Colonne 1

    // Affiche le texte contenu dans la variable 'text2=MATTHIS' sur la deuxième ligne du LCD
    Lcd_Out(2, 1, text2);        // Ligne 2, Colonne 1

    // --- Lecture et traitement de l'entrée D6 ---
    // Si le bit F6 du PORTD est à l'état haut (bouton)
    if (PORTD.F6 == 1) {

        // Inversion de l'état de la broche D7 (clignotement)
        // Si D7 = 1 → passe à 0 ; si D7 = 0 → passe à 1
        PORTD.F7 = ~PORTD.F7;

        // Délai d'attente de 1 seconde avant de répéter le test
        delay_ms(1000);

    } else {
        // Si D6 est à l'état bas, on force D7 à 0
        PORTD.F7 = 0;
    }
}
```

Oussama Dbibih, 3 days ago • add folders Oussama Dbibih [3 days ago] • add folders

Dans ce programme, la LED reliée à la broche D7 est contrôlée par un bouton poussoir connecté à D6. Tant que le bouton n'est pas appuyé (PORTD.F6 = 0), la LED reste éteinte. Lorsque le bouton est

pressé (PORTD.F6 = 1), le microcontrôleur inverse l'état de la sortie D7 à chaque passage dans la boucle, ce qui fait clignoter la LED avec une période d'environ 1 seconde (grâce à delay_ms(1000)).

2. Project n°2 : LCD maquette

le système affiche en continu les textes sur le LCD tout en utilisant le bouton D6 pour activer ou désactiver le clignotement de la LED sur D7.

```
// LCD module connections
sbit LCD_RS at RB0_bit;
sbit LCD_EN at RB1_bit;
sbit LCD_D4 at RB2_bit;
sbit LCD_D5 at RB3_bit;
sbit LCD_D6 at RB4_bit;
sbit LCD_D7 at RB5_bit;

sbit LCD_RS_Direction at TRISB0_bit;
sbit LCD_EN_Direction at TRISB1_bit;
sbit LCD_D4_Direction at TRISB2_bit;
sbit LCD_D5_Direction at TRISB3_bit;
sbit LCD_D6_Direction at TRISB4_bit;
sbit LCD_D7_Direction at TRISB5_bit;

char *text = "OUSSAMA";
char *text2 = "MATTHIS";

void main() {

    Lcd_Init();
    Lcd_Cmd(_Lcd_CLEAR);          // Clear display
    Lcd_Cmd(_Lcd_CURSOR OFF);    // Turn cursor off

    PORTD = 0;                   // Initialize PORTC
    TRISD = 0b01000000;          // Configure D6 en entr e et les autres en sorties

    while(1) {

        // --- Affichage sur l' cran LCD ---
        // Affiche le texte contenu dans la variable 'text=OUSSAMA' sur la premi re ligne du LCD
        Lcd_Out(1, 1, text);      // Ligne 1, Colonne 1

        // Affiche le texte contenu dans la variable 'text2=MATTHIS' sur la deuxi me ligne du LCD
        Lcd_Out(2, 1, text2);     // Ligne 2, Colonne 1
    }
}
```



```
// -----  
// 3 Calcul et affichage du bit de fraction (0.5°C)  
// -----  
// Le DS1631 encode la partie fractionnaire dans le bit 7 du LSB :  
// - Si ce bit = 1 → ajouter 0.5°C  
// - Si ce bit = 0 → partie fractionnaire = 0  
if (LSB & 0x80) {  
    msg[6] = '5';           // Ajouter ".5"  
} else {  
    msg[6] = '0';           // Ajouter ".0"  
}  
Lcd_Chrcp(msg[6]);          // Affiche la fraction (.0 ou .5)  
Lcd_Chrcp(223);             // Affiche le symbole ° (code ASCII 223)  
Lcd_Chrcp('C');             // Ajoute la lettre C (°C)  
  
// -----  
// 4 Gestion du bouton sur RD6 et de la LED sur RD7  
// -----  
// Si le bouton (ou interrupteur) sur RD6 est activé (niveau haut),  
// alors la LED sur RD7 bascule d'état (clignote chaque seconde).  
// Sinon, elle reste éteinte.  
if (PORTD.F6 == 1) {  
    PORTD.F7 = ~PORTD.F7;    // Inversion de l'état de la LED  
    Delay_ms(1000);          // Délai d'1 seconde entre deux basculements  
} else {  
    PORTD.F7 = 0;            // Si le bouton n'est pas appuyé → LED éteinte  
}
```

Initialisation du matériel

Le programme configure les ports du PIC16F877A pour piloter un écran LCD 16x2 sur le PORTB, et le capteur de température DS1631 via le bus I²C sur le PORTC.

Le LCD est initialisé (effacement, curseur désactivé), puis le bus I²C est paramétré à 100 kHz.

Le DS1631 est configuré en mode One-Shot avec une résolution de 12 bits, prêt à effectuer des mesures ponctuelles sur demande.

Lecture et affichage de la température

Dans la boucle principale, le PIC envoie une commande au DS1631 pour lancer une conversion de température, puis lit les deux octets renvoyés (MSB et LSB).

Le MSB contient la partie entière de la température, et le bit 7 du LSB indique la fraction (+0.5 °C).

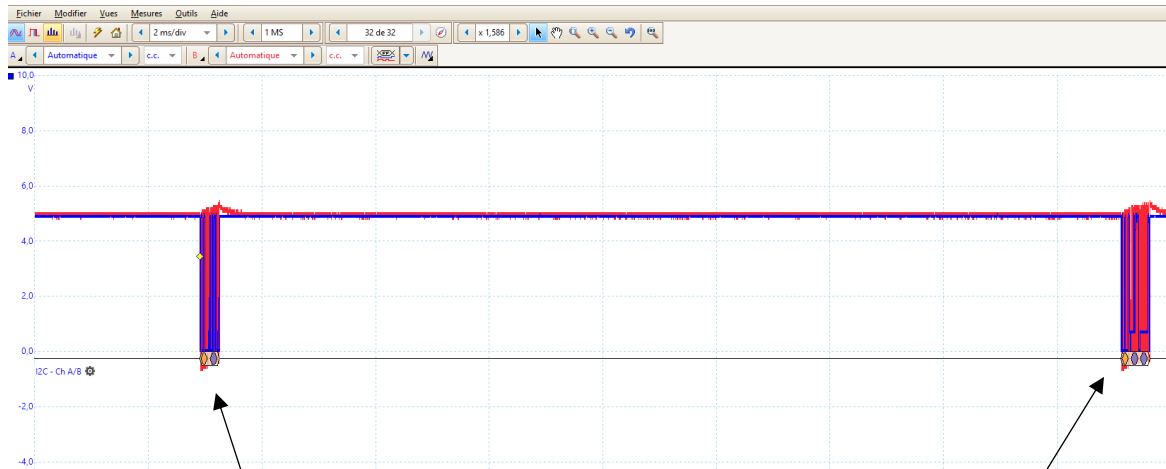
La valeur complète est ensuite convertie en texte et affichée sur le LCD sous la forme :

Temp: 23.5°C

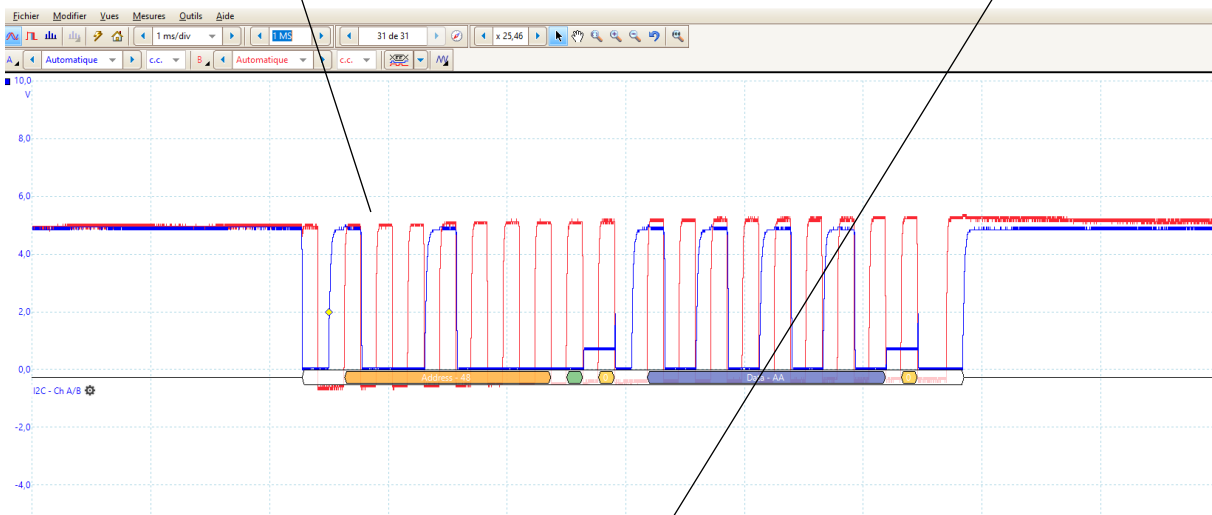
Gestion du bouton et de la LED

Le programme surveille en continu l'entrée RD6 reliée à un bouton poussoir.

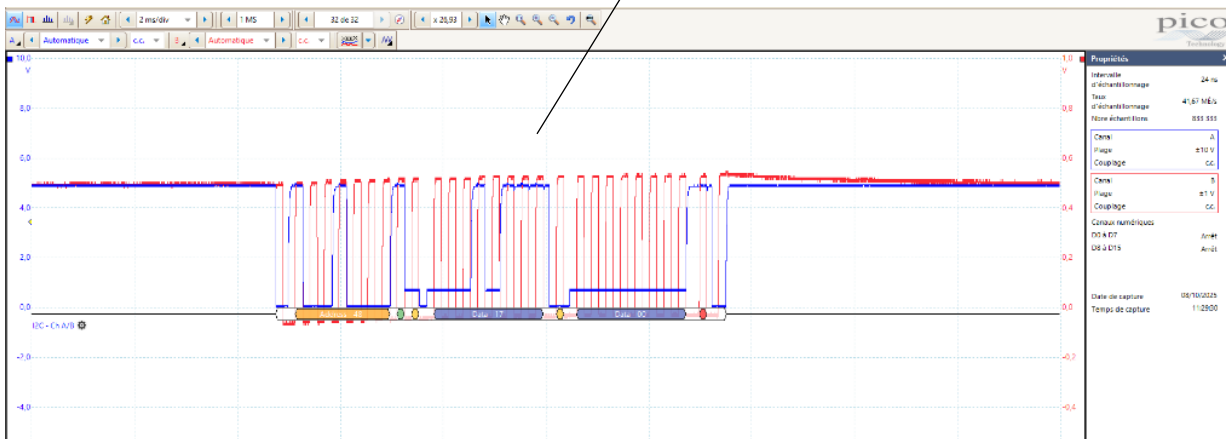
- Si le bouton est appuyé, la LED sur RD7 clignote avec une période d'environ 1 seconde.
- Si le bouton est relâché, la LED reste éteinte.
Cette partie permet de tester l'interactivité et le fonctionnement logique du port D tout en affichant la température mesurée.



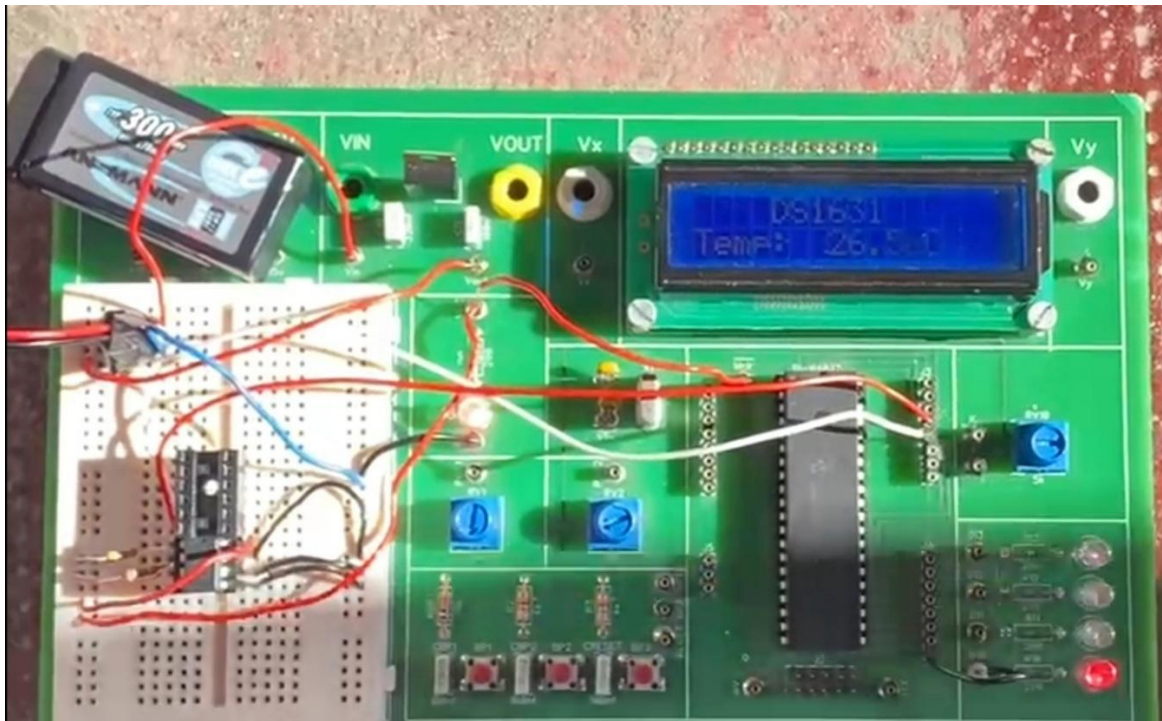
Cette trame correspond à une **écriture de commande** : le PIC envoie au DS1631 l'ordre de lire la température :



Le PIC lit la température 23.0 °C depuis le DS1631 (48h + 17h + 00h)



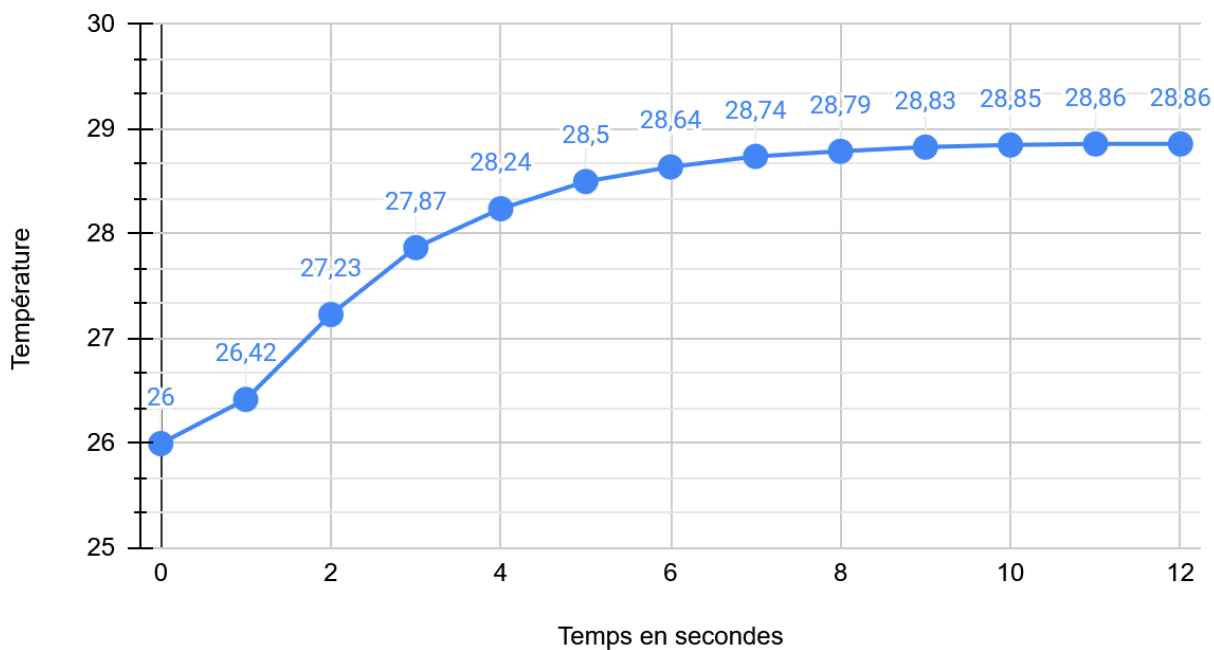
Sortir dehors avec la maquette :



Tracé du relevé de température en fonction du temps :

Temps en seconds	Temperature
0	26
1	26,42
2	27,23
3	27,87
4	28,24
5	28,5
6	28,64
7	28,74
8	28,79
9	28,83
10	28,85
11	28,86
12	28,86

Température par rapport au temps en secondes



En déduire le temps de réponse du capteur :

Calcul de la variation de température

$$\Delta T = T_{final} - T_{initial}$$

Dans notre cas :

$$T_{final} = 28,86^{\circ}\text{C} (\text{à } t = 12 \text{ s})$$

$$T_{initial} = 26^{\circ}\text{C}$$

$$\Delta T = 28,86 - 26 = 2,86^{\circ}\text{C}$$

Détermination du seuil à 95 %:

$$T_{95\%} = T_{initial} + 0,95 \times \Delta T$$

$$T_{95\%} = 26 + 0,95 \times 2,86 = 26 + 2,717 = 28,717^{\circ}\text{C}$$

Calcul du temps de réponse:

Le **temps de réponse** correspond au premier instant où la température mesurée atteint (ou dépasse) le seuil des **95 % de la variation totale**.

En observant le tableau expérimental:

La première valeur $\geq 28,717^{\circ}\text{C}$ apparaît à **t = 8 s**, où la température vaut **28,74 °C**.

TP1 b Envoie d'informations à distance à l'aide d'émetteur /récepteur XBEE

1. Projet n°5 : Faire converser 2 modules Xbee.

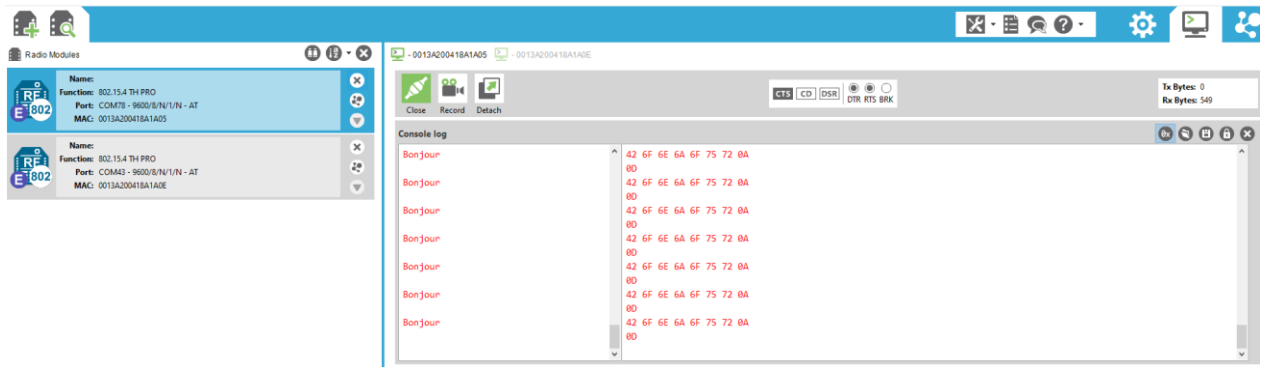
Les deux XBee doivent partager le même PAN ID et pointer mutuellement leurs adresses de destination pour échanger des données sans fil.

2. Project n°6 : bonjourxbee1

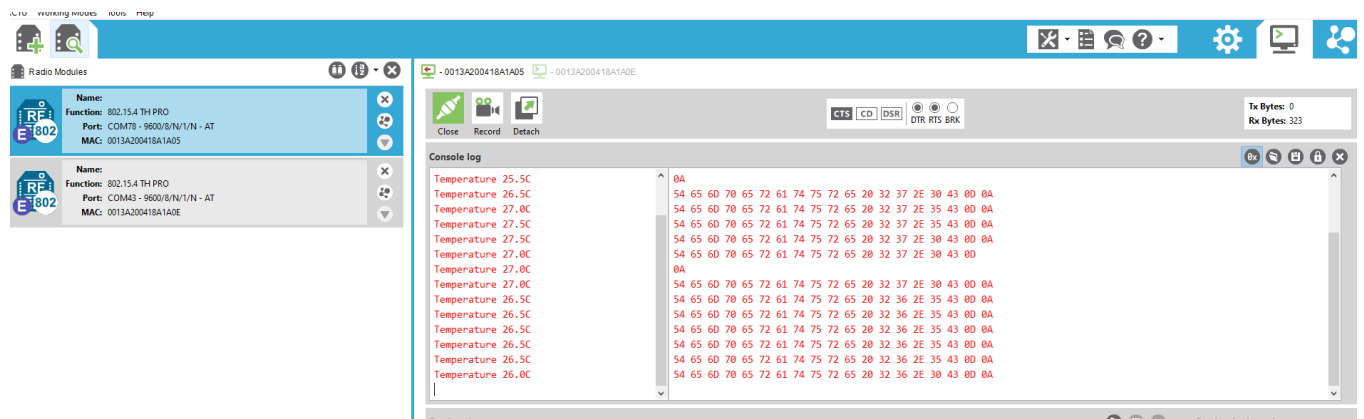
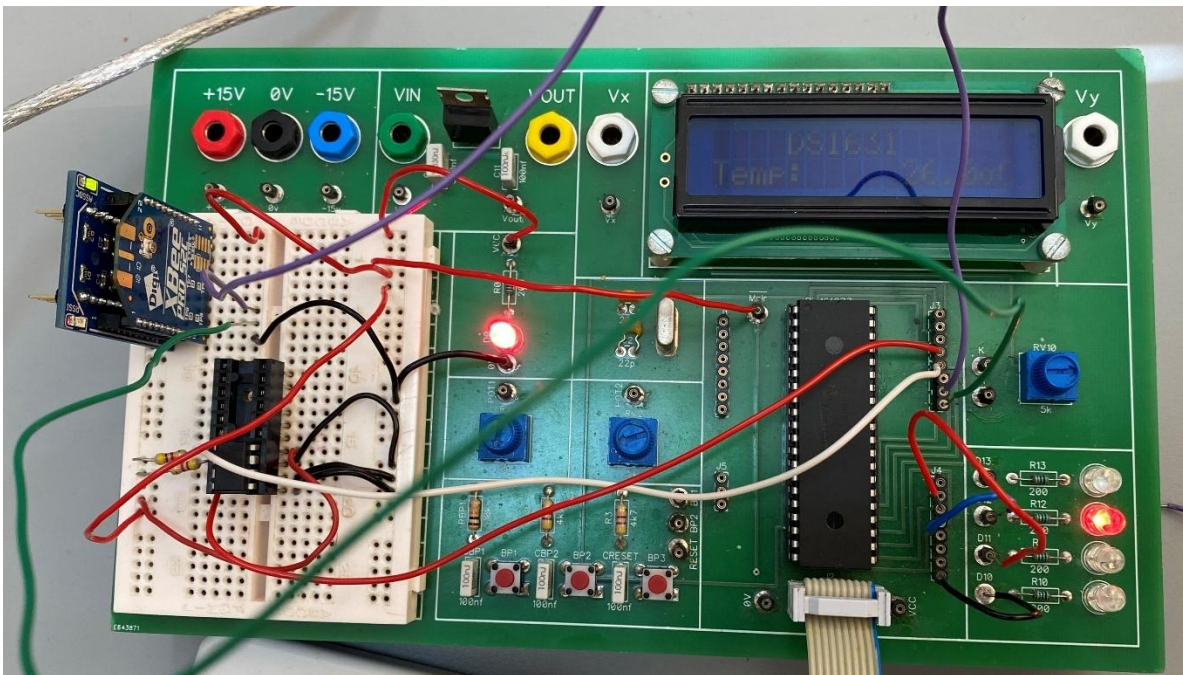
```
void main() {  
    // -----  
    // 1 Initialisation des ports  
    // -----  
    PORTD = 0;          // Valeur initiale du PORTD : tout à 0  
    TRISD = 0;          // Configure le PORTD comme sorties (pour LED de test)  
  
    // -----  
    // 2 Initialisation de la communication UART  
    // -----  
    // Configure le module UART1 à 9600 bauds, 8 bits, pas de parité, 1 stop bit  
    Uart1_Init(9600);  
    Delay_ms(100);      // Court délai pour stabiliser la liaison UART  
  
    do {  
        // -----  
        // ♦ Vérifie si une donnée a été reçue sur la liaison série  
        // -----  
        if (Uart1_Data_Ready()) {      // Si un octet est disponible sur RX  
  
            // --- Envoi du message "Bonjour" caractère par caractère ---  
            for (cpt = 0; cpt < 7; cpt++) {  
                i = tableau[cpt];      // Prend chaque lettre du tableau  
                Uart1_Write(i);        // L'envoi sur la ligne série  
            }  
  
            // --- Envoie un retour à la ligne pour aérer l'affichage ---  
            saut_ligne();  
        }  
  
        // -----  
        // ♦ Attente et clignotement du PORTD  
        // -----  
        Delay_ms(1000);      // Attente d'1 seconde  
        PORTD = ~PORTD;      // Inverse tous les bits du PORTD → LED clignote  
  
    } while (1);              // Boucle infinie  
}
```

You, 4 seconds ago • Uncommitted changes Not committed yet

Ce programme teste la communication série (UART) entre le microcontrôleur PIC et un module XBee : dès qu'une donnée est reçue sur le port série, le PIC envoie le message "Bonjour" caractère par caractère, ajoute un retour à la ligne pour une lecture claire, puis fait clignoter les LED du PORTD pour indiquer que la transmission a eu lieu. Il fonctionne en boucle continue, permettant ainsi de vérifier facilement que la liaison UART et le module XBee communiquent correctement.



3. Projet n°7 : Projet complet cad : envoi à distance de l'info température



Code :

```
void main() {
// =====
//  Boucle principale
// =====
while(1) {

// -----
//  ♦ Étape 1 : Lecture de la température via I2C
// -----
DS1631_Start(0);          // Lancer une conversion One-Shot
Delay_ms(750);            // Attendre la fin de conversion (résolution 12 bits)
temp = DS1631_Read_Temperature(0); // Lire la température (MSB + LSB)

// -----
//  ♦ Étape 2 : Traitement de la valeur lue
// -----
// Si le bit 7 du LSB = 1 → ajouter +0.5°C
if (((unsigned*)&temp)[0] & 0x80) {
    temp.decimal = 5;      // partie décimale = .5
} else {
    temp.decimal = 0;      // partie décimale = .0
}

// -----
//  ♦ Étape 3 : Affichage LCD
// -----
Lcd_Cmd(_LCD_CLEAR);
Lcd_Out(1, 5, "DS1631");
Lcd_Out(2, 1, "Temp: ");

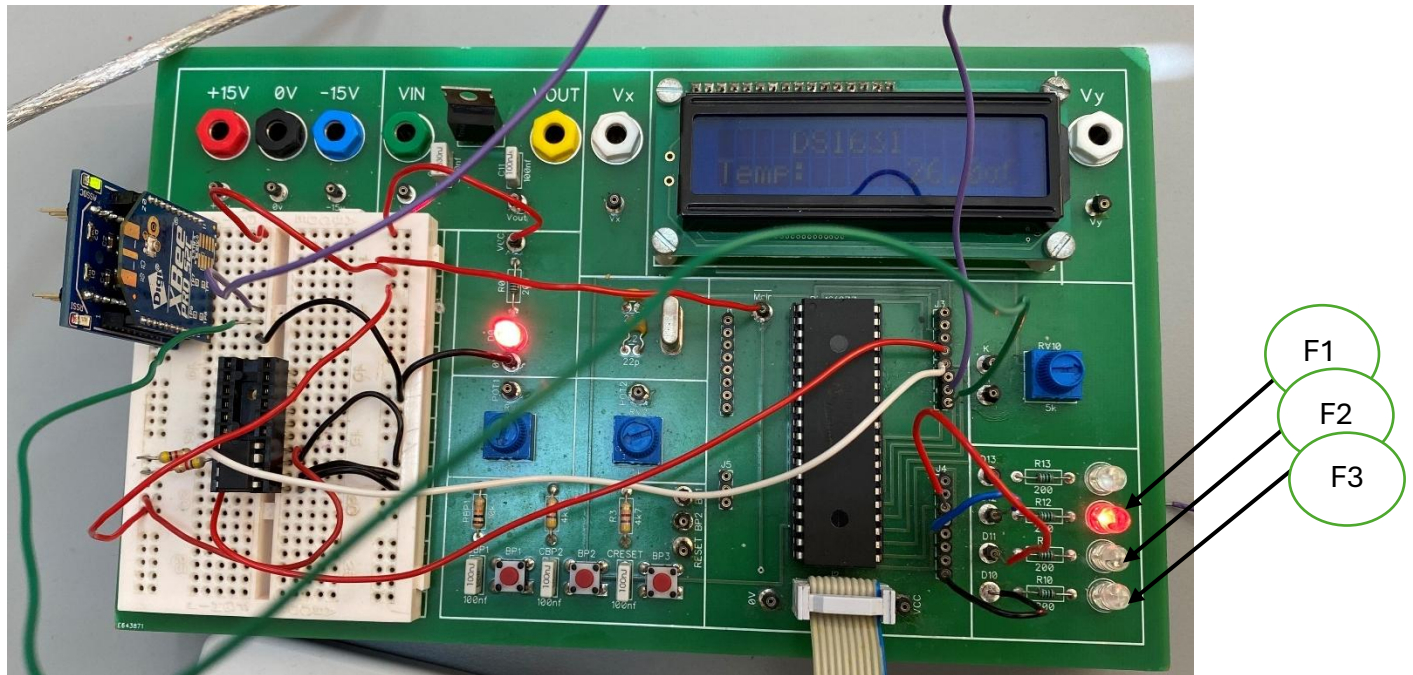
IntToStr(temp.entiere, msg); // Convertir la partie entière en texte
Lcd_Out_CP(msg + 3);        // Afficher sans les espaces
Lcd_Chrcp('.');              // Afficher le point décimal

msg[0] = temp.decimal + '0'; // Convertir la partie décimale (.0 / .5)
msg[1] = '\0';
Lcd_Out_CP(msg);             // Afficher .0 ou .5
Lcd_Chrcp(223);              // Afficher le symbole °
Lcd_Chrcp('C');              // puis "C"

xbee(11, tableau);           // Envoyer "Temperature "
xbee(3, msg + 3);            // Envoyer la partie entière
xbee(1, point);              // Envoyer le point
xbee(1, msg);                // Envoyer la partie décimale
xbee(1, degres);             // Envoyer "C"
xbee(2, saut);               // Saut de ligne
}
```

Ce programme permet de mesurer la température à l'aide du capteur DS1631, de l'afficher sur un écran LCD, et de l'envoyer sans fil via un module XBee.

4. Projet n°9: mise en veille de l'émetteur Xbee



Code :

Ce programme permet de mesurer la température à l'aide du capteur DS1631, de l'afficher sur un écran LCD, et de l'envoyer sans fil via un module XBee, tout en gérant un mode veille/réveil et une LED témoin.

void xbee_hibernate() : Cette fonction met le module XBee en mode veille (hibernation) pour économiser l'énergie.

Étapes :

1. Met la broche SLEEP_RQ (RD1) à 1 → demande de mise en veille.
2. Allume une LED témoin (RD3) pendant 2 secondes pour indiquer visuellement le passage en veille.
3. Éteint la LED après cette période.

```
void xbee_hibernate() {
    PORTD.F1 = 1;    // SLEEP_RQ = 1 => hibernation
    delay_ms(100);
    PORTD.F3 = 1;    // LED allumée pendant le sommeil
    delay_ms(2000);  // Laisse la LED allumée 2 secondes pour bien voir
    PORTD.F3 = 0;    // Puis éteinte
}
```

void xbee_wake() : Cette fonction réveille le module XBee après une période de sommeil.

Étapes :

1. Active la broche RESET (RD0) brièvement à 0, puis la relâche à 1 pour redémarrer le XBee.
2. Attends environ 500 ms pour laisser le temps au module de se réinitialiser.
3. Met la broche SLEEP_RQ (RD1) à 0 → sortie du mode veille.
4. Allume une LED témoin (RD2) pendant 1 seconde pour indiquer le réveil du module, puis l'éteint.

```
void xbee_wake() {  
    PORTD.F0 = 0;    // reset actif bas  
    delay_ms(10);  
    PORTD.F0 = 1;    // reset relâché  
    delay_ms(500);   // attendre que le XBee démarre  
  
    PORTD.F1 = 0;    // SLEEP_RQ = 0 => réveil  
    delay_ms(100);  
  
    PORTD.F2 = 1;    // Allume une LED pendant wake  
    delay_ms(1000);  // Durée visible (1s)  
    PORTD.F2 = 0;    // Éteint après  
}
```

Implémentation :

```
void main() {  
    while(1) {  
        // -----  
        // ♦ Étape 1 : Lecture de la température via I2C  
        // -----  
        DS1631_Start(0);    // Lancer une conversion One-Shot  
        Delay_ms(750);      // Attendre la fin de conversion (résolution 12 bits)  
        temp = DS1631_Read_Temperature(0); // Lire la température (MSB + LSB)  
  
        // -----  
        // ♦ Étape 2 : Traitement de la valeur lue  
        // -----  
        // Si le bit 7 du LSB = 1 → ajouter +0.5°C  
        if (((unsigned*)&temp)[0] & 0x80) {  
            temp.decimal = 5;    // partie décimale = .5  
        } else {  
            temp.decimal = 0;    // partie décimale = .0  
        }  
  
        // -----  
        // ♦ Étape 3 : Affichage LCD  
        // -----  
        Lcd_Cmd(_LCD_CLEAR);  
        Lcd_Out(1, 5, "DS1631");  
        Lcd_Out(2, 1, "Temp: ");  
  
        IntToStr(temp.entiere, msg); // Convertir la partie entière en texte  
        Lcd_Out_CP(msg + 3);         // Afficher sans les espaces  
        Lcd_Chrcp('.');              // Afficher le point décimal  
  
        msg[0] = temp.decimal + '0'; // Convertir la partie décimale (.0 / .5)  
        msg[1] = '\0';  
        Lcd_Out_CP(msg);             // Afficher .0 ou .5  
        Lcd_Chrcp(223);              // Afficher le symbole °  
        Lcd_Chrcp('C');              // puis "C"  
    }  
}
```



```
// -----  
// ♦ Étape 4 : Communication XBee (UART)  
// -----  
xbee_wake(); // Réveiller le module XBee  
xbee(11, tableau); // Envoyer "Temperature "  
xbee(3, msg + 3); // Envoyer la partie entière  
xbee(1, point); // Envoyer le point  
xbee(1, msg); // Envoyer la partie décimale  
xbee(1, degres); // Envoyer "C"  
xbee(2, saut); // Saut de ligne  
xbee_hibernate(); // Remettre le XBee en veille  
  
// -----  
// ♦ Étape 5 : Gestion du bouton sur RD6 et LED sur RD7  
// -----  
if (PORTD.F6 == 1) {  
    PORTD.F7 = ~PORTD.F7; // Inversion de la LED sur RD7  
    Delay_ms(1000); // Attente 1 seconde  
} else {  
    PORTD.F7 = 0; // LED éteinte si le bouton n'est pas appuyé  
}  
}
```

xbee_wake()

xbee_hibernate()

Cette méthode permet de réduire considérablement la consommation d'énergie du système en exploitant le mode veille matériel du module XBee : lorsque le microcontrôleur n'a plus besoin de communiquer, la fonction `xbee_hibernate()` met le XBee en hibernation (`SLEEP_RQ = 1`), ce qui coupe la plupart de ses circuits internes et abaisse sa consommation de plusieurs dizaines de milliampères à seulement quelques microampères. Lorsqu'une nouvelle mesure doit être transmise, la fonction `xbee_wake()` réactive le module (`RESET = 1`, `SLEEP_RQ = 0`) pour lui permettre d'émettre, avant de le remettre en veille. En alternant ces phases d'activité et de repos, le système reste opérationnel tout en économisant jusqu'à 90 % d'énergie, ce qui prolonge fortement l'autonomie d'une alimentation sur batterie.

Annexe :

Répertoire du Project + code : [Ossama1999-DEV/Xbee: ZigBee TP](https://github.com/Ossama1999-DEV/Xbee: ZigBee TP)

Ou : **git clone** <https://github.com/Ossama1999-DEV/Xbee.git>

QR-code :

