PROJET IMA4 SA-SC 2016-2017

Département Informatique - Microélectronique - Automatique

Réseau de capteur sur smartphone

Elèves : Wenyu SUN
Xinyue XU

Tuteurs : Alexandre BOE
Xavier REDON
Thomas VANTROYS

Introduction

Notre projet se situe au cœur d'une problématique de la surveillance en temps réel à côté du portable. Les smartphones et les applications existantes peuvent mettre à jour les informations d'environnement, mais avec l'accès d'internet. Même si le réseau est très répandu ces jours, il existe encore des problèmes de perdu du réseau. C'est pourquoi nous avons choisi ce sujet *Réseau de capteur sur smartphone*.

Ce sujet permets d'obtenir directement les informations d'environnement en mesurant avec un capteur, par exemple le température, le PM 2.5, etc. Celui-ci peut fonctionner sans considérant la condition du réseau, et assurer la fiabilité et la sécurité de la transmission données.

Dans ce rapport, nous commençons par faire une présentation plus détaillée du cahier des charges. Ensuite nous développons le déroulement du projet en deux parties : la partie électronique et la partie informatique. Enfin, nous finissons en tirant un bilan de ce projet, avec les difficultés rencontrées et les améliorations possibles.

I. Présentation du projet

I.1 Objectif du projet

L'objectif de ce projet est de réaliser une application qui affiche la température actuelle sur smartphone. La température doit être mesurée par le capteur et stocké dans le micro-contrôleur en données binaires. Ensuite, le micro-contrôleur doit lier avec le portable utilisant le câble USB afin d'envoyer la température avec la liaison série. Enfin, il faut construire une application qui peut lire la température quand on demande.

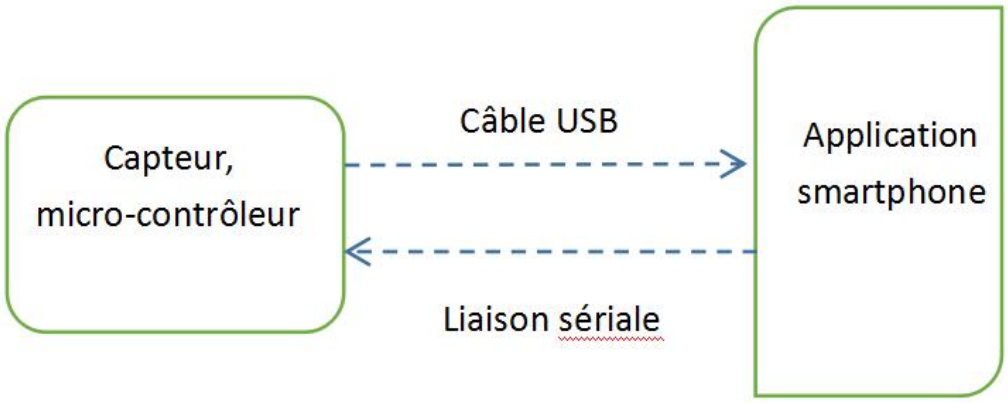


Figure 1. Concept de connection entre partrie électronique et laquelle informatique

On convertis la température(analogique) en digital tous les temps mais le update seulement quand le nouveau valeur est différent que l'ancien. On n'envoie pas le valeur que quand l'utilisateur demande de savoir le température. Quand l'utilisateur ne demande pas, le ADC du micro-contrôleur marche dans arrière-plan.

I.2 Choix techniques

matériel:

- Capteur de température
- Microntrôleur ATMega32U4

  On choisit le Atmega32u4 comme le microcontrôleur grâce à sa fonction de USB avec les broches UGND, UVcc, D- et D+.

Logiciel:

- Altium Designer

Pour dessiner le PCB.

- IDE

  IDE nous permets de tester Atmega32u4 avec Arduino Uno comme un ISP, et aussi de programmer le Atmega32U4 (Arduino Léonardo) comme AVR.
- Android Studio

  Pour construire l'application android.

II. Avancement du projet

II.1 Partie Électronique

II.1.1 Construction de la carte électronique

La carte électronique sert à lier le capteur température avec le Atmega32u4, et lier le câble USB avec Atmega32u4. On doit ajouter les composants électroniques (résistance, capacité, etc.) pour régler la tension d'alimentation et satisfaire les conditions de travail de Atmega32u4. On utilise le logiciel Altium Designer dans cette partie.

D'abord, on dessine le câblage entre les composantes qu'on a choisit. On prends le câblage de Arduino UNO comme la référence.[1] On a crée la library schématique pour Atmega32u4 et le capteur.

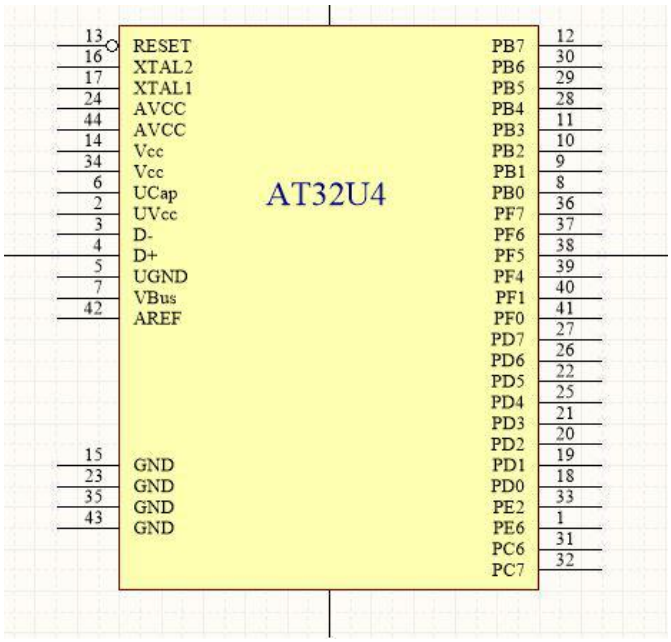


Figure 2. Library schématique Atmega32u4

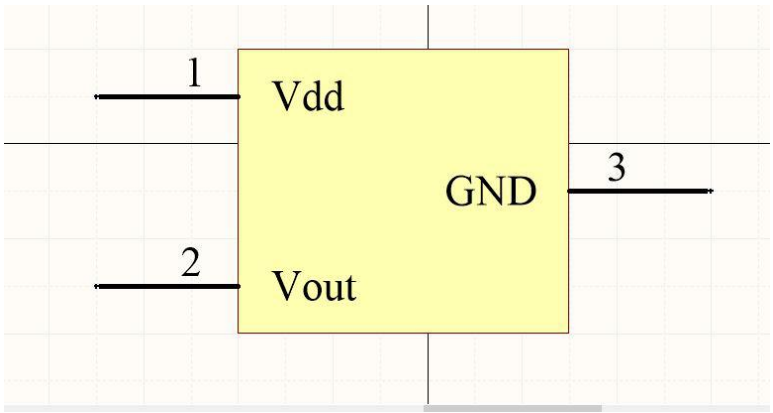


Figure 3. Library schématique capteur TC1047

[1] https://www.arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf

On dessine le schéma comme suivant.

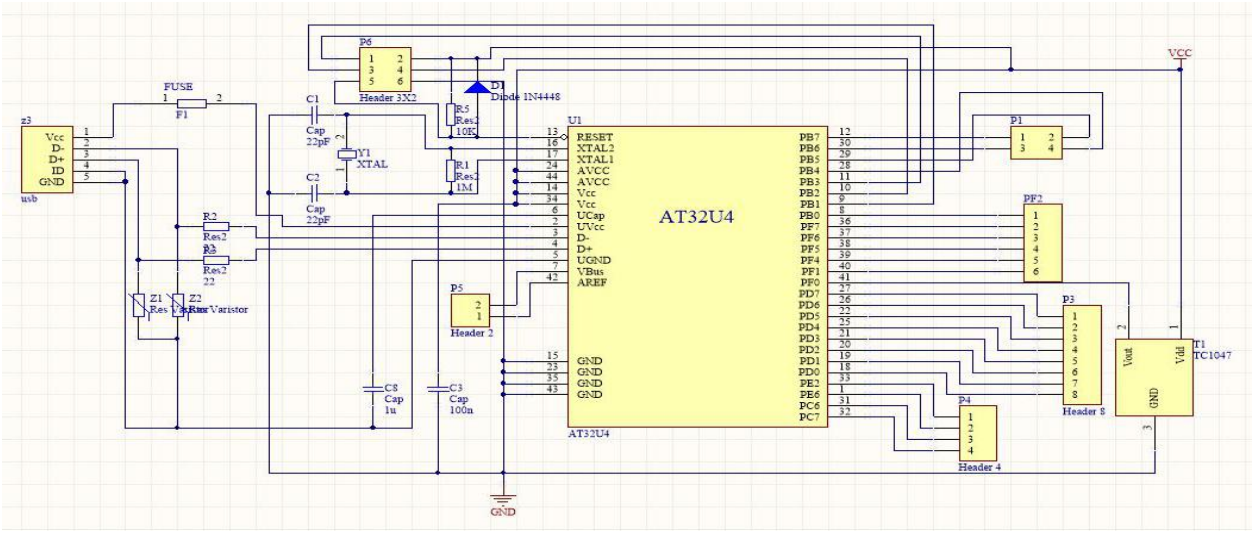


Figure 4. Dessin schématique

Ensuite, on crée le library de PCB de Atmega32u4, le capteur et le connecteur USB.

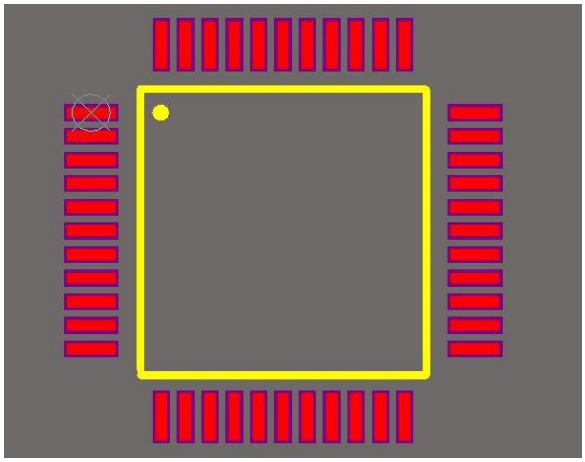


Figure 5. Library PCB Atmega32u4

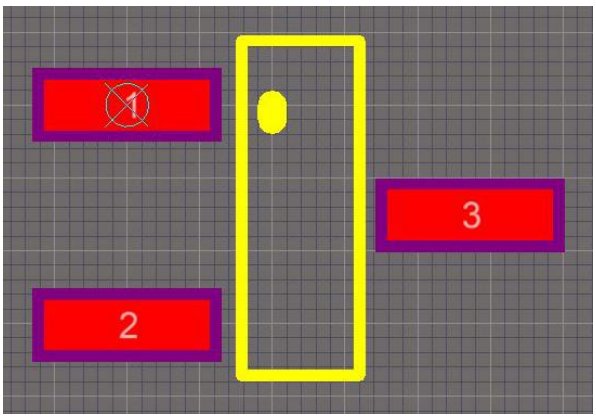


Figure 6. Library PCB capteur TC1047

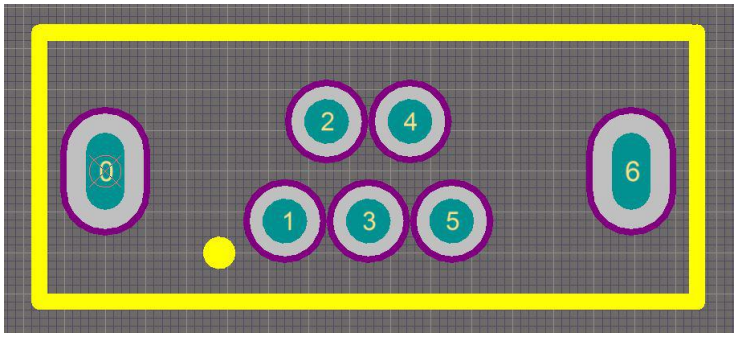


Figure 7. Library PCB USB connecteur

Avec les correspondance entre le library schématique et PCB, on transfère le dessin schématique à le dessin PCB, et on organise les câblage sur la carte. On fait la PCB de une seule couche, et on mis tous les headers et les insertions sur le bottom couche de la carte. Il y a 3 câbles qui ne peuvent pas être organisés, donc on les soude à la main.

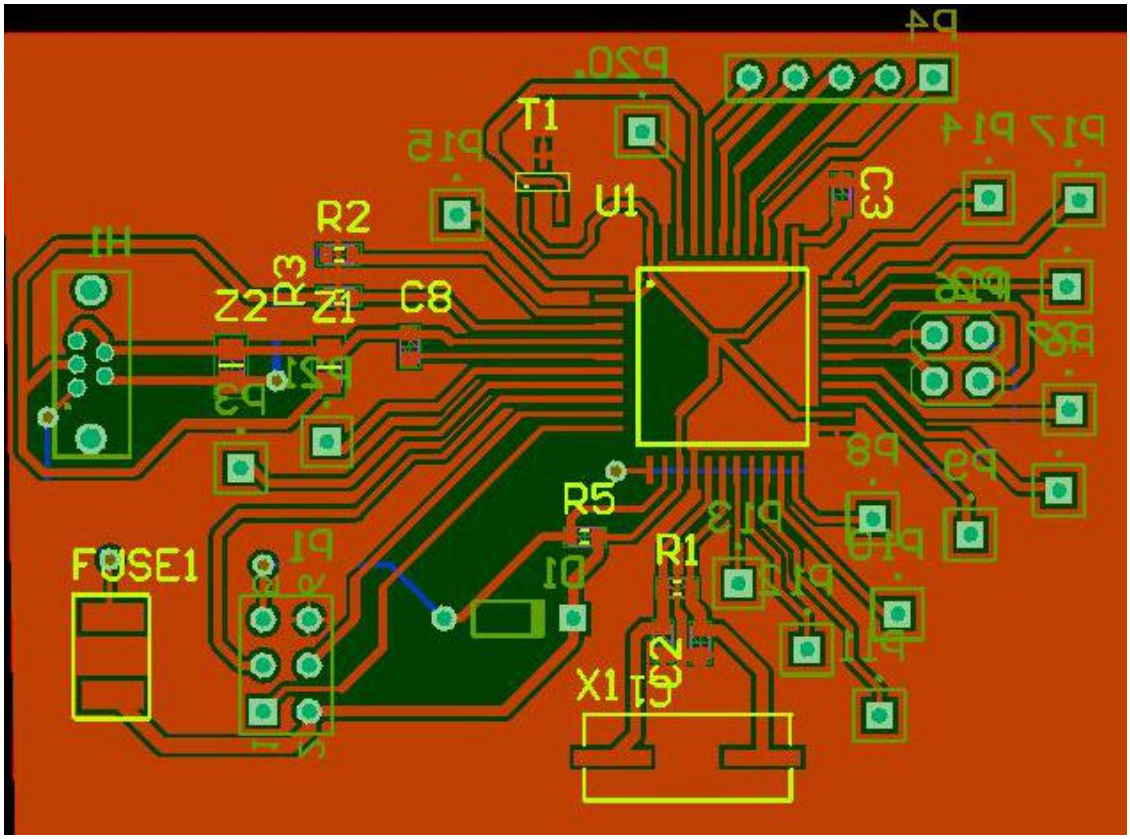


Figure 8. Dessin PCB

II.1.2 Corriger la carte

Pendant le test de notre carte, nous avons rencontré quelques problèmes et donc fait quelques changements sur notre carte.

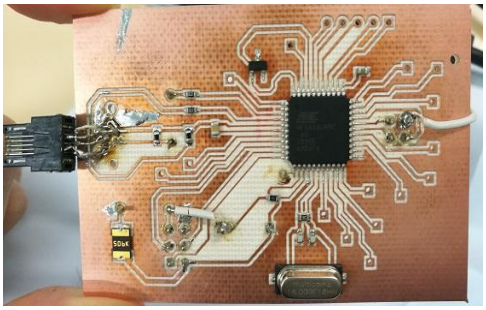
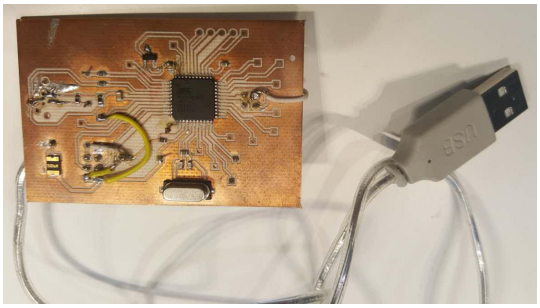



Figure 9. La carte avant et après modéfication

- **Ajout des câbles sur Vbus et Ugnd de Atmega32u4**

Quand on essaye de connecter le Atmega32u4 avec l'ordinateur, nous ne peut pas trouver ce chip avec la commande 'lsusb'. On trouve que quand on lie le carte avec un port USB, il n'est pas alimenté. Nous avons donc trouvé que le pin Vbus et Ugnd sur Atmega32u4 qui lie avec la porte doivent être respectivement 5V et à la masse.

- **Ajout de la résistance sur le capteur**

Quand on test la fonction du capteur, on trouve que la conversion entre la tension et le température fonctionne bizarrement. On a alors trouvé que l'alimentation du capteur est 5V , supérieur à sa portée 4.4V, donc on a ajouté un résistance de

II.2 Partie Informatique

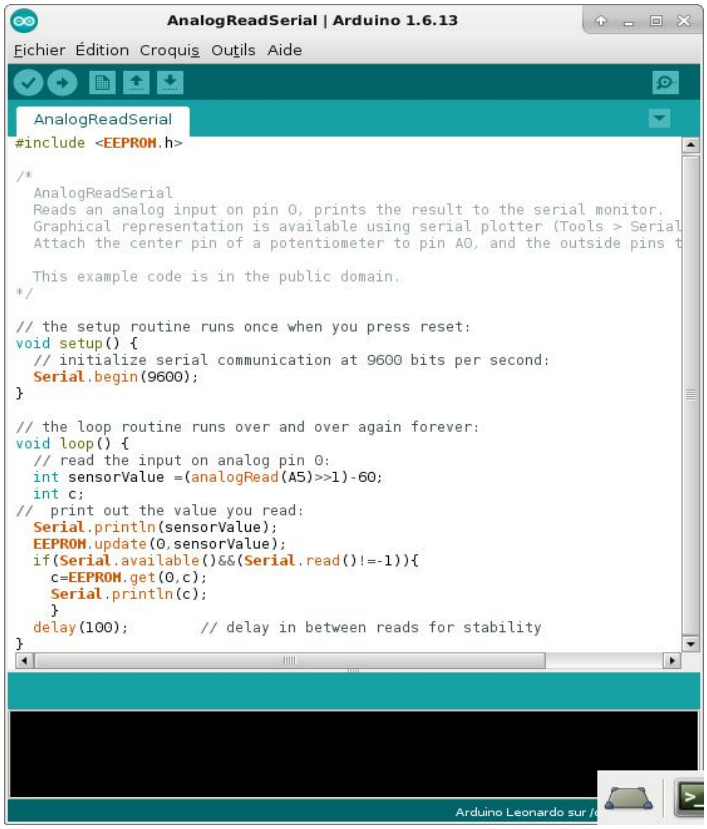
Pour la partie informatique, il y a deux côtés.

Du côté de la carte, il faut qu'on écrit un programme pour la carte qui lui permet de lire les données du capteur température et les sauvegarder dans eeprom. Une fois il est connecté avec le smartphone et il reçoit un message du smartphone, il va lire les données de température sauvegardé dans eeprom et les envoyer à le smartphone.

Du côté du smartphone, il faut qu'on crée un application sous android studio pour connecter le smartphone avec la carte et le demander d'envoyer les données de la carte à le smartphone. Une fois il reçoit les données, il les afficher sur l'écran.

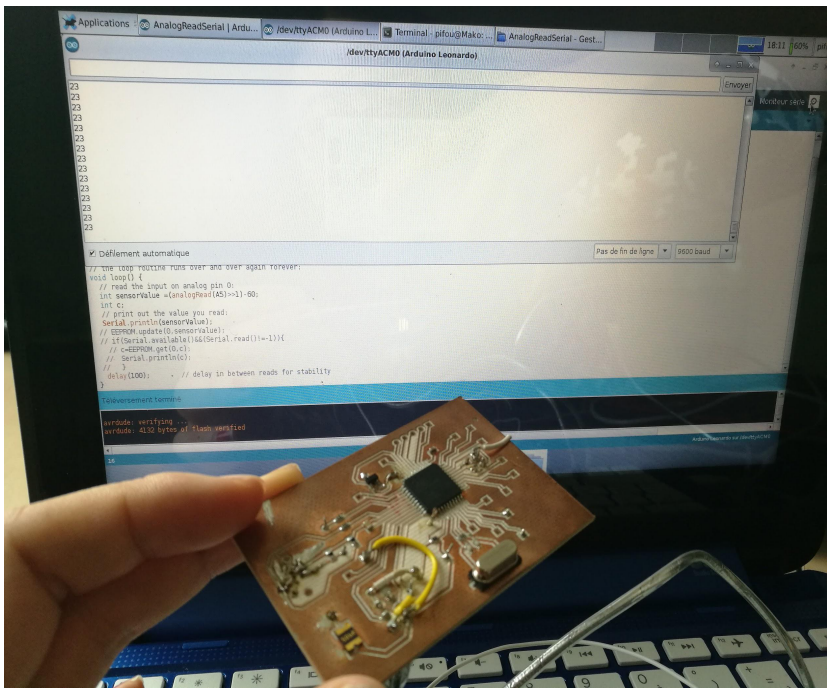
II.2.1 Programmation pour la carte Leonardo

Nous avons d'abord écrit un fichier de code c et un makefile pour coder la carte, mais quand on a rencontré beaucoup de problèmes, par exemple, il n'y a pas de library pour eeprom quand on les compilé. Après essayer plusieurs façons, on a réussi à réaliser make upload mais nous avons pas bien vu les résultats. Donc finalement nous avons décidé d'utiliser le logiciel IDE pour écrire notre program leonardo.

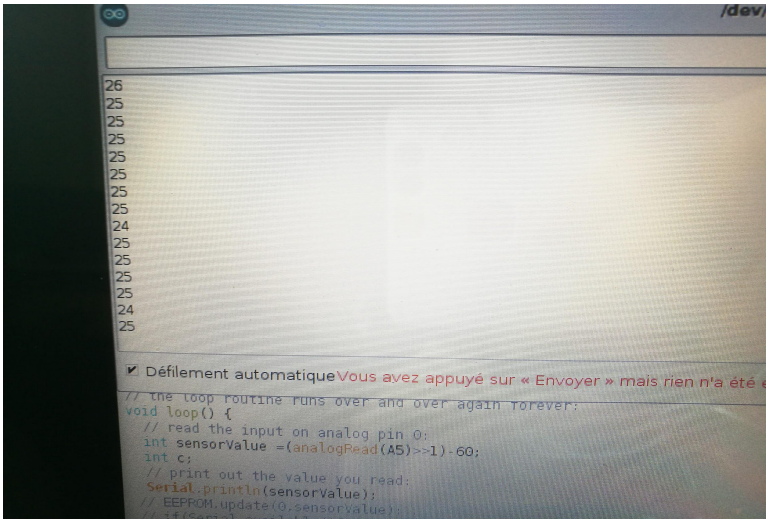


Un des avantages d'utiliser IDE est qu'on n'a pas besoin d'initialiser ADC manuellement. Nous avons utilisé ADC0, pour cela c'est tout simplement d'écrire 'analogRead(A5)'.

Nous avons testé notre carte leonardo et on a vu que la température est 23 degré.



Après, on a mis une main chaud proche et loin de la capteur et on a vu que les données varient bien correspondant.



II.2.2 Créer l'application sous Android Studio

Nous avons écrit un programme de android pour réaliser l'application dans le smartphone qui permet de lire les données de la température envoyées par notre carte et les afficher sur l'écran.

Le flux de notre programme est comme suivante:

Oncreate()

↓

Initialiser widgets et commencer Broadcast Receiver

↓

Attendre l'utilisateur cliquer connection

↓

Requête tous les périphériques connectés

↓

Vérifier ID de vendor correspondant

↓

Si notre carte est trouvée, demander user permission d'acceder à ce device

↓

Si permission est accordé, ouvrir l'appareil

↓

Créer la connexion série et définir des paramètres

↓

Une fois ça connecté, on clique 'get' pour envoyer un message à la carte

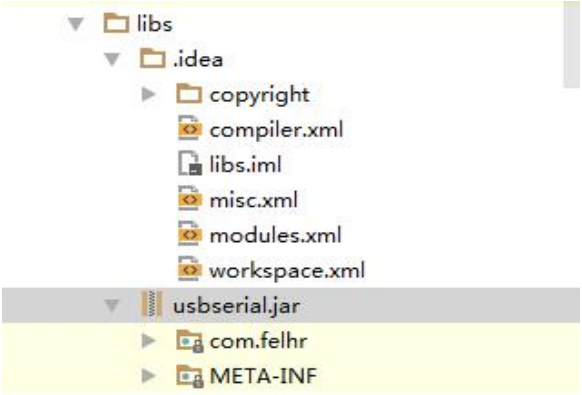
↓

On recevoit les données de la température et les affiche

Pour réaliser notre program de android studio, nous avons:

- **Ajouter library dans Android pour réaliser la connection série**

La mise en place d'une connexion série dans Android est vraiment compliqué, car il faudra que nous configurez manuellement beaucoup de choses. Nous avons cherché quelques bibliothèques qui font tout cela automatiquement, nous avons finalement trouvé un UsbSerial Library créé par l'utilisateur FelHR85 de Github[2]. On a téléchargé ce libraire et le déplacé dans le dossier 'libs' de notre projet android . Après, on a ajouté ce bibliothèque dans notre projet.

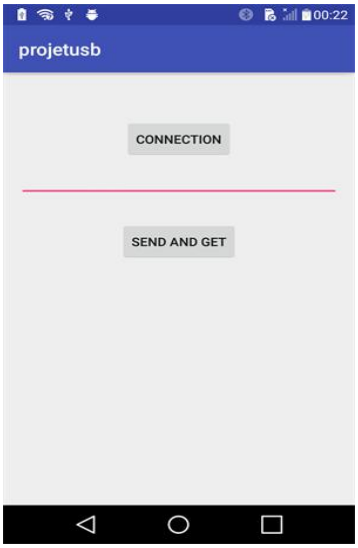


- **Définir Layout pour l'application**

Dans notre projet android, il y a trois fichiers principaux: MainActivity.java, activity_main.xml et AndroidManifest.xml.

Nous avons écrit le fichier activity_main.xml pour la mise en place de l'application[3]. On a créé 2 boutons, l'un des deux s'appel 'connection', pour demander la connection série entre le portable et la carte par la porte USB. L'autre bouton s'appel 'send and get', pour envoyer le message on a écrit à la carte et recevoir les données envoyées par la carte. Nous avons aussi créé un 'TextView' pour afficher les paramètres et un 'EditText' qui permet d'écrire un petit message.

L'interface de notre l'application est comme la figure suivante:



[2] réference: https://github.com/felHR85/UsbSerial/
[3] réference: http://mathias-seguy.developpez.com/tutoriels/android/utiliser-toolbar/

- **Donner la permission pour l'application**

Il existe déjà quelques permissions données à l'application, nous avons juste ajouté une commande dans le fichier AndroidManifest.xml :

```
<uses-feature android:name="android.hardware.usb.host" />
```

pour donner la permission de smartphone de devenir un hôte de USB.

Pour faire correspondre les propriétés du périphérique (notre carte leonardo), on a ajouté un dossier nommé 'xml' dans répertoire src / Main / res et met le texte dans un fichier 'device_filtre':

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <usb-device vendor-id="9025" />
    <!-- Vendor ID of Arduino -->
</resources>
```

(Le vendor_id est ce qu'on obtient par taper 'lsusb' sur linux avec notre carte 2341:8036 Arduino SA Leonardo (CDC ACM, HID))

Et on ajoute un ligne `android:resource="@xml/device_filter" />` pour indiquer au compilateur qu'il peut trouver les propriétés de l'appareil dans un fichier nommé device_filter dans src / main / res / xml .

- **réaliser les fonctionnements d'application**

C'est la partie plus importante pour notre projet andoid. Nous avons édit le fichier MainActivity.java pour cette partie.

D'abord, pour réaliser la connection, il faut cliquer le bouton 'connection'. Donc pour ce bouton on a créé une action: Lorsqu'il est cliqué, il devrait rechercher tous les périphériques connectés, puis vérifier si l'ID du fournisseur Arduino correspond à celle d'un périphérique connecté. S'il l'avez trouvé, l'autorisation doit être demandée à l'utilisateur.

```
public void onClickStart(View view) {
    tvAppend(textView, "commence connection!\n");
    HashMap<String, UsbDevice> usbDevices = usbManager.getDeviceList();
    if(usbDevices.isEmpty())
        tvAppend(textView, "usb device is empty!\n");
    else if (!usbDevices.isEmpty()) {
        boolean keep = true;
        for (Map.Entry<String, UsbDevice> entry : usbDevices.entrySet()) {
            device = entry.getValue();
            int deviceVID = device.getVendorId();
            if (deviceVID == 0x2341)//   Vendor ID
            {
                PendingIntent pi = PendingIntent.getBroadcast(this, 0, new Intent(ACTION_USB_PERMISSION), 0);
                usbManager.requestPermission(device, pi);
                keep = false;
            } else {
                tvAppend(textView, "wrong!\n");
                connection = null;
                device = null;
            }
            if (!keep)
                break;
        }
    }
```

Après nous définissons le 'BroadcastReceiver' pour recevoir la diffusion pour demander l'autorisation de l'utilisateur et aussi pour lancer la connexion automatiquement lorsqu'un périphérique est connecté et pour fermer la connexion lorsqu'il est déconnecté.

```
private final BroadcastReceiver broadcastReceiver = new BroadcastReceiver() { //Broadcast Receiver to automatically start an
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals(ACTION_USB_PERMISSION)) {
            boolean granted = intent.getExtras().getBoolean(UsbManager.EXTRA_PERMISSION_GRANTED);
            if (granted) {
                connection = usbManager.openDevice(device);
                serialPort = UsbSerialDevice.createUsbSerialDevice(device, connection);
                if (serialPort != null) {
                    if (serialPort.open()) { //Set Serial Connection Parameters.
                        setUiEnabled(true);
                        serialPort.setBaudRate(9600);
                        serialPort.setDataBits(UsbSerialInterface.DATA_BITS_8);
                        serialPort.setStopBits(UsbSerialInterface.STOP_BITS_1);
                        serialPort.setParity(UsbSerialInterface.PARITY_NONE);
                        serialPort.setFlowControl(UsbSerialInterface.FLOW_CONTROL_OFF);
                        serialPort.read(mCallback);
                        tvAppend(textView, "Serial Connection Opened!\n");
                    } else {
                        Log.d("SERIAL", "PORT NOT OPEN");    //debug
                    }
                } else {
                    Log.d("SERIAL", "PORT IS NULL");          //debug
                }
            } else {
                Log.d("SERIAL", "PERM NOT GRANTED");
            }
        } else if (intent.getAction().equals(UsbManager.ACTION_USB_DEVICE_ATTACHED)) { onClickStart(startButton);}
    }
```

Une fois il réalise la connection, l'utilisateur peut écrire un message et cliquer 'send and get' pour l'envoyer à la carte. Du côté de la carte, quand il reçoit une message, il va envoyer les données de la température à l'application (comme nous avons dit dans II.2.1). Pour écrire et envoyer le message on a écrit un action pour bouton 'send and get' on a une fonction comme suivant:

```
public void onClickget(View view) {
    String string = editText.getText().toString();
    serialPort.write(string.getBytes());
    tvAppend(textView, "\nYou have Sent : " + string + "\n");
    tvAppend(textView, "wait for temperature \n");
}
```

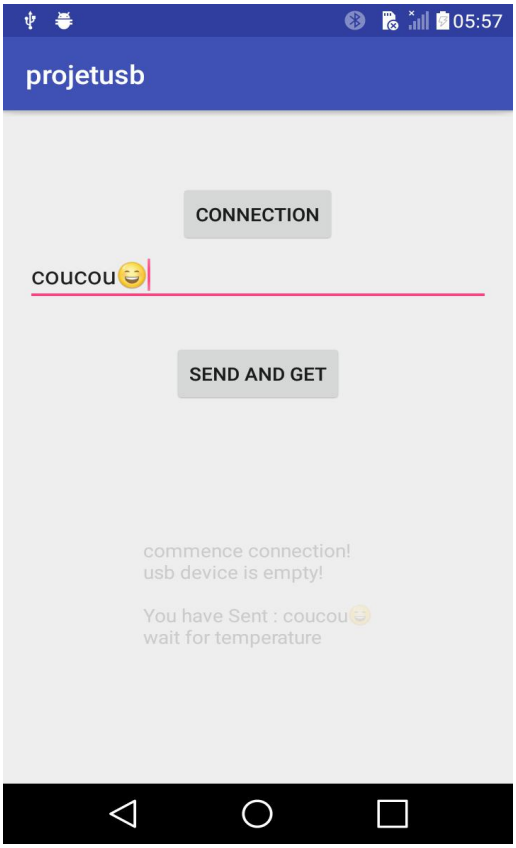
Pour recevoir les données de la carte, nous avons utilisé une fonction 'serialPort.read(mCallback)' dans le library ce qu'on a ajouté.

```
UsbSerialInterface.UsbReadCallback mCallback = new UsbSerialInterface.UsbReadCallback() {
    public void onReceivedData(byte[] arg0) {
        String data = null;
        try {
            data = new String(arg0, "UTF-8");
            data.concat("/n");
            tvAppend(textView, data);
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }
```

Ici quand on reçoit les données, on les affichera dans l'écran.

Les autres codes sont dans le répertoire ci-joint.

Nous avons testé l'application sans connecter la carte, et il fonctionne bien comme suivant:



Ici il a dit 'commence connection', et car on a pas lié avec la carte, donc il a dit 'usb device is empty'. Après, on tape 'coucou' et il a dit ce qu'on a tapé.

III. Problème et amélioration

- **Lier carte avec USB porte avec la câble au lieu de USB connecteur**

Avant de trouver qu'on se trompe sur Vbus et Ugnd, on a pris en compte la connection mauvaise du micro-USB connecteur, donc on a déssoudé le micro-USB connecteur et ressoudé un câble USB.

Mais quand on trouve c'est pas le cas de problème, on ne peut pas ressoudé le micro-USB car les pins de micro-USB connecteur on choisit est très fragile et pas pratique pour souder, malheureusement on n'a pas le temps de dessiner le PSB et souder une nouvelle carte avec un nouveau type de micro-USB connecteur.

C'est mieux si on peut lier le smartphone directement avec l'ordinateur avec un micro-USB connecteur. De plus, il faut que les pieds du micro-USB connecteur sont assez longue, sinon, il y aura un conflit entre les frontières du portable et la carte.

- **Echec lier Atmega32u4 avec le portable utilisant USB hub**

Pour lier la carte et le portable mais avec deux câble USB mâle, on a utilisé un USB hub. Mais quand on relever l'alimentation sur le USB hub, on trouve que le portable n'arrive pas à alimenter la carte. On a pris en compte de changer le smarthone du slave à le host en cablant le ID à la masse. On a douté que si le USB hub qu'on a utilisé peut créer le lien entre deux USB ports, mais on a pas du temps pour vérifier.

On va vérifier selon la façon suivant: On aliment un de port USB et on teste un autre, s'il y a de tension, on dit que les deux sont liées.

Sinon ou la tension est inférieur à 5V, on propose d'acheter une câble USB OTG (on the go) qui permets de faire fonctionner les périphéries.

Conclusion

Pour conclure, nous avons réalisé, dans le cadre d'un projet de 4ème année à Polytech Lille, une application qui affiche la température actuelle sur smartphone. La température est détecté par une mini carte avec un capteur de température soudé directement dedans. Les données de la température doivent être envoyé de la carte à l'application de smartphone à l'aide de la porte USB.

Durant ces 13 semaines, nous avons pu mettre en pratique nos connaissances théoriques et en développer certaines par le biais des technologies auxquelles nous avons été confrontés. Après des efforts nous avons fait pendant ce projet, nous avons appris comment coder avec les logiciels ce que nous n'avons jamais utilisé, comme Android Studio et IDE. Et nous avons aussi amélioré les compétences pour concevoir et fabriquer une carte électronique.

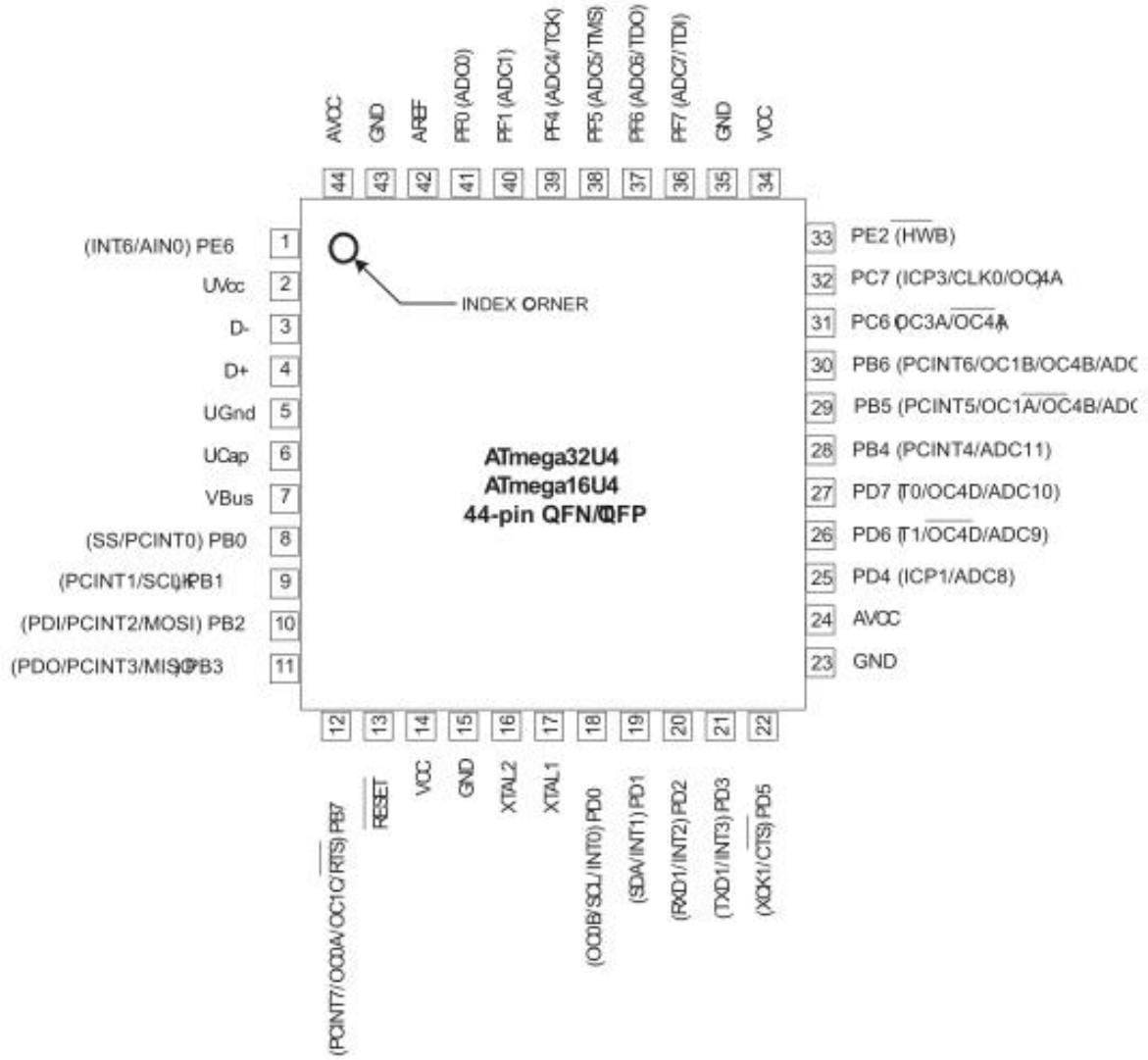
Comme nous avons dit dans le rubrique 'problème et améliorer' il reste beaucoup des efforts à faire à améliorer et corriger notre projet à la future. Nous devons continuer à réfléchir et améliorer nous même. Néanmoins, nous somme heureuses qu'on a déjà beaucoup amélioré dans plusieurs aspects.

Annexe

- Data sheet pour Atmega32U4:

1. Pin Configurations

Figure 1-1. Pinout



- Data sheet pour la capteur température:

MICROCHIP

TC1047/TC1047A

Precision Temperature-to-Voltage Converter

Features

- Supply Voltage Range:
  - TC1047: 2.7V to 4.4V
  - TC1047A: 2.5V to 5.5V
- Wide Temperature Measurement Range: -40°C to +125°C
- High Temperature Converter Accuracy: ± 2°C, Max, at 25°C
- Linear Temperature Slope 10mV/°C (typ.)
- Available in 3-Pin SOT-23B Package
- Very Low Supply Current:
  - 35µA Typical

Applications

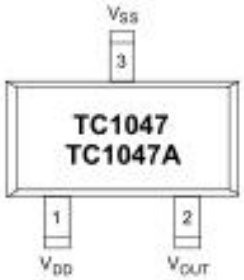
- Cellular Phones
- Power Supply Thermal Shutdown
- Temperature Controlled Fans
- Temperature Measurement/Instrumentation
- Temperature Regulators
- Consumer Electronics
- Portable Battery Powered Equipment

Device Selection Table

Part Number	Package	Temp. Range
TC1047VNB	3-Pin SOT-23B	-40°C to +125°C
TC1047AVNB	3-Pin SOT-23B	-40°C to +125°C

Package Type

3-Pin SOT-23B*



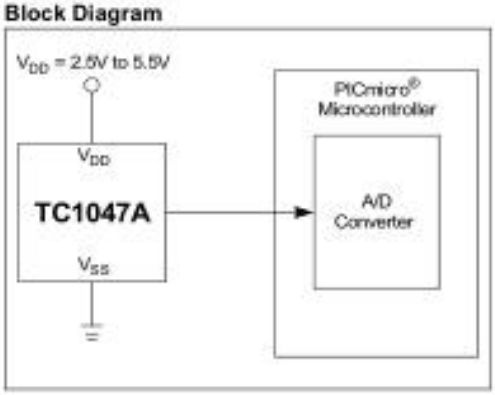
Note: * 3-Pin SOT-23B is equivalent to JEDEC TO-236

General Description

The TC1047 and TC1047A are linear voltage output temperature sensors whose output voltage is directly proportional to the measured temperature. The TC1047 and TC1047A can accurately measure temperature from -40°C to +125°C. With the TC1047, the supply voltage can vary between 2.7V and 4.4V. The power supply range of the TC1047A is from 2.5V to 5.5V.

The output voltage range for these devices is typically 100mV at -40°C, 500mV at 0°C, 750mV at +25°C, and 1.75V at +125°C. A 10mV/°C voltage slope output response allows for a predictable temperature measurement over a wide temperature range. The TC1047 and TC1047A are packaged in 3-Pin SOT-23B packages, making them ideal for space critical applications.

Block Diagram



3.0 DETAILED DESCRIPTION

The TC1047 and TC1047A have an output voltage that varies linearly with temperature in degrees Celsius. Figure 3-1 shows a plot of the output voltage versus temperature for the TC1047 and TC1047A. The temperature slope is fixed at 10mV/°C, and the output voltage at 0°C is 500mV.

FIGURE 3-1: OUTPUT VOLTAGE VS. TEMPERATURE