Système d'alarme

Jérôme PIZEL

Quentin COMTE-GAZ

Projet de 2A

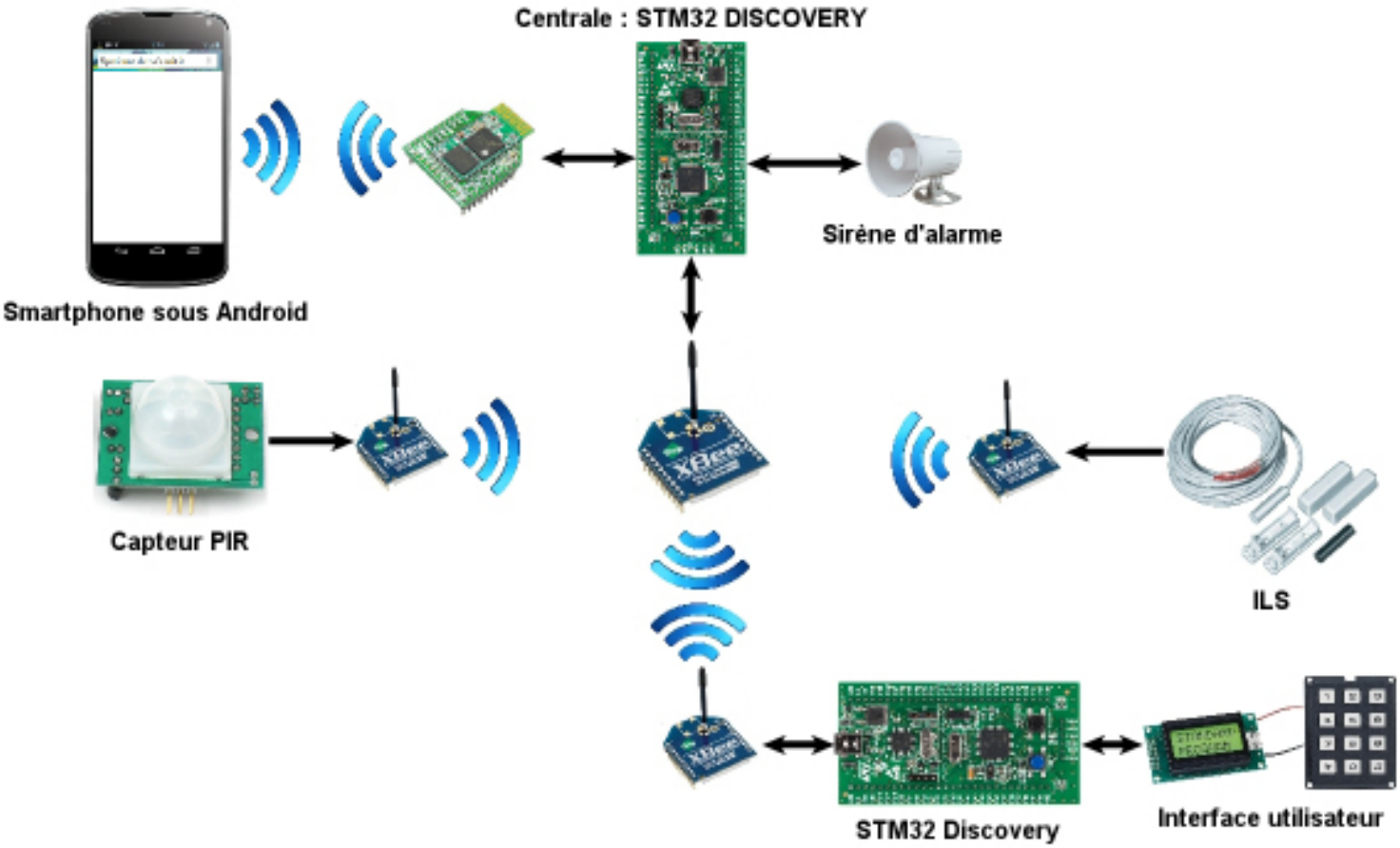
ENSEA

M.Monchal

Sommaire

Introduction

Notre projet consiste à réaliser un système d'alarme sans-fil permettant de protéger une habitation contre une intrusion.
Cette alarme aura initialement pour but de détecter la présence d'un intrus et déclenchera une sonnerie le cas échéant.
Cependant, ce système devra être évolutif afin de pouvoir y adjoindre des fonctions complémentaires.



I) Présentation du projet

1) Cahier des charges

Notre système d'alarme sera composé :
- d'un bloc de commande-calcul (nommé « centrale ») contrôlant le système complet
- de capteurs se trouvant dans des endroits stratégiques et en contact sans fil avec la centrale
- d'un bloc « alarme sonore » en contact filaire avec la centrale
- d'un bloc interface utilisateur permettant d'interagir avec l'utilisateur et la centrale
- de blocs annexes permettant d'améliorer le système (interaction avec l'utilisateur, système GSM permettant d'alerter par SMS le propriétaire en cas d'intrusion, …)

Le cahier des charges est le suivant :

Fonctions principales :

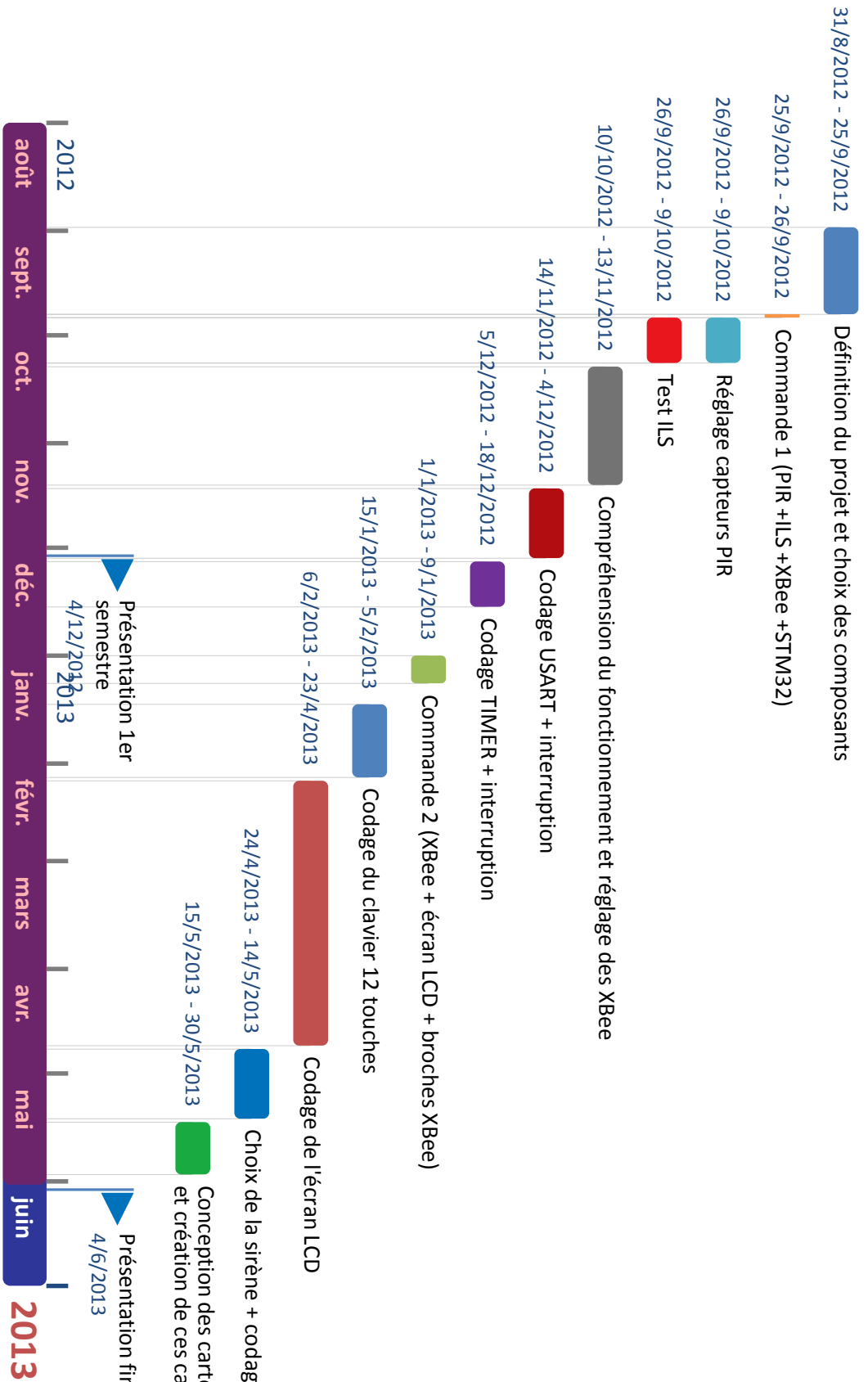
N°	Fonction	Critère	Niveau	Flexibilité	Piste pour la conception
1	Détecter la présence d'individus	- Distance - Mouvement - Chaleur humaine - Taille	>5m >0.1 km/h (doit détecter tout type de passage) > 30°C >30cm (ne pas détecter des animaux)	X	Capteur de présence PIR bien calibré
2	Détecter l'ouverture des fenêtres et des portes	- 2 états (Ouvert/Fermé)	X	X	- Interrupteur à Lames Souples (ILS) + aimant - Capteur de vibration (ressort+aimant+bobine)
3	Interagir avec l'utilisateur	- Afficher l'état de notre système - Possibilité de taper un code secret - Possibilité de déclencher l'alarme manuellement - S'adapter à l'utilisateur : - La maison est inoccupée et le système d'alarme fonctionne - Le propriétaire est dans la maison mais veut tout de même sécuriser quelques zones	X	X	- Afficheur LCD ou/et application JAVA sous Android - Interrupteur Tout ou rien (TOR) - Clavier numérique ou/et application JAVA sous android - Programmation de la centrale pour s'adapter à l'utilisateur

4	Transmettre les informations des capteurs et du bloc « interface utilisateur » vers le bloc « centrale »	- Sans fil (portée) - Transmission sécurisée - Type de données	- 20m - Cryptage 128 bits - Données de type « valeur de tension » pour les capteurs et « données réelles » pour l'interface utilisateur	- 5m - 56 bits - Aucune	- Module XBee - Module Bluetooth - Module radio fréquence basique (modulation/démodulation)
5	Alerter en cas d'intrusion	- Sonore	X	X	Sirène (haut-parleur)

Fonctions optionnelles :

N°	Fonction	Critère	Niveau	Flexibilité	Piste pour la conception
1	Système autonome en électricité OU Continuer à fonctionner même en cas de coupure d'électricité	- Durée --- - Durée - Coupures (transition alimentation secteur→alimentation autonome)	3 mois --- >4 heures <0.1 sec	1 mois --- 1 heure 0.1 sec	Batterie/Piles
2	Envoyer un SMS en cas d'intrusion	X	X	X	Module GSM

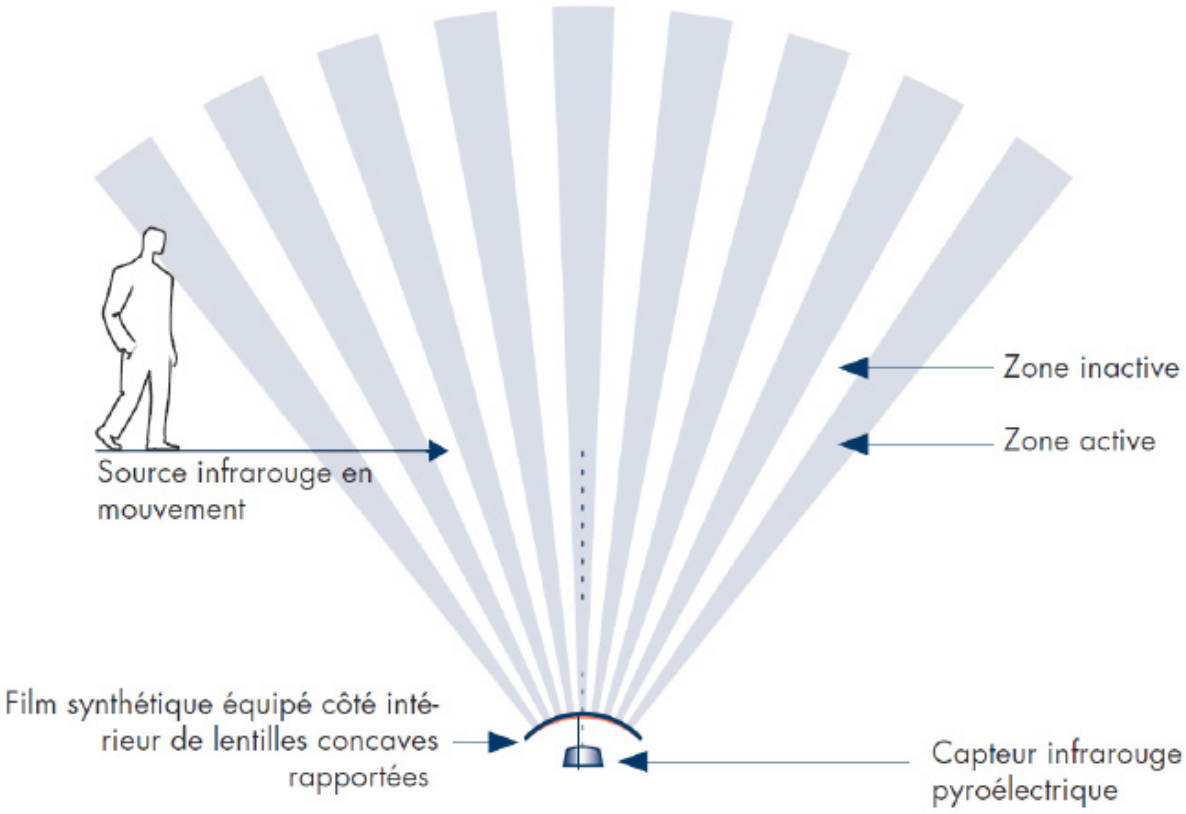
2) Diagramme de Gantt



II) Capteurs

1) Capteur PIR

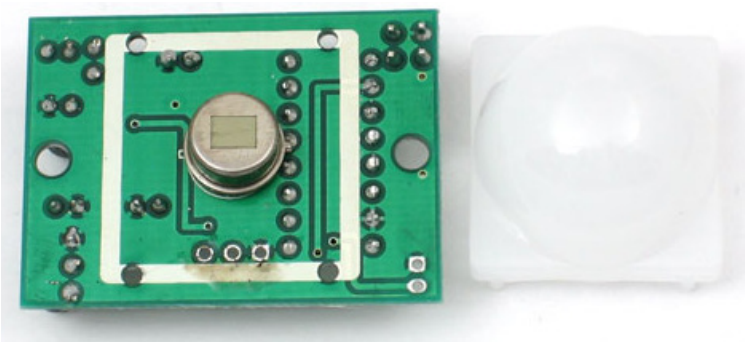
Les détecteurs de mouvement et de présence sont équipés pour détecter des sources de chaleur en mouvement dans leur zone de détection. Chaque objet, en particulier le corps humain, émet un rayonnement thermique dont l'intensité est fonction de sa température de surface. Le rayonnement thermique, aux températures habituelles, fait partie du domaine des rayons infrarouges, invisibles pour l'oeil humain. Un capteur pyroélectrique adapté au domaine infrarouge reçoit ce rayonnement et le convertit en tension électrique. Comme le capteur n'émet pas lui-même de rayonnement, on l'appelle capteur infrarouge passif (ou capteur PIR, de Passive Infra-Red).
Dans une pièce se trouvent en général d'autres objets qui sont plus chauds que la température ambiante, comme des lampes, radiateurs ou appareils électriques. Pour que le détecteur ne détecte que des sources infrarouges en mouvement, le capteur est placé derrière un système optique perfectionné.

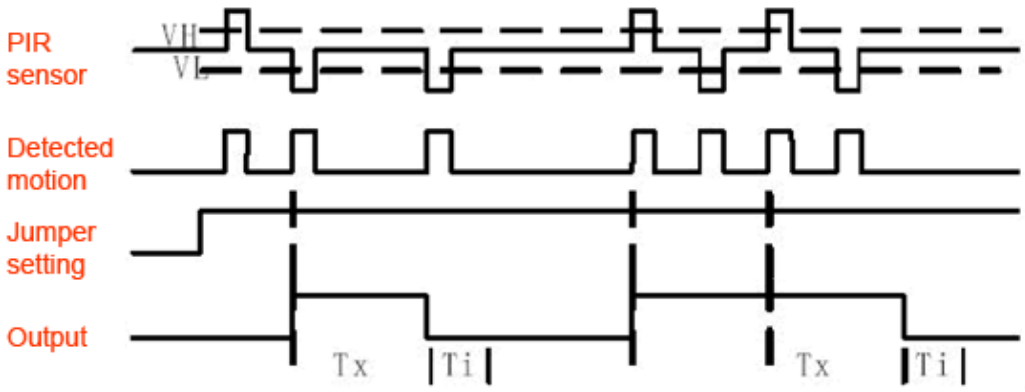


Ce système optique divise, grâce à des lentilles multiples, la zone de détection en un grand nombre de zones actives et inactives. Si une source de chaleur passe d'une zone à une autre, le capteur pyroélectrique envoie un signal qui est traité par un microprocesseur pour commander un système d'alarme.

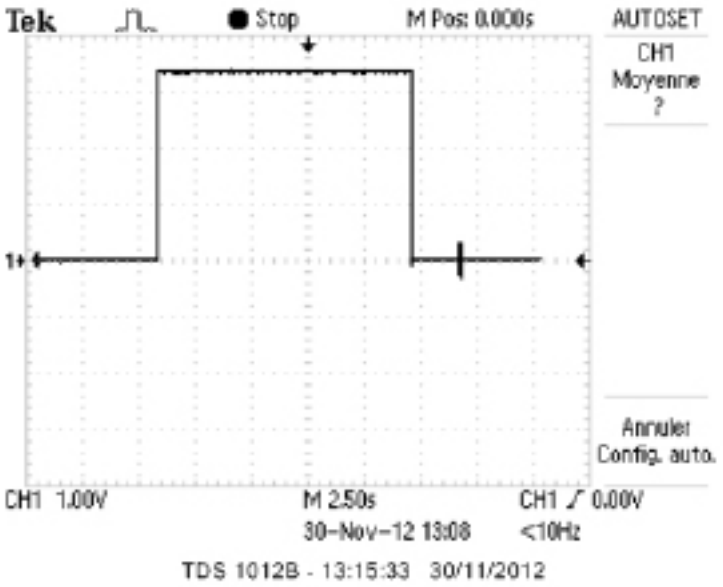
Test

Nous avons au préalable testé les capteurs dès réception, et nous les avons configuré en mode «retriggering» (cavalier en position H). Ainsi dès que le capteur détecte une présence, la sortie du capteur PIR reste activée à l'état haut pendant un temps réglé grâce aux potentiomètres, choisi à 10 secondes (cf le chronogramme ci-après).





Voici un oscillogramme de la sortie de notre capteur PIR : on distingue bien les deux temps, celui de la détection de mouvement, et l'absence de détection.



2) Interrupteur à lames Souples (ILS)

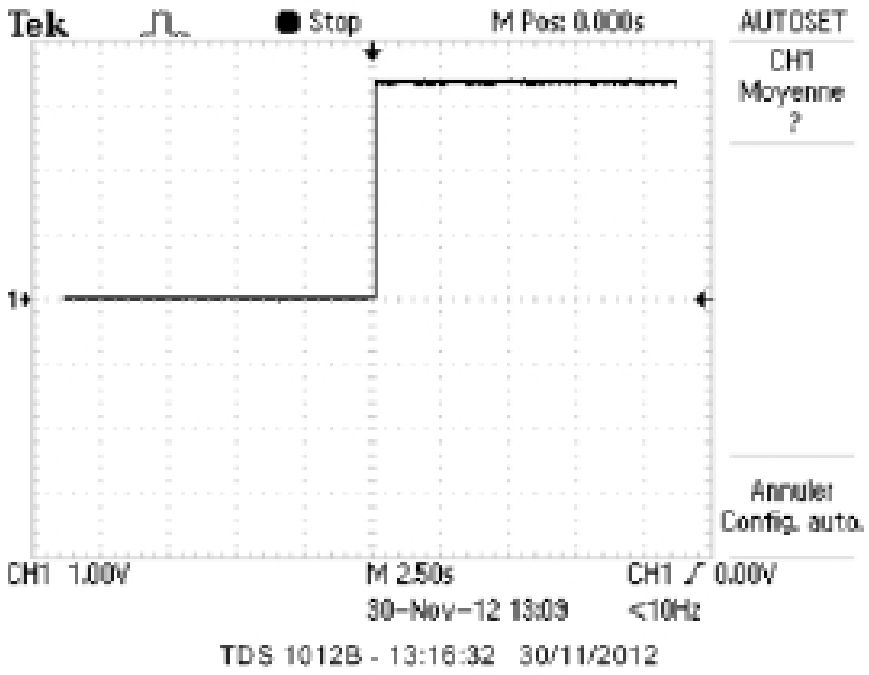
Un interrupteur à lames souples est un interrupteur dont les deux contacts sont en ferronickel, et en présence d'un aimant, les contacts s'aimantent par influence et sont attirés l'un par l'autre. Pour notre projet, les interrupteurs ILS permettront de rajouter une sécurité aux entrées de l'habitation, que ce soit aux portes ou aux fenêtres. En effet, dès que la porte ou la fenêtre sera ouverte, en sortie du ILS on observera un état haut, qui sera ensuite traité par la centrale.

Le contact à ouverture que nous avons choisi dispose d'une protection particulière contre le sabotage : si un aimant étranger au système est disposé à côté de l'aimant du système, ce dernier reconnaît le champ magnétique (modifié). Le contact Reed NC s'ouvre ensuite.

Test

Nous avons tout d'abord testé l'ILS afin de savoir si en sortie on observait bien un état haut (porte ouverte), ce qui était bien le cas. Ainsi il suffit de brancher l'ILS au module XBee afin de traiter l'information.

Voici un oscillogramme des différents états observés : porte ouverte (état haut) et porte fermée (état bas) :

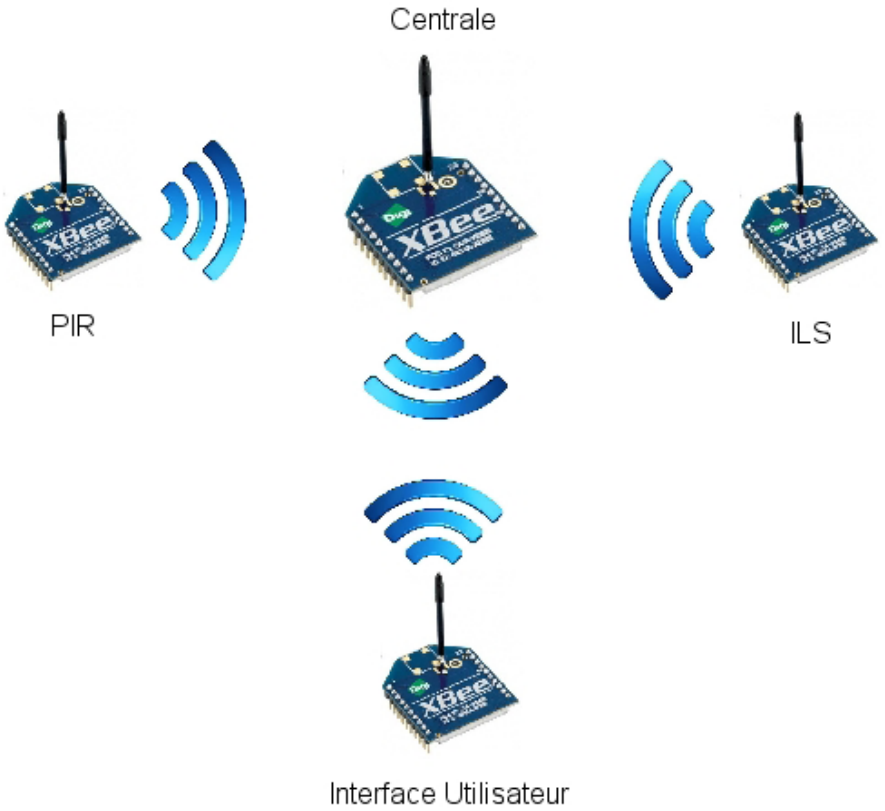


III) Modules XBee

1) Présentation

Pour la transmission des données entre les différents capteurs et la centrale, nous avons choisi d'utiliser des modules XBee XB24. Ces modules XBee ont comme avantage le travail en réseau. Ainsi, dans le cadre de notre projet, nous pouvons aisément faire communiquer les modules des capteurs et celui de la centrale. En effet, chaque module est défini en tant qu'émetteur et/ou récepteur et ne peut communiquer qu'avec les modules que nous avons choisi. De plus, les modules XBee ont une fonction de cryptage, ce qui permet de sécuriser le transfert des données et donc d'empêcher un piratage du système.

Nous avons aussi choisi d'utiliser ces modules XBee car ils consomment peu d'énergie et car ils possèdent un mode «sleep» permettant d'éteindre par intermittence le module. Un autre avantage est la portée qui peut aller avec certains modules XBee jusqu'à 1500 mètres, mais dans notre cas nous nous sommes limité à l'utilisation d'un module XBee disponible à l'école mais ayant une plus courte portée (dans les 20 mètres avec obstacles).
Le débit peut atteindre 250 kbps, mais si on les utilise pour réaliser une liaison série sans fil, les débits standards sont compris entre 9600 bps à 38410 bps. La vitesse et la bande passante ont des effets sur les erreurs de transmission et ne sont pas possibles à obtenir dans tous les environnements.



Réseau de XBee utilisé pour notre projet

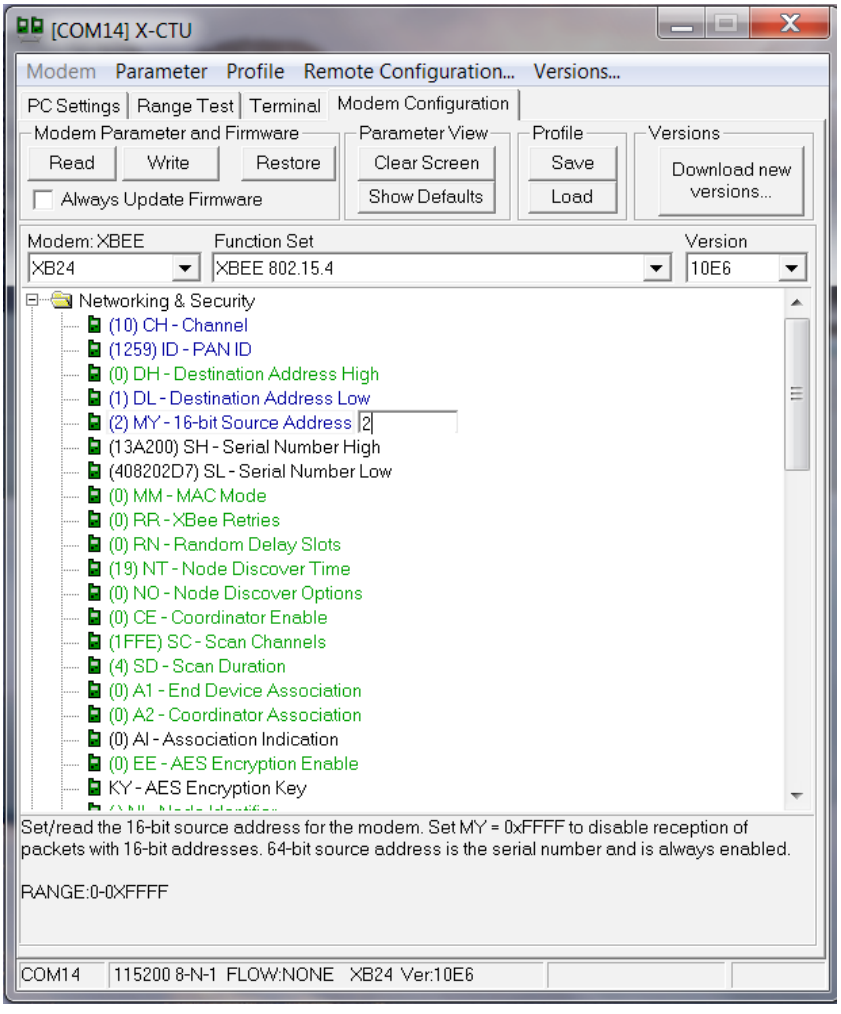
2) Configuration des modules XBee

Pour les modules XBee, nous avons été aidé par le tutoriel/TP de M. Tang (que nous remercions grandement au passage).
Les configurations choisies sur les XBee sont détaillées dans le tableau ci-dessous :

	Centrale	Capteur 1 (PIR)	Capteur 2 (ILS)	Interface utilisateur	Explications
CH	10	10	10	10	Définit le canal et l'identifiant de transmission commun aux XBee
ID	1259	1259	1259	1259	
MY	1	2	3	40	Identifiants des différents XBee. Notons que nous avons mis le MY de l'interface utilisateur à 40 pour avoir la possibilité d'ajouter d'autres capteurs
DL	40	1	1	1	La centrale envoie des données uniquement à l'interface utilisateur. Les autres envoient des données uniquement à la centrale. Rq : Il faut avoir 'FFFF' pour envoyer des données à tous les XBee
EE-AES	1	1	1	1	Activer un cryptage de type AES 128 bits (cryptage très sécurisé)
KY-AES	Qè&Sr5f-.»èqJ(ç+	Qè&Sr5f-.»èqJ(ç+	Qè&Sr5f-.»èqJ(ç+	Qè&Sr5f-.»èqJ(ç+	La clef de cryptage (identique pour tous nos Xbee)
Power lvl	4 -Higher	4 - Higher	4 - Higher	4 - Higher	Nos XBee n'ayant pas une très bonne portée, nous avons préféré augmenter la portée au détriment de l'autonomie
BD	7- 115200	7- 115200	7- 115200	7- 115200	Le Baud Rate permet de définir la vitesse de transmission de notre signal (qui est ici de 115200 Hz)
API	0	1	1	0	Le mode API permet : - de savoir qui a envoyé le message - de savoir à qui le message est destiné - de pouvoir communiquer en réseau et pas seulement entre 2 XBee - d'émettre des données utiles - de vérifier si nos données sont erronées Alors que pour la centrale et l'interface utilisateur il est bien plus aisé de les mettre en mode transparent : en effet avec ce mode ils communiquent seulement entre eux et de manière directe.

I/O D0	**X**	**2 - ADC**	**2 - ADC**	**X**	**On définit ici le port D0 du XBee :** **- La centrale et l'interface utilisateur doivent envoyer/recevoir des TRAMES sur deux ports UART (Dout et Din) adaptés à cet échange. Nous n'avons donc pas besoin de ports supplémentaires.** **- Les capteurs 1 et 2 envoient des signaux analogiques que l'on doit transmettre en numérique. On transforme donc notre signal analogique en signal numérique.**
I/O IU	**1 - Enable**	**1 - Enable**	**1 - Enable**	**1 - Enable**	**Permet d'envoyer les données sous forme de trames API.**
I/O IT Sample before TX	**X**	**1**	**1**	**X**	**On définit ici le nombre d'échantillons à récupérer avant d'envoyer les données :** **- On récupère 1 valeur pour le capteur PIR (ce capteur est à 1 uniquement lorsque la porte/fenêtre est ouverte ; Il faut donc prendre beaucoup d'acquisitions)** **→ 1 acquisition chaque seconde** **- On récupère 1 valeur pour le capteur (ce capteur est à 1 pendant 10 sec après que quelqu'un l'ai actionné; Il faut donc prendre peu d'acquisitions)** **→ 1 acquisition toute les 3 secondes** **- La centrale et l'interface utilisateur envoie des TRAMES. Il n'y a donc pas d'intérêt à faire des acquisitions.**
I/OIR Sample	**X**	**3000 ms**	**1000 ms**	**X**	

Ci-dessous se trouve une vue de XCTU : le logiciel qui nous a permis de configurer les différents modules XBee :



3) Communication avec le STM32

Pour la lecture des trames des XBee, nous avons d'abord activé et configuré l'USART1 qui est le composant permettant de faire la lisaion entre le STM32 et le port série qui reçoit la trame du XBee.


```
void ConfigurerUSART1(){ // Activation et configuration de l'USART (pour l'échange avec les XBee)
  RCC->APB2ENR |= 0x00000001;   // Activation de l'horloge pour les fonctions d'entrées-sorties
  AFIO->MAPR &= ~(1 << 2); // Remise à zéro du USART 1 remap
  RCC->APB2ENR |= 0x00000004;   // Activation de l'horloge du port GPIOA
  GPIOA->CRH = ((GPIOA->CRH & 0xFFFFF00F) | (0x000004B0)); /*USART 1 Tx se situe sur la broche PA9
                                                qui doit être configurée en
                                                « Alternate Output Push-Pull »,
                                                USART 1 Rx se situe sur la broche PA10,
                                                qui doit être configurée en « Input Floating »*/
  RCC->APB2ENR |= 0x00004000;   // Activation de l'horloge de l'USART 1
  USART1->BRR = 8000000 / 115200;   /* Avoir un Baud Rate de 115200Hz
                                    (USART1->BRR = 0b1000100* : USARTDIV 1 = 4 + (5 / 16) = 4,3125)*/
  USART1->CR1 = 0;
  USART1->CR2 = 0;
  USART1->CR3 = 0;
  USART1->CR1 |= 0x0000000C;   // Activation de l'émetteur et du récepteur
  USART1->CR1 |= 0x00002000;   // Activation de l'USART 1
  USART1->CR1 |= (1 << 5);    // Autorisation des interruptions
}

void initITUSART1(){   //Autorisation de l'interruption provoquée par l'USART 1
  NVIC->ISER[1] |= (1 << 5); /*L'USART 1 se trouve à la position 37 dans la table
                              des interruptions (donc position 5 de l'ISER[1])*/
}
```

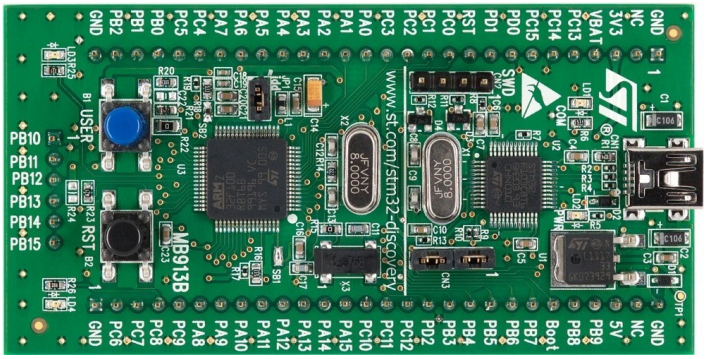
Ensuite, nous avons écrit trois fonctions permettant pour la première d'obtenir les octets reçus par le XBee, pour la deuxième d'envoyer des octets à l'USART1, et pour la dernière d'envoyer une chaîne de caractères.

```
int ObtenirOctetRecu()    /*Fonction qui attend que l'USART 1 ait reçu
                        une donnée (2 octets) pour renvoyer cette donnée*/

   {
      while (!(USART1->SR & USART_SR_RXNE));
                        /*On attend que l'USART 1 signale la réception d'un octet*/
      return ((int)(USART1->DR & 0x1FF));
                                  /*L'octet que l'USART 1 a reçu
                                  se trouve dans le registre « USART1->DR »*/
   }
```

```
void EnvoyerOctet(unsigned char donnee) /*Permet d'envoyer un octet par l'USART 1*/
   {
      while (!(USART1->SR & USART_SR_TXE));
     /*On attend que l'USART 1 soit prêt à envoyer une donnée*/
      USART1->DR = donnee;    /*L'envoi de l'octet par l'USART 1 démarre quand on
                                écrit l'octet dans le registre « USART1->DR »*/
   }


void EnvoyerPlusieursOctets(unsigned char *paquet)
                          /*Permet d'envoyer une chaîne de caractères par l'USART 1*/
   {
        while(*paquet != '\0')
          /*Tant que l'on n'est pas arrivé à la fin de la chaîne de caractères*/
         {
            EnvoyerOctet(*paquet);
           /*On envoie le caractère courant de la chaîne de caractères*/
            paquet++;
         }
   }
```

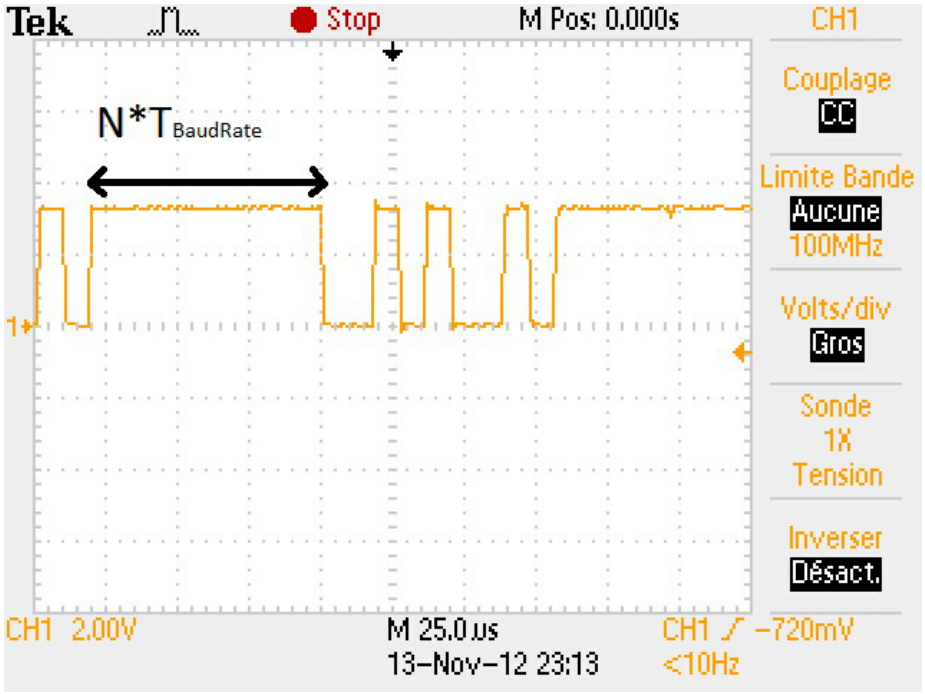
L'interruption suivante permet de stocker les trames dans le Buffer lorsqu'elles sont détectées.

```
void USART1_IRQHandler(void) __irq{    /*Fonction d'interruption de l'USART 1
                                       qui permet d'enregistrer dans le buffer
                                       les trames envoyées par les capteurs et
                                       par l'interface utilisateur*/
  if((USART1->SR & USART_SR_RXNE)){
      if(trame_en_cours==11) trame_en_cours=0;
        else trame_en_cours++;
    // Les trames recues par la centrale doivent commencer par 7E et ont une taille de 14
      if(trame_en_cours<10 && ObtenirOctetRecu()==126/*7E*/){
        int i;
        buffer[trame_en_cours][0]=126;/*"7E"*/
        for(i=1;i<14;i++)
          buffer[trame_en_cours][i]=ObtenirOctetRecu(); // récupération de la trame
      }
  }
  USART1->SR = 0;    /*Réarmement de l'interruption pour les trames à venir*/
}
```



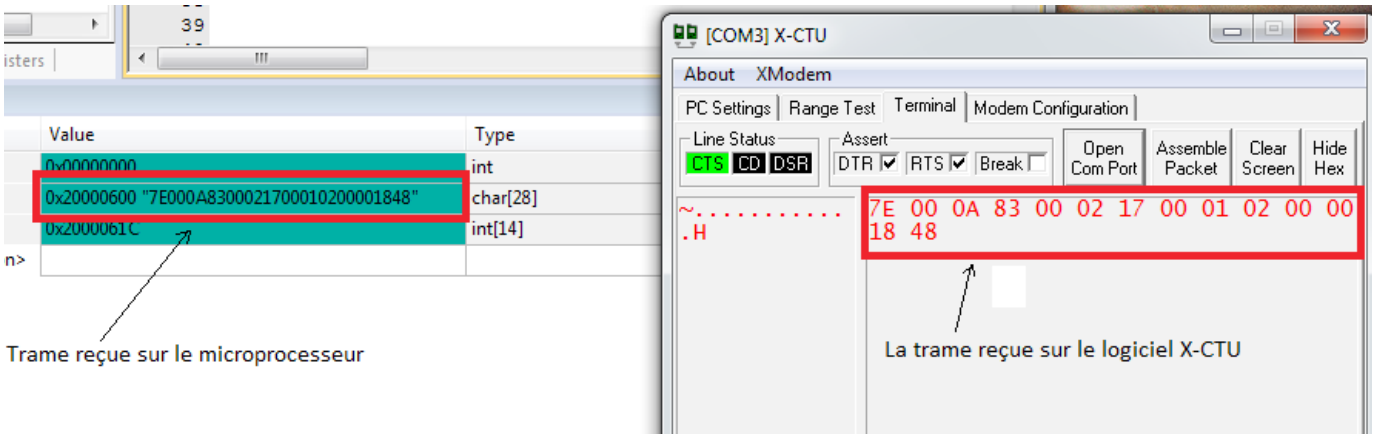
Test

Voici ce que l'on a obtenu à l'oscilloscope une fois que l'on s'est branché sur la pin DOut du module XBee de la centrale : on observe la trame reçue, et on peut retrouver le baud rate (fréquence d'émission du XBee) qui est de 115200.



T1bit = 0.35x25.10^-6sec, d'où f = 1/T1bit = 115 200 Hz
Ici, on en déduit que N = 9.

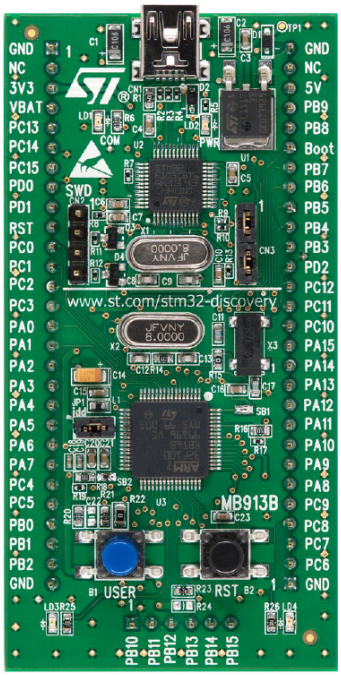
Et ci après on observe la trame complète reçue et lue sur la carte STM32 Discovery :



IV) Centrale

1) Présentation de la STM32

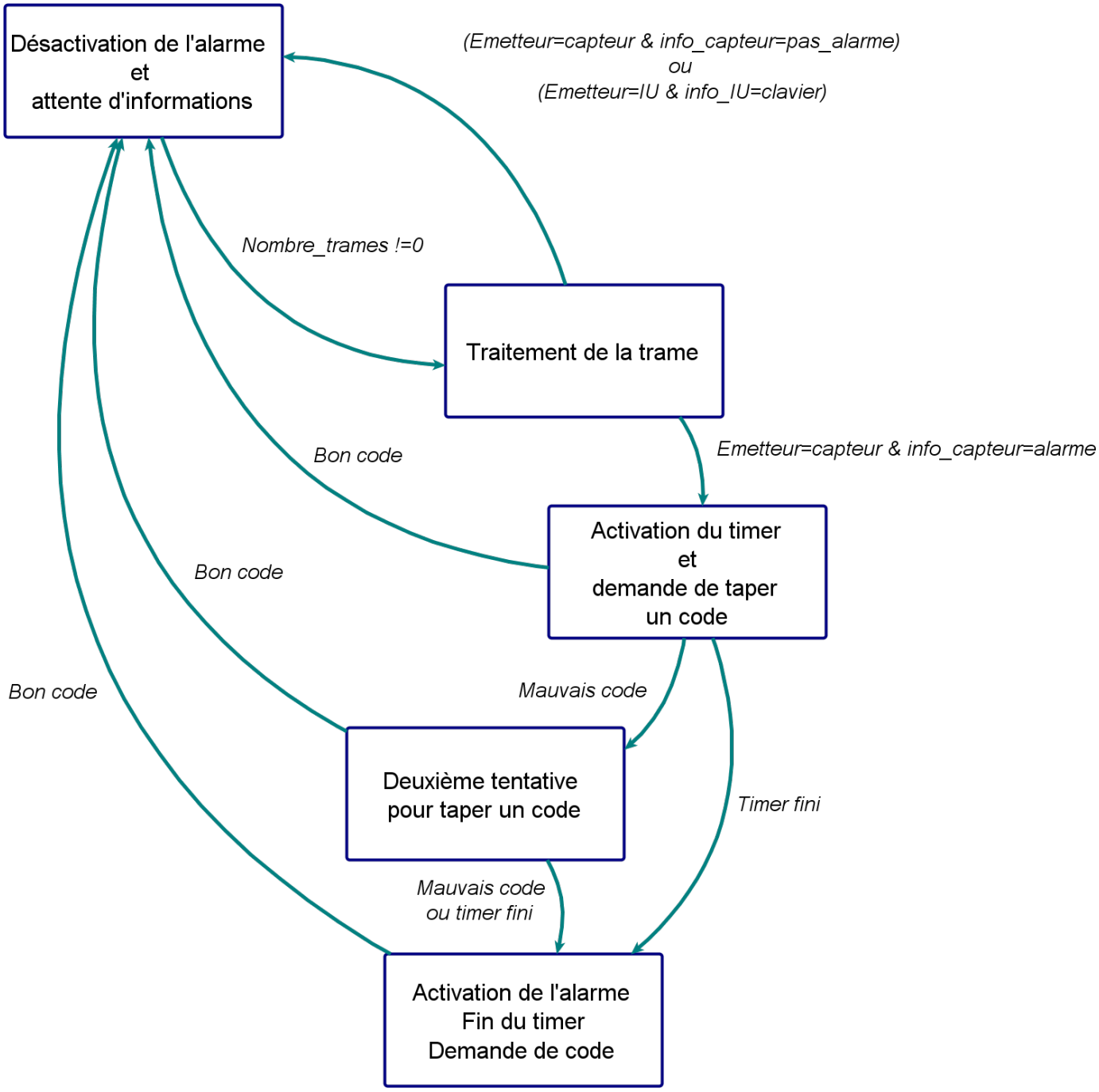
La carte STM32 Discovery est l'organe de contrôle et commande de notre système. En effet, elle va recevoir les trames des différents modules XBee et va ensuite les traiter : afin de connaître la position du capteur et les informations fournies par ce capteur. Elle va également échanger des données avec l'interface utilisateur, la sirène d'alarme, ...



Nous avons choisi cette carte, car elle est peu onéreuse, suffisamment puissante pour notre projet et car nous l'avons déjà utilisée sous une forme équivalente en TP de Microprocesseurs.
Cependant, nous aurions seulement pu utiliser un microcontrôleur et non la carte de développement complète.

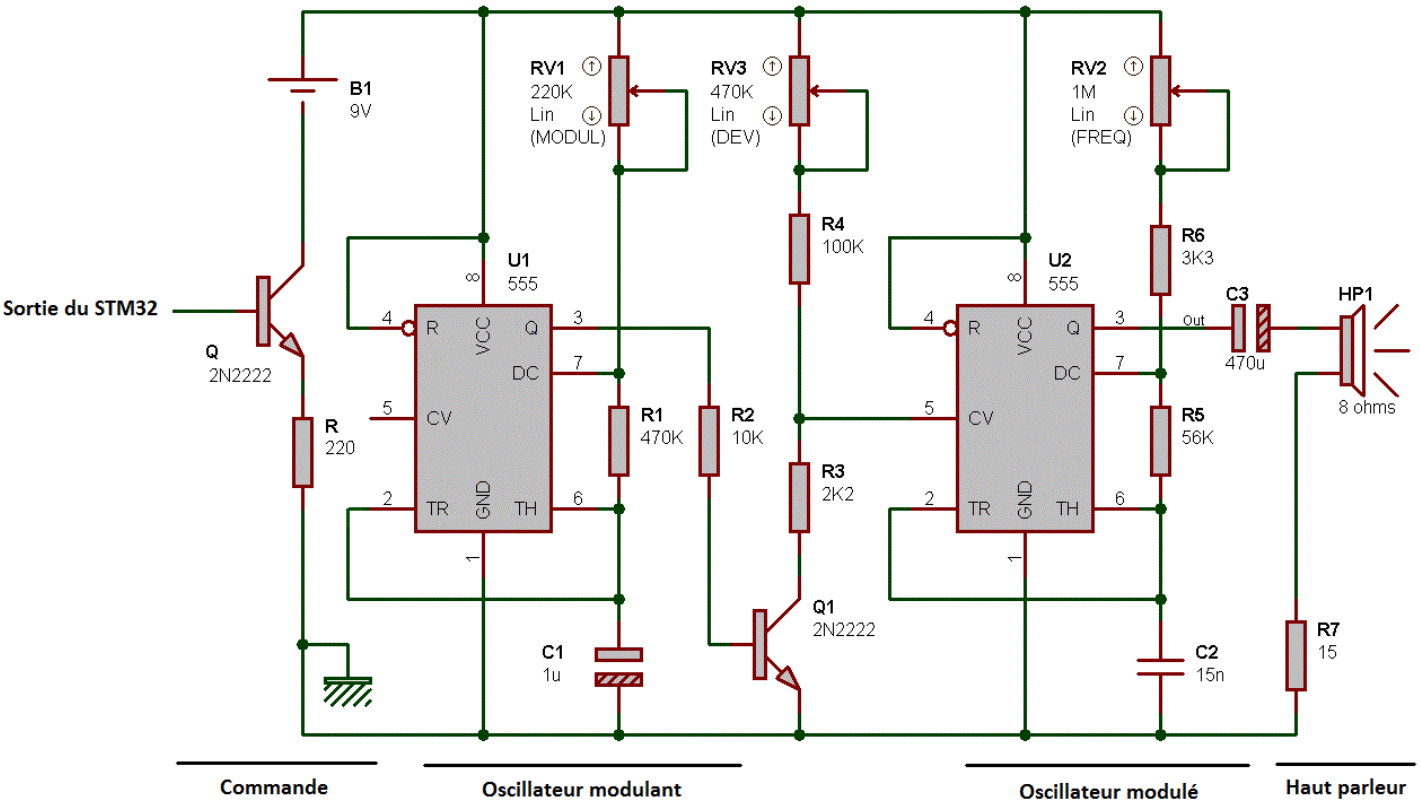
2) Codage du STM32

La centrale est la partie Contrôle et Commande de notre système.
Elle traite ainsi les trames reçues aussi bien envoyés par les capteurs que ceux transmis par l'interface utilisateur. Et en fonction de ces données, elle active ou désactive la sirène d'alarme et commande l'interface utilisateur.
Le fonctionnement global de cette centrale est expliqué par le diagramme ci-dessous.

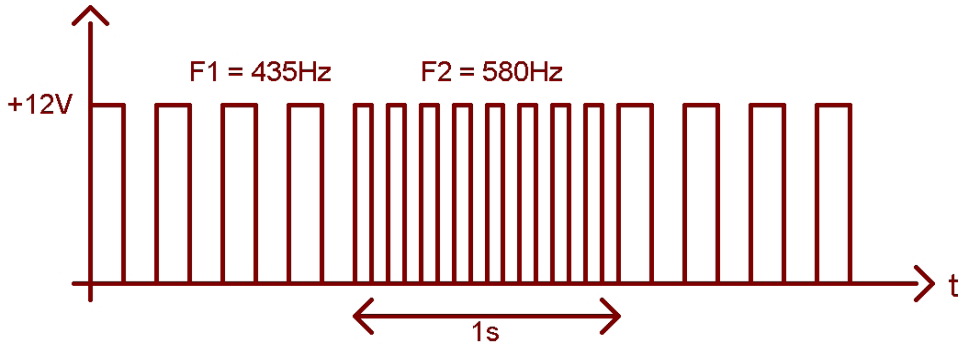


3) Sirène d'alarme

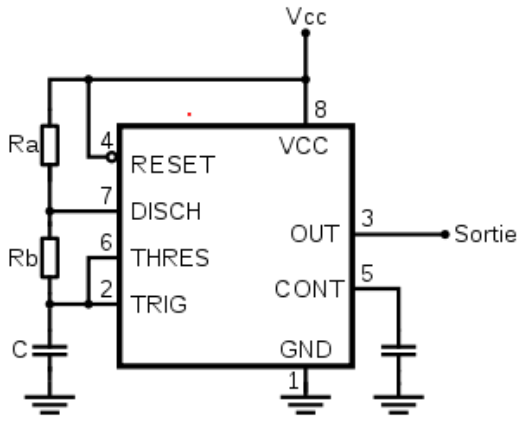
La sirène d'alarme permet d'avertir en cas d'intrusion.
Elle est présente sur la plaque de la centrale et est reliée au STM32 par le biais d'un transistor qui fait «interrupteur».
Ci-dessous se trouve le schéma de notre sirène d'alarme.



La sirène que nous avons conçue est de type bitonale. Cela signifie que l'on entend deux sons de fréquence fixe alternés (cf image ci-dessous). Il existe d'autres types d'alarmes comme les alarmes modulées qui ont une variation continue de la fréquence.



Nous utilisons ici deux NE555 en mode astable. Cela nous permet, avec une tension continue en entrée d'avoir un signal carré avec une fréquence et un rapport cyclique réglables en sortie.
Voici le branchement et les caractéristiques en sortie de ce mode de fonctionnement :



$$\alpha = 1 - \frac{R_b}{(R_a + 2R_b)}$$

$$f = \frac{1.44}{(R_a + 2R_b)C}$$

Suivant la sortie du premier NE555, le transistor (Q1) sera bloqué ou non.
S'il est bloqué, le deuxième NE555 reçoit une tension proche de la tension d'alimentation (en traversant RV3 et R4).
Si le transistor est passant, nous connectons R3 à la masse, ce qui a pour conséquence d'avoir un pont diviseur de tension : Ventrée 2ème NE555 = Vcc x R3/(RV3+R4+R3).
Ainsi, suivant l'état du transistor Q1 (et donc de la sortie du premier NE555), nous obtenons en entrée du 2ème NE555 deux tensions bien distinctes.
Cela implique qu'en sortie du 2nd NE555, nous obtenons deux tons bien distincts qui se répètent toutes les secondes.
De plus, sur les points clefs de notre circuit, nous avons utilisé des résistances variables afin de pouvoir régler la modulation et la fréquence des deux tons.

La sirène a une puissance en sortie de l'ordre de 3W.
Afin d'obtenir une sirène plus puissante, il faudrait ajouter à ce circuit un simple étage d'amplification. Nous ne l'avons pas fait pour éviter de réveiller tout le quartier.

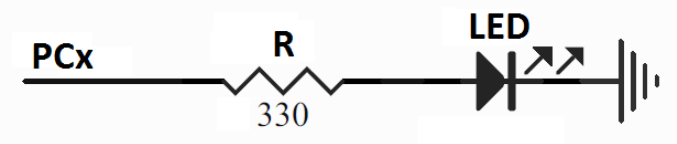
Les 2 fonctions suivantes permettent d'activer et de désactiver l'alarme en activant ou non PC8.

```
void ActiverSirene(){ // Activation de la sirène
  GPIOC->ODR |= (1 << 8);

}
void DesactiverSirene(){ // Extinction de la sirène
  GPIOC->ODR &= ~(1 << 8);
}
```

4) Débogueur

Pour débugger notre centrale et notre Interface Utilisateur sans Keil uVision, nous avons décidé d'utiliser 3 LEDs qui représentent l'état de l'alarme (alarme désactivée / personne détectée / alarme activée).
Les LEDs utilisées ont une consommation de 10mA et de 0.6V. Etant donné que l'on a branché ces LEDs sur les broches du STM32, la tension en sortie est de 3.3V.
Nous avons donc branché des résistances en série avec les LEDs d'une valeur de 330 ohms (3.3-0.6/0.01=270ohms --> 330ohms).

Voici le branchement :

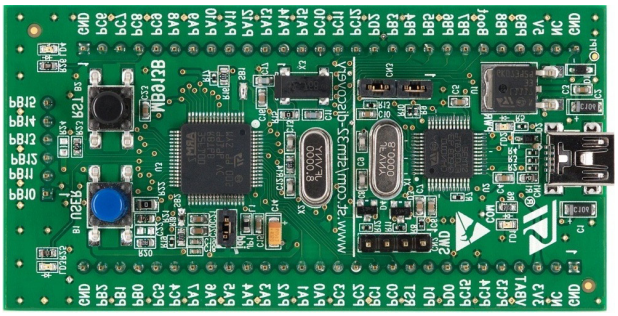


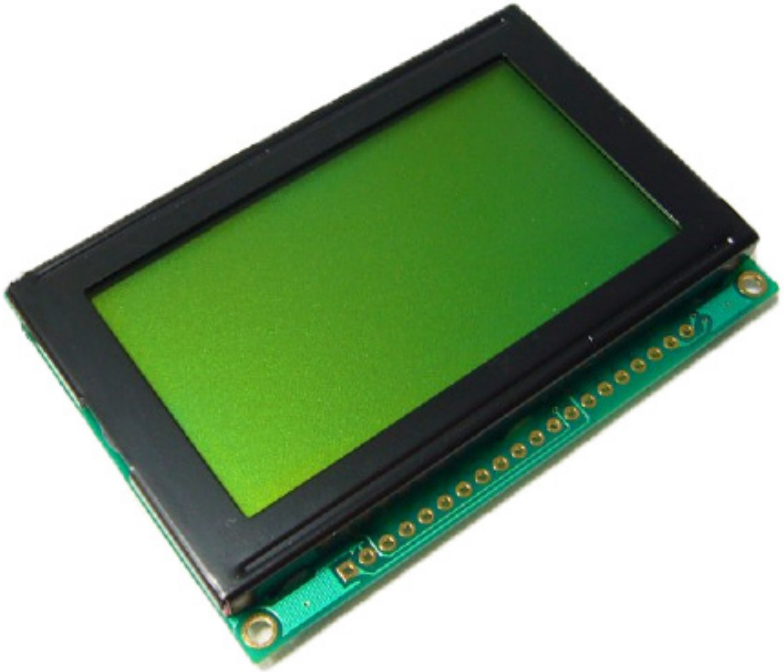
Voici également les différentes fonctions du STM32 pour activer-désactiver certaines LEDs :

```
void Activer_LED_Verte(){ // Activation de la led verte + désactivation des 2 autres leds
  GPIOC->ODR |= (1 << 11);
  GPIOC->ODR &= ~(1 << 12);
  GPIOC->ODR &= ~(1 << 13);

}
void Activer_LED_Orange(){ // Activation de la led orange + désactivation des 2 autres leds
  GPIOC->ODR |= (1 << 12);
  GPIOC->ODR &= ~(1 << 11);
  GPIOC->ODR &= ~(1 << 13);
}
void Activer_LED_Rouge(){ // Activation de la led rouge + désactivation des 2 autres leds
  GPIOC->ODR |= (1 << 13);
  GPIOC->ODR &= ~(1 << 12);
  GPIOC->ODR &= ~(1 << 11);
}

void Desactiver_les_LEDs(){ // Désactiver les 3 LEDs
  GPIOC->ODR &= ~(1 << 13);
  GPIOC->ODR &= ~(1 << 12);
  GPIOC->ODR &= ~(1 << 11);
};

void Activer_les_LEDs(void){ // Activer les 3 LEDs
  GPIOC->ODR |= (1 << 13);
  GPIOC->ODR |= (1 << 12);
  GPIOC->ODR |= (1 << 11);
};
```

V) Interface utilisateur

Ce système est composé d'un microcontrôleur STM32 (discovery),
d'un afficheur LCD (KS0108) et d'un clavier numérique.
Le tout communiquant avec la centrale par le biais d'un XBee.



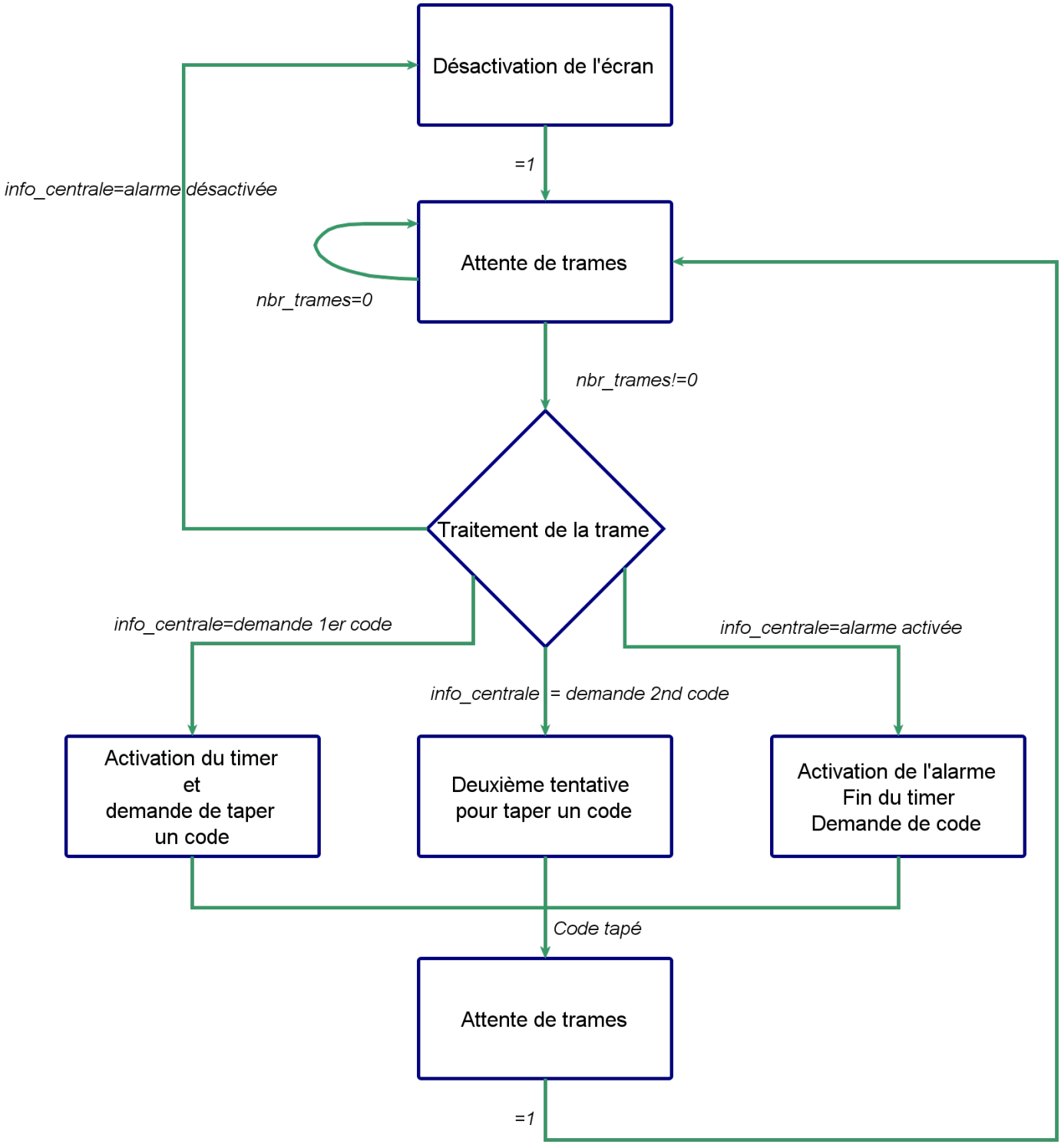


1) Codage du STM32

Le but de l'interface utilisateur est :

- de recevoir et d'interpréter les informations envoyées par la centrale
- d'émettre un message à afficher sur le mini écran LCD
- de recevoir et interpréter les données (code secret) envoyées par le clavier numérique
- de transmettre le code secret à la centrale
-

Le fonctionnement global de cette interface utilisateur est expliqué par le diagramme ci-dessous.



Nous utilisons un compteur du STM32 (Timer2) que l'on lance dès que l'on détecte une personne.
Nous activons alors une interruption toute les 2 secondes (grâce au compteur).
Tant que l'individu n'a pas tapé le bon code au clavier, le compteur continu de faire son travail.
Dans la routine d'interruption, on compte le nombre d'interruptions (et donc le temps écoulé depuis le début du comptage).

```
void ActiverTimer2(){ /*Activer le Compteur Timer2 (et donc, implicitement,
                      on active les interruptions du timer)*/
  TIM2->CR1 |= 0x1;
}

void DesactiverTimer2(){ /*Désactiver le Compteur Timer2 (et donc, implicitement,
                         on désactive les interruptions du timer)*/
  TIM2->CR1 &= ~(0x1);
}
```

```
void initTimer2(){ // Nous utilisons un compteur de 2 SEC pour creer des interruptions espacées de 2 SEC
  RCC->APB1ENR |= 0x1; //Activation du timer 2
  TIM2->CR1 = 0x00000094; // Configuration de base du Timer2
  TIM2->PSC = 0x01F0; /* Permet de baisser la fréquence de comptage
                       (bien pratique si l'on veut un compteur très lent)
                       Fréquence de comptage=fclk/(1+TIM2->PSC)*/
  TIM2->ARR = 0xFFFF; //On met le temps de comptage le plus long possible
  TIM2->DIER = TIM2->DIER | (1 << 0);   //Permettre la mise à jour sur une interruption
}

void initITTimer2(){ //Activation de l'interruption du TIMER2
  NVIC->ISER[0] |= (1 << 28);   /*L'interruption qui correspond au timer 2 est
                                située en position 28 dans la table des vecteurs*/
}
```

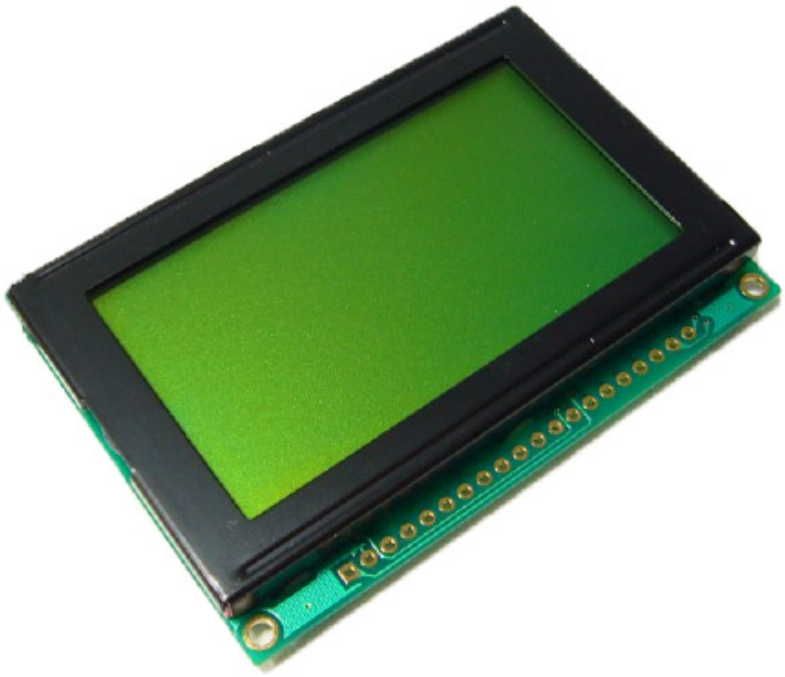
Quand 20 secondes se sont écoulées, on active la sirène. En effet, on laisse à la personne 20 secondes pour taper le bon code au clavier (2 essais).

```
void TIM2_IRQHandler(void) __irq{ /*Interruptions espacées de 2 sec. L'utilisateur a 20sec(=DELAIS)
                                  (avec une marge de 2sec) à partir du déclenchement de l'alarme
                                  pour taper le bon code (sinon on déclenche la sirène).*/
  if(alarme==1){
    compteur_timer++;
    if(compteur_timer>=(DELAIS/2)+1) compteur_utilise_fini=1; /* 20sec de comptage
                                                              --> On va activer la sirène dans le main*/
  }

  TIM2->SR &= (~1); //On réarme l'interruption du timer 2
}
```

2) Ecran LCD

L'écran LCD que nous utilisons a pour unique but d'informer l'utilisateur de l'état de l'alarme et de lui demander (si nécessaire) de taper un code au clavier.

Notre écran est un KS0108 de taille 128x64.
Cet écran consomme environ 1 Watt.
Ceci est un inconvenient majeur pour un système embarqué comme le notre.



Pour réduire la consommation de ce composant, nous avons décidé d'activer l'afficheur uniquement lorsque cela est nécessaire.

Le reste du temps, nous ne l'alimentons pas. Ainsi, l'écran consomme, en moyenne, très peu (en supposant que l'afficheur est actif 5 minutes par jour, ce qui est généralement le cas, nous ne consommons que 3.5 mW).

Nous avons choisi de ne pas vous présenter dans ce rapport toutes les fonctions de l'écran LCD. Nous nous limiterons donc aux fonctions essentielles.

Ainsi nous avons créé une fonction pour activer ou non l'afficheur, une fonction pour afficher un pixel à une position donnée, une fonction pour supprimer ce qu'il y a sur l'écran et enfin, une fonction pour afficher une image à l'écran.

```
void ActiverAfficheur(){ // Activation de l'écran LCD
  GPIOC->ODR |= (1 << 4);
}

void DesactiverAfficheur(){ // Extinction de l'écran LCD
  GPIOC->ODR &= ~(1 << 4);
}
```

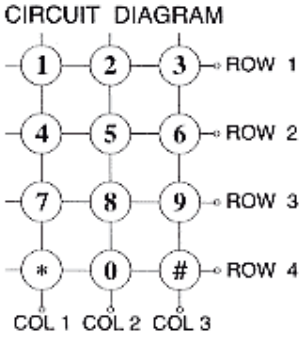
```
void GLCD_WriteData(unsigned char dataToWrite){ // Ecrire les données à la position actuelle
  while(GLCD_ReadStatus(screen_x / 64)&DISPLAY_STATUS_BUSY); // attente que l'écran soit pret
  //Définir le pin de sortie du STM pour pouvoir evoyer des données à l'afficheur :
  GPIO_StructInit(&GPIO_InitStructure);
  GPIO_InitStructure.GPIO_Pin = (0xFF << KS0108_D0);
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(KS0108_PORT, &GPIO_InitStructure);

  // L'écriture :
  GPIO_ResetBits(KS0108_PORT, KS0108_RW); // Relancer l'écriture
  GLCD_Delay();
  GPIO_SetBits(KS0108_PORT, KS0108_RS); // De même
  GLCD_Delay();
  GPIO_SetBits(KS0108_PORT, (dataToWrite << KS0108_D0)); // Envoyer les données
  dataToWrite ^= 0xFF;
  GPIO_ResetBits(KS0108_PORT, (dataToWrite << KS0108_D0));
  GLCD_Delay();
  GLCD_EnableController(screen_x / 64); // Activer le controleur
  GLCD_Delay();
  GPIO_SetBits(KS0108_PORT, KS0108_EN); // Activer les coordonnées
  GLCD_Delay();
  GPIO_ResetBits(KS0108_PORT, KS0108_EN);
  GLCD_Delay();
  GLCD_DisableController(screen_x / 64);
  screen_x++; // Aller à la position suivante de l'écran
}
```

```
void GLCD_ClearScreen(void) //Supprimer ce qu'il y a sur l'écran LCD
{
unsigned char i, j;
for(j = 0; j < KS0108_SCREEN_HEIGHT/8; j++) // On balaye l'écran complet (x et y)
  {
  GLCD_GoTo(0,j); // Aller à la position indiquée
  for(i = 0; i < KS0108_SCREEN_WIDTH; i++)
    GLCD_WriteData(0x00); // On écrit "rien" pour effacer
  }
}
```

```
void GLCD_Bitmap(char * bmp, unsigned char x, unsigned char y, unsigned char dx, unsigned char dy){
  /* Afficher une image BMP de la taille de l'écran LCD */
  unsigned char i, j;
  for(j = 0; j < dy / 8; j++){ // On balaye l'écran
    GLCD_GoTo(x,y + j);
    for(i = 0; i < dx; i++)
      GLCD_WriteData(GLCD_ReadByteFromROMMemory(bmp++)); /*On écrit sur le  pixel de l'écran
                                                          la valeur du niveau de gris de l'image*/
  }
}
```

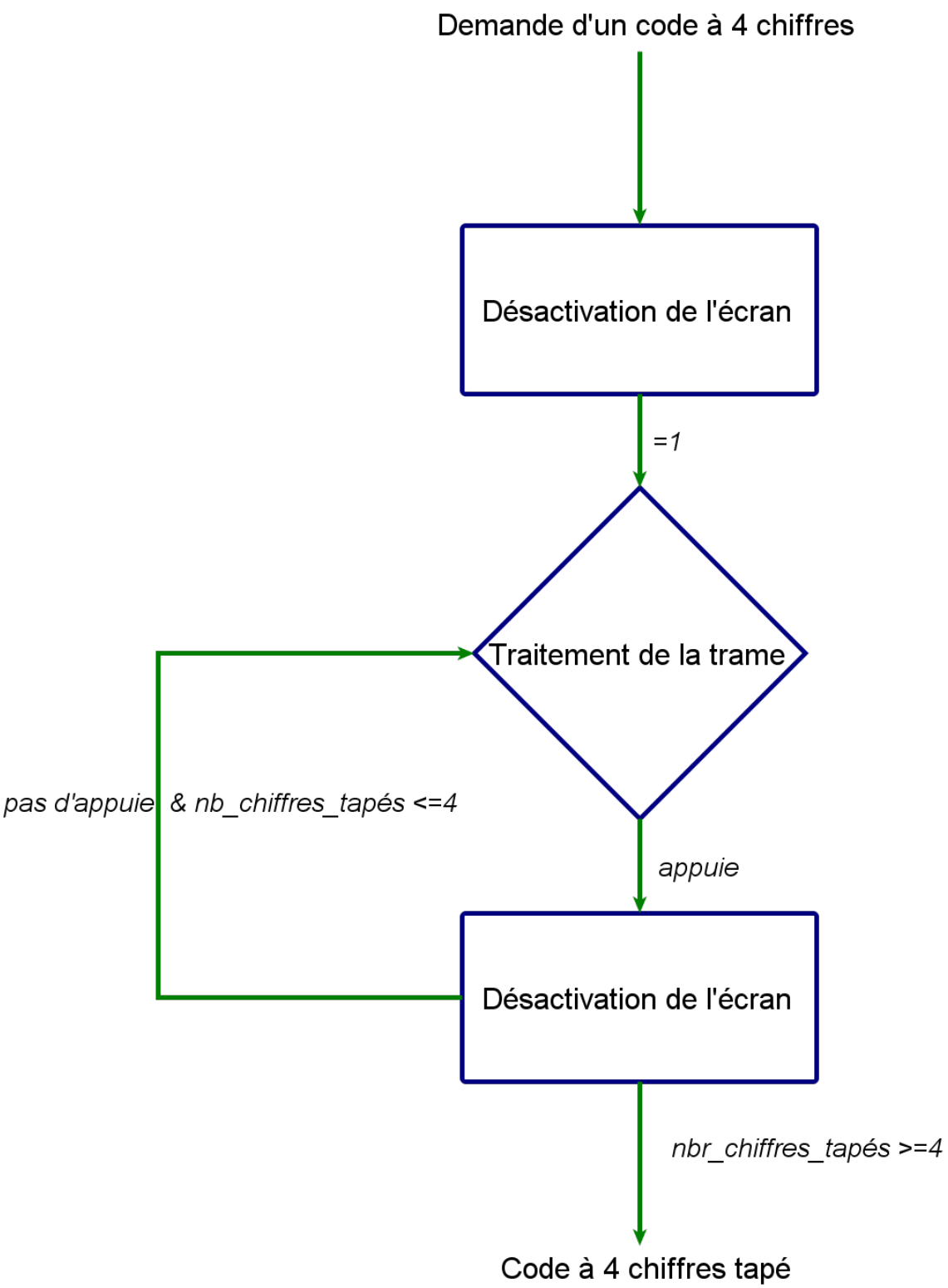
3) Clavier 12 touches

Pour que les personnes puissent taper un code, nous avons choisi un clavier de 12 touches afin de pouvoir taper un code de 4 chiffres.
Ce type de clavier n'a pas un fonctionnement très complexe.
Il y a une patte par colonne et une patte par ligne. Ainsi, ce clavier comporte 7 pattes.



Pour savoir si une personne appuie ou non sur une touche, il faut se positionner sur la colonne en question (on envoie un 1 logique sur la colonne choisi et un 0 logique sur les deux autres).
On regarde ensuite si nous retrouvons un 1 logique sur l'une des pattes colonne. Si oui, la patte comportant un 1 logique, correspond à un appui.
Un point important est qu'un appui par un utilisateur classique dure environ 150ms et le temps entre deux chiffres est de l'ordre de 300ms.
Ainsi, nous avons réalisé la machine à état suivante afin d'éviter de récuperer plusieurs fois de suite la même touche.

```
void attente_appuie(int appuie2){
  //Pour savoir si une personne appuie ou non sur une des touches du clavier
  while(appuie==appuie2){
    GPIOC->BSRR = (1 << 10);  // On met à 1 une des trois entrées.
    GPIOC->BSRR = (1 << (11+16)); // les deux autres sont à 0
    GPIOC->BSRR =(1 << (12+16));
    attente(2); // temps de reponse du clavier (très faible)
    if(!((GPIOC->IDR & 0x8)+(GPIOC->IDR & 0x4)+(GPIOC->IDR & 0x2)+(GPIOC->IDR & 0x1)))
    // Si aucune des touches n'est appuyé
       poursavoirsiappuie++;

    GPIOC->BSRR = (1 << (10+16)); //De même
    GPIOC->BSRR = (1 << 11);
    GPIOC->BSRR =(1 << (12+16));
    attente(2);
    if(!((GPIOC->IDR & 0x8)+(GPIOC->IDR & 0x4)+(GPIOC->IDR & 0x2)+(GPIOC->IDR & 0x1)))
       poursavoirsiappuie++;

    GPIOC->BSRR = (1 << (10+16)); //De même
    GPIOC->BSRR = (1 << (11+16));
    GPIOC->BSRR =(1 << 12);
    attente(2);
    if(!((GPIOC->IDR & 0x8)+(GPIOC->IDR & 0x4)+(GPIOC->IDR & 0x2)+(GPIOC->IDR & 0x1)))
       poursavoirsiappuie++;

    if(poursavoirsiappuie==3)appuie=0;
    // Si la personne a appuyé sur un boutton, appuie=1
    else appuie=1;

    poursavoirsiappuie=0;
  }
}
```


Demande d'un code à 4 chiffres
Désactivation de l'écran
=1
Traitement de la trame
pas d'appuie & nb_chiffres_tapés <=4
appuie
Désactivation de l'écran
nbr_chiffres_tapés >=4
Code à 4 chiffres tapé

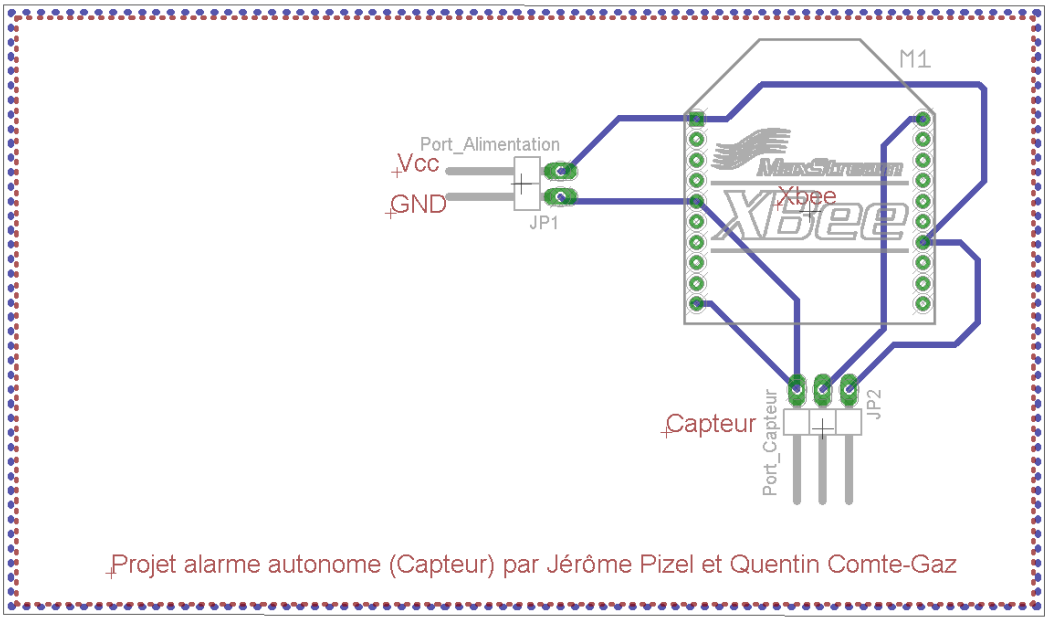
```
void recuperation_chiffre_clavier(void){
/*Récuperer la touche
(rapide, il faut donc la combiner avec attente_appuie)*/
  //Pour la premère colonne :
  GPIOC->BSRR = (1 << 10);  // Mettre à 1 PC10 :
  GPIOC->BSRR = (1 << (11+16)); // Mettre à 0 PC11
  GPIOC->BSRR =(1 << (12+16));  // Mettre à 0 PC12
  attente(2); // Temps de reponse du clavier (très faible)
   if(GPIOC->IDR & 0x1) //si PC0 est à 1
     clavier[i]=1; //Alors l'utilisateur appuie sur la touche "1"
   else if(GPIOC->IDR & 0x2)  //si PC1 est à 1
       clavier[i]=4;
   else if(GPIOC->IDR & 0x4)  //si PC2 est à 1
     clavier[i]=7;
   else if(GPIOC->IDR & 0x8)  //si PC3 est à 1
      clavier[i]=10;

  //De même pour la 2ème colonne :
  GPIOC->BSRR = (1 << (10+16));
  GPIOC->BSRR = (1 << 11);
  GPIOC->BSRR =(1 << (12+16));
  attente(2);
  if(GPIOC->IDR & 0x1)
    clavier[i]=2;
  else if(GPIOC->IDR & 0x2)
    clavier[i]=5;
  else if(GPIOC->IDR & 0x4)
    clavier[i]=8;
  else if(GPIOC->IDR & 0x8)
    clavier[i]=0;

  //De même pour la 3ème colonne :
  GPIOC->BSRR = (1 << (10+16));
  GPIOC->BSRR = (1 << (11+16));
  GPIOC->BSRR =(1 << 12);
  attente(2);
  if(GPIOC->IDR & 0x1)
    clavier[i]=3;
  else if(GPIOC->IDR & 0x2)
    clavier[i]=6;
  else if(GPIOC->IDR & 0x4)
    clavier[i]=9;
  else if(GPIOC->IDR & 0x8)
    clavier[i]=11;
}
```

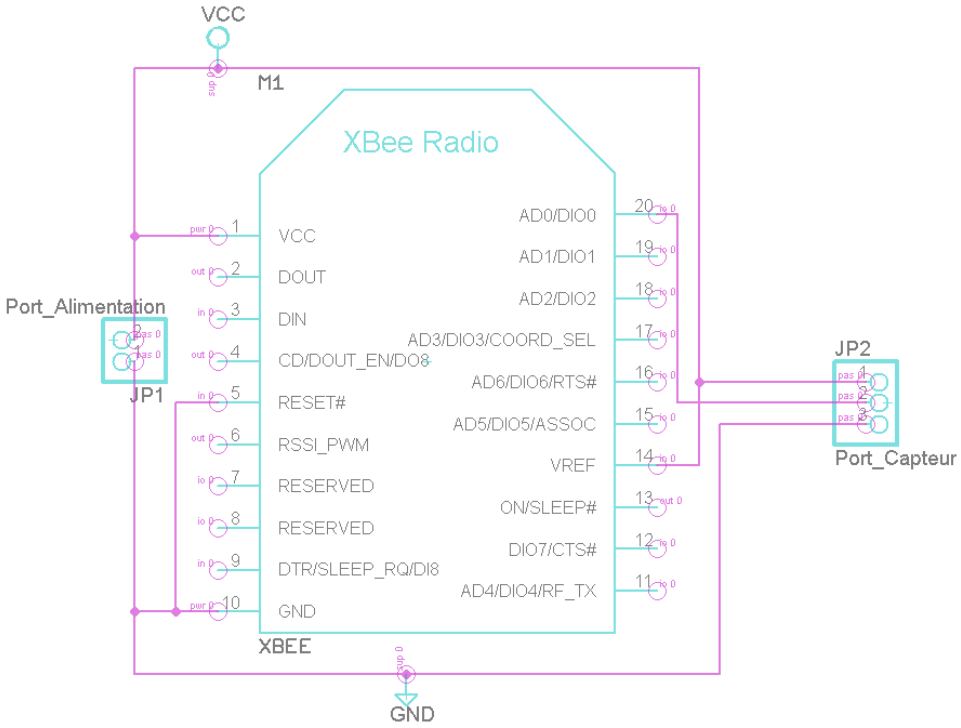
VI) Conception de circuits imprimés

L'étape finale de notre projet consistait à concevoir des circuits imprimés pour nos différents blocs (capteurs, centrale et interface utilisateur).
Nous avons utilisé le logiciel Eagle afin de concevoir les différents tipons permettant la conception de nos circuits imprimés.
Nous avons d'abord fait le schéma du circuit puis nous avons fait le routage de la carte.

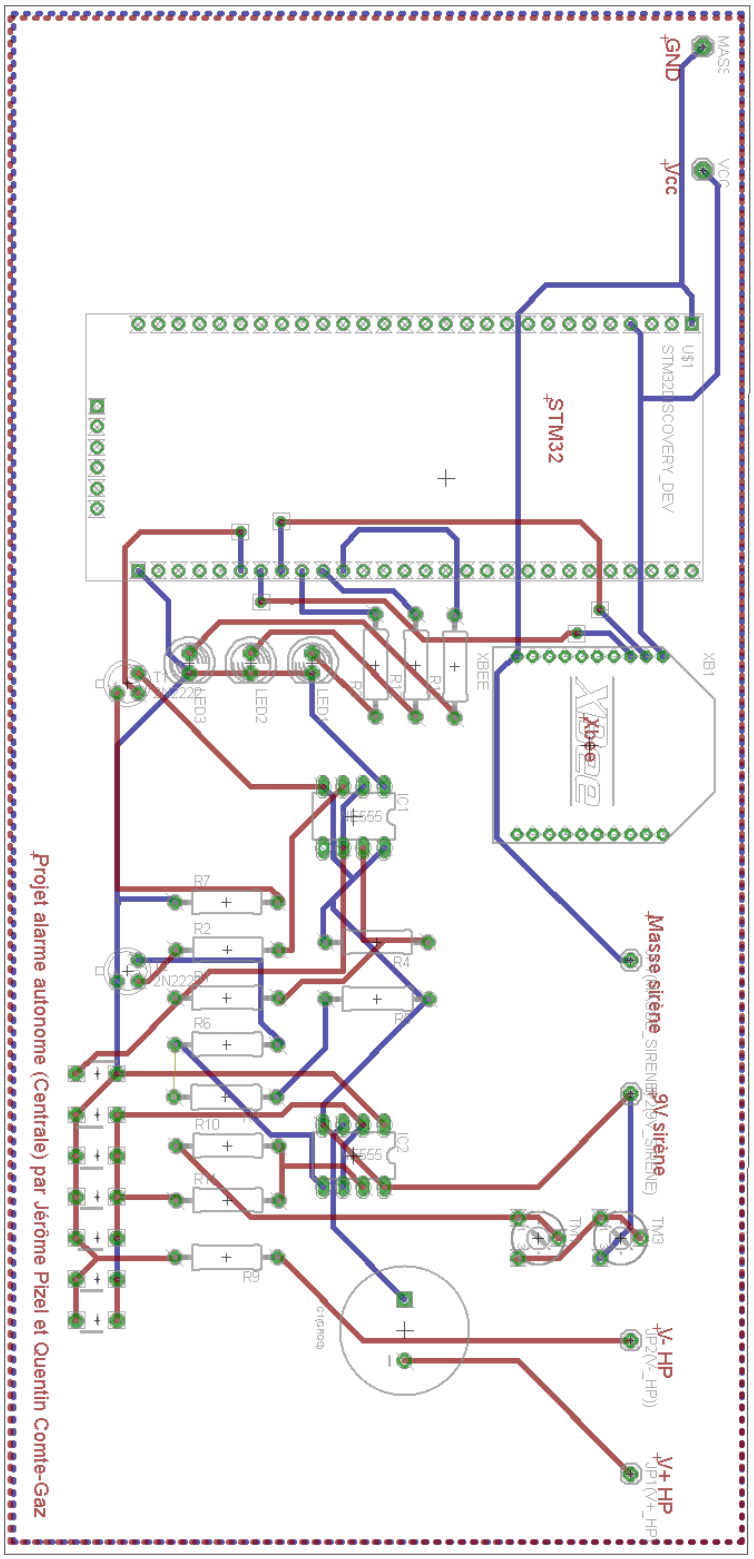
Nous avons rencontré de nombreux problèmes dans la réalisation de ces plaques :
Après plusieurs vérifications de nos tipons, nous avons fait faire les circuits imprimés correspondants.
Dès le perçage, nous nous sommes rendu compte de certains problèmes sur nos cartes : nous avions fait des vias de 0.6mm de diamètres au lieu de 0.8mm. Cela a eu pour conséquence de rendre plus difficile le perçage et le soudage.
Ensuite, nous avons également fait des pastilles pour certains composants à 0.6mm au lieu de 0.8mm. Cela nous a ralenti lors de la mise en place de certaines résistances.
Un autre problème est que nous avons dû souder des deux côtés de l'écran LCD. En effet, il était difficile, même en faisant de nombreux vias, de relier les câbles à l'écran sur un seul côté de la plaque.
Tous ces problèmes auraient pu être évités si nous avions prévu plus de temps pour cette étape de cette étape de conception sous Eagle. Nous avions prévu de travailler au moins 2 mois sur la réalisation de ces plaques mais avons été ralenti par le codage de l'écran LCD.
Cela a eu pour conséquence de faire de nombreuses petites erreurs qui, cumulées, ont fait que le circuit imprimé de l'interface utilisateur ne fonctionnait pas. La carte de la centrale fonctionne en partie (la sirène fonctionne mais la liaison entre le STM32 et l'alarme ne semble pas fonctionner).
Heureusement, la carte du capteur fonctionne parfaitement.

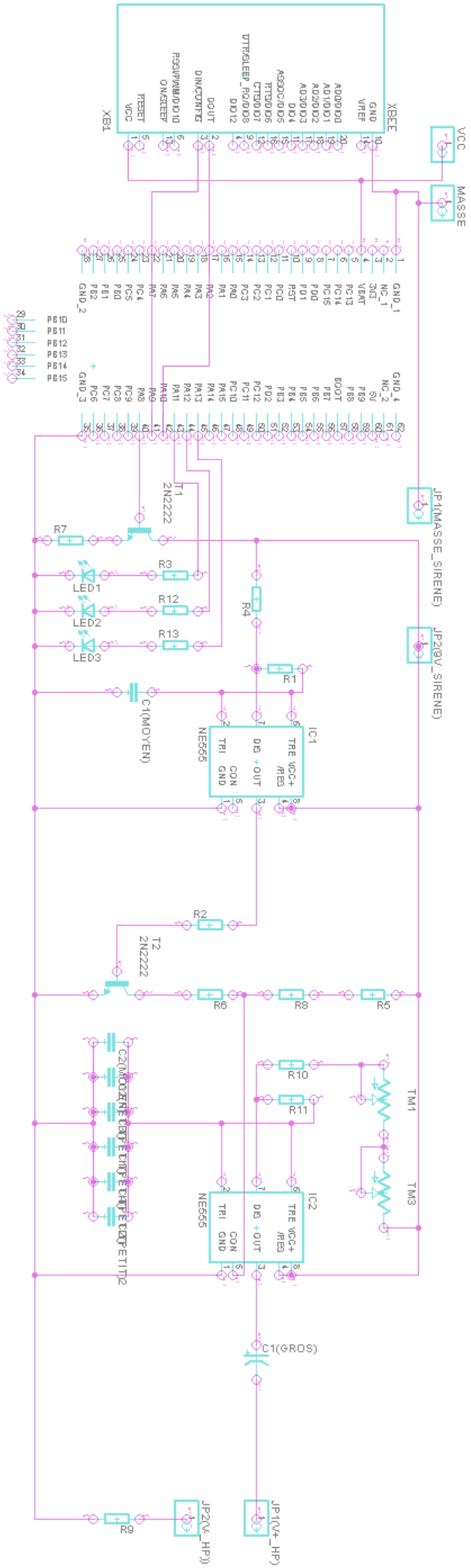
1) Capteur


M1
Port_Alimentation
Vcc
GND
JP1
Xbee
Capteur
Port_Capteur
JP2
Projet alarme autonome (Capteur) par Jérôme Pizel et Quentin Comte-Gaz

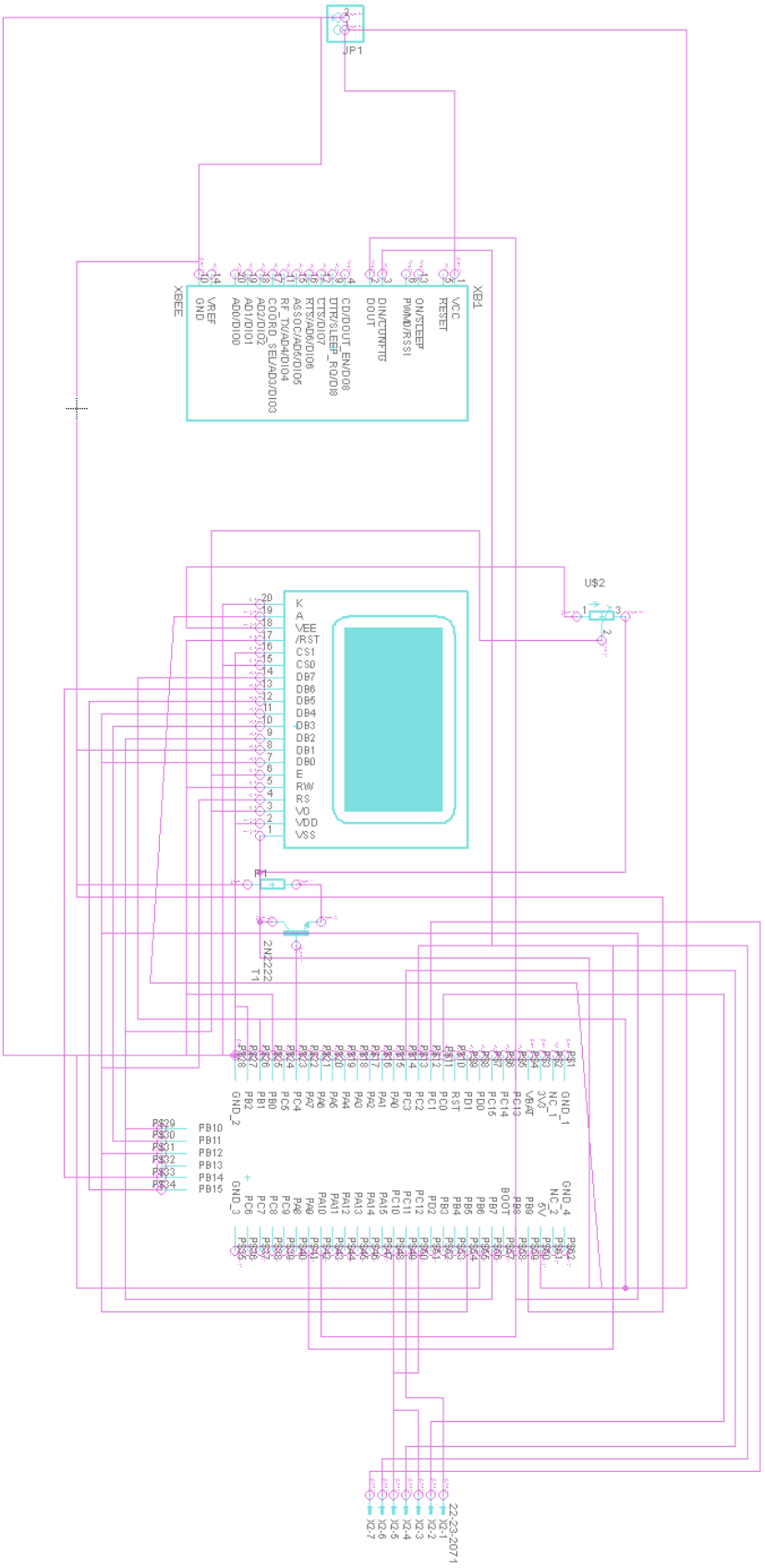

VCC
M1
XBee Radio
VCC
DOUT
DIN
CD/DOUT_EN/DO8
RESET#
RSSI_PWM
RESERVED
RESERVED
DTR/SLEEP_RQ/DI8
GND
AD0/DIO0
AD1/DIO1
AD2/DIO2
AD3/DIO3/COORD_SEL
AD6/DIO6/RTS#
AD5/DIO5/ASSOC
VREF
ON/SLEEP#
DIO7/CTS#
AD4/DIO4/RF_TX
XBEE
Port_Alimentation
JP1
JP2
Port_Capteur
GND

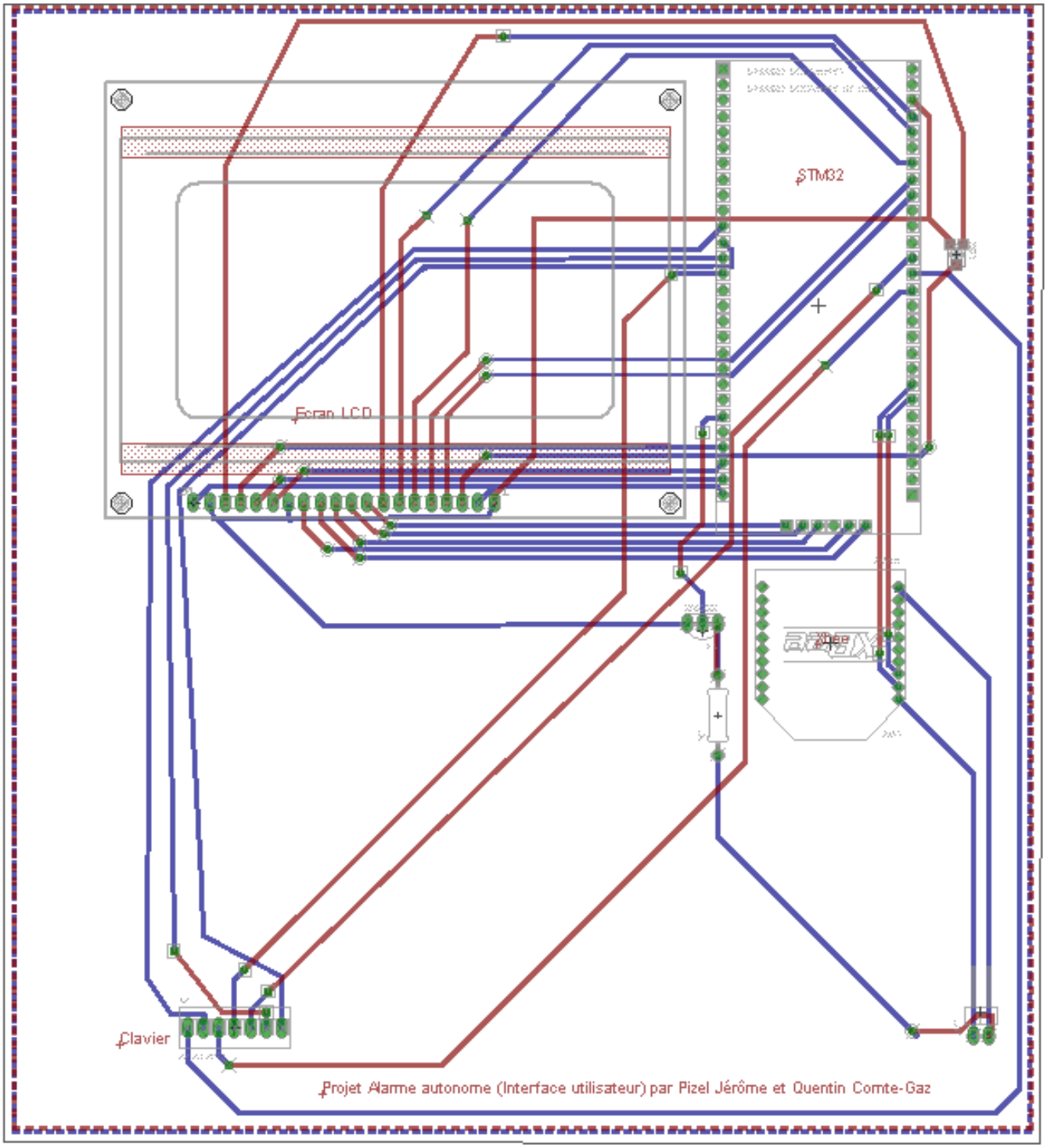
2) Centrale


GND
Vcc
STM32
Masse sirène
9V sirène
V- HP
V+ HP
Projet alarme autonome (Centrale) par Jérôme Pizel et Quentin Comte-Gaz


VCC
MASSE
JP1(MASSE_SIRENE)
JP2(9V_SIRENE)
T1
2N2222
T2
2N2222
LED1
LED2
LED3
C1(MOYEN)
C1(GROS)
NE555
IC1
IC2
TM1
TM3
JP2(V-_HP)
JP1(V+_HP)

3) interface utilisateur




STM32
Ecran LCD
Clavier
Projet Alarme autonome (Interface utilisateur) par Pizel Jérôme et Quentin Comte-Gaz

IV) Améliorations envisageables

1) Système autonome

Cette partie va consister à trouver les batteries les plus adaptées.
Nos composants du bloc « Capteurs » consomment très peu. En effet, l'Interrupteur à lames souples ne consomme RIEN car c'est un interrupteur commandé par un aimant. Le capteur PIR consomme peu (de l'ordre du mW). Enfin, les XBee consomment uniquement lors des acquisitions (de l'ordre d'une dizaine de nW) et lors de l'envoi des trames API (de l'ordre du mW).
Nos composants du bloc « Centrale » consomment aussi très peu. Telle que la carte STM32 discovery qui consomme environ 1,6 mW en permanent.
Le XBee de cette partie consomme également un peu plus que les autres (environ 2-3 mW) car il est plus sollicité que les autres. Le bloc « interface utilisateur » consomme encore un peu plus car nous utilisons un afficheur (environ 4 mW), une carte STM32 discovery (environ 1,6 mW), un module XBee (environ 1 mW) et un clavier numérique (quelques nW).
Remarquons qu'il serait judicieux d'éteindre l'afficheur si celui-ci n'est pas utilisé. En effet, cela augmenterait fortement l'autonomie. Le bloc « Sirène d'alarme » est celui qui consomme le plus en énergie mais est très peu utilisé : il consomme de l'ordre d'une dizaine de Watts.

Pour le calcul de chacun des blocs et la stabilisation de la tension veuillez vous référer à l'annexe 1.

2) Système GSM

Le système GSM doit permettre d'envoyer un SMS ou message vocal automatiquement en cas d'intrusion.
Ce système GSM sera composé d'un émetteur GSM, d'une carte SIM valide et sera connecté et commandé directement par notre centrale.
L'interaction entre la centrale et ce module n'est pas très compliqué à programmer.
En effet, si nous voulons envoyer un SMS, nous n'avons qu'à envoyer les données et le numéro téléphonique sous forme de trame à une fréquence (BaudRate) prédéfinie, et ce de la même manière qu'avec la configuration de l'USART1.
Notons que les émetteurs GSM sont trop onéreux pour notre projet (~100 €), nous ne pouvons donc pas en utiliser.

3) Interface Android

Cette interface consisterait à communiquer entre la centrale et un smartphone sous Android afin de pouvoir activer / désactiver l'alarme, ou activer / désactiver certaines zones de l'habitation. La centrale et l'utilisateur s'échangeraient donc des données via un module Bluetooth.
Notons que cet échange de données serait relativement aisé car nous avons déjà fait l'expérience d'émission / réception de trames via l'USART.

Conclusion

La réalisation d'une alarme sans fil n'est pas nécessairement une tâche aisée. D'autant plus lorsque la transmission sans-fil entre les différents modules de notre alarme est réalisée par le biais de modules XBee, technologie assez récente qu'il nous a fallu maîtriser.

L'alarme sans-fil que nous avons réalisée a été notre premier projet longue durée : cela nous a permis d'apprendre à gérer notre temps sur une longue période et d'améliorer notre capacité à travailler en équipe.

Cependant, nous avons commis l'erreur de ne pas prévoir de composants de secours, et la perte de plusieurs d'entre eux en travaillant à la réalisation d'une carte électronique aurait pu être rédhibitoire sans l'aide apportée par différents groupes qui nous ont permis de remplacer au dernier moment des composants hors-service.

De plus, ce projet nous a permis de nous familiariser avec les STM32 Discovery, les modules XBee, le fonctionnement des claviers numériques, le codage et l'utilisation d'écran LCD ainsi qu'à la réalisation de cartes électroniques sous Eagle.

Nous tenons à remercier tout particulièrement M. Tang qui nous a fait découvrir les modules XBee et surtout M. Monchal qui nous a aidé et qui a eu le courage de nous supporter pendant une année complète.

Annexe 1 : Autonomie de notre alarme

I) Consommation des différents blocs

Nos composants du bloc « Capteurs » consomment très peu. En effet, l'Interrupteur à lames souples ne consomme RIEN car c'est un interrupteur commandé par un aimant. Le capteur PIR consomme environ 2 mW.
Enfin, les XBee consomment uniquement lors des acquisitions (de l'ordre d'une dizaine de nW) et lors de l'envoi des trames API (environ 1 mW).
De plus, nous avons une acquisition et un envoi chaque seconde pour le bloc ILS et une toute les 3 secondes pour le bloc PIR avec un baudrate de 115200 et une trame de 224 bits(28 octets) d'où les calculs suivants donnant la consommation d'un XBee coté capteur :

% consommation XBee
= Temps de consommation(sec)/Temps entre chaque acquisition(sec)
= Longueur_trame x Temps_d'émission_d'un_bit/ Temps entre chaque acquisition
= (224 x 1/115200)/3
= 6.5.10-4

- **Consommation XBee = 6.5.10-4 x 1.10-3=6.5.10-7W**

Ainsi, la consommation des XBee associés aux capteurs ne consomment presque rien (de l'ordre du nW). En pratique, le XBee reste éveillé tout le temps.
D'où une consommation du bloc capteur PIR de 3 mW.
De la même manière, le bloc ILS consomme 1 mW (seul le XBee consomme).

Nos composants du bloc « Centrale » consomment plus que ceux des bloc capteurs.
En effet, la carte STM32 discovery consomme 1.6mW (200 µA/MHz et nous tournons à 8 MHz). Le XBee de cette partie consomme également un peu plus que les autres (environ 1.2mW en moyenne, pour les mêmes raisons que les XBee des blocs capteurs) car il est bien plus sollicité que les autres.
Ajoutons à cela un transceiver Bluetooth pour les communications avec un téléphone sous Android (1 mW en moyenne).
Le bloc centrale consomme alors en moyenne environ 3 ,8 mW.

Le bloc « interface utilisateur » consomme encore un peu plus car nous utilisons un afficheur (environ 3mW en moyenne si on coupe l'écran quand il n'est pas utile), une carte STM32 discovery (1.6mW), un module XBee (quasi-nul car peu utilisé) et un clavier numérique (quelques nW).

Remarquons donc qu'il serait judicieux d'éteindre l'afficheur si celui-ci n'est pas nécessaire. En effet, cela augmenterait fortement l'autonomie (la consommation réel de l'afficheur est de l'odre de 1 Watt).
Supposons donc que l'afficheur est éteint 99.99% du temps (logique puisqu'on l'allumera uniquement lorsque l'on tapera le code ou lorsqu'il y aura un risque d'intrusion).
L'interface utilisateur consomme donc en moyenne environ 5.6 mW.

Le bloc « Sirène d'alarme » est le plus consommateur en énergie mais est très peu utilisé.
Ce bloc consomme quelques dizaines de Watts.
En supposant que cette alarme se déclenche pendant 10 min chaque mois, alors le bloc sirène consommera en moyenne 1 mW.
Le bloc « Système GSM » consomme de l'ordre de quelques dizaines de mW lorsqu'il est allumé, or celui-ci ne l'est quasiment jamais (quelques secondes lorsqu'il y a effraction). Nous pouvons donc négliger sa consommation par rapport au bloc centrale à qui il soutirera l'énergie nécessaire pour son fonctionnement.

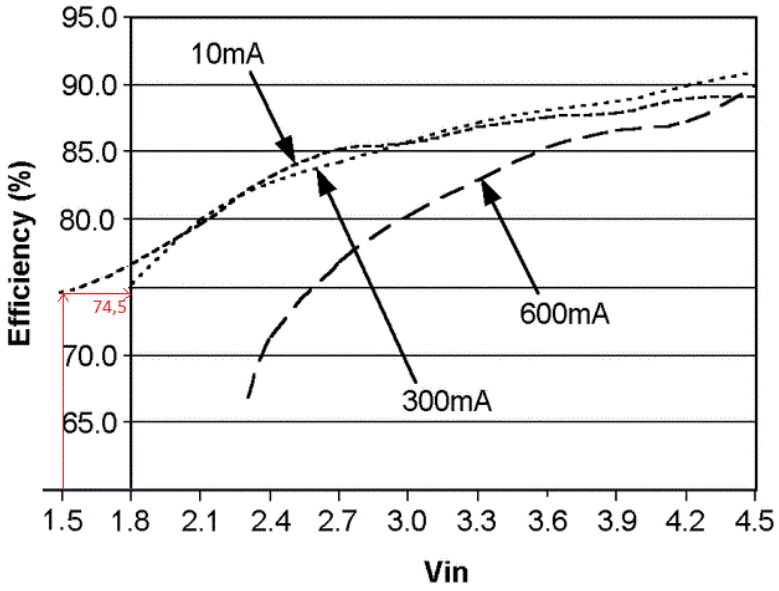
Ainsi, nous obtenons la consommation théorique suivante:

Bloc ILS	Bloc PIR	Bloc centrale + sirène + GSM (même source de tension)	Bloc interface utilisateur
1 mW	3 mW	3.8+1 mW	5.6 mW

II) Consommation réelle des différents blocs

Notre alimentation sera constitué de piles AAA (1.5V, 1.175 A.h)
Nos différents blocs auront besoin de tensions 3.3v ou 5v.
Nous allons donc devoir transformer du 4.5v en 3.3v (ou 5v) :
Nous allons donc utiliser un composant créé à cet effet : un LM2623 qui permet de transformer un signal continu 4.5v en 3.3v (ou 5v) à partir d'une source de tension quelconque.

Dans notre cas, ce composant possède un rendement compris soit de 75% (1.5v = une pile) soit de 85% (3v = 2 piles en série ou 4 piles) soit 90% (4.5v = 3 piles en série ou 6 piles) :



cf http://www.ti.com/product/LM2623

Ainsi, nous obtenons la consommation réelle suivante :

	Bloc ILS	Bloc PIR	Bloc centrale + sirène + GSM (même source de tension)	Bloc interface utilisateur
Puissance :	**1.17mW**	**3.5 mW**	**5.3 mW**	**6.2 mW**
Autonomie :	**8 mois**	**6 mois**	**6 mois**	**5 mois**
Nombre de piles utilisées :	**2 piles (3V)**	**4 piles (3V)**	**6 piles (4.5v)**	**6 piles (4.5v)**

Annexe 2 : Circuits imprimés