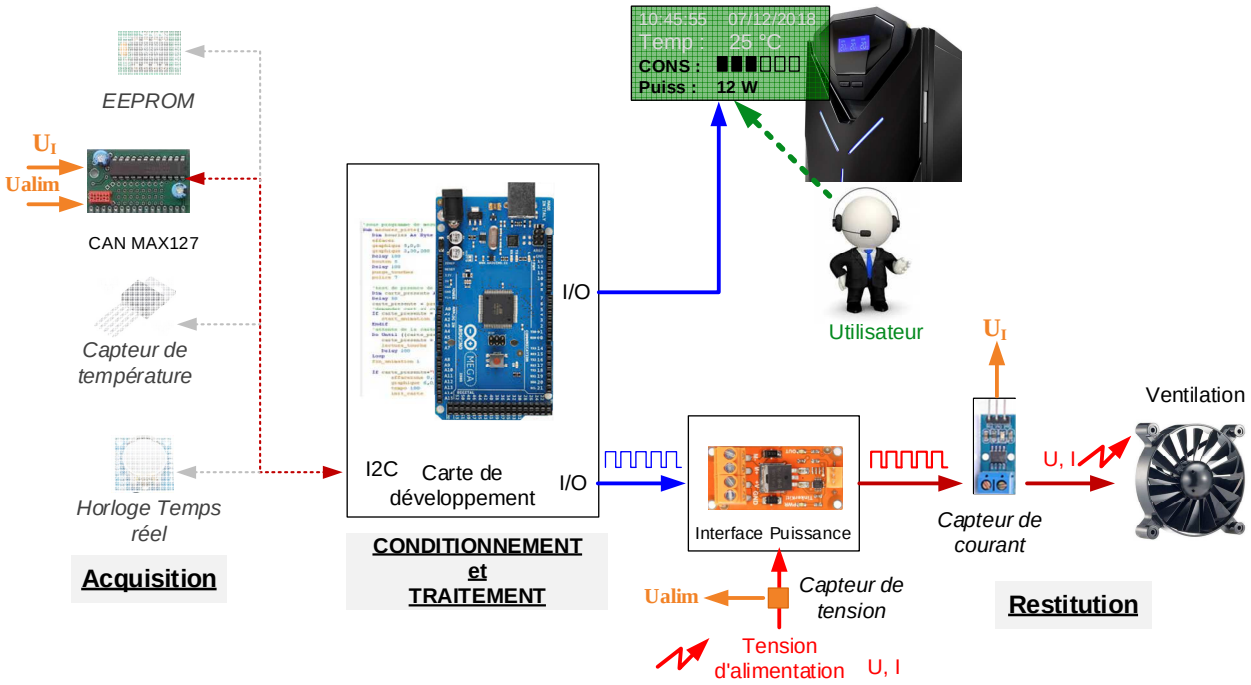
♦ PRESENTATION DU PROJET :

On désire **Mesurer** la puissance électrique consommée par le montage **toute les secondes**. Pour cela, il vous faudra mettre en place l'acquisition et la conversion du courant consommé par la partie puissance grâce à un capteur à effet hall et un convertisseur A/N sur bus I2C.

Conditions de fonctionnement :

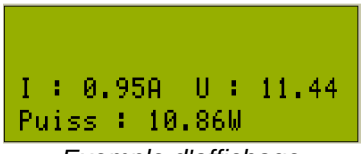
- Mesure et affichage du courant consommé toutes les secondes,
- Mesure du courant et affichage de la puissance électrique consommée toute les secondes



Problématique posée : *Comment réaliser un bilan énergétique sur un système microprogrammé ?*

L'objectif est donc de mesurer et d'afficher en **temps réel** la valeur du courant, de la tension et de calculer ainsi la puissance électrique instantanée :

- Affichage sur la 3ème ligne de la valeur du courant en Ampère (A) et de la tension (V),
- Affichage sur la 4ème ligne de la puissance électrique en watt (W),



Exemple d'affichage

Cette Activité est évaluée par revue de projet individuelle !

♦ Programmation sous Arduino du circuit I2C MAX127

La bibliothèque **Wire** permet de communiquer sur le bus I2C en Arduino. Il est nécessaire de l'importer dans votre programme :

```
#include <Wire.h>
```

Description des fonctions I2C :

Wire.**begin**() : initialise le mode de communication I2C. À écrire une fois dans la procédure **setup**( )

Wire.**beginTransmission**(**adress**) : débute une transmission avec un circuit I2C. Cette fonction permet de prendre la main sur le bus, et d'indiquer, par la variable **adress,** avec quel circuit l'on désire communiquer.

Wire.**requestFrom** (***adress***, ***quantity***, ***STOP***) : Requête envoyée à l'esclave (***adress***) lui demandant de placer sur le bus I2C les données de son ou de ses registres. Le bit R/W est alors automatiquement mis à 1 (Lecture).

Quantity correspond au nombre **d'octets** à lire. ***STOP*** est un booléen (True/False) indiquant si l'on doit placer un stop après exécution de la requête pour libérer le bus.

Wire.**available**() : vérifie si les données sont disponibles sur le BUS après le lancement d'une requête. Cette fonction renvoie le nombre d'octets disponibles.

Wire.**read**() : Permet de lire un ou plusieurs octets placés par l'esclave sur le bus. Cette fonction ne peut fonctionner que si un début de transmission est effectué et si les données sont disponibles.

Wire.**endTransmission**( ) : Termine une transmission en cours sur le bus I2C. Cette fonction permet de libérer le bus en plaçant un STOP sur la trame.

Wire.**write**() : Permet d'écrire une donnée sur le bus. Cette fonction ne peut fonctionner que si un début de transmission est effectué. Le bit R/W est alors automatiquement mis à 0.

Programmation Lecture de la valeur de conversion N : procédure **getCAN**

L'algorithme de la procédure **int getCAN (byte canal)** est le suivant :

data1 et data2 sont des entiers signés sur 16 bits

Début de la transmission I2C avec adresse MAX127_I2C_adresse (START)

*Écriture de **registre** : conversion sur l'entrée **canal***

Fin de transmission (STOP)

Début de la transmission I2C avec adresse MAX127_I2C_adresse (START)

Requête de lecture des 2 octets du registre avec STOP final

SI Données disponibles = 2 ALORS
*   **data1** = donnée lue sur bus I2C*
*   **data2** = donnée lue sur bus I2C*
SINON
*   transmission série "Erreur Lecture CAN !"*
FIN

Fin de transmission (STOP)

décalage de 4 bits vers la gauche de data1

décalage de 4 bits vers la droite de data2

***N** = (data1 OU data2) ET % 0000 1111 1111 1111*

Retourner N

Remarques :

⇨ Le **ET logique** avec le **OU logique** de **data1** et **data2** permet de reconstruire la donnée de conversion N sur 12 bits en éliminant le quartet de poids fort inutilisé lié aux 16 bits nécessaires pour identifier **Nconv** (entier codé sur 16 bits). **Cette méthode n'est pas obligatoire, elle assure simplement la bonne mise à 0 des bits non-utilisés lors du décalage !**

⇨ L'opérateur de décalage en Arduino est **>>** pour un décalage à droite et **<<** pour un décalage à gauche :

```
int val  = 0b1111000011110000;
val = val << 4  ;                    \\val = 0b0000111100000000 : décalage de 4 bits à gauche
```

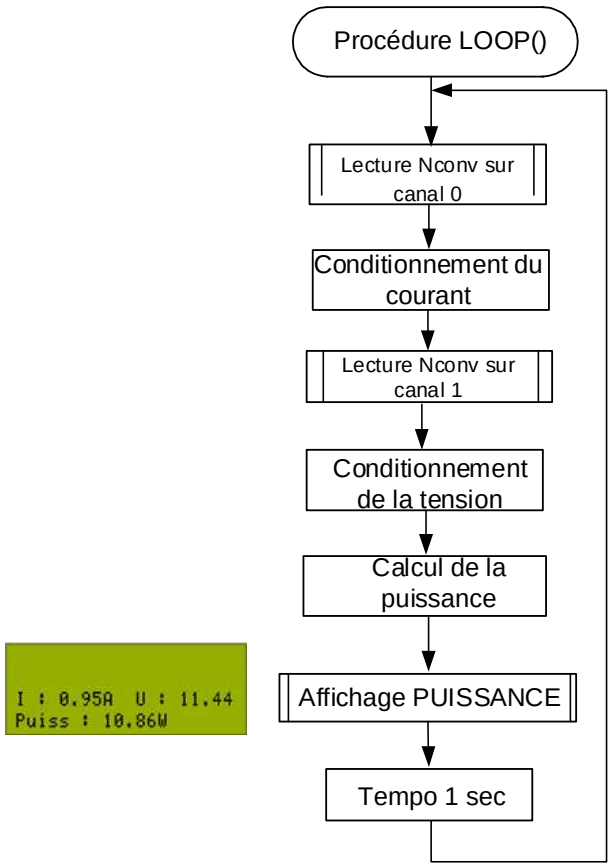
Exemple : la conversion donne la valeur $**89B** -> % **1000 1001 1011**

Conversion :	data1 = % 0000 0000 **1000 1001**
	data2 = % 0000 0000 **1011 0000**
décalage de data1 :	data1 = % 0000 **1000 1001** 0000
décalage de data2 :	data2 = % 0000 0000 0000 **1011**
OU logique entre data1 et data2	: % 0000 **1000 1001 1011**
suppression avec ET logique	: % **0000 1111 1111 1111**
Ce qui donne pour Nconv	: % **0000 1000 1001 1011** donc bien $**89B**

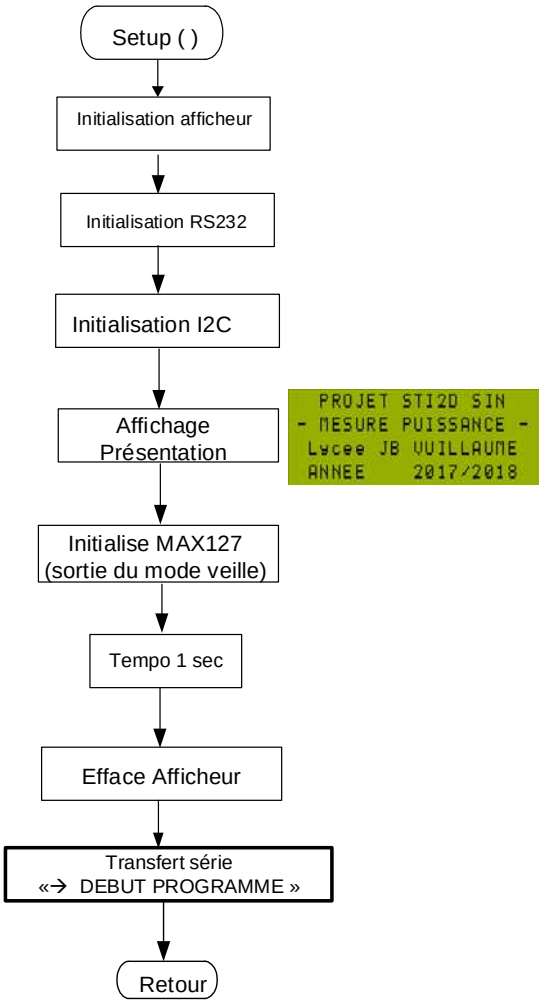
⇨ Le **courant** sera mesuré sur le **canal 0** et la **tension** sur le **canal 1** pour le MAX127

♦ ORGANIGRAMMES :

Programme PRINCIPAL (procédure LOOP) :



SP Initialisation () :



♦ REALISATION du PROGRAMME :

☞ Ouvrez le programme « *TP3_Arduino_Mega.ino* » contenu dans le dossier de ressources « *\programme* »

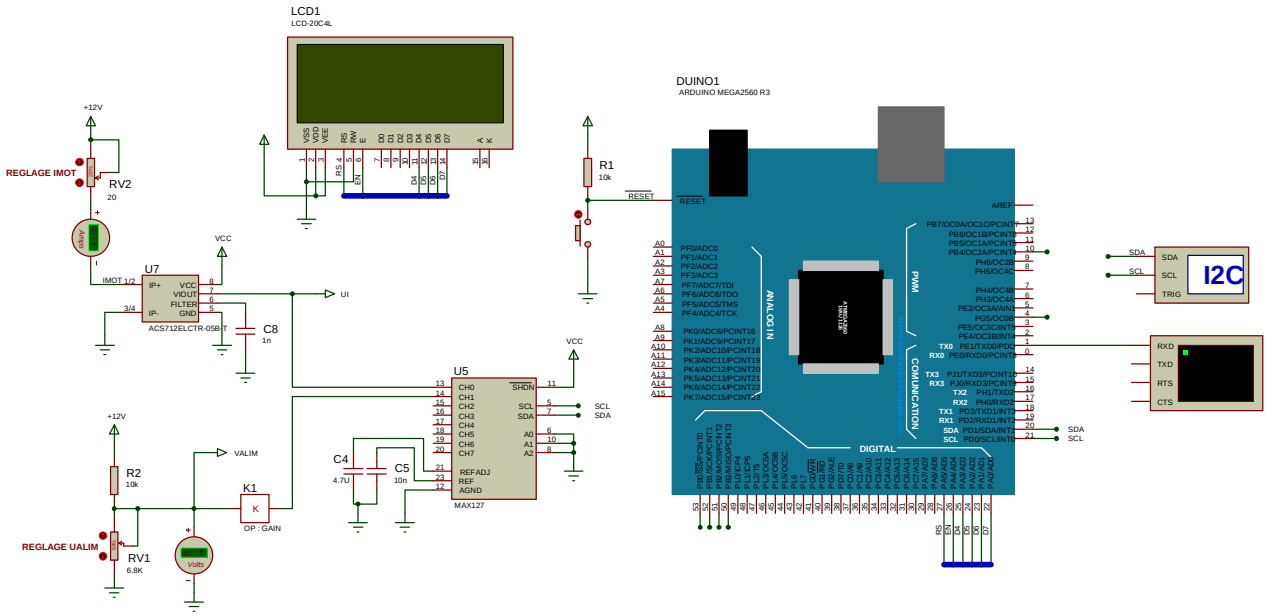
Celui-ci est organisé en Onglets :

☞ Ouvrez le fichier Arduino puis complétez les différentes parties du programme en vous aidant du cahier des charges, à savoir :

- Procédure d'initialisation **setup**()
- Programme principal **loop**()
- Procédure **affichePuissance** (*double* ***puissance***, *double* ***courant***, *double* ***tension***)
- Fonction int **getCAN** (byte **canal**)

♦ SIMULATION et MISE AU POINT du PROGRAMME :

☞ Ouvrez le fichier de simulation **Proteus ISIS** « *TP3-MESURE PUISSANCE.DSN* » contenu dans le dossier de ressources « \simulation »



☞ Placez le programme ***TP3_Arduino_Mega.ino.HEX*** dans le modèle de simulation Arduino, et simulez le montage. Mettre au point le programme afin de répondre au cahier des charges.

TRAVAILLEZ PAR ÉTAPE !!! Utilisez les opérateurs /* et */ pour mettre en commentaire les fonctions ou parties de programme en attentes d'écriture.

Utilisez le débogueur I2C pour analyser le contenu des trames !