| | | |
|---|---|---|
| | **Téléinformation compteur EDF : affichage sur Rpi** | |

| NOMS : | Date : |
|---|---|
| | |

## Objectifs :

- Faire l'acquisition, sur une carte Raspberry Pi, des données produites par le compteur EDF et les afficher en mode console.

## Compétences abordées :

| | |
|---|---|
| Concevoir | C3.4 : Valider le choix d'une architecture matérielle/logicielle<br>C3.8 : Produire la documentation nécessaire à la conception du produit |
| Réaliser | C4.3 : Analyser la structure logicielle. Procéder aux modifications logicielles.<br>C4.5 : Tester et valider un matériel |
| Installer | C5.2 : Exécuter des mesures et tests appropriés. |

## Savoirs abordés :

| **Savoir** | **Description** |
|---|---|
| S3.1. Modélisation orientée objet | Objets, classes.<br>Constructeurs, destructeurs. |
| S4.1. Principes de base | Représentation et codage des informations.<br>Organisation des programmes : point d'entrée et arguments de la ligne de commande, prototypes, fonctions, paramètres, valeur de retour. |
| S4.2. Algorithmique | Modèle canonique de gestion d'E/S : ouvrir, lire, écrire, fermer<br>Bibliothèque standard (ANSI C) |
| S4.3. Structure et gestion des données | Structures de données et méthodes d'accès directe et/ou indirecte : tableau |
| S4.5. Programmation par flux de données | Définition des flux d'entrée et de sortie (signaux, données)<br>Définition des interfaces d'entrée, de sortie et de restitution de l'information |
| S4.6. Programmation orientée objet (Support : C++) | Instanciation d'objets [new, delete, etc.) |
| S4.7. Langages de programmation | C++<br>Utilisation d'un langage objet (Java, C#, C++, etc.) |
| S7.1. Concepts fondamentaux de la transmission | Codage, débit binaire |
| S8.1 Instruments de mesure | Analyseur logique |

## Moyens :

| | |
|---|---|
| – Compteur EDF<br>– Carte Raspberry Pi 2 avec configuration minimale<br>– Carte d'interfaçage entre Compteur EDF et Raspberry Pi | – Ordinateur disposant de Putty, VNC-Viewer, WinSCP, Fritzing.<br>– Analyseur logique Saleae ou Logicport.<br>– Un radiateur 500/1000Wh ou sèche cheveux. |

## Conditions :

- Travail en binôme.
- Durée : 1 séance de 4H
- Compte rendu remis en fin de séance.

## Prérequis :

- Avoir effectué le TP portant sur la téléinformation sur compteur EDF.
- Avoir effectué le TP portant sur la mise en service de la carte Raspberry Pi 2.
- Notions de base sur la communication UART.

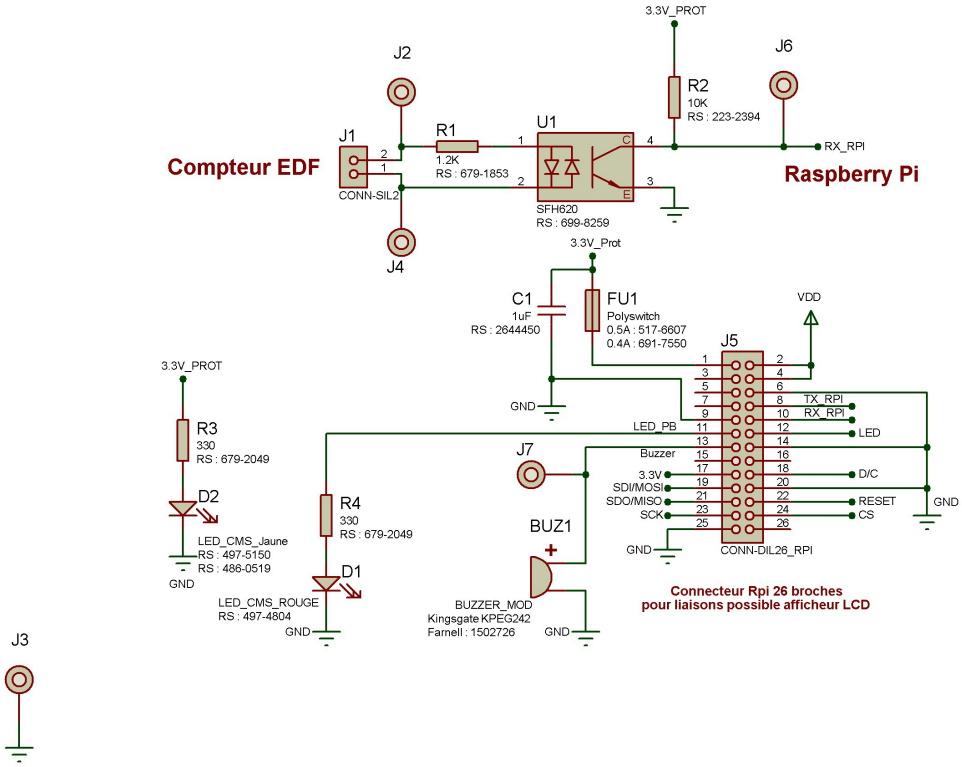
# Mise en situation

Le but de ce TP est :

- de récupérer les informations générées par la sortie de téléinformation d'un compteur EDF sur une carte Raspberry Pi 2, par l'intermédiaire de sa liaison série,
- de vérifier que le circuit SFH620 permet à lui seul d'effectuer l'interfaçage entre le compteur et la carte Rpi,
- d'afficher ces données brutes sur une console, puis de modifier leur présentation,
- d'effectuer les mêmes mesures en utilisant une programmation orientée objet,
- d'adapter le programme pour effectuer une alerte en cas de dépassement d'un seuil de consommation.

## Optocoupleur SFH620 utilisé comme interface entre le compteur EDF et la carte Rpi

- La documentation du compteur ainsi que les spécifications de la norme de transmission en sortie du compteur figurent sur le site.
- La documentation de l'optocoupleur figure sur le site.
- Le schéma de la carte d'interfaçage entre le compteur et la carte Rpi figure ci-dessous.
- La carte Raspberry Pi 2 doit disposer des configurations minimales. En particulier sa liaison UART doit être en service et la librairie wiringPi serial installée *(voir TP sur la mise en service de la carte Rpi)*.




| | | |
|---|---|---|
| | TITLE: Interface téléinformation compteur EDF Version 3 pour Rpi<br>BY: Hortolland C. REV: | DATE: 01/08/16<br>PAGE: 1/1 |

1 Disposer soigneusement la carte d'interfaçage sur la Rpi avant toute mise sous tension.

2 L'amplitude du signal en sortie du compteur étant au maximum de ±5V, calculer l'intensité maximale qui circule dans les LEDs de l'optocoupleur. Cette valeur est-elle compatible avec les valeurs indiquées dans la documentation du composant *(Doc sur le site)* ?

→

→

3 Utiliser la documentation du compteur EDF pour rappeler le format de transmission des données transmises par le compteur : vitesse, nombre de bits de chaque donnée, et bit de parité.

→

4 Configurer l'analyseur logique (Saleae ou Logicport) pour qu'il soit compatible avec le format de transmission du compteur.

5 D'après le schéma structurel quelle est la référence du picot permettant de relever le signal en sortie du photocoupleur ( = entrée UART de la carte Rpi) , et la référence du picot relié au 0V de la carte.

→ →

6 Visualiser le signal en sortie de l'optocoupleur avec un analyseur logique. Vérifier que les données affichées par l'analyseur logique sont conformes.

***Faites constater***

**Acquisition et affichage des données « brutes » en mode console sur Rpi**

7 Télécharger depuis le site du TP le fichier « Compteur_EDF_1.cpp » et le stocker dans un sous-répertoire /home/pi/Travail/TP/Teleinfo. Effectuer une copie du fichier sous un nom du type : «*EDF_1_votre_nom.cpp »,* copie sur laquelle le travail sera effectué.

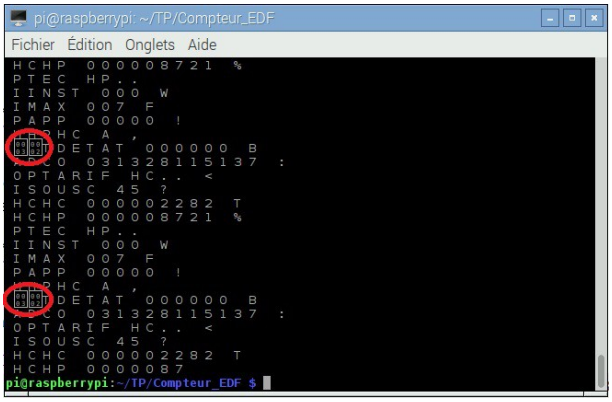
8 Mettre en œuvre le programme, son listing figure sur le Document ANNEXE 1. Une vidéo illustrant le résultat attendu figure sur le site *(Vidéo 1)*. Relancer le programme 2 ou 3 fois de suite pour voir si la capture est la même à chaque fois.

→

***Faire constater***

*A noter : des caractères ASCII non identifiables apparaissent dans les captures.*

*Ils permettent de repérer le début et la fin de chaque trame, comme indiqué dans le paragraphe 1.5.2 du document « Sorties de télé-information client des appareils de comptage électroniques utilisés par ERDF » qui figure sur le site.*



9 Surligner sur le Document ANNEXE 1 les extraits du programme permettant de prendre en compte les 2 informations : vitesse et nombre de bits.

Donner le nom de la technique utilisée pour ne garder que le nombre de bits utiles

→

Invalider momentanément le formatage du nombre de bits, pour voir quelle est l'incidence sur les données récupérées.

→

Le tableau des codes ASCII figure sur le site. Compte-tenu du format de transmission du compteur, quelle peut-être l'origine du problème ? *Le recours à l'analyseur logique peut également permettre de répondre à cette question.*

→

10 Analyser le programme et indiquer combien d'octets sont récupérés et affichés ?

→

Combien d'octets sont-ils vraiment nécessaires pour obtenir une capture complète de la trame ? →

Indiquer la méthode qui vous a permis de répondre à cette question *(plusieurs méthodes, et donc plusieurs bonnes réponses, sont possibles).*

→

**Synchronisation de la capture des données**

On souhaite faire des captures régulières de la trame transmise par le compteur. Leur affichage sera synchronisé sur la détection de la suite de caractères «IM» correspondant à la valeur de l'intensité maximale atteinte par le compteur depuis sa première mise en service.

11 Télécharger depuis le site le fichier « Compteur_EDF_2.cpp ». Le listing du programme figure sur le Document ANNEXE 2. Effectuer une copie du programme sous un nom du type : « *EDF_2_votre_nom.cpp »*, copie sur laquelle le travail sera effectué. Mettre en œuvre ce programme. Une vidéo illustrant le résultat attendu figure sur le site *(Vidéo 2)*.

On souhaite faire évoluer ce programme de la façon suivante :
- Il serait plus logique que la synchronisation se fasse sur le début de la trame, c'est à dire sur l'adresse du compteur.
- On souhaite que l'affichage ne soit plus défilant mais statique.
- Une vidéo illustrant le résultat attendu figure sur le site *(Vidéo 3)*.

12 Modifier le programme sous une version appelée « *EDF_2_mod_votre_nom.cpp »* pour qu'il fonctionne comme indiqué.
*A noter :*
- *la succession de caractères « A » suivi de « D » n'a qu'une seule occurrence.*
- *l'affichage statique sera obtenu en utilisant la « commande système » « clear » dans le programme. Consulter si nécessaire le site indiqué (onglet « liens d'aide à la programmation ») pour connaître la librairie qu'il est nécessaire d'inclure au projet.*

***Faire constater le bon fonctionnement***

**Localisation de chacun des caractères dans la trame.**
Dans les évolutions à venir du programme il va être très pratique de localiser chronologiquement où se trouvent certains caractères. Une vidéo illustrant le résultat attendu figure sur le site *(Vidéo 4)*.

13 Modifier le programme sous une version appelée « *EDF_2_mod_cpt_votre_nom.cpp »* pour qu'il fonctionne comme indiqué. Ajouter une variable (par exemple booléenne) pour que le programme s'arrête après une seule capture complète de la trame.

**Utilisation d'une classe « compteurEDF »**
La création d'une classe et des méthodes associées vont permettre l'utilisation de fonctions pour faire une capture de la trame, et en extraire certaines informations à volonté.

14 Télécharger le fichier Compteur_EDF_3.zip depuis le site. *(La version papier figure en ANNEXES 3, 4 et 5).* Le désarchiver dans le sous-répertoire de travail, effectuer la compilation et lancer l'exécution.

15 Analyser les différents fichiers. Surligner le constructeur et le destructeur d'objets de type compteurEDF. Surligner l'instanciation du compteur.

16 Modifier le programme afin que l'intensité instantanée soit ajoutée à l'affichage. Le prototype est donné, il reste à compléter la méthode et à l'utiliser dans le main(). Une vidéo illustrant le résultat attendu figure sur le site *(Vidéo 5)*. La variation de l'intensité consommée peut-être obtenue en agissant sur la puissance consommée par le radiateur ou le sèche cheveux. Relever les 2 seuils, ils seront utilisés dans la suite → →

**Détection d'un dépassement de seuil et avertissement par LED ou Buzzer**
On souhaite faire évoluer le programme de la façon suivante :
- La détection d'une consommation instantanée dépassant un premier seuil *(1ère vitesse du sèche cheveux ou du radiateur)* doit mettre en service la LED rouge.
- Le dépassement d'un second seuil doit mettre en service le buzzer seul.

Une vidéo illustrant le résultat attendu figure sur le site *(Vidéo 6)*.

17 Quelles broches du GPIO *(notation wiringPi)* correspondent respectivement à la LED et au Buzzer ?

→ LED : BUZZER :

18 Une broche du GPIO peut fournir au maximum 16mA. Justifier, par calcul si nécessaire, si la LED et le buzzer peuvent être correctement pilotés *(Documentations sur le site)*.

→
→

19 Modifier le programme pour qu'il fonctionne comme indiqué.
*A noter : pour la configuration et la commande du GPIO avec wiringPi il est conseillé de s'inspirer du fichier « blink.c » qui est fourni avec la librairie wiringPi et également téléchargeable sur le site.*

***Faire constater le bon fonctionnement***

20 Les fichiers des différents programmes modifiés sont à rendre en fin de séance.

## Document ANNEXE 1 : Programme *Compteur_EDF_1.cpp*

```
// Récupération des données d'un compteur EDF par Rpi
#include <iostream>
#include <string.h>
#include <errno.h>

#include <wiringPi.h>
#include <wiringSerial.h>

using namespace std;

// g++ -o Compteur_EDF_1 -lwiringPi Compteur_EDF_1.cpp  // Syntaxe de compilation

int main ()
{
 int fd ;

  if (wiringPiSetup () == -1)
 {
        cout<<"Impossible de démarrer wiringPi. Erreur : "<<strerror (errno);
        return 1 ;
 }

 if ((fd = serialOpen ("/dev/ttyAMA0", 1200)) < 0)
 {
        cout<<"Impossible d'ouvrir le port /dev/ttyAMA0. Erreur : "<<strerror (errno);
        return 1 ;
 }

char data;
for (int i(0); i<400; i++)
        {
        data = serialGetchar(fd);               // Lecture et stockage dans data
        data&=127;
        cout<<data<<" ";
        }
cout<<endl;

serialFlush(fd);                // Effacement buffers série avant lecture Rx
serialClose(fd);                // Fermeture du port série
return 0 ;
}
```

## Document ANNEXE 2 : Programme *Compteur_EDF_2.cpp*

```
// Récupération des données d'un compteur EDF par Rpi
#include <iostream>
#include <string.h>
#include <errno.h>
#include <wiringPi.h>
#include <wiringSerial.h>
using namespace std;

// g++ -o Compteur_EDF_2 -lwiringPi Compteur_EDF_2.cpp  // Syntaxe de compilation

int main ()
{
 int fd ;

 if (wiringPiSetup () == -1)
 {      cout<<"Impossible de démarrer wiringPi. Erreur : "<<strerror (errno);
        return 1 ;  }

 if ((fd = serialOpen ("/dev/ttyAMA0", 1200)) < 0)
 {      cout<<"Impossible d'ouvrir le port /dev/ttyAMA0. Erreur : "<<strerror (errno);
        return 1 ;  }

int Nb_captures=0;
char data[400];

while(1)
        {
        data[0] = serialGetchar(fd);            // Lecture et stockage dans data
        data[0]&=127;
        if (data[0]=='I')
                {
                data[1] = serialGetchar(fd);            // Lecture et stockage dans data
                data[1]&=127;
                if (data[1]=='M')
                        {
                        for (int i(2); i<170; i++)
                                {
                                data[i] = serialGetchar(fd);    // Lecture et stockage dans data
                                data[i]&=127;
                                }
                        Nb_captures++;
                        cout<<"Capture N° : "<<Nb_captures<<endl;
                        for (int i(0); i<170; i++) cout<<data[i]<<" ";
                        cout<<endl;
                        cout<<endl;
                        }
                }
        }
serialFlush(fd);                // Effacement buffers série avant lecture Rx
serialClose(fd);                // Fermeture du port série
return 0 ;
}
```

## Document ANNEXE 3 : Fichier *compteurEDF.h*

```
#pragma once

class compteurEDF
{
int fd;             // File Descriptor
char* ADCO;
char* IINST;
char* IMAX;
char* PAPP;
char* HCHC;
char* HCHP;

char data[170];          // 170 octets sont transmis dans chaque trame

public :
        compteurEDF();
        ~compteurEDF();
        void set_fd(int);
        int get_fd();
        void set_data(int, char);
        char get_data(int);
        void affiche_ADCO();
        void affiche_IINST();
};
```

## Document ANNEXE 4 : Fichier *compteurEDF.cpp*

```
#include "compteurEDF.h"
#include <string>
#include <iostream>
using namespace std;


compteurEDF::compteurEDF()
{
fd=0;
for(int i(0); i<170; i++) data[i]=0;
ADCO = new char[13];
IINST = new char[6];
IMAX = new char[6];
PAPP = new char[9];
HCHC = new char[13];
HCHP = new char[13];
}

compteurEDF::~compteurEDF()
{
delete ADCO;
delete IINST;
delete IMAX;
delete PAPP;
delete HCHC;
delete HCHP;
}

void compteurEDF::set_fd(const int val)
{
fd=val;
}

int compteurEDF::get_fd()
{
return fd;
}

void compteurEDF::set_data(int ad, char val)
{
data[ad]=val;
}

char compteurEDF::get_data(int ad)
{
return data[ad];
}

void compteurEDF::affiche_ADCO()
{
for (int i(0); i<17; i++) cout<<data[i]; // Récupération des octets correspondant à ADCO
cout<<endl;
}
```

```
void compteurEDF::affiche_IINST()
{




}
```

## Document ANNEXE 5 : Programme *Compteur_EDF_3.cpp*

```
#include <iostream>
#include <string.h>
#include <errno.h>
#include <stdlib.h>     // system
#include <wiringPi.h>
#include <wiringSerial.h>
#include "compteurEDF.h"

using namespace std;

// g++ -o Compteur_EDF_3 -lwiringPi Compteur_EDF_3.cpp compteurEDF.cpp

int main ()
{
compteurEDF compt1 ;          //

  if (wiringPiSetup () == -1)
  {
    cout<<"Impossible de démarrer wiringPi. Erreur : "<<strerror (errno);
    return 1 ;
  }

compt1.set_fd(serialOpen ("/dev/ttyAMA0", 1200));
  if (compt1.get_fd ()< 0)
  {
          cout<<"Impossible d'ouvrir le port /dev/ttyAMA0. Erreur : "<<strerror (errno);
          return 1 ;
  }

int Nb_captures=0;
```

```
char data[400];

while(1)
        {
        do
                {
                do
                        {
                        compt1.set_data(0,127&serialGetchar(compt1.get_fd ()));  // Capture et stockage du 1er caractère ...
                        }
                while (compt1.get_data(0)!='A');        // ... jusqu'à capture d'un 'A'
                compt1.set_data(1,127&serialGetchar(compt1.get_fd ()));  // Capture et stockage du 2ème caractère
                }
        while ((compt1.get_data(0)!='A')||(compt1.get_data(1)!='D'));         // ... jusqu'à capture d'un 'A' suivi d'un 'D'

        for (int i(2); i<170; i++)
                {
                compt1.set_data(i,127&serialGetchar(compt1.get_fd ()));  // Capture et stockage de tous les autres caractères de la trame
                }
        Nb_captures++;
        system("clear");
        cout<<"Capture N° : "<<Nb_captures<<endl;
        compt1.affiche_ADCO();                          // Affichage de l'adresse du compteur
        }

serialFlush(compt1.get_fd ());                  // Effacement buffers série avant lecture Rx
serialClose(compt1.get_fd ());                  // Fermeture du port série
return 0;
}
```