

G9D

Rapport final

Projet numérique semestre 1 – Electronique et Signal

Yassine LARAIEDH, William Alexander MBOLLO MBASSY,
Théotime POICHOTTE, Benjamin POMBET, Christophe
SAURY, Pierre TOMEI

Table des matières

Contexte.....	3
Cahier des charges	3
Le Micro	4
Le capteur	4
Traitement du Signal.....	4
Étape 1 : Suppression des silences	5
Étape 2 : Calcul de la puissance	7
Étape 3 : Filtrage numérique	7
Étape 4 : Vérification des conditions sur le signal	8
[OBJ]Résultats expérimentaux	Erreur ! Signet non défini.
Implémentation en langage C.....	9
Le capteur I2C	11
Capteur de température et d'humidité GoTronic	12
Capteur de fréquence cardiaque	14
Le Capteur	14
Traitement du signal	18
Implémentation en langage C.....	19
Interface.....	21
Programmation de l'envoi de la trame	22
Gestion de l'énergie sur la carte	23
Conclusion.....	25
Annexe	26

Contexte

Notre client, *Infinite Measure*, souhaite une solution clé en main, modulaire, paramétrable et bon marché pour une diffusion de masse pour la partie Hardware.

Pour répondre à son besoin, notre entreprise *Tech'n Logic* a choisi de développer son produit *Tech'n Health*. Notre produit est une solution déterminant la qualité de l'environnement de travail des ouvriers sur les chantiers et ainsi mesurant leur niveau de stress sur leur lieu de travail. Le but de ces mesures est de mettre en place, à terme, des solutions pour améliorer les conditions de travail de ces ouvriers.

Cahier des charges

Pour cela, nous avons créé des capteurs mesurant différents paramètres pour répondre au cahier des charges de notre client. Nous avons donc la capacité de :

- Détecter le taux de *CO2* (confort) grâce à un capteur de *CO2*
- Mesurer la puissance sonore captée par un micro (confort)
- Mesurer la fréquence cardiaque en utilisant un capteur cardiaque (indispensable)
- Mesurer la température corporelle (confort)

Pour l'interface utilisateur de notre capteur, nous avons utilisé l'écran *OLED* pour créer un menu qui permet de naviguer entre les différents capteurs. Ce menu possède les fonctionnalités suivantes :

- Affichage du logo de notre entreprise (indispensable)
- Navigation dans le menu grâce à un bouton (indispensable)

Pour traiter différentes données, nous avons implémenté des algorithmes de traitement du signal :

- Un algorithme permettant de mesurer la puissance du signal sonore capté par le micro (confort)
- Un algorithme détectant les signaux *ECG* et renvoyant la fréquence cardiaque de l'utilisateur (indispensable)

Enfin notre carte a la capacité de transmettre les trames binaires à la passerelle via une liaison Bluetooth (indispensable).

Le Micro

Le capteur

Ajouter un microphone à notre solution nous permet de mesurer les dangers dus à un niveau sonore trop élevé sur le chantier. Le but du micro est de mesurer à la fois le niveau sonore, mais aussi de détecter la présence de dangers pour ouïe de l'ouvrier. Le microphone est intégré à notre solution mobile. Notre objectif est de pouvoir obtenir des valeurs de puissance des signaux sonores par le micro, les affichés sur un écran *OLED* intégré à notre système et les envoyés à la passerelle. Il faut également détecter les sons jugés comme étant pénible et allumer une *LED* intégrée à notre système à chaque dépassement du seuil de bruit pénible.

Dans un premier temps, les valeurs sont mesurées et renvoyées par le microphone vers le microprocesseur via les sorties analogiques. Elles peuvent ensuite être affichées dans la console. Cependant, ces informations ne correspondent qu'à des valeurs prélevées à intervalles de temps réguliers et n'ont aucune signification physique. Seul le traitement des signaux acoustiques récupérés peut nous fournir les informations attendues sur les risques liés au lieu de travail.

Traitement du Signal

Le but de notre capteur est de détecter ce qu'on qualifie de « bruits pénibles » dans un signal. Pour cela, nous nous sommes appuyés sur des enregistrements de signaux environnementaux pour déterminer des paramètres adaptés au traitement des signaux que nous récupérerons sur les chantiers. Le but est de mettre en évidence les signaux environnementaux comprenant des composantes spectrales pénibles pour l'oreille humaine. Nous proposons une conception d'un système de détection de ces signaux pénibles en se basant sur le filtrage numérique. La détection de ces bruits, jugés dangereux, est définie par un ensemble de paramètres et de conditions. Ces paramètres seront évoqués par la suite.

Le micro utilisé possède une sensibilité égale à $-55dB$ et amplifie le signal reçu avec un gain de $30dB$.

L'analyse et le traitement des signaux audios se fait en plusieurs étapes, celles-ci sont présentées ci-dessous :

- Récupérer le signal et supprimer les silences, pour ne conserver que le bruit utile
- Calculer la puissance moyenne de ce signal
- Appliquer un filtre numérique au signal
- Vérifier les conditions données afin de déterminer s'il s'agit ou non d'un bruit

Les signaux concernés sont considérés comme « bruit pénible » si et seulement si, plus de 20% de leur puissance est comprise dans les fréquences supérieures à $2KHz$, avec une puissance sonore totale supérieure à $74 dB SLP$.

Étape 1 : Suppression des silences

Parmi les signaux captés par notre micro, certains se composent de plages de silences. Ces dernières modifient à la baisse un grand nombre d'outils de traitements des signaux, notamment ceux basés sur des valeurs moyennes. Ces silences sont à l'origine d'erreurs de calculs et tendent à diminuer, donc fausser, les résultats d'analyse de signaux (puissance, valeur moyenne, valeur efficace...). C'est la raison pour laquelle nous avons mis en place un algorithme de suppression des silences.

L'algorithme que nous avons produit se compose dans un premier temps de l'échantillonnage du signal récupéré par le micro. Les échantillons vont ensuite être analysés pour ensuite être classés en groupes de silences ou de bruits. Ces deux termes sont définis ci-dessous :

- Les « bruits » sont les sons étendus dans le temps sur une certaine durée, supérieure au seuil Dt fixé, et de puissance supérieure à celle définie comme seuil (P_{SPL}).
- Les « silences » sont les sons de puissance inférieure au seuil P_{SPL} , et/ou de durée inférieure au seuil Dt .

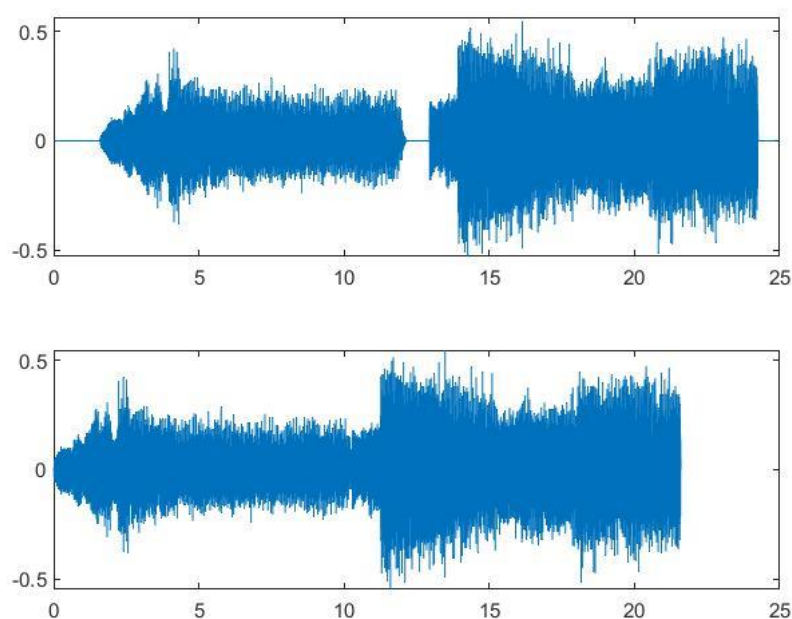
Pour différencier les bruits et les silences, nous parcourons tous les échantillons du signal (au minimum, ceux contenus dans une fenêtre glissante). Si un groupement d'échantillons possède une puissance moyenne supérieure à P_{SPL} s'étendant sur une durée supérieure à

Dt , alors ces échantillons sont qualifiés de bruits. Si cette condition n'est pas respectée, les échantillons sont qualifiés de silences. L'application de cette méthode sur l'ensemble du signal nous permet de distinguer d'une part les silences, et d'autre part les bruits.

L'étape suivante consiste à supprimer les plages de silences. Pour cela, on reprend le vecteur obtenu après les manipulations décrites précédemment. Pour chaque échantillon (ici d'indice x), si l'échantillon fait partie d'une plage de silence, l'échantillon prend la valeur 0. Si l'échantillon est situé dans les plages de bruits, on lui donne comme valeur son indice ($f(x) = x$). Ce vecteur se compose uniquement des indices des éléments situés dans les plages de bruits. En supprimant tous les 0 de ce vecteur (une fonction programmée nous le permet), nous obtenons la liste des indices des échantillons du signal original présents dans les plages de bruit. De cette liste d'indices, nous récupérons les valeurs du signal enregistré par le micro. Cela correspond aux éléments du signal audio aux instants $f(x) * T_e$ où x parcourt la liste des éléments du vecteur précédent.

On récupère par conséquent le signal dont les plages de silences ont été supprimées. En quelque sorte, on peut dire qu'on a réussi à concaténer les plages de bruits.

Les signaux ainsi obtenus sont représentés sur les figures ci-dessous. Le signal initial et le signal traité sont représentés dans le domaine temporel. La suppression de la plage de silence entre les instants $t_1 = 12,11s$ et $t_2 = 12,94s$ prouve l'efficacité de notre algorithme.



Étape 2 : Calcul de la puissance

Pour calculer la puissance totale de notre signal sans tenir compte des silences nous pouvons procéder deux manières différentes : soit calculer la puissance totale du nouveau signal qui ne contient plus de bruits, soit calculer la puissance totale de chaque fréquence et en faire la somme. Cette seconde méthode s'apparente au calcul de la densité spectrale. Notre signal étant stochastique, on ne peut pas calculer la puissance de façon analytique ; pour cela on estime sa puissance sur une fenêtre glissante correspondant à une durée de temps pour laquelle le signal est supposé stationnaire. Voici la formule :

$$P(n) = \frac{1}{2K + 1} \sum_{n-K}^{n+K} x(k)^2$$

L'intervalle $(2K + 1)T_e$ correspond à la durée de la fenêtre temporelle pour l'estimation.

La mise en place de cette opération sur des fenêtres glissantes nous permet de calculer la puissance du signal.

Étape 3 : Filtrage numérique

L'étape suivante du traitement de notre signal consiste au filtrage de celui-ci. Pour cela, nous avons utilisé l'application *Filter Designer*. Il s'agit d'un créateur de filtre programmé sur *MATLAB*. Nous avons décidé de programmer un filtre passe-bas. De par la définition du bruit pénible, la fréquence de coupure est fixée à $2kHz$. Les valeurs des paramètres appliqués sont les suivantes :

- $f_{Pass} = 2000Hz$
- $f_{Stop} = 2100Hz$
- $A_{Pass} = 1dB$
- $A_{Stop} = 80dB$

La détermination des valeurs des paramètres se fait en fonction de l'ordre du filtre choisi. Nous avons décidé de conserver un ordre autour de 1000. Un ordre trop élevé impliquerait un temps de calcul trop long par le microcontrôleur. Nous perdrons toute notion de « fréquence cardiaque en temps réel ».

Notre filtre passe-bas coupe toutes les fréquences supérieures à 2kHz. Pour déterminer si au moins 20% de la puissance est comprise dans les fréquences supérieures à 2kHz, nous regarderons si la puissance totale du signal filtré est strictement inférieure à 80% de la puissance totale du signal avant filtrage.

Étape 4 : Vérification des conditions sur le signal

Nous appliquons le filtre à notre signal et nous comparons les valeurs de puissances à celles du signal d'origine. Pour le calcul de la puissance, nous recourrons à la densité spectrale. Pour mesurer si 20% de la puissance du signal se situe dans les fréquences supérieures à 2kHz, nous effectuons un rapport entre les puissances.

$$\frac{\text{Puissance moyenne du signal filtré}}{\text{Puissance moyenne du signal originel}}$$

Si ce rapport est supérieur à 80%, le bruit n'est pas pénible.

Nous avons testé notre algorithme sur les différents signaux environnementaux qui nous étaient mis à disposition.

Nous avons également choisi de modéliser un passe-haut et nous avons comparé les valeurs obtenues. Les valeurs varient en général de quelques pourcents. Les valeurs, quant à elles restent donc à peu près les mêmes. Cela nous certifie que les valeurs trouvées sont cohérentes et proches de la valeur exacte. Effectivement, le filtre numérique n'étant pas idéal, une très petite partie des fréquences situées au-dessus de 2kHz sont conservées après filtrage. Celles-ci se retrouvent dans le calcul de la puissance.

Implémentation en langage C

Nous cherchons maintenant à réaliser un code en langage C implémentable sur notre carte *TIVA* qui aura pour but de répliquer les processus réalisés en traitement du signal pour pouvoir obtenir un résultat correct. Pour cela, on commence par réaliser un code permettant de lire les valeurs de notre capteur. On commence pour cela par définir dans notre code le port analogique correspondant à la broche de notre micro. On utilise ensuite la fonction *analogRead(MicPin)* pour lire la valeur obtenue par le micro. On envoie ces valeurs vers notre afficheur *OLED*.

Nous voulons maintenant ajouter une fonction à notre code qui lui permettra de juger des bruits comme pénibles, de les détecter et de prévenir l'utilisateur en allumant une LED. Pour cela, nous voulons obtenir la puissance de notre signal en entrée. Cette puissance nous permettra d'augmenter le rapport signal à bruit tout en nous donnant une bonne estimation de l'évolution du signal sans qu'elle soit faussée par un bruit soudain isolé. Ayant un signal stochastique, nous décidons d'utiliser la technique de fenêtre glissante pour pouvoir estimer la puissance moyenne du signal. Pour pouvoir mettre en point cette technique, nous devons créer une liste à partir des valeurs en sortie du micro. Ces valeurs sont mises à jour à l'arrivée de chaque nouvelle valeur. Le dernier élément de la liste est supprimé et les autres valeurs sont décalées d'un cran pour faire de la place à la nouvelle valeur du micro. On effectue ensuite un calcul de la puissance moyenne de ces échantillons. Nous utilisons ensuite une fonction conditionnelle *if()* avec comme condition la vérification que la puissance moyenne dépasse ou non un seuil défini. Ce seuil est pour l'instant défini en tâtonnant après l'implémentation du programme. Pour avoir quelque chose de plus précis pour la valeur du seuil, on aurait pu trouver un capteur de son qui donne la valeur en dB et utilisé ça en lançant notre programme et récupérer la valeur que nous retourne notre programme lorsqu'on passe un son à plus de 70 dB. Lorsque la condition de notre boucle *if()* est vérifiée, on allume une LED à l'aide de *digitalWrite(LEDPin, HIGH)*.

Nous avons maintenant un code valable pour le fonctionnement du micro, il suffit maintenant d'ajouter une LED à notre montage et trois résistances de 100 ohms sur l'emplacement dédiés sur le circuit imprimé. Nous avons choisi de mettre ces résistances car

il faut en avoir pour éviter que la LED ne s'endommage mais il ne faut pas non plus avoir des valeurs de résistance trop haute par peur que l'intensité de la LED en pâtisse. De plus, ces trois résistances étaient les trois résistances à base valeur immédiatement disponible à nous lorsque nous cherchions à les ajouter à notre montage.

Pour revenir au code, il était fonctionnel lorsque nous le testions sur plaquette, mais nous avons rencontré des difficultés lorsque nous l'avons ajouté aux autres codes et que nous l'avons testé sur notre montage. Le problème est dû au fait que le microcontrôleur a d'autres tâches à effectuer lorsque nous ajoutons d'autres capteurs et que le code du micro n'a pas été conçu pour boucler autant que ce qu'on lui demande. Un autre problème est causé par la sensibilité du micro qui n'est pas terrible.

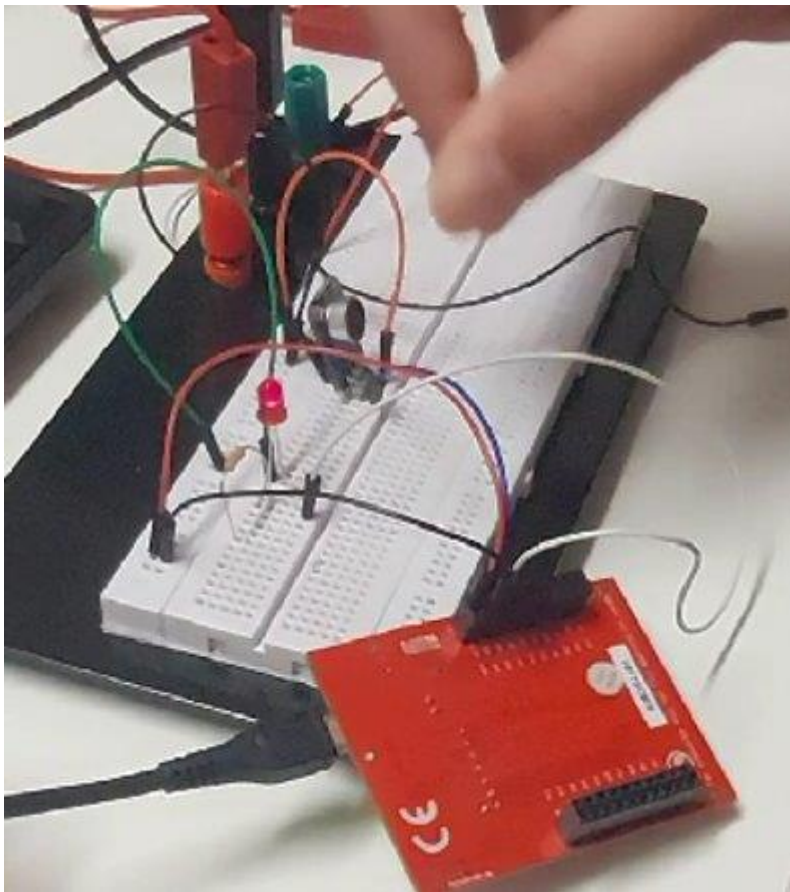


Photo d'un test du programme lorsqu'on effectue un claquement de doigt

On peut voir que la LED s'allume en détectant un son.

Le capteur I2C

Le capteur *I2C* nous est utile à la détection de microparticules, comme ici pour notre projet, le *CO2*.

Il fonctionne de telle manière à renvoyer les données récoltées sur notre écran *OLED*.

Il effectue cela à l'aide d'un protocole dit *I2C* :

Le protocole *I2C* (Inter Integrated Circuit) est un protocole permettant la communication entre un composant maître et plusieurs esclaves. Elle s'effectue à l'aide d'un bus de communication. La communication n'a lieu qu'entre maître et esclave. Un maître peut envoyer un ordre à tous ses esclaves. Le protocole est le plus épuré des bus de communication entre un processeur et ses esclaves. Il s'appuie sur la masse commune entre les équipements. Il a une communication bidirectionnelle (où chacun ne parle qu'à son tour), reposant ainsi sur une communication série synchrone.

Le capteur de carbone aurait permis de recueillir les valeurs du *CO2* dans un chantier pour savoir s'il faut aérer ou non et avoir un indicateur d'un facteur pouvant potentiellement nuire à la santé des ouvriers.

Le capteur *I2C* que nous avons reçu ne fonctionne pas, on estime que c'est causé par un défaut de conception. Ainsi dans la partie intégration, nous n'avons pas pris en compte le capteur *I2C* pour l'affichage sur l'écran *OLED* et pour l'envoi de trames.

Le capteur de température et d'humidité *GoTronic*

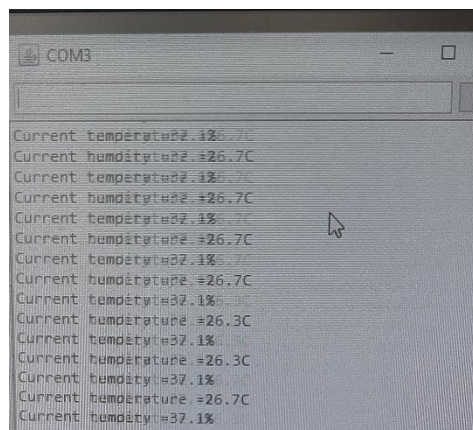
Ce "double" capteur va nous aider à prendre la température ambiante du chantier ainsi que de transmettre le taux d'humidité du corps sur notre écran.

Nous avons un capteur de température *LM35*. Pour le capteur de température *LM35*, la précision est au degré près. Une hausse de la tension de *10mV* correspond à une hausse de *1°* de température. Voici la formule avec laquelle on obtient une température en degrés.

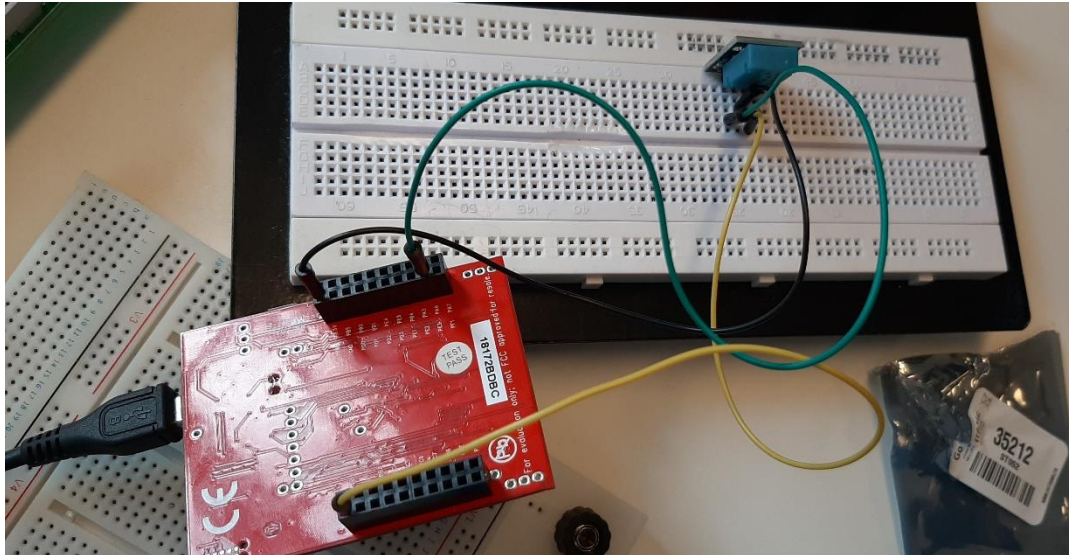
$$\text{Température mesurée} = \frac{\text{nombre renvoyé} * 3,3 * 100}{4095}$$

Avec ici, *3.3V* étant la tension maximale en entrée des ports analogiques de notre microcontrôleur, *4095* correspond $2^n - 1$ avec $n = 12 \text{ bits}$.

Ce capteur lit à la fois l'humidité et la température. Il renvoie les valeurs sur 5 bits. Les deux premiers chiffres affichés sont pour l'humidité et les deux derniers sont pour la température. Cependant, ce capteur, contrairement au précédent, lit la température seulement de 0 à 50 degrés. Il nécessite une tension de *5V*. Pour lire la valeur, on utilise la fonction `digitalread()` et pour l'afficher sur la console, on utilise la fonction `println`



Cette photo nous montre l'affichage des valeurs de la température et du taux d'humidité sur notre console. Ci-dessous nous avons également le montage.



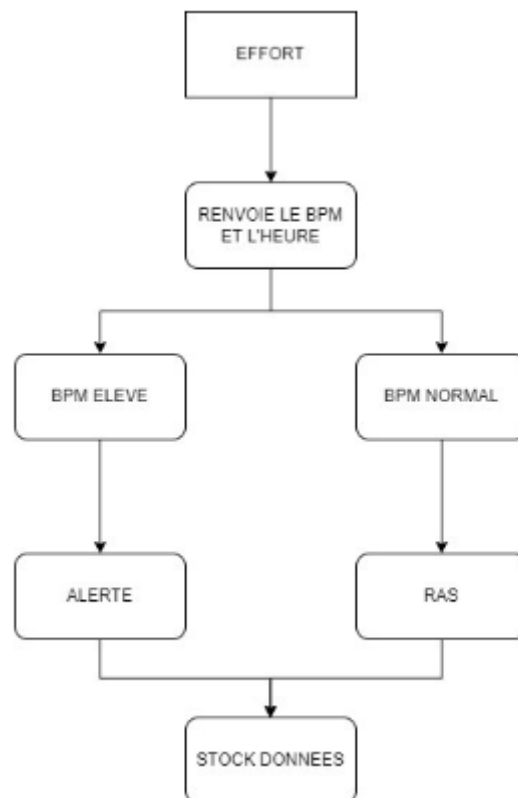
Ce capteur nous permet de recueillir les conditions thermiques de l'environnement d'un chantier. Cela permet à l'employeur de mieux pouvoir gérer ses employés et pouvoir organiser des temps de pause et imposer un minimum d'eau à boire à chacun en fonction de la température ambiante. Le taux d'humidité est aussi intéressant à mesurer car il est lié au ressenti de la chaleur. En effet, le mécanisme de transpiration fonctionne moins bien lorsque l'humidité est très élevée.

On utilise ainsi la formule précédente et le fait que le capteur renvoie les valeurs des deux capteurs sur 5 bits pour pouvoir réaliser un code en langage C permettant de lire la valeur du capteur et la convertir en degrés Celsius. On utilise notamment la fonction *analogRead(Pin_Capteur)* pour lire les valeurs en sortie du port analogique. On assigne ces valeurs à une variable. On convertit la valeur de cette variable en degré Celsius et c'est cette valeur qu'on va afficher et envoyer à la passerelle.

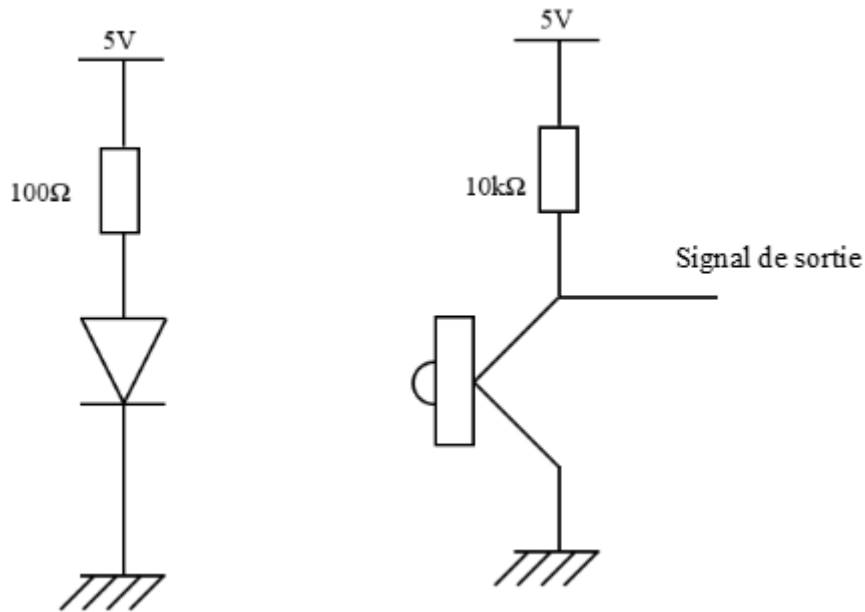
Capteur de fréquence cardiaque

Le Capteur

Pour répondre à la demande de notre client *Infinite Measure*, nous avons conçu un capteur de fréquence cardiaque destiné aux travailleurs du secteur du *BTP*. Pour ce faire nous disposons des composants suivants : une *LED*, un phototransistor, des amplificateurs et un bracelet. Enroulé autour du doigt des travailleurs, ce capteur mesurera leur fréquence cardiaque pendant leur travail afin de mesurer leur niveau de stress.

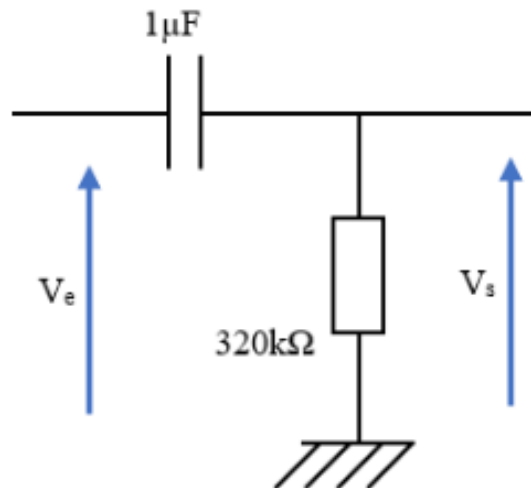


Notre dispositif fonctionne grâce à une *LED* et un phototransistor placé de part et d'autre du doigt de l'ouvrier.



La lumière, traversant le doigt et arrivant au phototransistor, fluctue en raison du passage du sang. Cette fluctuation est responsable d'une variation de l'intensité lumineuse reçue par le phototransistor. Il faut savoir que plus le phototransistor est éclairé, plus la tension qu'il délivre est faible, jusqu'à être nulle. Ainsi les augmentations d'amplitude dans le signal correspondent au passage du sang dans le doigt et la fréquence du signal observé est celle du passage du sang dans les veines, soit la fréquence cardiaque.

Étant donné qu'il existe des bruits parasites provenant des battements du cœur, il nous a été demandé de réaliser un filtrage analogique. C'est pour cela que nous avons mis en entrée de notre circuit un filtre passe-haut et une structure de Rauch. Comme passe-haut nous avons décidé d'utiliser un circuit *RC* « *C en-tête* », car c'est un montage facile à réaliser et qui nous permet de déterminer la fréquence de coupure très aisément.



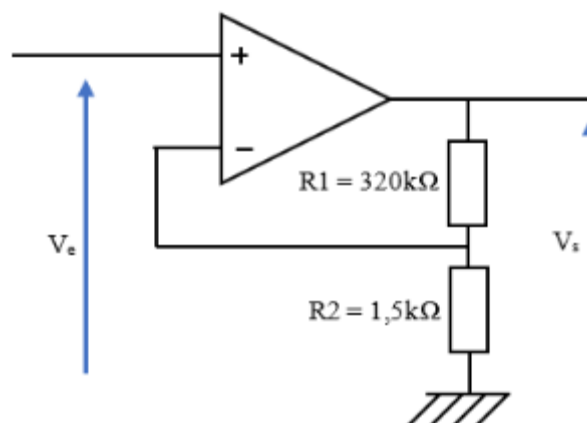
La fréquence de coupure pour ce type de montage est :

$$f_c = \frac{1}{2\pi R * C}$$

Sachant que le cœur humain peut battre jusqu'à 3 battements par seconde, (correspondant à 3 Hz), nous avons choisi des valeurs de résistance et des capacités telles que f_c soit proche de 3Hz. Ainsi :

$$R = 320k\Omega \text{ et } C = 1\mu F$$

Le signal reçu possède une amplitude crête-à-crête de quelques millivolts seulement. Pour pouvoir implémenter un code mesurant la fréquence cardiaque, nous devons amplifier le signal. Nous avons choisi d'utiliser un amplificateur non-inverseur. Voici ci-dessous le schéma de l'amplificateur utilisé :



Pour un amplificateur non-inverseur le gain est donné comme suit :

$$G = 1 + \frac{R_1}{R_2}$$

Pour des valeurs de $R_1 = 320 \text{ k}\Omega$ et $R_2 = 1,5 \text{ k}\Omega$ nous avons obtenu un gain de 210.

Pour le passe-bas actif il nous a été conseillé d'utiliser la structure de Rauch qui est la suivante :

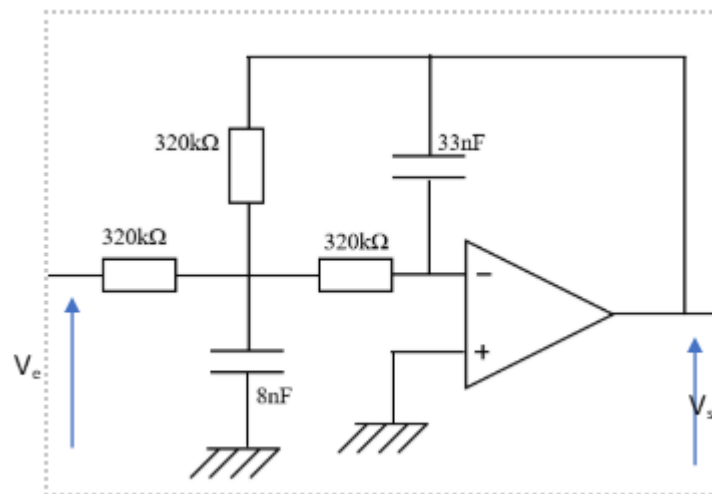
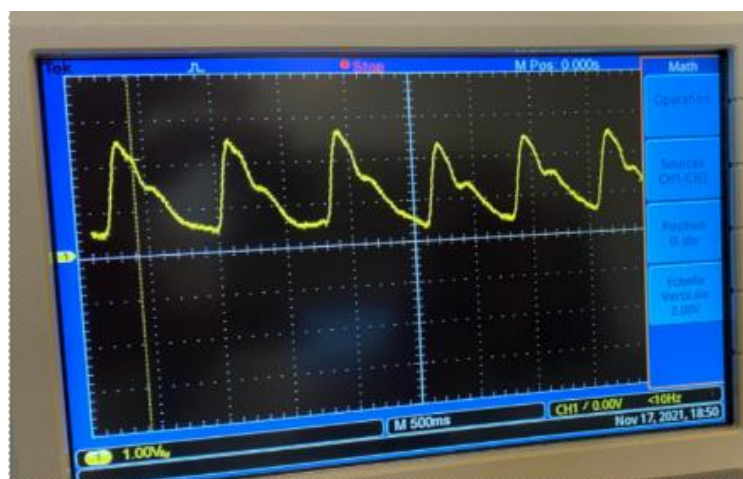
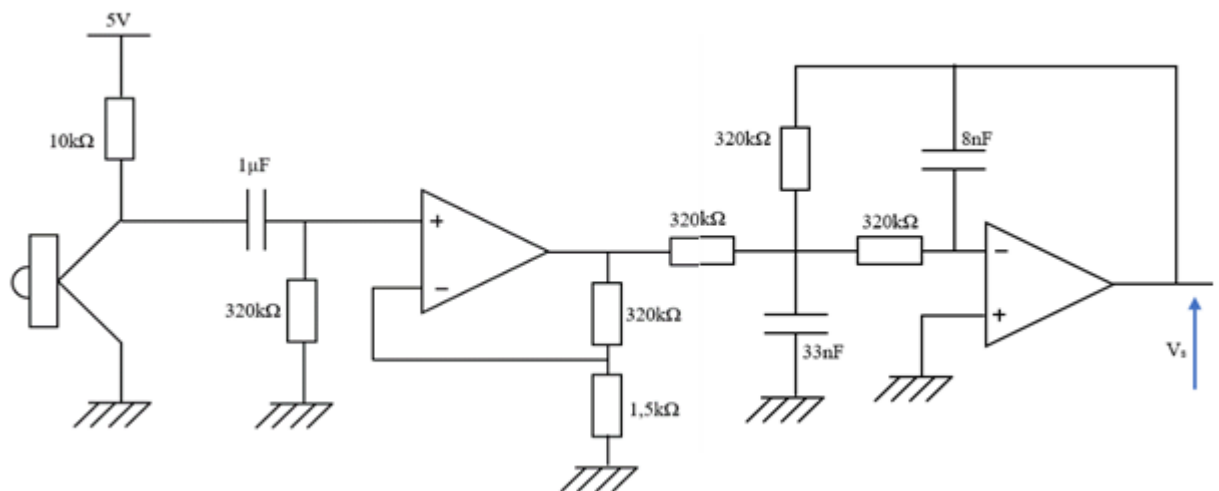


Schéma de la structure de Rauch



Nous avons pris pour valeurs $R = 320 \text{ k}\Omega$, $C_1 = 8\text{nF}$ et $C_2 = 33\text{nF}$ et obtenu une fréquence de coupure de 15 Hz. Ceci nous permet de ne pas atténuer les fréquences cardiaques les plus élevées. Nous terminons par présenter ci-dessous l'ensemble de notre montage.



Traitement du signal

Nous avons conçu deux algorithmes de mesure de fréquence cardiaque en temps réel sur *MATLAB*.

Le premier algorithme s'appuie sur une analyse fréquentielle du signal, le second sur une analyse temporelle. Dans le but d'en sélectionner un seul pour l'implémentation en langage C sur notre microcontrôleur, nous les avons comparées en termes de robustesse, de complexité algorithmique et de précision. Pour l'algorithme procédant par analyse fréquentielle, nous appliquons une Transformée de Fourier Discrète pour obtenir la fréquence. Cet algorithme est particulièrement robuste aux variations de fréquences. Il fait également preuve d'une grande précision. Par cela, l'algorithme utilise le principe des fenêtres glissantes pour calculer en temps réel, à partir d'une étude spectrale du signal, la fréquence cardiaque de l'utilisateur. Une partie du signal est placée dans la mémoire tampon du microcontrôleur en attente d'être traitée.

Pour le second algorithme, les temps correspondant aux pics sont mesurés. La période cardiaque, égale à l'intervalle de temps entre deux pics consécutifs est calculée, de sorte qu'en la passant à l'inverse, on retrouve la fréquence cardiaque de l'utilisateur.

Pour les deux algorithmes, la précision est très bonne. Nous avons choisi d'implémenter la méthode par analyse temporelle, jugée moins coûteuse en complexité algorithmique et en calculs. La méthode par analyse fréquentielle nécessite davantage de mémoire tampon.

Dans le programme que nous avons élaboré, un battement cardiaque est défini par un maximum local (mesuré à partir de l'étude du coefficient directeur de la pente en entre deux points), dont la valeur en tension est supérieure à la 90% de la valeur maximale mesurée sur l'ensemble du signal. Il sera nécessaire de revoir à la baisse cette valeur, puisque certains phénomènes électroniques inattendus pourraient fausser la valeur du maximum. Pour les premiers échantillons, la valeur maximale enregistrée sera faible, et non-révélatrice, entraînant par conséquent un nombre de détections de battements très élevés. En revanche, dès la première mesure de battement cardiaque, la valeur du maximum sera suffisamment élevée pour stabiliser le capteur cardiaque.

Implémentation en langage C

Nous cherchons maintenant à réaliser un code en langage C nous permettant de détecter les battements de cœur obtenu par le capteur cardiaque et à déterminer la fréquence de battements de cœur. Pour détecter les battements de cœur, cela est très simple algorithmiquement. Il suffit d'un dépassement de seuil de la valeur de la tension de notre capteur cardiaque. Ce qui est plus long c'est la définition du seuil de cette valeur et l'obtention de la fréquence cardiaque.

Pour obtenir la fréquence des battements de cœur, nous avons commencé par chercher à obtenir l'autocorrélation de notre signal en entrée. Pour cela, nous avons réutilisé le code que nous avons pour le micro pour de nouveau appliquer la technique des fenêtres glissantes pour avoir des valeurs de puissances d'un signal stochastique. A partir de cela, nous avons utilisé la formule de l'autocorrélation pour obtenir l'autocorrélation du signal et son délai. Le but était d'utiliser l'autocorrélation pour obtenir la fréquence du signal mais nous nous sommes rendu compte en mettant notre programme en pratique que cette méthode prenait trop de temps à effectuer ce qui lui est demandé. Nous avons ainsi décidé de ne pas utiliser le calcul d'autocorrélation que nous avons réalisé pour déterminer la fréquence du signal. Nous avons voulu simplifier la détermination de la fréquence.

Dans un premier temps nous initialisons à zéro deux variables globales une pour la précédente valeur du signal à l'instant $t-1$ (*valeurPrécédente*) et l'autre pour la valeur du temps mesuré

lors de l'échantillonnage $n-1$ (*TempsPrécédent*) ; puis nous créons une boucle dans laquelle nous définissons la valeur actuelle (*valeurActuelle*) du signal à l'instant t pendant l'échantillonnage en la récupérant sur le port analogique de notre choix. Dans la suite il faut vérifier deux conditions :

- Si la valeur actuelle du signal est supérieure au seuil (*if ValeurActuelle > valeurSeuil*), vérifier si la valeur précédente du signal est inférieure au seuil (*if valeurPrécédente < valeurSeuil*), ce qui signifie que le signal est croissant depuis l'échantillon $n-1$ et vient de dépasser le seuil.

Si c'est bien le cas nous fixons une variable qui correspond à l'instant de détection (*TempsDétection*). Si le temps écoulé entre l'instant de détection du signal et l'instant de détection précédent est supérieur à une certaine durée (*if TempsDétection - TempsPrécédent > durée*), nous considérons que la valeur du signal détectée n'est pas un bruit. Nous calculons la fréquence cardiaque en BPM (nombre de battement par minute) et l'affichons à l'écran. La formule qui nous donne la fréquence en BPM que nous avons utilisée est :

$$f \text{ en BPM} = f \text{ en Hz} * 60 = \left\lfloor \frac{60 * 1000}{(TempsDétection - TempsPrécédent) \text{ ms}} \right\rfloor$$

Puis nous terminons notre itération en transférant dans la variable *TempsPrécédent* la valeur de *TempsDétection*.

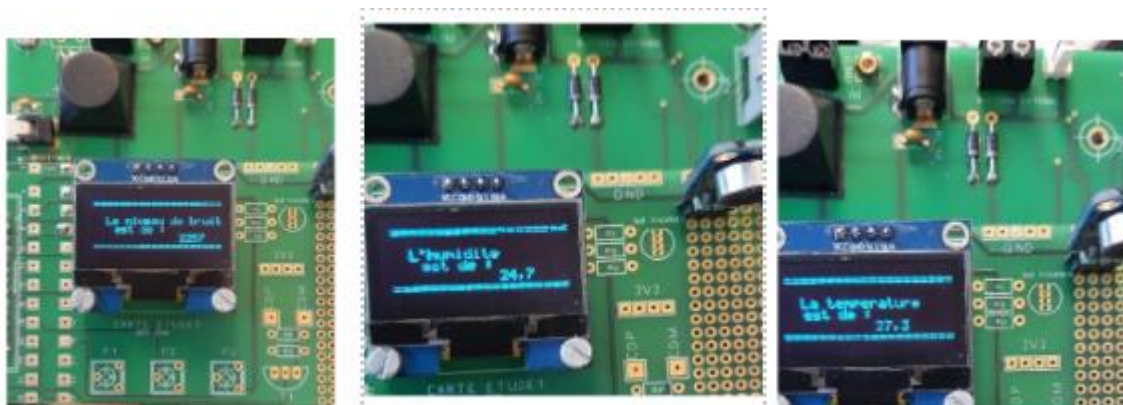
- Que les conditions précédentes soient vérifiées ou pas, à la fin de ces tests, la valeur du signal récupérée est stockée dans la variable *valeurActuelle*. Elle conservera cette valeur pour la prochaine itération du processus.
- Nous avons fixé le seuil de tel sorte que la valeur du signal obtenu ne soit pas négligeable devant la valeur maximale de 4096 que peut renvoyer la carte TIVA. Il fallait donc vérifier que $valeurSeuil > 0,1 * 4096 = 409,6$.
- Ensuite nous savons qu'en une seconde il y a un peu plus d'un battement cardiaque, pour des sportifs de haut niveau lorsque le rythme cardiaque est très élevé pendant l'effort il peut atteindre des records de plus de trois battements par secondes (3Hz). Nous avons fixé notre durée pour être fidèle au signal et pour établir un critère sur le filtrage du bruit. Il fallait donc capter pas plus d'un battement cardiaque durant cette durée. Pour une fréquence cardiaque record de 240 bpm nous captons 1,2 battements

en 0,3s et 0,8 battement en 0,2s. Nous avons établi notre durée de vérification à 0,2s, ainsi tout signal détecté avant cette durée de 0,2s est presque sûrement un bruit.

Le capteur cardiaque ne fonctionnant pas lorsqu'il est monté sur notre système, nous avons décidé de ne pas intégrer le code faisant fonctionner ce capteur, celui détectant les battements de cœur et celui qui obtient la fréquence des battements au reste de notre code. Les codes de la partie cardiaque, bien que pas intégrés à notre solution finale, ne sont pas perdus car ils pourront être utilisés dès que nous arriverons à faire fonctionner notre capteur ou lorsque nous en recevons un nouveau.

Interface

Afin de pouvoir transporter nos capteurs, nous les avons soudés sur une carte électronique et avons réalisé une interface consistant en un menu qui permet à un utilisateur de visualiser sur un écran *OLED* tour à tour les valeurs de chacun des capteurs qui fonctionnent (le capteur de température, le microphone, et le capteur d'humidité). Pour ce faire, il dispose d'un bouton poussoir qui permet de naviguer entre les différents capteurs. Ce bouton ainsi que l'écran *OLED* sont connectés à une carte électronique sur laquelle nous avons implémenté le code de chaque capteur. Par la suite, nous avons programmé le menu de sorte qu'il actualise un minimum d'éléments. Ensuite nous avons défini une zone qui actualise en temps réel la valeur de chaque capteur. Nous avons également ajouté une *LED* et trois résistances de 100Ω à notre montage pour permettre la détection de bruit jugé comme nuisible.



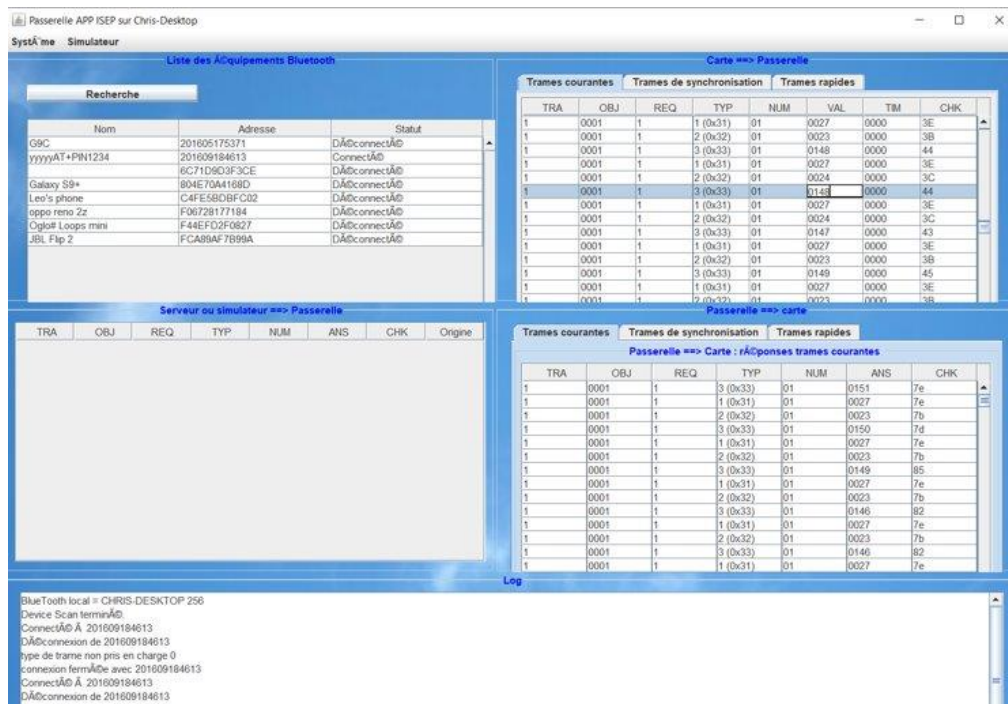
Différentes valeurs des capteurs affichées à l'écran

Programmation de l'envoi de la trame

Pour envoyer les valeurs de nos capteurs sur la passerelle nous devons au préalable les convertir en format de trame. Cette dernière comprend des parties fixes qui sont le type de trame envoyé et le nom de notre groupe mais également des parties variables comme les identifiants et valeurs des capteurs. La nouvelle passerelle étant en cours de construction, nous n'avons pas reçu de code pour l'envoi de trames depuis la carte. Malgré cela, nous voulions tout de même tester l'envoi de trames depuis notre système complet. Ainsi, nous avons décidé de le coder nous-même.

Pour coder la trame, nous avons utilisé une variable trame qui est initialisée à une valeur par défaut correspondant à la partie fixe de la trame. Lorsqu'un capteur mesure une valeur, la fonction permettant la lecture de cette valeur appelle une fonction qui donne un début de trame et une autre fonction qui ajoute à notre trame actuelle l'identifiant du capteur et sa valeur convertie en hexadécimal. On calcule à partir de ceci le *checksum* nous permettant de savoir si l'information a été envoyée correctement et on l'ajoute à la trame. Ces trames sont envoyées en direct sur la plateforme via Bluetooth.

On obtient les résultats suivants :



The screenshot displays the 'Passerelle APP ISEP' software interface. It features a 'Liste des Équipements Bluetooth' (Bluetooth Equipment List) on the left, a central area for 'Trames courantes' (Current Frames) and 'Trames de synchronisation' (Synchronization Frames), and a 'Log' window at the bottom.

Liste des Équipements Bluetooth:

Nom	Adresse	Statut
G9C	201605175371	DÃ©connectÃ©
yyyyyAT+PIN1234	201609184613	ConnectÃ©
6C71D803F3CE	DÃ©connectÃ©	
Galaxy S9+	804E70A4168D	DÃ©connectÃ©
Leo's phone	C4FE5BDBFC02	DÃ©connectÃ©
oppo reno 2z	F06728177184	DÃ©connectÃ©
Oglo® Loops mini	F44EFD2F0827	DÃ©connectÃ©
JBL Flip 2	FCA89AF7899A	DÃ©connectÃ©

Trames courantes:

TRA	OBJ	REQ	TYP	NUM	VAL	TM	CHK
1	0001	1	1 (0x31)	01	0027	0000	3E
1	0001	1	2 (0x32)	01	0023	0000	3B
1	0001	1	3 (0x33)	01	0148	0000	44
1	0001	1	1 (0x31)	01	0027	0000	3E
1	0001	1	2 (0x32)	01	0024	0000	3C
1	0001	1	3 (0x33)	01	0148	0000	44
1	0001	1	1 (0x31)	01	0027	0000	3E
1	0001	1	2 (0x32)	01	0024	0000	3C
1	0001	1	3 (0x33)	01	0147	0000	43
1	0001	1	1 (0x31)	01	0027	0000	3E
1	0001	1	2 (0x32)	01	0023	0000	3B
1	0001	1	3 (0x33)	01	0149	0000	45
1	0001	1	1 (0x31)	01	0027	0000	3E
1	0001	1	2 (0x32)	01	0023	0000	3B

Trames de synchronisation:

TRA	OBJ	REQ	TYP	NUM	ANS	CHK
1	0001	1	3 (0x33)	01	0151	7e
1	0001	1	1 (0x31)	01	0027	7e
1	0001	1	2 (0x32)	01	0023	7b
1	0001	1	3 (0x33)	01	0150	7d
1	0001	1	1 (0x31)	01	0027	7e
1	0001	1	2 (0x32)	01	0023	7b
1	0001	1	3 (0x33)	01	0149	85
1	0001	1	1 (0x31)	01	0027	7e
1	0001	1	2 (0x32)	01	0023	7b
1	0001	1	3 (0x33)	01	0146	82
1	0001	1	1 (0x31)	01	0027	7e
1	0001	1	2 (0x32)	01	0023	7b
1	0001	1	3 (0x33)	01	0146	82
1	0001	1	1 (0x31)	01	0027	7e

Log:

```

Bluetooth local = CHRIS-DESKTOP 258
Device Scan terminÃ©
ConnectÃ© Ã 201609184613
DÃ©connexion de 201609184613
type de trame non pris en charge 0
connexion terminÃ©e avec 201609184613
ConnectÃ© Ã 201609184613
DÃ©connexion de 201609184613

```

On peut voir ici que la plateforme et notamment l'envoi de trames fonctionne bien.

Gestion de l'énergie sur la carte

On avait la possibilité d'utiliser une batterie ou d'utiliser une cellule solaire qui nous a été fournie par l'ISEP. On cherche à savoir si la batterie et le panneau solaire suffisent à alimenter le système. Pour cela, nous avons besoin de connaître la consommation de tous les éléments du système.

Consommation des composants :

Consommation du système (éléments tous à 5V)					
Carte TIVA	Puissance	460	mW	0,46	W
Bluetooth	Puissance	1		0,001	
Amplificateur Opérationnel x4	Puissance	100		0,1	
Diode	Puissance	75		0,075	
Capteur Cardiaque	Puissance	250		0,25	
Capteur Température	Puissance	0,3		0,0003	
Afficheur OLED	Puissance	200		0,2	
Micro	Puissance	2,5		0,0025	
Total	Puissance	1088,8	mA	1,0888	A
	Courant	217,76		0,21776	

Nous avons commencé par calculer la puissance totale du système et on a obtenu $1.0138W$.

Ensuite, nous avons calculé l'intensité du système en divisant la puissance par la tension (5V).

Nous avons obtenu une intensité de $0.2027A$.

Batterie	Capacité	5	Ah
Total Système	Courant	0,21776	A
Autonomie Système	Temps	22,961058	H

Nous avons cherché une batterie en lithium de 5V en ligne peu coûteuse. Nous en avons trouvé une de 5V ayant une capacité de 5Ah. A partir des caractéristiques de la batterie trouvée, nous avons vérifié l'autonomie du système. Pour cela nous avons divisé la capacité de la batterie par le courant calculé pour le système. On obtient 22 heures et 57 minutes.

Panneau Solaire original	Puissance	0,5	W
	Tension	5	V

Le panneau solaire permet de fournir une puissance de $0.5W$. Cela n'est pas suffisant pour alimenter le système comprenant la carte TIVA et tous les composants. Il faut donc trouver un nouveau panneau solaire ayant une tension de 5V et une puissance supérieure à $1.0888W$. Ainsi on a trouvé un panneau solaire ayant une puissance de 2W en ligne (voir le tableau ci-dessous). Ce dernier est suffisant pour alimenter le système. Cependant, un panneau solaire

a besoin de lumière, ce qui peut poser un problème. Il faudra ainsi impérativement l'utiliser avec la batterie trouvée lors de la mission 1.

Nouveau Panneau Solaire	Puissance	2	W
	Tension	5	V

On cherche l'équivalent carbone de la consommation de la carte sachant qu'en France 1kWh produit environ $0.1kg$ d'équivalent CO_2 .

On réalise un produit en croix : $1.088W * 0.1kCO_2 / 1KW = 108\text{ mg}$

L'équivalent carbone de la consommation de la carte est de $46mg$ par heure.

Conclusion

Pour conclure, on peut dire que notre projet a plutôt été réussi, malgré les difficultés qu'on a pu rencontrer. Le but de notre entreprise était de développer son produit *Tech'n Health*. Pour déterminer la qualité de l'environnement de travail des ouvriers sur les chantiers et ainsi mesurer leur niveau de stress sur leur lieu de travail. Le but de ces mesures est de mettre en place, à terme, des solutions pour améliorer les conditions de travail de ces ouvriers.

Nous avons donc créé des capteurs mesurant différents paramètres pour satisfaire le besoin de notre client. Un capteur de CO_2 pour mesurer le taux de dioxyde de carbone, un micro pour mesurer la puissance sonore, un capteur cardiaque pour mesurer la fréquence cardiaque, et enfin un capteur de température pour mesurer la température corporelle.

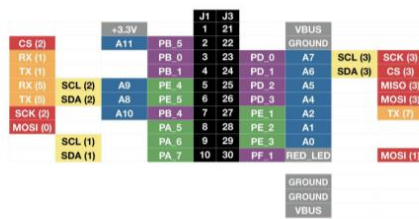
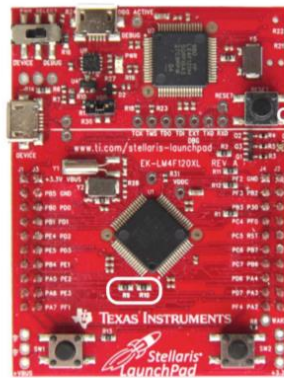
Durant notre projet d'entreprise, nous avons pu faire le nécessaire afin de satisfaire au maximum notre client ainsi que bien respecter le cahier de charges fonctionnel. Cependant, tout n'a pas été parfait. En effet, notre groupe est rempli de qualités, mais la vie d'un groupe, ce n'est pas toujours évident. Quelques malentendus ont eu lieu notamment sur l'organisation, et cela est dû essentiellement à un manque de communication au tout début du projet. C'était le point à améliorer, ce qu'on a plutôt réussi à le faire avec également l'équité du travail. Certains ont plus travaillé sur la partie Electronique que sur la partie Signal, d'autres plus sur la partie Signal qu'Electronique. Malgré cela, nous étions mis à la page sur tout ce que nous avons fait, afin de construire un groupe le plus solide, et le plus homogène possible, nous avons également mis au point certains malentendus afin d'assurer le très bon fonctionnement du groupe et enfin nous avons pris de nouvelles résolutions pour nos projets à venir, même si notre projet actuel a été plutôt bien fait. Un point très positif est à noter, notre groupe est toujours au rendez-vous et réponds aux besoins du client bien à temps.

Annexe



Flash	256	KB
SRAM	32	KB
Serial	hardware	
ADC	12	bits
Use pins numbers only!		

LaunchPad with LM4F120H5QR LaunchPad with TM4C123GH6PM Revision 1



Hardware
Pin number
Other pin number
PC
Serial UART
SPI
analogRead()
digitalRead() and digitalWrite()
digitalRead() and digitalWrite() and analogWrite()



Rei Vito, 2012-2015
embeddedcomputing.weebly.com
version 1.5 2015-07-20

00 shunt
23 SCL (3) PD.0 R9 PB.6 14
24 SDA (3) PD.1 R10 PB.7 15

Remove shunts for compatibility

VBUS detection	PD.7
----------------	------