```
/*  ex_i2cA - Noeud A
 *  Ce programme est un exemple de communication I2C
 *  entre un coordonnateur (Pi) et un neoud (Arduino).
 *  Ce noeud est capable de transférer vers le coordonnateur:
 *
 *    - la valeur de la température interne;
 *    - le numéro de l'échantillon.
 *
 *  De plus, le noeud est capable de recevoir les commandes suivantes
 *  du coordonnateur:
 *    - STOP: arrêter l'échantillonnage;
 *    - GO:   démarrer l'échantillonnage.
 *
 *  Dans cet exemple, l'arrêt de l'échantillonnage remet à zéro le numéro
 *  de l'échantillon.
 *
 *  GPA788 Conception et intégration des objets connectés
 *  T. Wong
 *  06/2018
 *  08/2020
 *  11/2021
 */
#include <Wire.h>                       // Pour la communication I2C
#include <util/atomic.h>                // Pour la section critique
#include "dhtlib_gpa788.h"

/* ---------------------------------------------------------------
   Globales pour la classe ChipTemp
   --------------------------------------------------------------- */
dhtlib_gpa788 DHT(7);

/* ---------------------------------------------------------------
   Globales pour la communication I2C
   --------------------------------------------------------------- */
const uint8_t ADR_NOEUD{0x44};        // Adresse I2C de ce noeud
const uint8_t NB_REGISTRES{11};        // Nombre de registres de ce noeud

/* La carte des registres ------------------------------------------ */
union CarteRegistres {
  // Cette structure: Utilisée par le noeud pour lire et écrire
  //                  des données.
  struct {
    // Taux d'échantillonnage (1 octet)
    volatile uint8_t Ts;
    // Nombre d'échantillons (2 octets)
    volatile int16_t nb_echantillons;
    // Température interne du processeur ATMEGA en celsius
    // (4 octets)
    volatile float temperature;
    volatile float humidity;
  } champs;
  // Ce tableau: Utilisé par le coordonnateur pour lire et écrire
  //             des données.
  uint8_t regs[NB_REGISTRES];
};

union CarteRegistres cr;       // Une carte des registres
float temperature;                      // Variable intermédiaire pour mémoriser la
température
```

```
59  float humidity;
60  uint8_t adrReg;                         // Adresse du registre reçue du coordonnateur
61  enum class CMD { Stop = 0, Go, Pause};        // Commandes venant du coordonnateur
62  volatile CMD cmd;                       // Go -> échantilloner, Stop -> arrêter
63  const uint8_t MIN_Ts = 5;               // Période d'échantillonnage min (sec)
64  const uint8_t MAX_Ts = 200;             // Période d'échantillonnage max (sec)
65
66  /* ------------------------------------------------------------------
67     Initialisation
68     ------------------------------------------------------------------ */
69  void setup()
70  {
71    // Pour la communication série
72    Serial.begin(115200);
73
74    // Sur le VS Code, l'ouverture du port série prend du temps et on
75    // peut perdre des caractères. Ce problème n'existe pas sur l'IDE
76    // de l'Arduino.
77    waitUntil(2000);
78
79    // Initialiser les champs de la carte des registres
80    cr.champs.Ts = MIN_Ts;
81    cr.champs.nb_echantillons = 0;
82    cr.champs.temperature = -1;
83    temperature = -1;
84    cr.champs.humidity = -1;
85    humidity = -1;
86    // Initialiser les variables de contrôle de la
87    // communication I2C
88    cmd = CMD::Stop;
89    adrReg = -1;
90    // Réglage de la bibliothèque Wire pour le I2C
91    Wire.begin(ADR_NOEUD);
92    // Fonction pour traiter la réception de données venant du coordonnateur
93    Wire.onReceive(i2c_receiveEvent);
94    // Fonction pour traiter une requête de données venant du coordonnateur
95    Wire.onRequest(i2c_requestEvent);
96
97    // Indiquer que le noeud est prêt
98    Serial.print(F("Noeud à l'adresse 0x")); Serial.print(ADR_NOEUD, HEX);
99    Serial.println(F(" prêt à recevoir des commandes"));
100 }
101
102 /* ------------------------------------------------------------------
103    Boucle principale
104    ------------------------------------------------------------------ */
105 void loop()
106 {
107   // Échantillonner la température interne si la commande est Go
108   if (cmd == CMD::Go) {
109     DHT.read11(7);
110     temperature = DHT.getTemperature();
111     humidity = DHT.getHumidity();
112     // Section critique: empêcher les interruptions lors de l'assignation
113     // de la valeur de la température à la variable dans la carte des registres.
114     // Recommandation: réaliser la tâche la plus rapidement que possible dans
115     //                 la section critique.
116     ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
117       // Assigner la température lue dans cr.champs.temperature
118       cr.champs.temperature = temperature;
```

```
      cr.champs.humidity = humidity;
      // Augmenter le compte du nombre d'échantillons
      cr.champs.nb_echantillons++;
    }
    Serial.print(F("Échantillon #")); Serial.print(cr.champs.nb_echantillons);
    Serial.print(F("  Humidité relative : ")); Serial.print(cr.champs.humidity);
    Serial.print(F("  Température : ")); Serial.print(cr.champs.temperature);
Serial.print(F("  "));
    Serial.print(cr.champs.Ts); Serial.print(F(" sec\n"));

    // Attendre la prochaine période d'échantillonnage
    waitUntil(cr.champs.Ts * 1000);
  }
}

/* ---------------------------------------------------------------
   i2c_receiveFunc(int n)
   Cette fonction est exécutée à la réception des données venant
   du coordonnateur. Le paramètre n indique le nombre d'octets reçus.
   ---------------------------------------------------------------
   Note: Normalement on ne doit pas afficher des messages utilisant
         le port série - il y a risque de conflit entre les inter-
         ruptions. Donc, après débogage, n'oubliez pas de mettre les
         Serial print en commentaires ;-).
   --------------------------------------------------------------- */
void i2c_receiveEvent(int n) {
  // Traiter les commandes ou le numéro d'un registre (1 octet)
  if (n == 1) {
    // Un seul octet reçu. C'est probablement une commande.
    uint8_t data = Wire.read();
    switch (data) {
    case 0xA1:
        cmd = CMD::Stop;
        Serial.println(F("commande 'Arrêter' reçue"));
        break;
    case 0xA2:
        cmd = CMD::Go;
        cr.champs.nb_echantillons = 0;
        Serial.println(F("Commande 'Démarrer' reçue"));
        break;
    case 0xA5:
        cmd = CMD::Pause;
        Serial.println(F("Communication en Pause"));
        break;
    case 0xA6:
        cmd = CMD::Go;
        Serial.println(F("Communication en Redémarré"));
        break;

    default:
        // Sinon, c'est probablement un numéro de registre
        if ((data >= 0) && (data < NB_REGISTRES)) {
          adrReg = data;
        }
        else
          adrReg = -1; // Il y sans doute une erreur!
    }
  }
  else if (n == 2) {
    // Deux octets reçus. C'est probablement pour changer le
```

```
178     // taux d'échantillonnage.
179     uint8_t data1 = Wire.read();
180     uint8_t data2 = Wire.read();
181     if ((data1 == 0xA0) && (data2 >= MIN_Ts) && (data2 <= MAX_Ts)) {
182       Serial.println(F("Commande 'Changer Ts' reçue"));
183       cr.champs.Ts = data2;
184       Serial.print(F("La nouvelle valeur est: "));
185       Serial.print(cr.champs.Ts); Serial.println(F(" secondes"));
186     }
187   }
188   else {
189     // Ignorer la réception n > 2 octets.
190     Serial.println(F("Erreur: ce noeud n'accepte\
191     pas de communication/commande à trois octets"));
192   }
193 }
194 
195 /* ---------------------------------------------------------------
196    i2c_requestEvent()
197    Cette fonction est exécutée à la réception d'une requête de
198    données venant du coordonnateur.
199    ---------------------------------------------------------------
200    Note: Normalement on ne doit pas afficher des messages utilisant
201          le port série - il y a risque de conflit entre les inter-
202          ruptions. Donc, après débogage, n'oubliez pas de mettre les
203          Serial print en commentaires ;-).
204    --------------------------------------------------------------- */
205 void i2c_requestEvent(){
206   // Le coordonnateur veut la valeur d'un registre. L'adresse du
207   // registre a été reçue précédemment.
208   if ((adrReg >= 0) && (adrReg < NB_REGISTRES)){
209     Serial.print("");
210     // Envoyer le contenu du registre au coordonnateur
211     Wire.write(cr.regs[adrReg]);
212   }
213 }
214 
215 /* ---------------------------------------------------------------
216    Fonction pour créer un delai de w millisecondes
217 
218    La fonction delay() est utilisée dans bien des tutoriels pour
219    créer un delai temporel. On peut aussi créer notre propre délai
220    et utiliser une unité de temps à notre guise.
221    --------------------------------------------------------------- */
222 void waitUntil(uint32_t w) {
223   uint32_t t{millis()};
224   // Attendre w millisecondes
225   while (millis() < t + w) {}
226 }
227 
```