

## **Rapport de projet : Détection Zombies**

### **Sommaire**

#### **1) Introduction**

- Présentation générale du projet
- Use case & Répartition des tâches
- Schéma fonctionnel & Diagramme de Gantt

#### **2) Hardware**

- Schématique
- Routage
- Soudure & Tests préliminaires

#### **3) Software**

- Capteur ultrason
- Capteur de fréquence cardiaque
- Ecran
- Tests préliminaires

#### **4) Partie mécanique**

- Principe

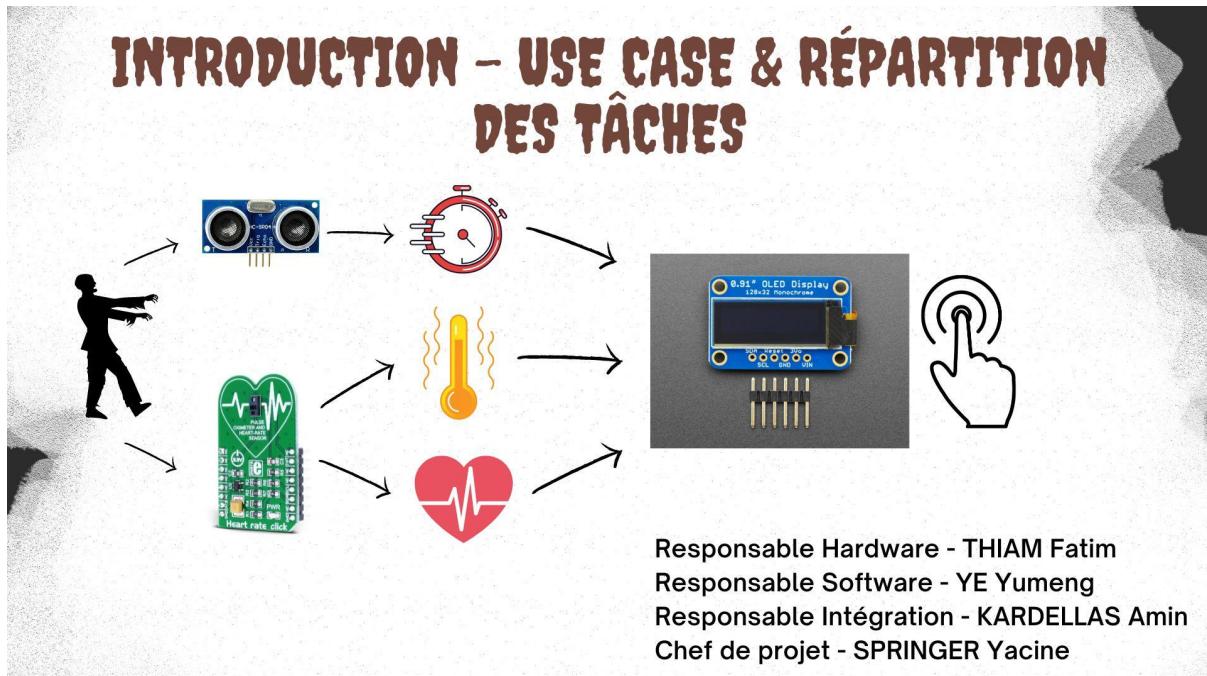
#### **5) Conclusion globale**

- Obstacles rencontrés
- Conclusion
- Démonstration du résultat

## Introduction - Présentation générale

Bienvenue dans un monde où l'impensable est devenu réalité : une pandémie mondiale a transformé des individus autrefois humains en zombies affamés de cerveaux. Ces créatures déambulent dans les couloirs de notre école, semant la terreur parmi les étudiants et les professeurs. Face à cette situation cauchemardesque, notre équipe a pris les devants pour lutter contre cette épidémie de zombies. Notre projet vise à détecter ces zombies et à les différencier des humains en utilisant des données clés telles que la vitesse de déplacement, la fréquence cardiaque et la température corporelle. En combinant la technologie et l'ingéniosité de notre équipe, nous espérons apporter une lueur d'espoir dans ce monde sombre et protéger notre école des griffes des morts-vivants. Préparez-vous à plonger dans une aventure unique où la science rencontre le surnaturel dans une lutte désespérée pour la survie.

## Introduction - Use case & Répartition des tâches



Notre use case consiste à détecter les zombies en utilisant trois critères clés : la vitesse, la fréquence cardiaque et la température.

Nous mesurons la vitesse de déplacement de la personne suspecte, en la comparant à une référence inférieure à la vitesse de marche moyenne des humains, par exemple 1 m/s.

De plus, nous analysons sa fréquence cardiaque, en la comparant à une valeur inférieure à la fréquence cardiaque moyenne des individus sains, dans ce cas on prend 60 battements par minute.

Enfin, nous relevons sa température corporelle, en identifiant une valeur correspondant à une température froide comme 18 °C. Ces données seront accessibles sur différents menus de l'écran OLED.

En combinant ces mesures, nous pouvons rapidement déterminer si la personne est un zombie ou non, permettant ainsi de garantir la sécurité de tous.

## Introduction - Schéma fonctionnel & Diagramme de Gantt

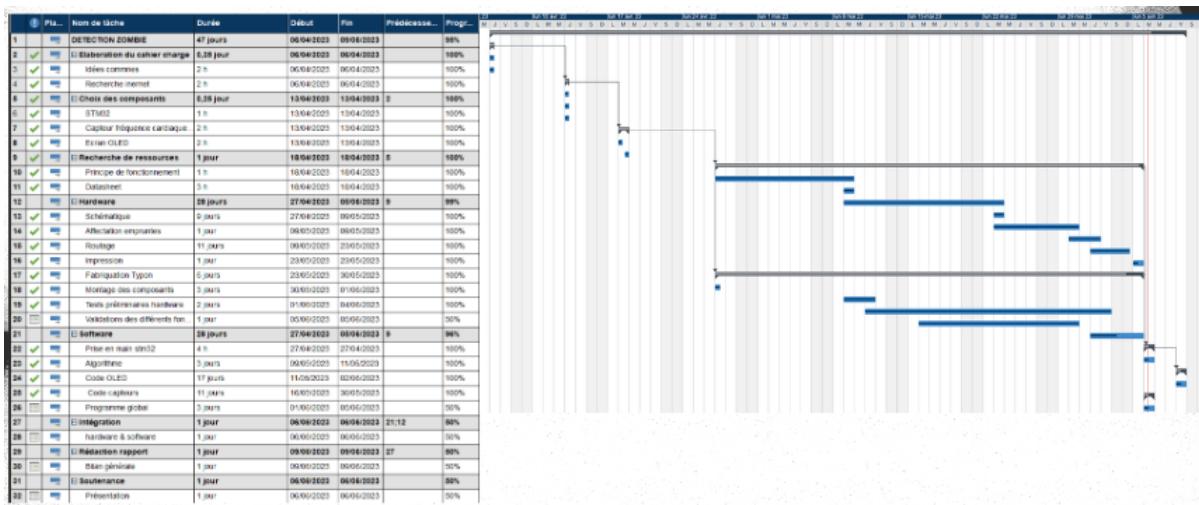
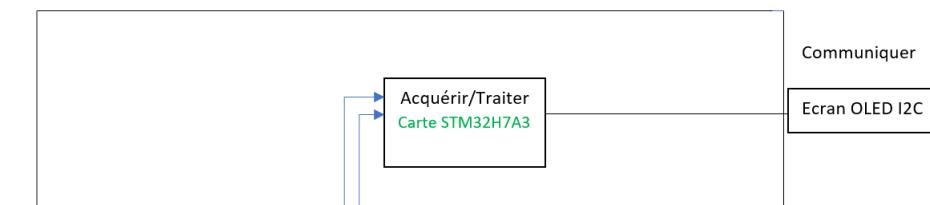
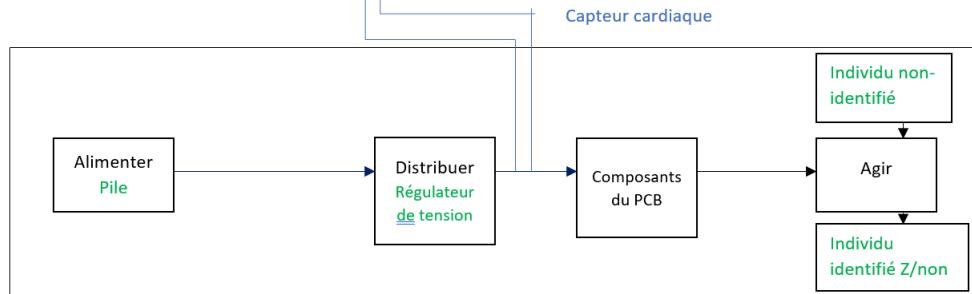


Schéma fonctionnel

Chaîne d'information



Chaîne d'Energie

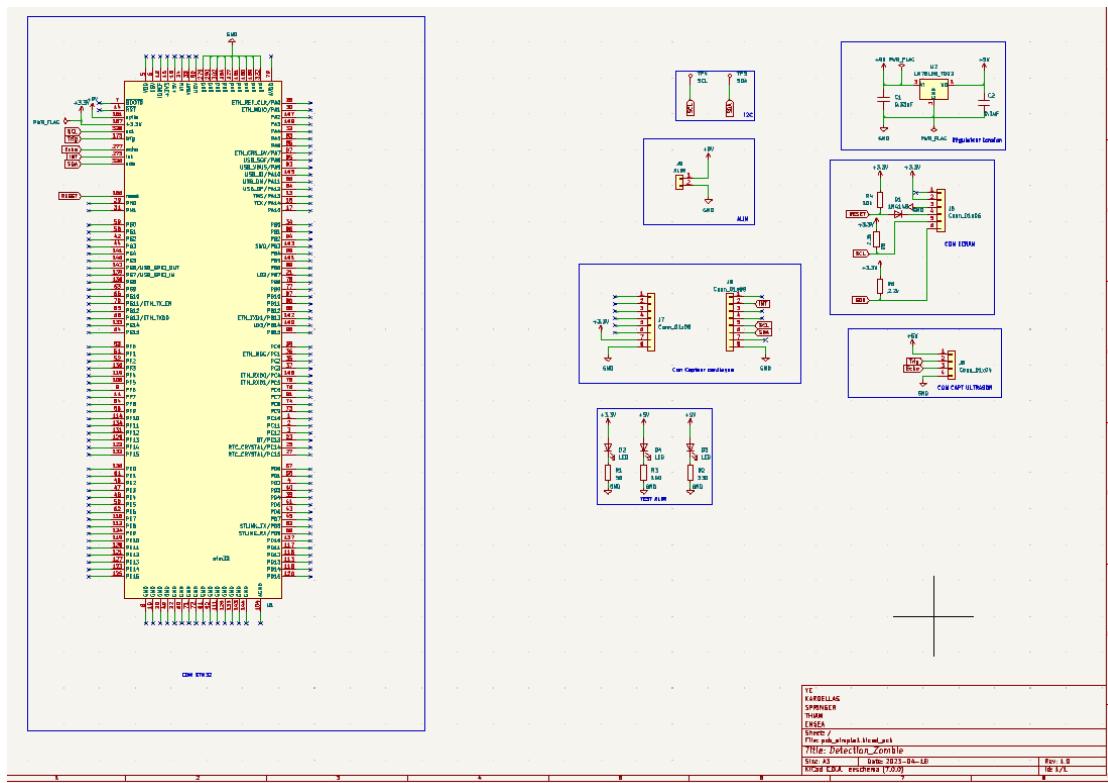


## Hardware - Schématique

Pour la réalisation du pcb, nous avions besoin :

- D'une alimentation externe : pile 9 V
- D'un régulateur 5V
- D'un capteur fréquence cardiaque et d'un capteur ultrason pour connaître la vitesse de la personne
- D'un écran OLED pour afficher les mesures et dire si la personne est un zombie ou pas
- D'un microcontrôleur : le STM32H7A3

La tension 5V en sortie du régulateur a permis d'alimenter le capteur ultrason. Concernant le capteur fréquence cardiaque et l'écran OLED, elles sont alimentées en 3.3V issue du régulateur interne du microcontrôleur. Une fois que les composants ont été définis, on s'est basé sur les datasheets pour effectuer le schématique ci-dessous sur Kicad:



## Hardware - Attribution des empreintes et routage

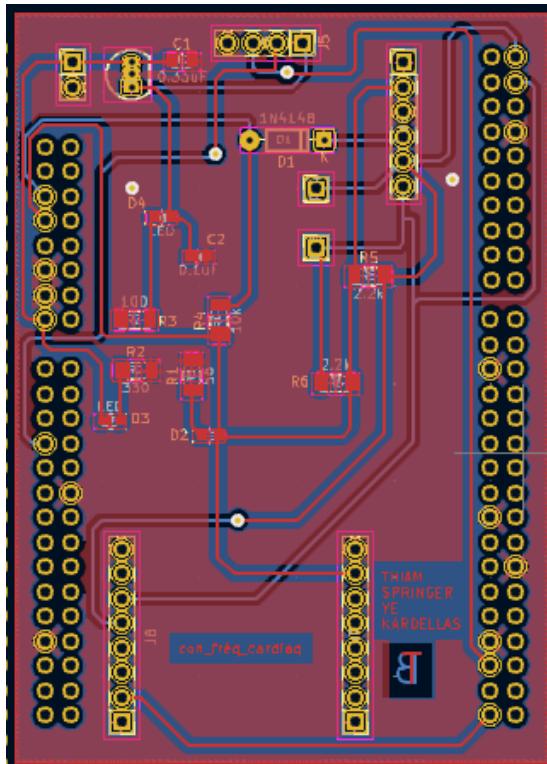
Après avoir vérifié l'absence d'erreur sur le schématique, on a assigné à nos différents composants des empreintes qui représentent les dimensions du composant.

Symbole: Attribution Empreintes	
1	C1 -
2	C2 -
3	D1 -
4	D2 -
5	D3 -
6	D4 -
7	J5 -
8	J6 -
9	J7 -
10	J8 -
11	J9 -
12	R1 -
13	R2 -
14	R3 -
15	R4 -
16	R5 -
17	R6 -
18	TP4 -
19	TP5 -
20	U1 -
21	U2 -
	0.33uF : Capacitor_SMD:C_0805_2012Metric_Pad1.18x1.45mm_HandSolder
	0.1uF : Capacitor_SMD:C_0603_1608Metric_Pad1.08x0.95mm_HandSolder
	1N4148 : Diode_THT:D_DO-35_SOD27_P7.62mm_Horizontal
	LED : LED_SMD:LED_0603_1608Metric_Pad1.05x0.95mm_HandSolder
	LED : LED_SMD:LED_0603_1608Metric_Pad1.05x0.95mm_HandSolder
	LED : LED_SMD:LED_0603_1608Metric_Pad1.05x0.95mm_HandSolder
	Conn_01x04 : Connector_PinHeader_2.54mm:PinHeader_1x04_P2.54mm_Vertical
	Conn_01x06 : Connector_PinHeader_2.54mm:PinHeader_1x06_P2.54mm_Vertical
	Conn_01x08 : Connector_PinHeader_2.54mm:PinHeader_1x08_P2.54mm_Vertical
	Conn_01x08 : Connector_PinHeader_2.54mm:PinHeader_1x08_P2.54mm_Vertical
	ALIM : Connector_PinHeader_2.54mm:PinHeader_1x02_P2.54mm_Vertical
	56 : Resistor_SMD:R_1206_3216Metric_Pad1.30x1.75mm_HandSolder
	330 : Resistor_SMD:R_1206_3216Metric_Pad1.30x1.75mm_HandSolder
	100 : Resistor_SMD:R_1206_3216Metric_Pad1.30x1.75mm_HandSolder
	10k : Resistor_SMD:R_1206_3216Metric_Pad1.30x1.75mm_HandSolder
	2.2k : Resistor_SMD:R_1206_3216Metric_Pad1.30x1.75mm_HandSolder
	2.2k : Resistor_SMD:R_1206_3216Metric_Pad1.30x1.75mm_HandSolder
	SCL : Connector_PinHeader_2.54mm:PinHeader_1x01_P2.54mm_Vertical
	SDA : Connector_PinHeader_2.54mm:PinHeader_1x01_P2.54mm_Vertical
	stm32 : zi_connector:zio_connector
	LM78L05_TO92 : Package_TO_SOT_THT:TO-92_Inline

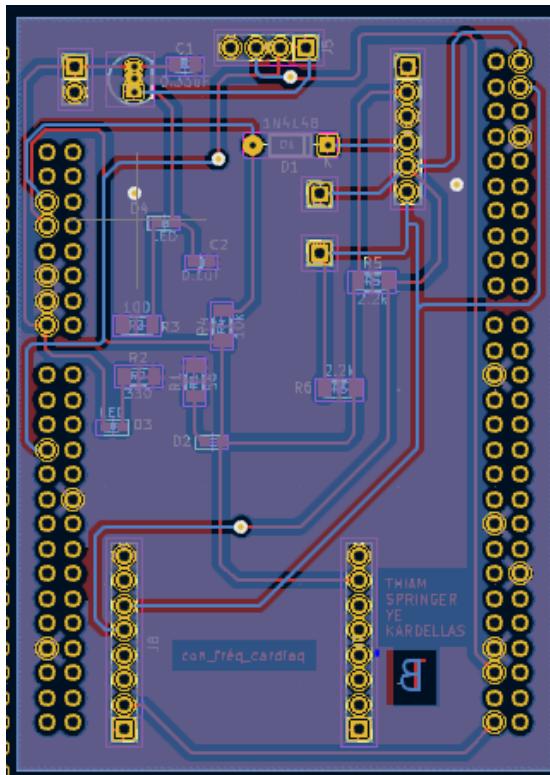
Une fois que les empreintes sont assignées, nous sommes passés d'abord à l'importation des composants, ensuite à l'emplacement d'un contour et puis au réglage aux classes d'équipotentielle ci-dessous.

Nom	Isolation	Largeur Piste	Diam Via	Trou de via	Diamètre uVia	Trou de microvia	Largeur Paire Diff	Dist Paire Diff
Default	0,3 mm	0,4 mm	1,2 mm	0,6 mm	0,3 mm	0,1 mm	0,2 mm	0,25 mm
ALIM	0,3 mm	0,4 mm	2,2 mm	1 mm	0,3 mm	0,1 mm	0,2 mm	0,25 mm

On a par la suite réalisé le routage en respectant les critères suivants :



Sur le TOP



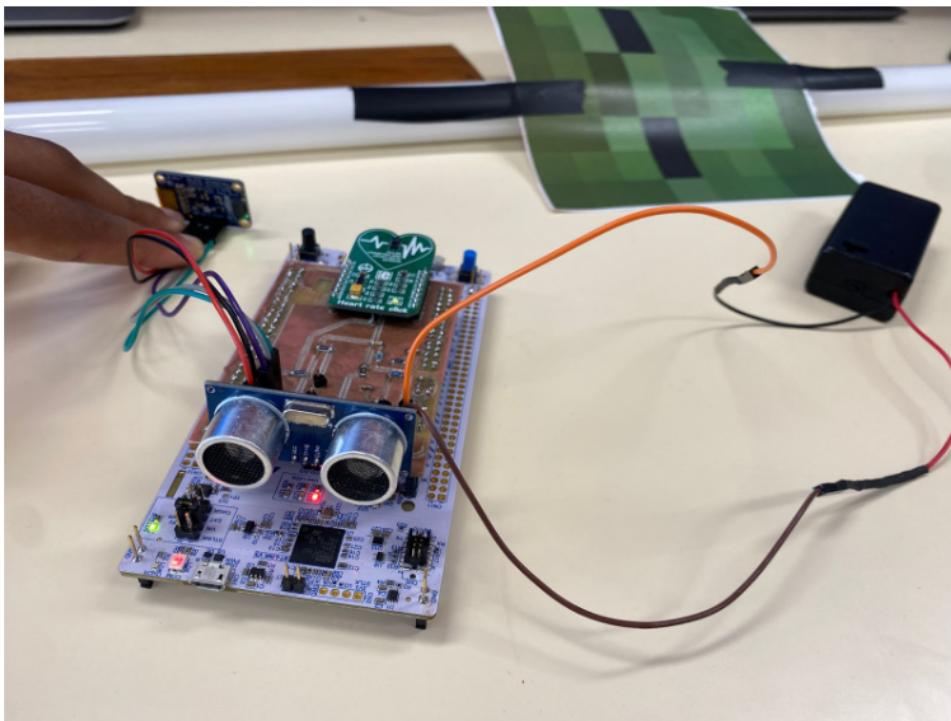
Sur le Bottom

Remarque : Comme les connecteurs du STM32 utilisés n'étaient pas visibles sur Kicad, ces symboles et empreintes sont créées en modifiant une autre nucléo et en reliant les pins aux signaux dont on avait besoin.

## Hardware - Soudure & Tests préliminaires

Après que la carte soit fabriquée et les trous percés, nous avons grâce à de l'étain et du fer à souder soudés les leds, résistances et condensateurs (cms), ensuite les connecteurs des différents capteurs, de l'écran, de l'alimentation et la diode (traversants). Les pattes de la diode en excès ont été utilisées pour les vias.

Lorsque les différents composants ont été soudés, nous avons d'abord vérifié la continuité des pistes grâce à un multimètre, ensuite en alimentant avec la pile mesuré la tension de sortie de régulateur ainsi que les tensions aux bornes des différents capteurs et de l'écran OLED.



On remarque que les leds de l'alimentation du capteur fréquence cardiaque et l'écran OLED sont allumés : il n'y a aucun problème vis-à-vis des continuités de tension ou du PCB.

## Software - Capteur ultrason

### Présentation du capteur ultrason HC-SR04 (Datasheet)

- Principe de fonctionnement : Le capteur HC-SR04 utilise des ondes ultrasonores pour mesurer les distances. Il émet un signal sonore ultrasonore et détecte le temps écoulé jusqu'à ce que le signal soit réfléchi par un objet et revienne au capteur.
- Plage de mesure : Le capteur peut mesurer des distances allant de 2 cm à 400 cm avec une précision de  $\pm 3$  mm.
- Interface : Le capteur utilise une interface de type GPIO (General Purpose Input/Output) pour déclencher la mesure et recevoir les données de distance.
- Signaux de contrôle : Pour démarrer une mesure, il faut envoyer une impulsion de déclenchement (Trig) d'une durée minimale de 10  $\mu$ s. Le capteur émet ensuite une série d'impulsions ultrasonores et attend la réception de l'écho. La durée de l'écho est ensuite convertie en distance.
- Sortie de données : Le capteur renvoie la distance mesurée en utilisant un signal de type impulsion. La durée de l'impulsion correspond à la durée de l'écho.

### Code (msu05.h)

Le code commence par inclure les fichiers d'en-tête nécessaires, y compris "msu05.h" et "main.h", ainsi que les fichiers spécifiques à la gestion du timer (TIM) pour mesurer le temps. Il déclare également deux variables globales : "distance" pour stocker la distance mesurée et "vitesse" pour calculer la vitesse en fonction des variations de distance.

Ensuite, nous avons une fonction "msu05\_Trig" qui envoie une impulsion de 10 ms sur la broche de déclenchement (TRIG\_PIN). Cette impulsion est nécessaire pour activer la mesure de distance par le capteur.

La fonction suivante est "msu05\_Echo" qui effectue la mesure réelle de la distance. Elle utilise une boucle pour attendre que le signal d'écho du

capteur soit activé (HIGH). Une fois que le signal d'écho est détecté, elle enregistre le temps de départ (startTime) en utilisant le timer, puis attend que le signal d'écho revienne à l'état bas (LOW). Elle enregistre ensuite le temps actuel (timeStamp) à l'aide du timer.

La distance est calculée en soustrayant le temps de départ du temps actuel et en le divisant par un facteur de conversion (58.9 dans ce cas) pour obtenir la distance en centimètres, conformément aux spécifications du datasheet du capteur

De plus, la vitesse est calculée en utilisant la différence de distance entre les mesures précédentes (old\_distance) et actuelles, divisée par la différence de temps correspondante (old\_Time et timeStamp). Cela permet d'estimer la vitesse de déplacement de l'objet détecté.

```

8 #include "msu05.h"
9 #include "main.h"
10 #include <stm32h7xx_hal_tim.h>
11 #include <stm32h7xx_hal_tim_ex.h>
12 extern TIM_HandleTypeDef htim2;
13 extern float distance;
14 float vitesse;
15
16 //Fonction Trig
17 void msu05_Trig (void) {
18     HAL_GPIO_WritePin(TRIG_PORT, TRIG_PIN, 1);
19     HAL_Delay(10); //envoyer une impulsion de 10ms à pin trig
20     HAL_GPIO_WritePin(TRIG_PORT, TRIG_PIN, 0);
21     HAL_Delay(1000);
22 }
23 //Fonction Mesure
24 void msu05_Echo (void) {
25     static int startTime=0;
26     static int old_distance=0;
27     static int old_Time;
28     static int timeStamp;
29
30     while (HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0)==0){}
31     startTime =__HAL_TIM_GET_COUNTER(&htim2);
32     while (HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0)==1){}
33     timeStamp=__HAL_TIM_GET_COUNTER(&htim2);
34     distance = (timeStamp-startTime)*1.7/58.9; //datasheet en cm
35
36     vitesse = (distance-old_distance)/(timeStamp-old_Time);
37
38     old_distance = distance;
39     old_Time=timeStamp;
40 }
```

## Code(main.c)

Ce qui est vraiment intéressant et un peu difficile à faire est la mise en œuvre des tâches parallèles dans le main.c afin de garantir un fonctionnement efficace et simultané.

Nous avons créé deux tâches distinctes, à savoir 'StartTask02' et 'StartTask03'. La tâche 'StartTask02' est responsable de déclencher régulièrement les mesures en envoyant une impulsion au capteur HC-SR04 à l'aide de la fonction 'msu05\_Trig()'. Cette tâche permet de synchroniser les envois d'impulsions de déclenchement.

La tâche 'StartTask03' s'occupe de récupérer régulièrement les mesures de distance en exécutant la fonction 'msu05\_Echo()'. Cette fonction réalise les mesures réelles de distance en utilisant le signal d'écho du capteur. Grâce à cette tâche, nous pouvons obtenir des mises à jour périodiques des valeurs de distance mesurées.

En utilisant des tâches parallèles, nous assurons un fonctionnement simultané et indépendant des opérations de déclenchement et de mesure du capteur HC-SR04. Cela nous permet d'optimiser les ressources du microcontrôleur et de maintenir une exécution fluide des autres parties de notre programme.

Il est important de noter que nous avons veillé à configurer correctement notre système d'exploitation et les priorités des tâches pour garantir un fonctionnement cohérent et synchronisé.

```
> void StartDefaultTask(void *argument)
{
    /* USER CODE BEGIN 5 */
    /* Infinite loop */
    for(;;)
    {
        osDelay(1);
    }

void StartTask02(void *argument)
{
    for(;;)
    {
        msu05_Trig();
    }
}

void StartTask03(void *argument)
{
    long startTime = 0;
    for(;;)
    {
        msu05_Echo();
    }
}
```

## Software - Capteur de fréquence cardiaque

### Présentation du capteur de fréquence cardiaque MAX30100 (Datasheet)

- Le capteur utilise une combinaison de LEDs (rouge et infrarouge) et d'un photodétecteur pour capturer les variations de la lumière réfléchie par la peau, fournissant ainsi des informations sur la fréquence cardiaque.
- En plus de la mesure de la fréquence cardiaque, le capteur MAX30100 intègre un capteur de température qui permet de mesurer la température corporelle avec précision.
- Les adresses de différents registres sont importantes pour la lecture des données dans le code :

```

#define DEVICE_ADDRESS_WRITE 0xAE // Adresse du capteur MAX30100 (mode écriture)
#define DEVICE_ADDRESS_READ 0xAF // Adresse du capteur MAX30100 (mode lecture)
#define READ_MODE_MASK 0x01 // Masque pour le mode lecture
#define FIFO_WR_PTR 0x02 // Adresse de FIFO_WR_PTR
#define FIFO_DATA 0x05 // Adresse de FIFO_DATA
#define FIFO_RD_PTR 0x04 // Adresse de FIFO_RD_PTR
extern I2C_HandleTypeDef hi2c1;
  
```

### Code(getComponentID)

Cette fonction permet de lire l'identifiant du composant MAX30100 en envoyant une demande de lecture à l'adresse spécifiée. Si l'identifiant correspond à la valeur attendue (0x11 dans ce cas), la fonction renvoie 1, sinon elle renvoie 0.

```

int getComponentID(void){
  char toReceive[1]={0};
  HAL_StatusTypeDef dummyStatus;
  dummyStatus=HAL_I2C_Mem_Read(&hi2c1,0xAE,0xFF,1,toReceive,1,-1);
  if (toReceive[0]==0x11)
    return 1;
  return 0;
}
  
```

### Code(sendConfig)

La fonction "sendConfig()" a pour but d'envoyer une configuration spécifique au capteur afin de le mettre dans un état de fonctionnement optimal pour la

collecte des données de fréquence cardiaque et de température. Ici nous avons choisi la configuration représentée par la valeur 0x0a dans la variable "toSend".

```
> void sendConfig(){
    char toSend[1]={0x0a};
    HAL_StatusTypeDef dummyStatus;
    dummyStatus=HAL_I2C_Mem_Write(&hi2c1,0xAE,0x06,1,toSend,1,-1);
}
```

### **Code(getPulse)**

La fonction "getPulse()" est responsable de la mesure de la fréquence cardiaque à partir des données captées par le capteur. Voici comment le code fonctionne :

#### 1) Lecture du registre "interruptstatus" :

- La variable "interrupt" est initialisée à 0.
- Une boucle est utilisée pour lire le registre "interruptstatus" tant que le bit 7 (0x80) n'est pas activé (FULL).
- Cela permet d'attendre que de nouvelles données soient disponibles dans le registre.

#### 2) Stockage des données IR :

- Un tableau "irdata" de 16 éléments est créé pour stocker les données infrarouges captées par le capteur.
- Un tableau temporaire "data" de 4 octets est utilisé pour lire les données depuis le registre "FIFO\_DATA".
- Une boucle est utilisée pour lire et extraire les données IR (16 bits) dans le tableau "irdata".

#### 3) Tri des données IR :

- Une fois les données IR capturées, elles sont triées dans l'ordre croissant.
- Cela permet de déterminer la valeur médiane, qui est un indicateur plus robuste pour estimer la fréquence cardiaque et réduire les erreurs causées par des fluctuations temporaires.

#### 4) Calcul de la fréquence cardiaque :

- La valeur médiane des données IR est calculée en additionnant les éléments d'indice 7 et 8, puis en les divisant par 2.

Yumeng YE

THIAM Fatima Tall

SPRINGER Yacine

KARDELLAS Amin

Encadrant : M.Tauvel

- Cette valeur médiane est considérée comme une estimation de la fréquence cardiaque.

### 5) Renvoi de la valeur de la fréquence cardiaque :

- La valeur médiane est renvoyée par la fonction "getPulse()" pour être utilisée dans d'autres parties du code.

```

int getPulse(){
    //on lit le registre interruptstatus
    char interrupt[1]={0};
    HAL_StatusTypeDef dummyStatus;

    do{
        dummyStatus=HAL_I2C_Mem_Read(&hi2c1, 0xAE, 0x00, 1, interrupt, 1, -1);
    }while((interrupt[0]&0x80)!=0x80);

    char irdata[16]={0}; // Tableau pour stocker les données IR

    char data[4]={0}; //// Tableau temporaire pour lire les données depuis FIFO_DATA
    for (int i = 0; i < 16; i++)
    {
        HAL_StatusTypeDef dummyStatus;
        dummyStatus=HAL_I2C_Mem_Read(&hi2c1, 0xAE, 0x05, 1, data, 4, -1);

        // Extraction des données IR (16 bits) et stockage dans data
        irdata[i] = (data[0] << 8) | data[1];
    }

    for (int i = 0; i < 15; i++) {
        for (int j = i + 1; j < 16; j++) {
            if (irdata[j] < irdata[i]) {
                int temp = irdata[i];
                irdata[i] = irdata[j];
                irdata[j] = temp;
            }
        }
    }

    int mediane = (irdata[7]+irdata[8])/2;
    return mediane;
}

```

### Code(getTemperature)

La fonction "getTemperature()" est responsable de la mesure de la température à partir des données fournies par le capteur MAX30100. Voici comment le code fonctionne :

#### 1) Lecture du registre de température :

- Un tableau "toReceive" de 2 octets est créé pour stocker les données de température.

- La fonction "HAL\_I2C\_Mem\_Read()" est utilisée pour lire les données du registre spécifique (0x16) à partir de l'adresse du capteur MAX30100 (0xAE).

- Les données lues sont stockées dans le tableau "toReceive".

## 2) Renvoi de la valeur de température :

- La fonction "getTemperature()" renvoie la première valeur du tableau "toReceive", qui correspond à la mesure de température captée par le capteur.

```
int getTemperature(){
    char toReceive[2]={0};
    HAL_StatusTypeDef dummyStatus;
    dummyStatus=HAL_I2C_Mem_Read(&hi2c1,0xAE,0x16,1,toReceive,2,-1);
    return toReceive[0];
}
```

## Code(main.c)

La tâche "StartTask04" fonctionne en conjonction avec les tâches "echo" et "trig". Le code de cette tâche effectue les opérations suivantes :

- Au démarrage de la tâche, les variables "test", "temperature" et "pulse" sont initialisées pour la gestion des mesures de température et de fréquence cardiaque.
- Dans une boucle infinie, le code effectue les opérations suivantes :
  - 1) Attendre 100 ms pour réguler la fréquence des mesures.
  - 2) Configurer le capteur en appelant la fonction "sendConfig()".
  - 3) Mesurer la température en appelant la fonction "getTemperature()" et stocker la valeur dans "temperature".
  - 4) Mesurer la fréquence cardiaque en appelant la fonction "getPulse()" et stocker la valeur dans "pulse".

```
void StartTask04(void *argument)
{
    int test = getComponentID();
    char temperature;
    char pulse;
    for(;;)
    {
        osDelay(100);
        sendConfig();
        temperature=getTemperature();
        pulse=getPulse();
    }
}
```

## Software - Ecran



Seules 4 pattes nous intéressent pour l'écran OLED MCOT 096016AY-WI , à savoir GND, SDA pour les données, SCL pour la synchronisation et Vin pour l'alimentation. Deux écrans étaient disponibles mais celui-ci s'est imposé puisqu'il ne pose aucun problème de continuité de tension avec le PCB, en effet il est alimenté en 3V3. On souhaite à l'aide d'un bouton poussoir afficher différents menus : le premier avec un message d'accueil, trois autres avec les valeurs mesurées et enfin un dernier avec le message "alerte zombie" si l'individu traité est reconnu comme étant un zombie. L'affichage d'un texte ce fait comme suit :

On définit tout d'abord le texte à afficher puis l'écran OLED doit être initialisé aux coordonnées où l'on souhaite afficher celui-ci. Après chaque affichage l'écran doit être "rafraîchi" pour accueillir le texte suivant :

```
int main(void)
{
    /* USER CODE BEGIN 1 */
    //      char myText[] = "Detecteur Zombie";
    //      char retVal;

    /* Affichage d'un texte predefini*/
    //  ssd1306_Init();
    //  ssd1306_SetCursor(5,5);
    //  retVal = ssd1306_WriteString(myText, Font_7x10, White);
    //  ssd1306_UpdateScreen();
```



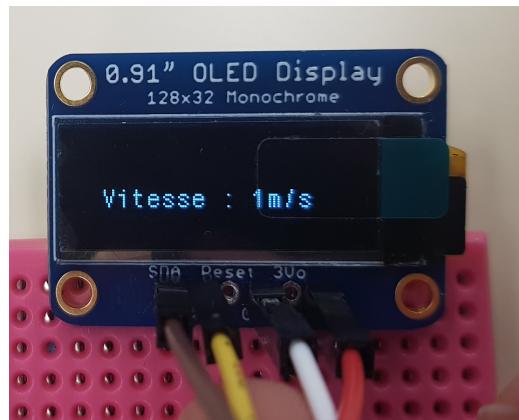
Une subtilité persiste, les bibliothèques nécessaires pour utiliser l'écran OLED doivent être téléchargées au préalable et ajoutées au fichier contenant le projet. Maintenant, il faut générer la partie du code qui nous permettra d'accéder aux différents menus d'affichage de l'écran OLED. Dans ce code mis à jour, nous avons ajouté une variable "numButtonPresses" pour suivre le nombre d'appuis sur le bouton. Lorsqu'un appui sur le bouton est détecté, nous incrémentons cette variable. Si numButtonPresses est égal à 1, nous inverserons simplement l'état de la LED comme auparavant. Si numButtonPresses est égal à 2, nous allumerons constamment la LED. Enfin, si numButtonPresses est égal à 3 (zombie détecté ou pas?!), nous ferons clignoter la LED en la faisant s'allumer et s'éteindre à intervalles réguliers pendant 10 cycles, puis nous réinitialiserons numButtonPresses. Ci-dessous nous pouvons trouver les configurations des GPIO pour la LED :

```
// Configuration du GPIO pour la LED
GPIO_InitStruct.Pin = GPIO_PIN_0;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

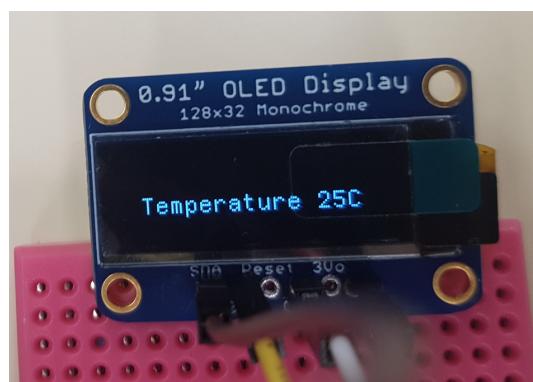
// Configuration du GPIO pour le bouton
GPIO_InitStruct.Pin = GPIO_PIN_13;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
```

On obtient donc les résultats suivants :

- Appui une première fois



- Appui une deuxième fois



- Appui une troisième fois



## Software - Tests préliminaires

### Capteur ultrason



Nous avons effectué des tests préliminaires avec le capteur ultrason pour mesurer la vitesse des objets se déplaçant devant le capteur. L'objectif était de déterminer si le capteur fournissait des mesures précises et fiables dans différentes plages de distance.

Les résultats des tests ont démontré que le capteur ultrasonore était efficace pour mesurer la vitesse dans la plupart des cas, en fournissant des valeurs cohérentes et précises. Cependant, des limitations ont été observées pour les objets situés à une distance inférieure à 2 cm, ce qui est conforme aux spécifications du datasheet du capteur.

Une observation intéressante a été faite pour les objets situés à une distance supérieure à 15 cm. Dans ces cas, les mesures de vitesse ont montré une certaine imprécision et une moins grande fiabilité. Des écarts significatifs ont été observés entre les valeurs mesurées et les vitesses réelles des objets en mouvement.

Dans l'ensemble, malgré ces limitations, le capteur ultrason HC-SR04 s'est révélé prometteur pour la détection de la vitesse des objets, offrant des résultats satisfaisants dans une plage de distance appropriée.

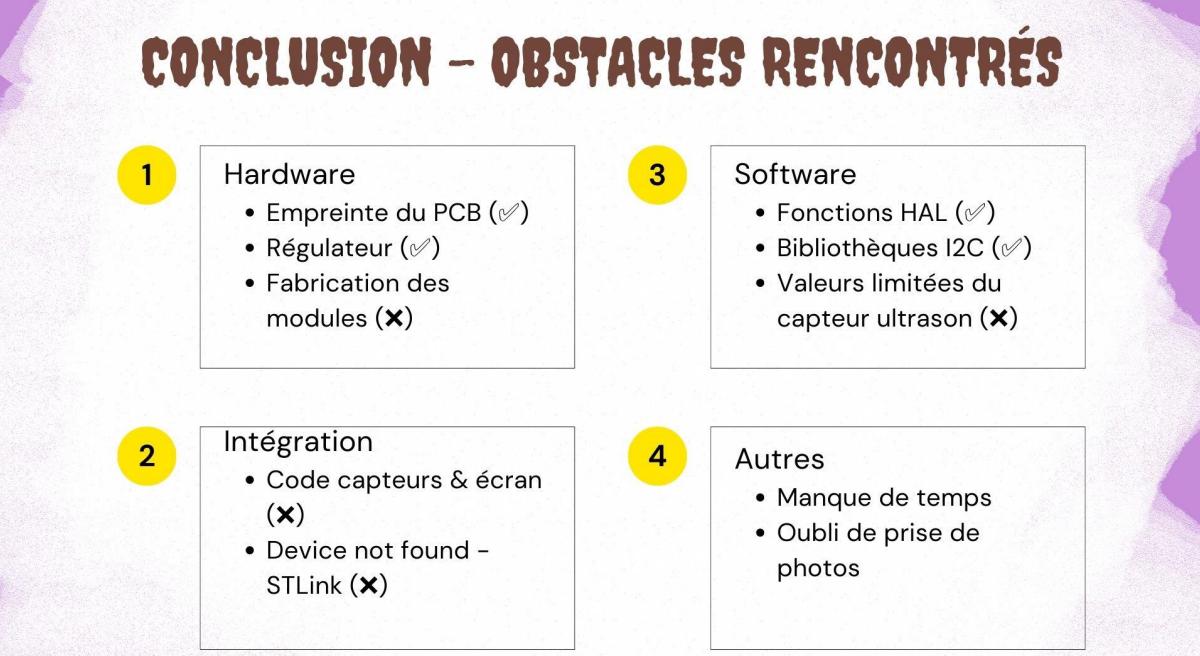
## Partie mécanique



Il s'agit d'un collet mécanique en métal qui fonctionne par télescopage : un cylindre en métal emboîté dans un manche métallique coulisse selon une liaison pivot pour resserrer un lacet en fibre de carbone. Deux intérêts majeurs se présentent à nous : une grande malléabilité de l'outil pour venir piéger l'individu au niveau du poignet pour réaliser les tests et une protection de l'intégrité physique de celui-ci. En effet, une première approche était d'utiliser des fils en nylon pour piéger de la même manière l'individu au niveau de la gorge mais pour des raisons de sécurité cette piste a été abandonnée. La solution retenue permet d'ailleurs de réaliser les tests en protégeant le suspect tout comme notre équipe.

## Conclusion globale

### Obstacles rencontrés



**CONCLUSION – OBSTACLES RENCONTrés**

<b>1</b>	<b>Hardware</b> <ul style="list-style-type: none"><li>• Empreinte du PCB (✓)</li><li>• Régulateur (✓)</li><li>• Fabrication des modules (✗)</li></ul>	<b>3</b>	<b>Software</b> <ul style="list-style-type: none"><li>• Fonctions HAL (✓)</li><li>• Bibliothèques I2C (✓)</li><li>• Valeurs limitées du capteur ultrason (✗)</li></ul>
<b>2</b>	<b>Intégration</b> <ul style="list-style-type: none"><li>• Code capteurs &amp; écran (✗)</li><li>• Device not found – STLink (✗)</li></ul>	<b>4</b>	<b>Autres</b> <ul style="list-style-type: none"><li>• Manque de temps</li><li>• Oubli de prise de photos</li></ul>

### Conclusion globale

Pour finir, durant ce projet nous avons rencontré certaines difficultés, autant pour la conception du PCB que pour la réalisation des programmes en software.

Pour la partie hardware, un des problèmes majeurs rencontrés est celui du choix de l'empreinte pour la réalisation du routage. Le problème étant que l'empreintes de notre stm32 n'existe pas sur Kicad. Nous avons pris beaucoup de temps pour choisir la solution la plus adaptée afin d'obtenir un PCB qui se clipse à notre board stm32.

Yumeng YE  
THIAM Fatima Tall  
SPRINGER Yacine  
KARDELLAS Amin  
Encadrant : M.Tauvel

Finalement, après avoir résolu toutes ces difficultés imprévues pour la partie hard et soft, il ne nous restait plus énormément de temps pour implémenter nos codes dans la board stm32 et donc plus beaucoup de temps pour la phase d'intégration.

En plus du manque de temps, nous avons immédiatement fait face à un souci majeur au début de la phase d'intégration. Un message d'erreur apparaissait sur le logiciel “device ST-Link is not found”. Lorsque nous branchions notre PCB à la board stm32 il nous était impossible d'implémenter les codes des 2 capteurs ainsi que ceux de l'écran OLED. Toutefois, nous avons persisté jusqu'au dernier moment sans abandonner et nous avons réussi à trouver la raison de ce message d'erreur. Au début nous pensions que nos cartes étaient sûrement grillées suite à un court-circuit causé par le PCB. Mais en réalité c'était un faux contact dû au fait que les pins du PCB étaient tordus et créaient un “faux contact” avec les broches femelles du stm32.

Pour finir, nous avons pu résoudre ce problème, nous avons implémenté les codes mais aussi affiché des messages avec l'écran OLED directement soudés à notre PCB qui lui-même est alimenté avec notre pile. Le bouton poussoir fonctionne également et nous permet de passer d'un message affiché à un autre.

Pour conclure, ce projet a été une source d'épanouissement pour chacun d'entre nous. Cela nous a permis de mettre en pratique nos connaissances mais aussi d'en apprendre plus sur le métier d'ingénieur en électronique. Nous tenons à remercier M.Tauvel pour nous avoir permis de mener à bien ce projet ainsi que Mme.Kittel sans qui la réalisation de notre PCB n'aurait pas été possible.

Yumeng YE  
THIAM Fatima Tall  
SPRINGER Yacine  
KARDELLAS Amin  
Encadrant : M.Tauvel



**Démonstration (vidéo d'affichage)**

[https://drive.google.com/file/d/17Rh9QX5GrmEFSrZYWYiDqz654mX4QuSX/view?usp=drive\\_link](https://drive.google.com/file/d/17Rh9QX5GrmEFSrZYWYiDqz654mX4QuSX/view?usp=drive_link)