

Dossier de Projet



Membre de l'équipe :

IR : Hery Corentin
Dimier Benjamin
Bruccoleri Aloïs

EC : Charier Nicolas
Founas Yanis

Professeur Responsable : Antoine Philippe

Sommaire

I Partie commune	4
Présentation du projet :	4
Contexte du projet :	4
Cahier des charges :	5
Début du projet :	5
Diagramme :	6
Diagramme des cas d'utilisation :	6
Diagramme de déploiement :	7
Travail à effectué par membre :	8
Fiche de spécification :	10
Spécification des informations à paramétrer sur l'interface graphique :	10
Diagramme de gantt pour la revue 3 :	11
II Partie personnelle Bruccoli Aloïs	12
Présentation rapide de la partie	12
Travail à effectuer	13
Résumé de la Première revue :	14
Travail effectué pour la deuxième revue :	15
Programme pour l'utilisation de l'EEPROM Interne :	15
Programme pour l'utilisation de l'EEPROM externe :	16
Début de la programmation de la carte :	21
Travail effectué après la revue 2 :	29
Physique :	33
Présentation du bus SPI :	33
Exemple de trame :	35
Partie personnelle Dimier Benjamin	37
Format des commandes	44
Partie personnelle Hery Corentin	54
Présentation de la partie IR1	54
Travail effectué :	55
Format des commandes	56
Voici les diagrammes de séquences de chaque fonctionnalité de la carte	63
Partie personnel Yanis Founas	76
Constitution d'une trame UART	81
Le transistor	86
Annexes	101
Mode d'emploi	108
Description du produit :	109
Installation :	109
Utilisation du dispositif :	109

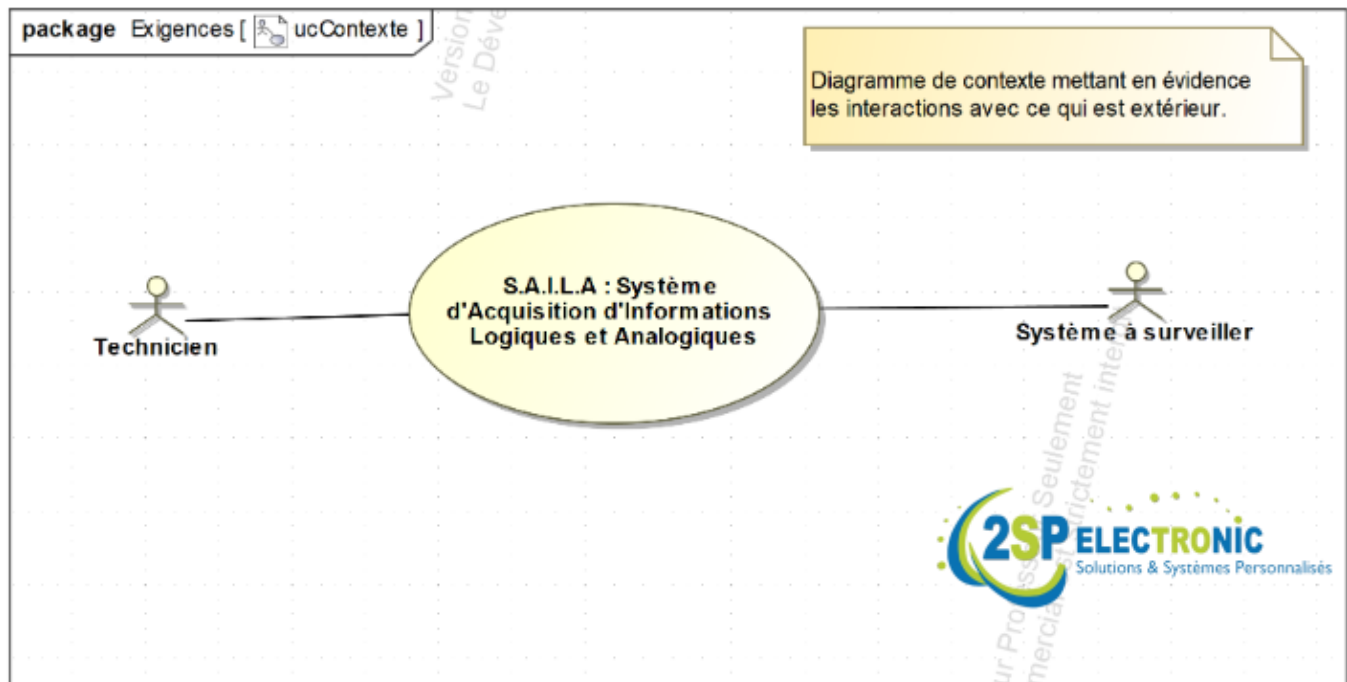
Connexion :	109
Paramétrage d'une mission :	109
Mettre en route les acquisitions :	109
Déchargement des données :	110
Reset de la carte :	110
Charger une mission :	110
Dépannage :	110

I Partie commune

Présentation du projet :

En quoi consiste notre Projet ?

Il s'agit de concevoir un outil portable permettant de faire l'acquisition cyclique de tensions analogiques, d'une température et de stocker ces informations en mémoire après numérisation. Puis lors de la connexion de cet outil sur le port USB d'un ordinateur, les données stockées sont récupérées sauveées et exploitées dans un logiciel tableur. Le logiciel PC permettra de configurer l'appareil lors d'une première connexion USB. Ci-après le cahier des charges fourni par l'entreprise.



Contexte du projet :

L'entreprise 2SPeectronic a été créée par Dominique Morin. Experte en solutions et systèmes personnalisés, l'entreprise accompagne les TPE et PME vers des solutions électronique. Le projet a pour but d'aider l'entreprise dans la maintenance préventive et curative qu'elle effectue.



Cahier des charges :

Le projet se décompose donc en 2 parties :

- Une carte électronique :
 - Pouvant ou non être alimentée par le système existant
 - Dotée d'une RTC et d'un microcontrôleur PIC (Mirochip)
 - Liaison USB
 - Entrées configurables selon :
 - Les tensions analogiques à relever
 - Présence secteur
 - Niveaux logiques
- Une application informatique :
 - Lecture et enregistrement des données dans un tableur Excel
 - Configuration
 - Ecart de temps
 - Nombres et spécificités des données
 - Reset

Début du projet :

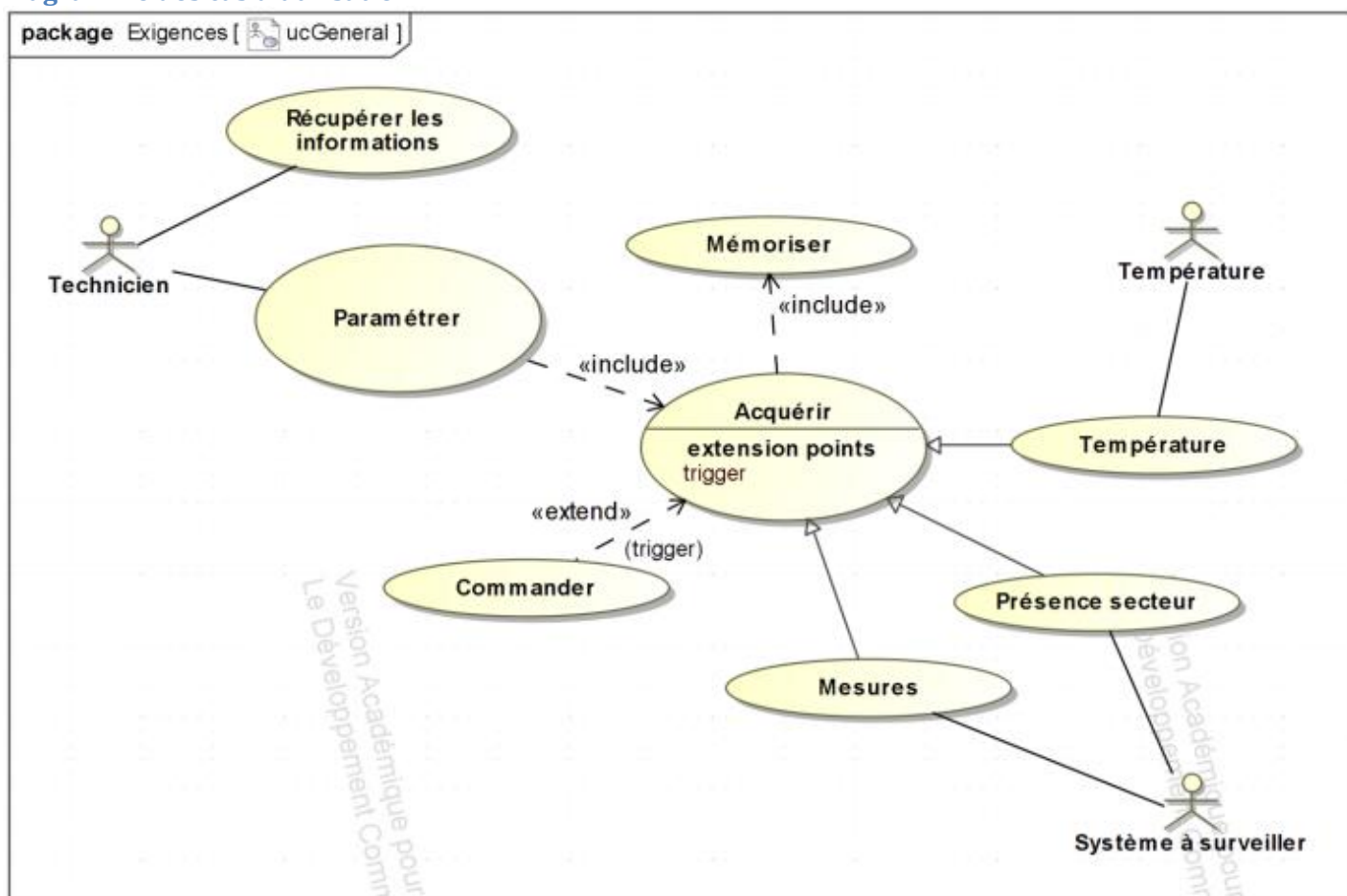
Dès la première séance, nous avons fait une réunion de groupe, où nous avons discuté, de notre projet, et défini les caractéristiques techniques pour notre compte rendu écrit, (taille de police, logiciel) et nous avons défini un logo pour notre projet.



Ensuite nous avons fait notre diagramme de Gantt et notre plan de développement distinctif où chacun a déterminé grâce à son dossier ses travaux.

Diagramme :

Diagramme des cas d'utilisation :



« Récupérer les informations »

Ce cas est pris en charge par le logiciel PC. Lors d'une connexion de la carte sur un port USB, le logiciel détermine qu'il faut récupérer les données. Les données récupérées sont sauveées dans un fichier au format CSV. Une fois les données récupérer ou non, la carte peut être réinitialisée pour retourner à l'état paramétrage.

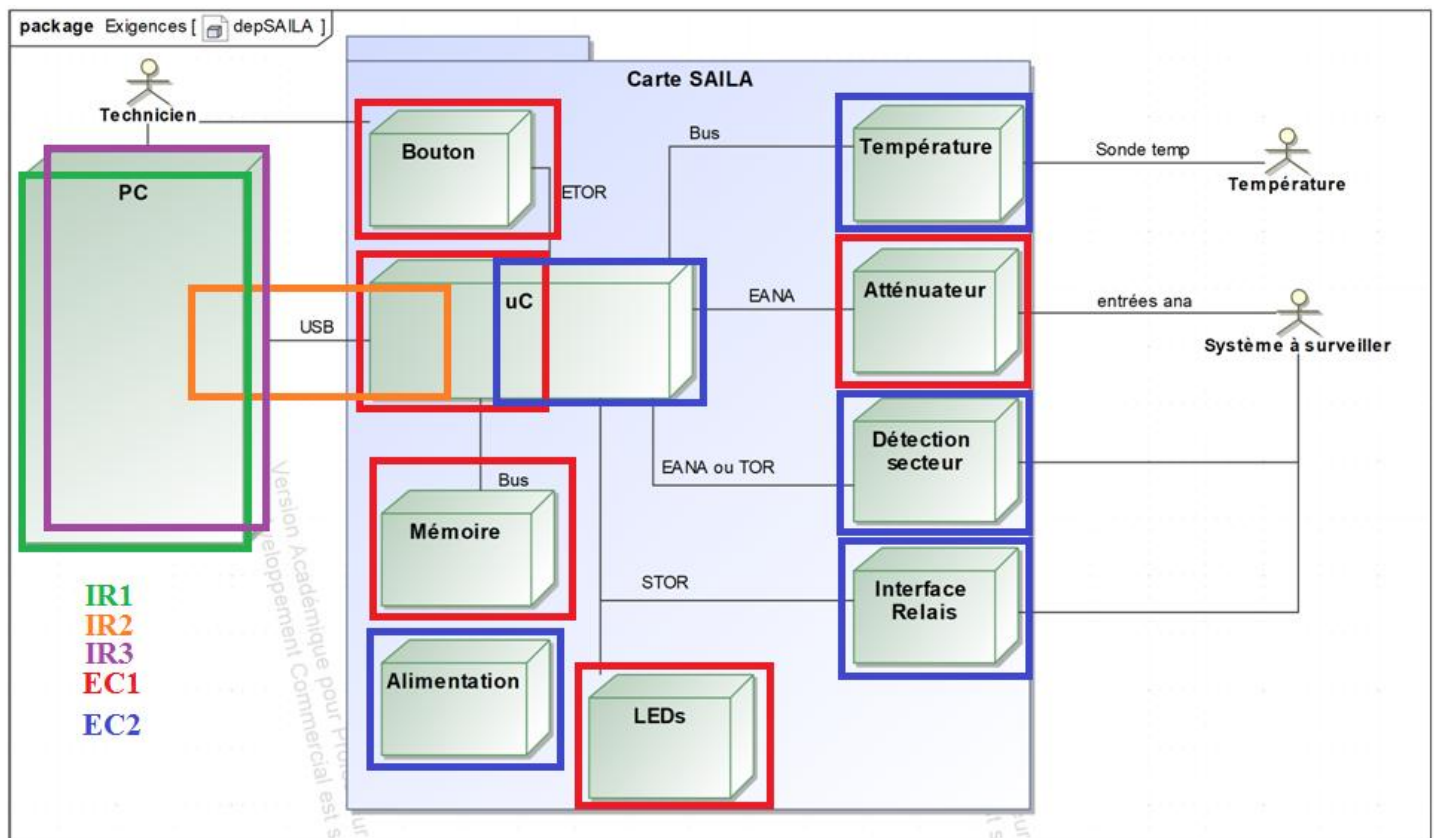
« Paramétrer »

Ce cas est pris en charge par le logiciel PC. Le paramétrage est effectué par le biais du logiciel développé sur PC. A la connexion, le programme récupère l'état de la carte pour déterminer si il faut paramétrer le système pour une nouvelle acquisition ou récupérer les informations suite a une mission. Si un paramétrage a lieu, les paramètres seront envoyés dans la carte et sauvegarder dans un fichier.

« Acquérir »

Ce cas est pris en charge par la carte SAILA. Suite à la déconnexion USB et si l'état du système est « En attente d'acquisition », les acquisitions commencent. Les informations acquises seront stockées en mémoire pour un déchargement ultérieur. Il y a également la possibilité de contrôle un relais grâce au paramétrage d'un trigger dans la phase de paramétrage de la carte.

Diagramme de déploiement :



Sur ce diagramme de déploiement, on met en évidence la partie de chacun.

Travail à effectué par membre :

Étudiant 1 IR 1	<i>Liste des tâches assurées par l'étudiant</i> Assurer le développement du logiciel PC Windows permettant : - Le paramétrage de la carte SAILA, - La sauvegarde du paramétrage dans une base de données et dans un fichier CSV. - Le rapatriement des données acquises et leur sauvegarde dans une base de données et dans un fichier CSV. - La communication avec la carte SAILA (en collaboration avec l'étudiant IR2).	Installation : Système d'exploitation Windows, EDI, driver USB. Mise en œuvre : Communication USB, BDD, microcontrôleur. Configuration : EDI. Réalisation : Logiciel PC Windows de gestion de la carte SAILA. Documentation : Manuel d'installation, de fonctionnement.
Étudiant 2 IR 2	<i>Liste des tâches assurées par l'étudiant</i> Assurer le développement (C/C++) du logiciel sur le microcontrôleur PIC18F45K50 permettant : - L'acquisition des 8 entrées analogique. - L'acquisition d'une température. - La commande du relais. - La communication USB avec le PC pour le paramétrage et le transfert des informations acquises.	Installation : EDI du microcontrôleur. Mise en œuvre : Développement des E/S du microcontrôleur. Configuration : EDI. Réalisation : Logiciel contenu dans la carte SAILA. Documentation : Manuel d'utilisation et de maintenance.
Étudiant 3 IR 3	<i>Liste des tâches assurées par l'étudiant</i> Assurer le développement du logiciel PC Linux permettant : - Le paramétrage de la carte SAILA, - La sauvegarde du paramétrage dans une base de données et dans un fichier CSV. - Le rapatriement des données acquises et leur sauvegarde dans une base de données et dans un fichier CSV. - La communication avec la carte SAILA (en collaboration avec l'étudiant IR2).	Installation : Système d'exploitation Linux, EDI, driver USB. Mise en œuvre : Communication USB, BDD, microcontrôleur. Configuration : EDI, Réalisation : Logiciel Linux de gestion de la carte SAILA. Documentation : Manuel d'installation, de fonctionnement.
Étudiant 4 EC 1	<i>Liste des tâches assurées par l'étudiant</i> - Le schéma structurel d'une solution possible, non testée, sera fourni à l'étudiant. - Sur un outil de développement à microcontrôleur PIC, effectuer des essais pour mettre en œuvre les convertisseurs analogiques numériques permettant l'acquisition de 8 entrées analogiques et le stockage de leur amplitude numérisée dans une mémoire externe. Le déclenchement se fera par l'appui sur un bouton poussoir, le remplissage complet de la mémoire sera signalé par une LED. Ces essais pourront amener à apporter toute modification nécessaire au schéma structurel proposé. - Conjointement avec l'étudiant EC2 finaliser le schéma structurel complet et le soumettre au client. - Effectuer la saisie du schéma et le routage de la solution proposée complète (routage individuel). Produire les fichiers Gerber afin que la fabrication du PCB soit sous-traitée. - Câbler le PCB de la carte et effectuer les essais. - Documenter la mise en service de la carte finalisée.	Installation : MPLAB X IDE sous Windows. Prise en main d'une carte de développement à microcontrôleur PIC18F45K50. Mise en œuvre : Tester/valider/modifier une partie du schéma structurel d'un outil d'acquisition et de stockage de grandeurs analogiques. Utiliser pour cela la carte de développement à microcontrôleur, associée à des composants extérieurs câblés en prototypage rapide. Réalisation : Suite aux essais préalables, proposer un schéma structurel complet conjointement avec l'autre étudiant EC dont les essais sont complémentaires. Après validation de la solution par le client, concevoir un circuit imprimé devant être fabriqué industriellement. Documentation : Schémas de câblage rapide (Fritzing) pour documenter la phase d'essais. Documents de fabrication des cartes (KiCad). Ces documents devront avoir un niveau de qualité permettant une fabrication industrielle du circuit imprimé. Programmes en C/C++ permettant de valider le fonctionnement des différents éléments de la carte, accompagnés des commentaires et diagrammes nécessaires à leur compréhension et mise en œuvre.

Étudiant 5 EC 2	<i>Liste des tâches assurées par l'étudiant</i> - Le schéma structurel d'une solution possible, non testée, sera fourni à l'étudiant. - Sur un outil de développement à microcontrôleur PIC, effectuer des essais pour mettre en œuvre la mesure d'une température externe, la commande d'un relais et la détection de la présence secteur. L'alimentation de la carte sera également à tester. Ces essais pourront amener à apporter toute modification nécessaire au schéma structurel proposé. - Conjointement avec l'étudiant EC1 finaliser le schéma structurel complet et le soumettre au client. - Effectuer la saisie du schéma et le routage de la solution proposée complète (routage individuel). Produire les fichiers Gerber afin que la fabrication du PCB soit sous-traitée. - Câbler le PCB de la carte et effectuer les essais. - Documenter la mise en service de la carte finalisée.	Installation : MPLAB X IDE sous Windows. Prise en main d'une carte de développement à microcontrôleur PIC18F45K50. Mise en œuvre : Tester/valider/modifier une partie du schéma structurel d'un outil d'acquisition et de stockage de grandeurs analogiques. Utiliser pour cela la carte de développement à microcontrôleur, associée à des composants extérieurs câblés en prototypage rapide. Réalisation : Suite aux essais préalables, proposer un schéma structurel complet conjointement avec l'autre étudiant EC dont les essais sont complémentaires. Après validation de la solution par le client, concevoir un circuit imprimé devant être fabriqué industriellement. Documentation : Schémas de câblage rapide (Fritzing) pour documenter la phase d'essais. Documents de fabrication des cartes (KiCad). Ces documents devront avoir un niveau de qualité permettant une fabrication industrielle du circuit imprimé. Programmes en C/C++ permettant de valider le fonctionnement des différents éléments de la carte, accompagnés des commentaires et diagrammes nécessaires à leur compréhension et mise en œuvre.
Tous les étudiants	✓ <i>Tâches à traiter par l'ensemble des étudiants de l'équipe projet pour le développement de la solution</i> - Documents de vie du projet : - Fiches de lecture croisée - Comptes rendus de réunion. ✓ <i>Domaines de physique à traiter par l'ensemble des étudiants de l'équipe projet :</i> - Traitement du signal analogique - Signaux : Puissance et énergie - Numérisation de signaux - Lignes de transmission	Intégration de la solution et livraison au client du matériel/logiciel/sources/manuels. Le logiciel sera installable facilement chez le client en suivant une procédure écrite.

Fiche de spécification :

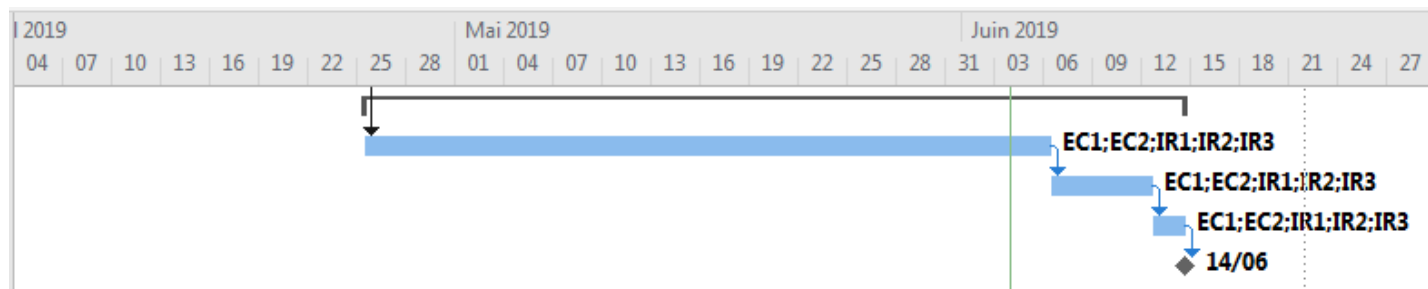
Spécification des informations à paramétrer sur l'interface graphique :

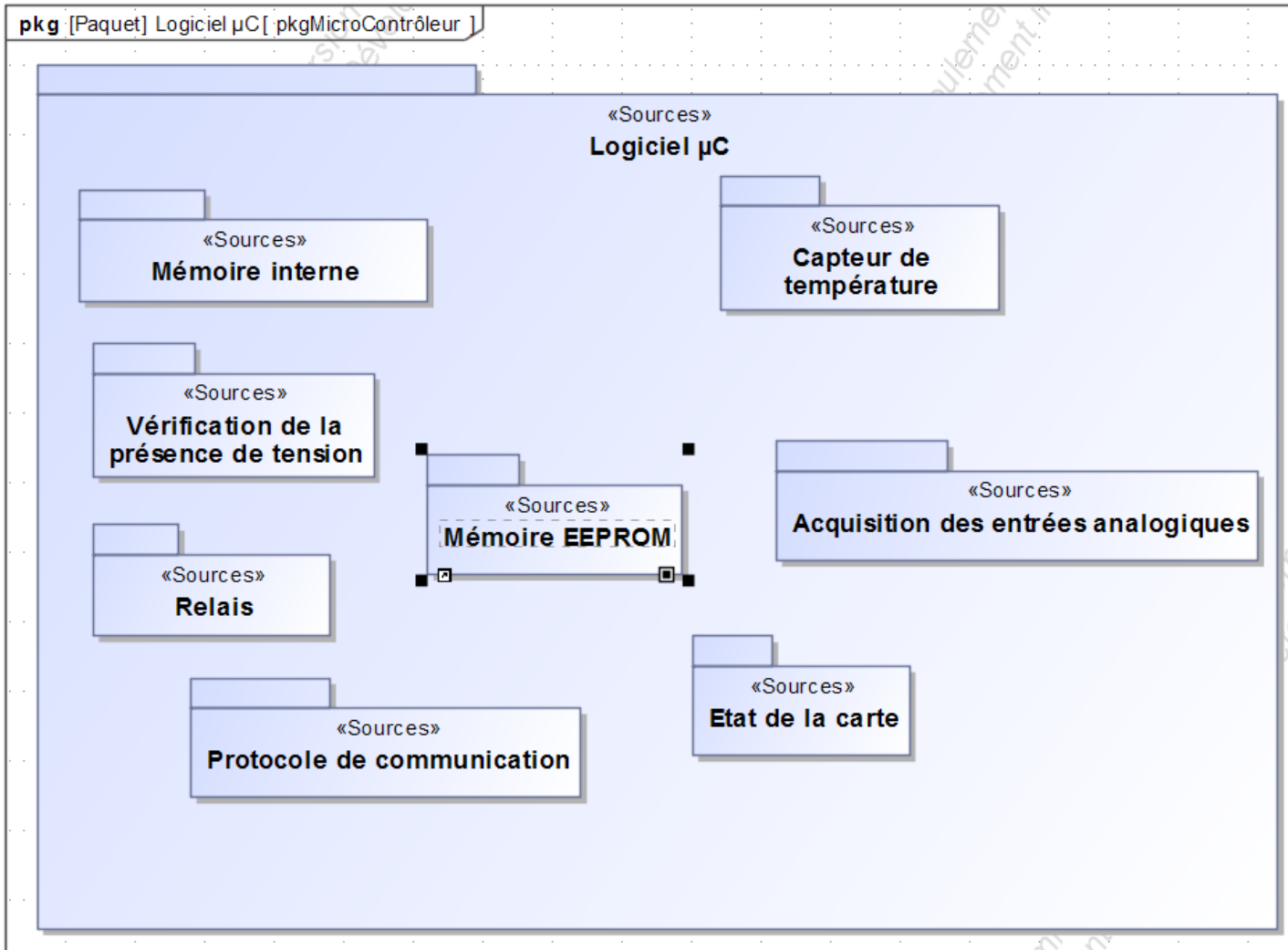
- Paramétrage du bouton poussoir
- Gérer le paramétrage des données
 - Temps entre chaque acquisition en seconde (amélioration en hh/mm/ss)
 - Durée totale d'acquisition en seconde (amélioration en hh/mm/ss)
 - Suppression/reset des données auparavant stockées
 - Paramétrage des sorties analogiques à utiliser
- Gérer la récupération des données pour les mettre dans un fichier CSV
- Le port série
- Paramétrage du relais en 0 à 255 (Si on le met en place ou pas. Si oui quand le déclencher (si on dépasse le seuil mais si on passe en dessous aussi))
- Les 9 entrées en analogique dont le capteur de température et une surveillance secteur
- Paramétrer le nom de la mission

Charger un fichier CSV existant

Diagramme de gantt pour la revue 3 :

Nom de la tâche	Durée	Début	Fin	Prédécesseurs	Noms ressources
▸ Revue de projet finale	84 h	Jeu 25/04/19	Ven 14/06/19		
Correction des bugs	64 h	Jeu 25/04/19	Jeu 06/06/19	55	EC1;EC2;IR1;IR2;IR3
Mise en place du mode d'emploie	10 h	Jeu 06/06/19	Mer 12/06/19	59	EC1;EC2;IR1;IR2;IR3
création des fiches recette	10 h	Mer 12/06/19	Ven 14/06/19	60	EC1;EC2;IR1;IR2;IR3
Revue finale	0 h	Ven 14/06/19	Ven 14/06/19	61	EC1;EC2;IR1;IR2;IR3





Ce diagramme de packages me permet d'avoir une vue d'ensemble sur les paquets logiciels à implémenter en se servant de l'IDE MPLAB X.

Travail à effectuer

Mon travail est constitué de plusieurs fonctionnalités à programmer.

Je dois programmer :

- La gestion de l'état de la carte
- Des LED qui indiqueront ces états.
- L'acquisition des 8 entrées analogiques et d'une température
- Le transfert des données vers le PC
- La prise en compte des paramètres effectués par l'utilisateur via le logiciel pc.
- Le déclenchement d'un relais suivant des conditions paramétrées.
- Stocker temporairement les données acquises dans une mémoire.

Étudiant 2

IR 2

Liste des tâches assurées par l'étudiant

Assurer le développement (C/C++) du logiciel sur le microcontrôleur PIC18F45K50 permettant :

- L'acquisition des 8 entrées analogique.
- L'acquisition d'une température.
- La commande du relais.
- La communication USB avec le PC pour le paramétrage et le transfert des informations acquises.

Plan des itérations :

N° itération	Activités	Justification
1	Mise en œuvre de plusieurs programmes : • Allumage de LED. • Bouton poussoir (par interruption). • Entrée analogique. • Communication série TTL.	Apprentissage de la programmation micro contrôleur et de son IDE.
2	Codage du programme principal du microcontrôleur : • Communication avec le pc • Acquisition des entrées analogiques • Acquisition de présence tension	Objectif principal de projet qui de pouvoir utiliser la carte pour réaliser des acquisitions.
3	• Programmation de la gestion de la mémoire flash. • Programmation du déclenchement relais • Intégration de toutes les parties du projet. • Création des notices et des manuels d'utilisation.	Aider le client à utiliser le matériel et l'application et présenter le projet.

Résumé de la Première revue :

J'ai effectué de nombreuses mises en œuvre pour la première revue. Elles m'ont permis de me familiariser une très grande partie des capacités de MPLAB et des capacités du microcontrôleur. Au début j'ai fait que faire clignoter une LED, ensuite l'interruption bouton, puis les interruptions timer.

```

void TMR0_DefaultInterruptHandler(void) {
    // add your TMR0 interrupt custom code
    // or set custom function using TMR0_SetInterruptHandler()
    IO_RA6_Toggle(); //change l'état de la LED
}

```

Change l'état de la led a chaque interruption due au timer.

```

void IOCB4_DefaultInterruptHandler(void) {
    // add your IOCB4 interrupt custom code
    // or set custom function using IOCB4_SetInterruptHandler()
    static int cpt=2;

    if (cpt%2) { // si le reste de la division est égale à 1 alors
        IO_RA7_Toggle(); // on change l'état la led;
        cpt=2; // et on remet cpt à 2
    } else
        cpt++; // incrémentation de cpt
}

```

Change l'état de la LED à chaque interruption due au bouton.

La condition permet de supprimer la double interruption.

Travail effectué pour la deuxième revue :

Pour la deuxième revue j'ai fait des mises en œuvre pour apprendre à me servir de la mémoire EEPROM interne et de la mémoire EEPROM externe, et ensuite programmer la carte pour la rendre quasi fonctionnelle pour le cahier des charges fourni.

Programme pour l'utilisation de l'EEPROM Interne :

Dans cette mise en œuvre, j'ai utilisé le timer, le bouton poussoir et la voie série pour faire fonctionner la EEPROM. J'ai programmé l'interruption timer pour pouvoir écrire des valeurs toutes les secondes dans la mémoire interne puis le bouton poussoir pour commencer l'envoi des données sur la voie série.

```
void TMR0_DefaultInterruptHandler(void) {
    // add your TMR0 interrupt custom code
    // or set custom function using TMR0_SetInterruptHandler()
    IO_RA7_Toggle();
    value = ADC_GetConversion(channel_AN0); //récupère la valeur du potentiomètre
    value >>= 8; //effectue un décalage de 8 bit pour avoir une valeur sur 8 bit
    DATAEE_WriteByte(EESAddr, (uint8_t)value); // écrit la valeur dans la mémoire

    EESAddr ++; //incrémente l'adresse de la mémoire
    if (EESAddr > 0xEE){          //si l'adresse atteint 0xEE
        TMR0_StopTimer();      //on arrête le timer
    }
}
```

A chaque interruption timer, on récupère la valeur du potentiomètre puis on la stocke dans l'EEPROM. Si l'adresse dépasse EE, on arrête le timer.

```
void IOCB4_DefaultInterruptHandler(void) {
    // add your IOCB4 interrupt custom code
    // or set custom function using IOCB4_SetInterruptHandler()
    uint8_t valuread;
    uint8_t i = 0;
    static int etat = 0;
    etat++;

    if (etat==3) {          //annule l'interruption créé par la mise sous tension

        TMR0_StopTimer();      //stop le timer
        do{
            valuread = DATAEE_ReadByte(i++); //lit la valeur dans la mémoire
            printf("valeur : %d \r\n",valuread); // écrit la valeur sur la voie série
            __delay_ms(1000);
        }while(i < EESAddr); //quand tout a été écrit
        EESAddr = 0x00;      // on remet l'adresse a 0
        etat = 1;
        TMR0_StartTimer();  //et on relance le timer
    } // if 3
}
```

Si il y a une interruption bouton, on arrête le timer et on envoie toute les données stockées dans la mémoire sur la voie série.

Programme pour l'utilisation de l'EEPROM externe :

Pour utiliser l'EEPROM externe, j'ai utilisé le bus spi. J'ai donc installé le module MCC pour le spi. Puis, j'ai implémenté le timer pour qu'il écrive via le bus spi dans la mémoire toute les 2 secondes et le main pour lire dans la mémoire toute les 2 secondes.

```
void TMR0_DefaultInterruptHandler(void){
    // add your TMR0 interrupt custom code
    // or set custom function using TMR0_SetInterruptHandler()
    static uint8_t cpt = 0;
    static uint32_t adrW =0;
    uint8_t tab[14]={0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}; //initialisation du tableau
    IO_RA5_SetLow(); //broche du chip select
    SPI1_Exchange8bit(0x06); // activation du bit WEN (write enabled)
    IO_RA5_SetHigh();
    if (adrW < 0x40000){ //si l'adresse est inférieure a 0x40000
        IO_RA7_Toggle();
        tab[0]=0x02; //commande pour engager une écriture
        tab[1]=(uint8_t) (adrW>>16); //adresse
        tab[2]=(uint8_t) (adrW>>8); //adresse
        tab[3]=(uint8_t) adrW; //adresse
        tab[4]=cpt; //donné a écrire dans la mémoire
        IO_RA5_SetLow();
        //do{
        SPI1_Exchange8bitBuffer(tab,5,NULL); //écriture sur le bus pour la mémoire
        //} while(total < 14);
        IO_RA5_SetHigh();
        IO_RA7_Toggle();
        adrW ++;
        cpt++;
        if (adrW ==255) adrW=0x00; //si l'adresse est égale à 250 la remettre a 0
    }//if
}
```

Dans le timer, on écrit dans la mémoire la valeur de la variable cpt.

```
while (1)
{
    // Add your application code

    __delay_ms(2000);
    IO_RA6_Toggle();
    if((adrR-6)>0){
        datain[0]=0x03; //commande pour la mémoire qui active la lecture
        datain[1]=(uint8_t) (adrR>>16); //adresse
        datain[2]=(uint8_t) (adrR>>8); //adresse
        datain[3]=(uint8_t) adrR; //adresse
        IO_RA5_SetLow();
        //do{
        SPI1_Exchange8bitBuffer(datain,5,dataout); //envoi de la trame puis réception des données
        IO_RA5_SetHigh();
        //__delay_ms(1000);
        printf("%d\r\n",dataout[4]); //affichage des données
        adrR++;

        if(adrR == 255) adrR = 1;
    }//if
    else{
        //printf("erreur \r\n");
    }//else

    IO_RA6_Toggle();
} //while
}
```

Dans le « main » on lit la valeur du timer dans la mémoire toutes les 2 secondes.

Diagramme de gantt prévisionnel :

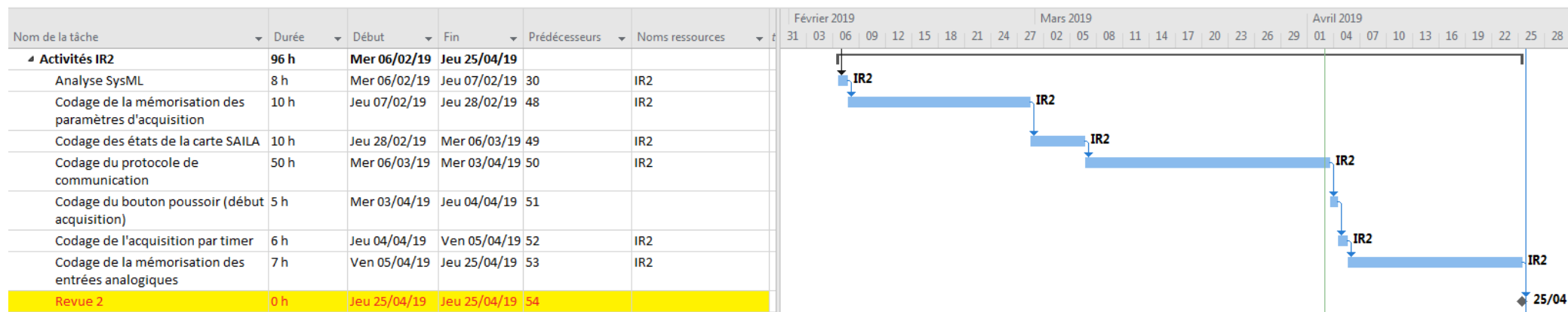
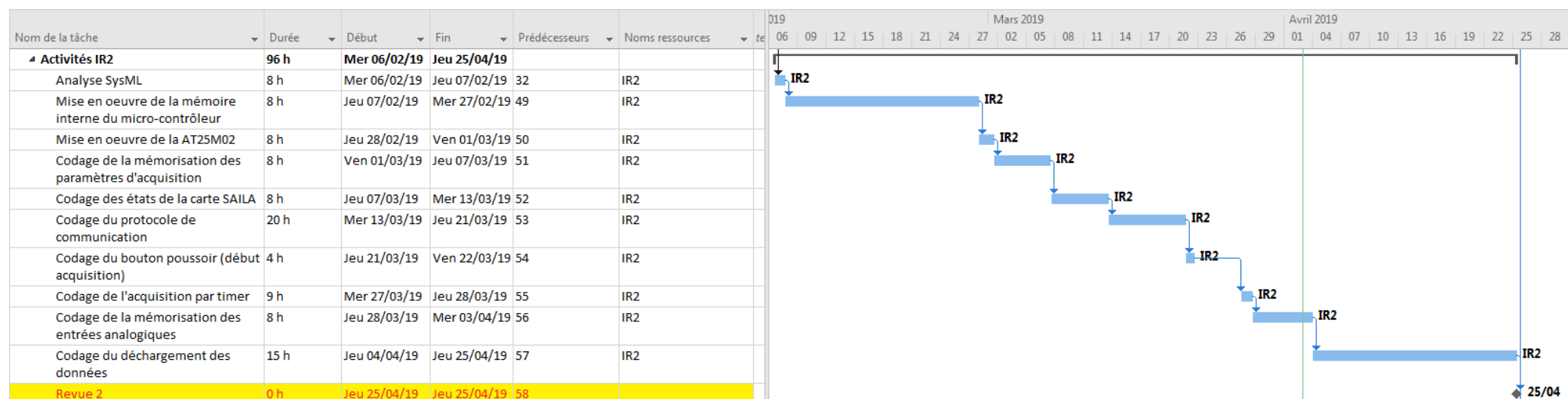


Diagramme de gantt prévisionnel



interaction ssdGeneral[ ssdGeneral]

 Diagramme de séquence système :
 Fonctionnement général du système
: Technicien «Système»
: SAILA: Système à surveiller **opt**

[Préparation]

1: Connecter USB

2: Paramétrer

3: OK

4: Déconnecter

[Acquisition]

loop

[Intervalle A]

5: Acquérir

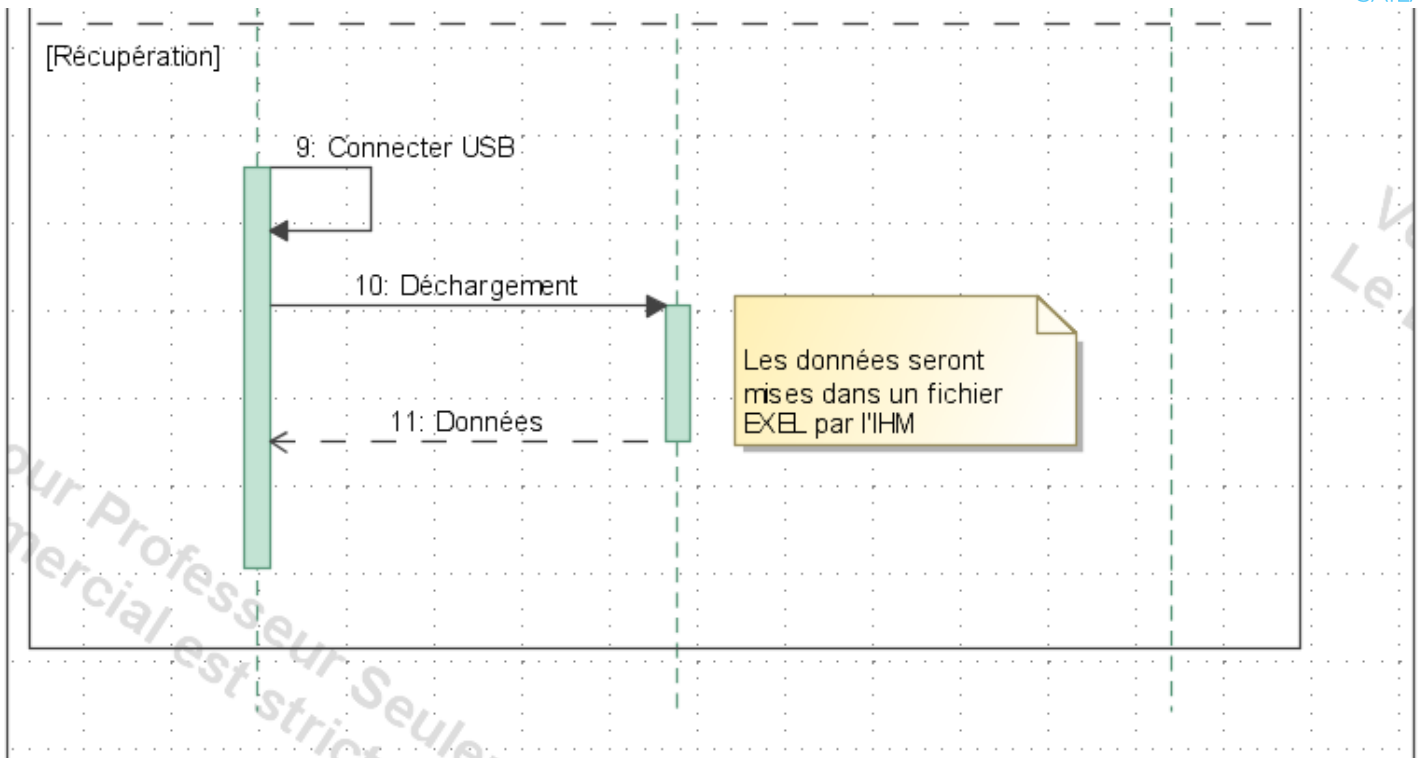
6: Mesures

7: Mémoriser

opt

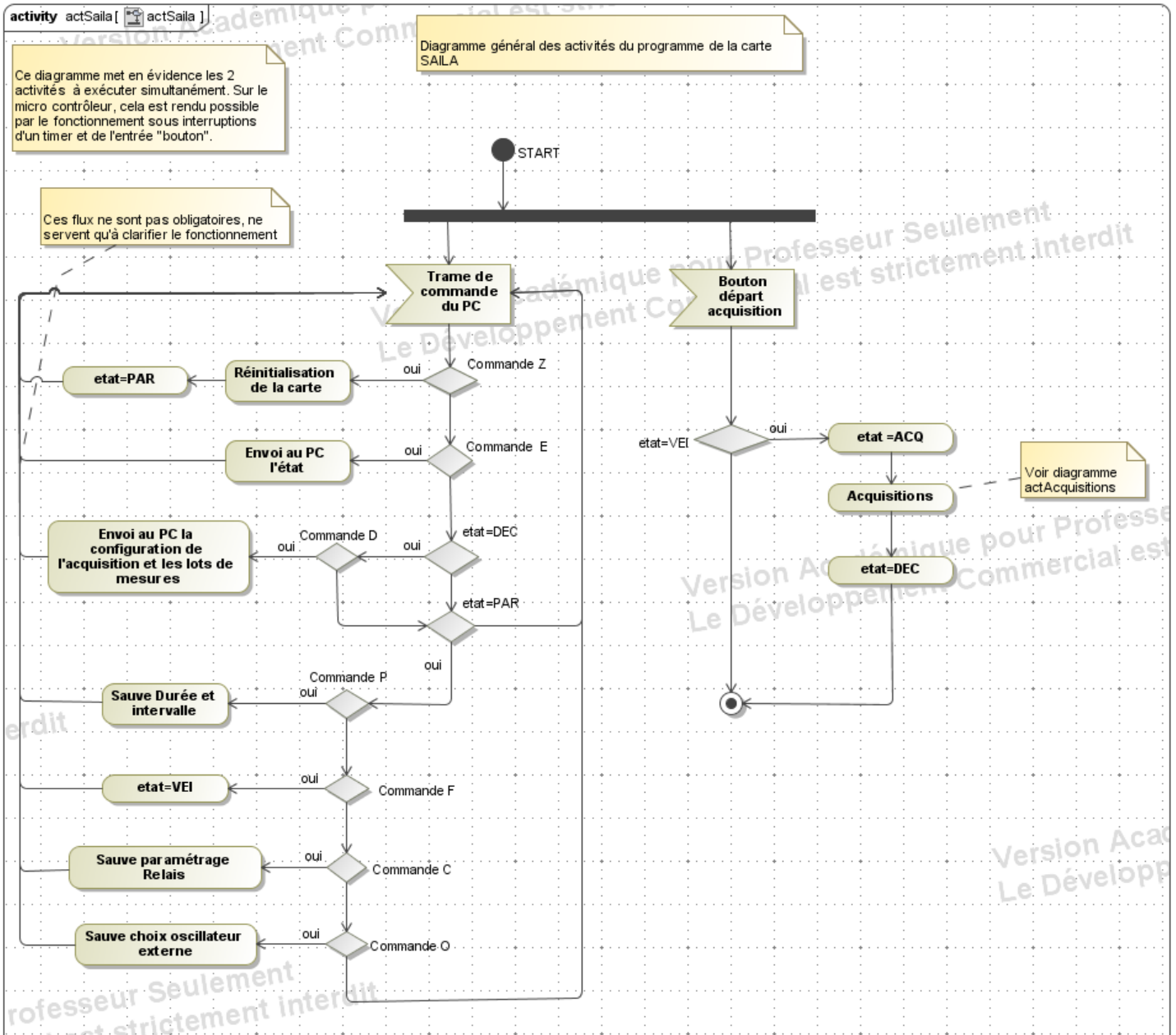
[trigger]

8: Commander



Ce diagramme de séquence permet d'avoir une vue d'ensemble du fonctionnement du système.

Puis, on a ce diagramme d'activité qui résume les actions de la carte.



Début de la programmation de la carte :***Mise en place de la ZDC :***

La ZDC (zone de mémoire commune) est située dans l'EEPROM interne au microcontrôleur.
Elle est agencée comme ceci :

Fiche

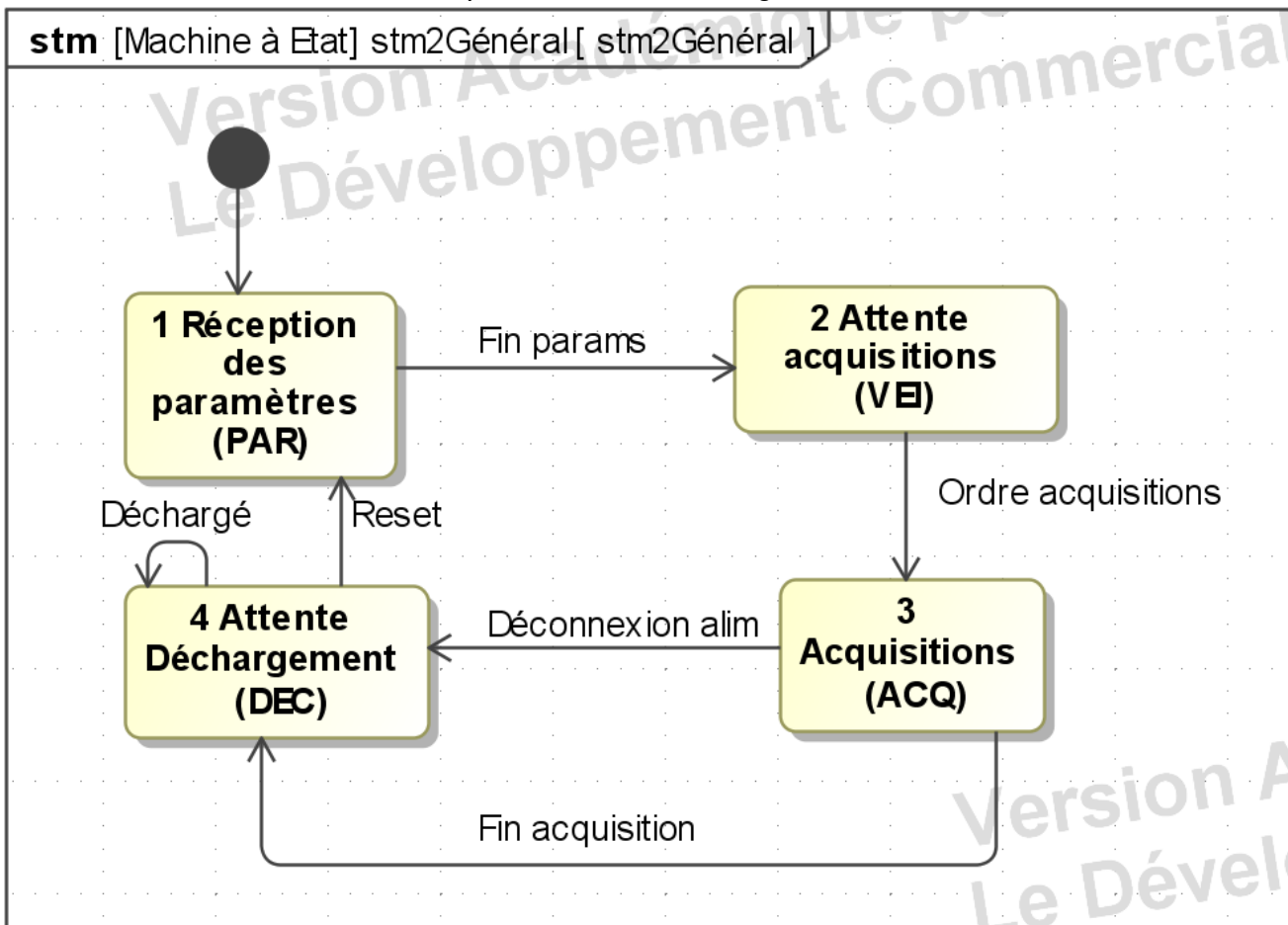
Zone de Mémoire Commune (EEPROM du MC)

Contenu de la zone mémoire :

Adresse	Nombre d'octets	Désignation
0x00	1	Etat de la carte
0x01	4	Nombre de lots *
0x05	2	Durée d'acquisition *
0x07	2	Intervalle d'acquisition *
0x09	4	Adresse dans la mémoire additionnel
0x0C	3	Paramétrage du relais
0x0F	1	Valeur instantanée du Relais
0x10	1	Valeur instantanée de la présence secteur

* Poids fort en premier

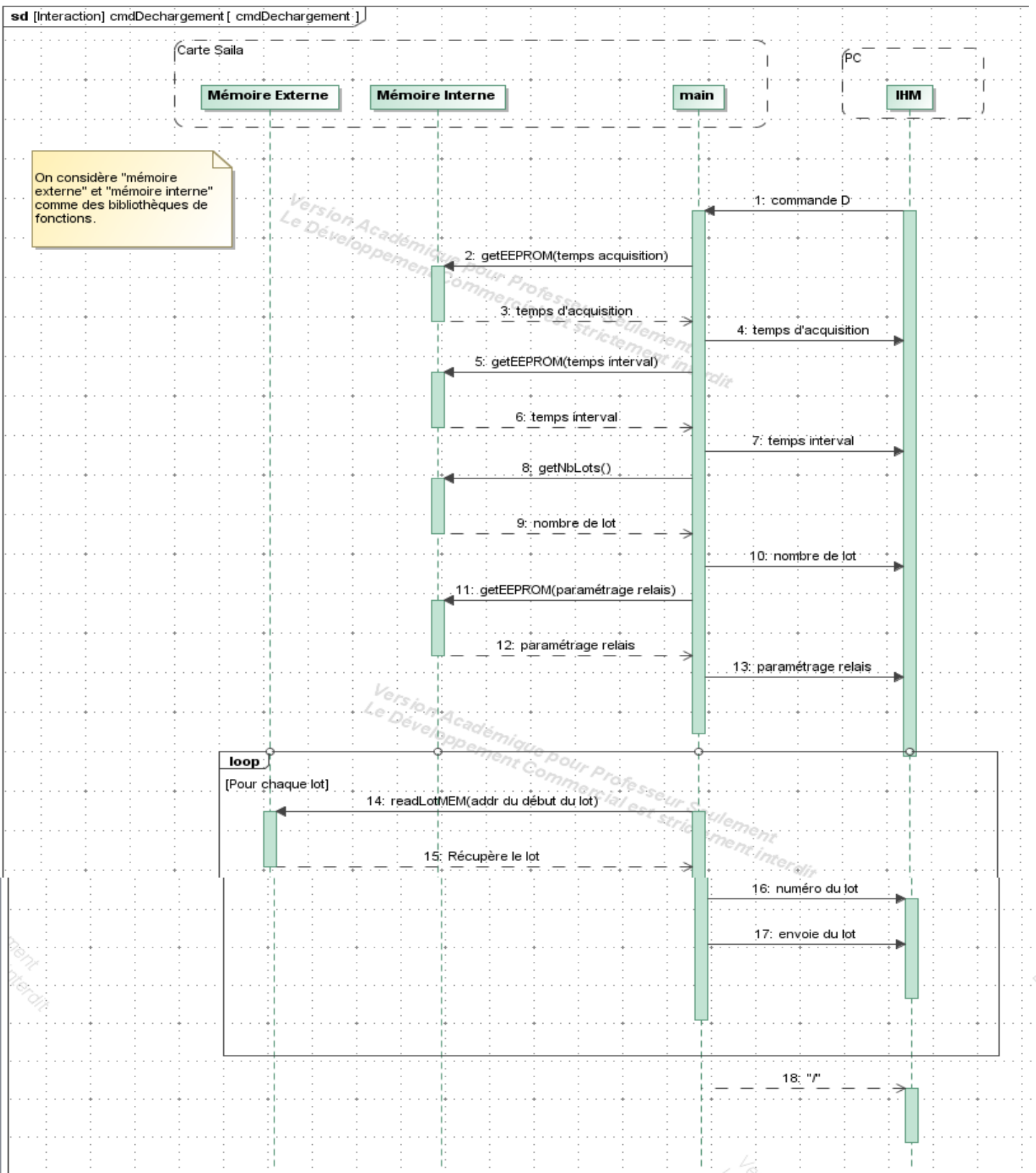
Pour les états de la carte, on peut les voir sur ce diagramme d'état :



L'état PAR est l'état dans lequel se trouve la carte par défaut. Dans cet état, elle attend de recevoir les paramètres d'acquisitions qui lui seront envoyés par le pc. Suite à la fin du paramétrage la carte va passer en VEI. De cette état, elle attend l'appui sur le bouton poussoir pour commencer les acquisitions, ce qui va la passer dans l'état ACQ. A la fin des acquisitions ou suite a une déconnexion à l'alimentation, elle va passer dans l'état DEC, dans lequel elle va attendre une demande d'envoi de données ou de reset.

Mise en place de la communication :

Dans les fichiers « mémoire » se trouvent les fonctions permettant l'envoi de données sur la voie série.



Dans ce diagramme de séquence, on peut voir le déroulement de la commande Déchargement.

Dans le main se trouve le protocole de communication :

Les commandes se trouvant dans un switch, il y a un case pour chaque :

- La commande Z qui permet le reset de la carte

```
case 'Z':
    resetCarte();
    reponse1C(getEtat());
    break;
```

Tout d'abord il y a un reset de la carte puis la carte répond l'état

- La commande E renvoie l'état de la carte

```
case 'E':
    reponse1C(getEtat());
    break;
```

La carte répond l'état

- La commande F permet de terminer le paramétrage

```
case 'F':
    if(getEtat() == PAR){
        setEtat(VEI);
        reponse1C(getEtat());
    }
    break;
```

Si l'état est PAR, la carte passe dans l'état VEI et renvoie l'état

- La commande P permet de récupérer le paramétrage de l'intervalle et de la durée pour ensuite l'enregistrer dans l'EEPROM interne

```
case 'P':
    if(getEtat() == PAR){
        while(EUSART1_is_rx_ready() != 4);
        for (i=0; i<4; i++)
            tdi[i]= EUSART1_Read();
        parDureeInterv(tdi);
        reponse1C('O');
    }//if
    break;
```

Si la carte est en état PAR, on attend qu'il y ai 4 octets dans le buffer de la liaison série . Ensuite on met ce qu'on a récupéré dans un tableau qu'on enregistre dans la mémoire interne du microcontrôleur. Puis la carte renvoie 'O' pour dire que tout s'est bien passé.

- La commande D est celle qui déclenche le déchargement des données

```

case 'D':

    //temps acquisition
    getEEPROM(ADR_DUREEACQ,2,tacq);
    tempsAcq = (uint16_t) (tacq[0]<<8);
    tempsAcq += tacq[1];
    EUSART1_Write16bits(tempsAcq); //envoi le temps d'acquisition

    //intervalle d'acquisition
    getEEPROM(ADR_DUREEINT,2,tinter);
    tempsInter = (uint16_t) (tinter[0]<<8);
    tempsInter += tinter[1];
    EUSART1_Write16bits(tempsInter); //envoi l'intervalle d'acquisition

    //nombre de lot
    nbLots = getNbLots();
    EUSART1_Write32bits(nbLots); //envoi le nombre de lot

    //paramétrage relais
    getEEPROM(ADR_PARARELAIS,3,tabre);
    //tparRelais = (uint24_t) (tabre[0]<<16);
    tparRelais = tabre[0];
    tparRelais <= 16;
    tparRelais += (uint24_t) (tabre[1]<<8);
    tparRelais += tabre[2];
    EUSART1_Write24bits(tparRelais); //envoi le paramétrage relais

    addr = 0;
    for (i=0;i<nbLots;i++){
        readLotMEM(addr,tablot);
        addr += TAILLE_LOT;
        EUSART1_Write32bits(i); //envoi le numéro du lot
        for (j=0;j<TAILLE_LOT;j++){
            EUSART1_Write(tablot[j]); //envoi le lot
        }
    }
    EUSART1_Write('/'); //termine le déchargement
    break;

```

Envoie du paramétrage de la durée d'acquisition

Envoie du paramétrage de la durée d'intervalle

Envoie le nombre de lot

Envoie du paramétrage de la durée

Pour chaque lots, on lit dans la mémoire le lot, on envoie le numéro du lot puis on envoie le lots.

Envoie du '/' pour dire que le déchargement est terminé.

- La commande C permet de récupérer le paramétrage du relais

```

case 'C':

    if(getEtat() == PAR){
        while(EUSART1_is_rx_ready() != 3);
        for (i=0;i<3; i++){
            tabre[i]= EUSART1_Read();
        }
        paraRelais(tabre);
        reponse1C('O');
    }
    break;

```

Si l'état est PAR, on attend qu'il y ai 3 octets dans le buffer de la liaison série puis on les met dans un tableau pour les écrire dans la mémoire interne du microcontrôleur. Puis la carte renvoie 'O' pour dire que tout s'est bien passé.

Pour envoyer sur la voie série les paramètres lors du déchargement, j'ai rajouté des fonctions dans le fichier eusart1 pour envoyer 16, 24 et 32 bits.

Les fonctions créées utilisent la fonction de base créée par MCC mais en plusieurs fois en décalant des bits.

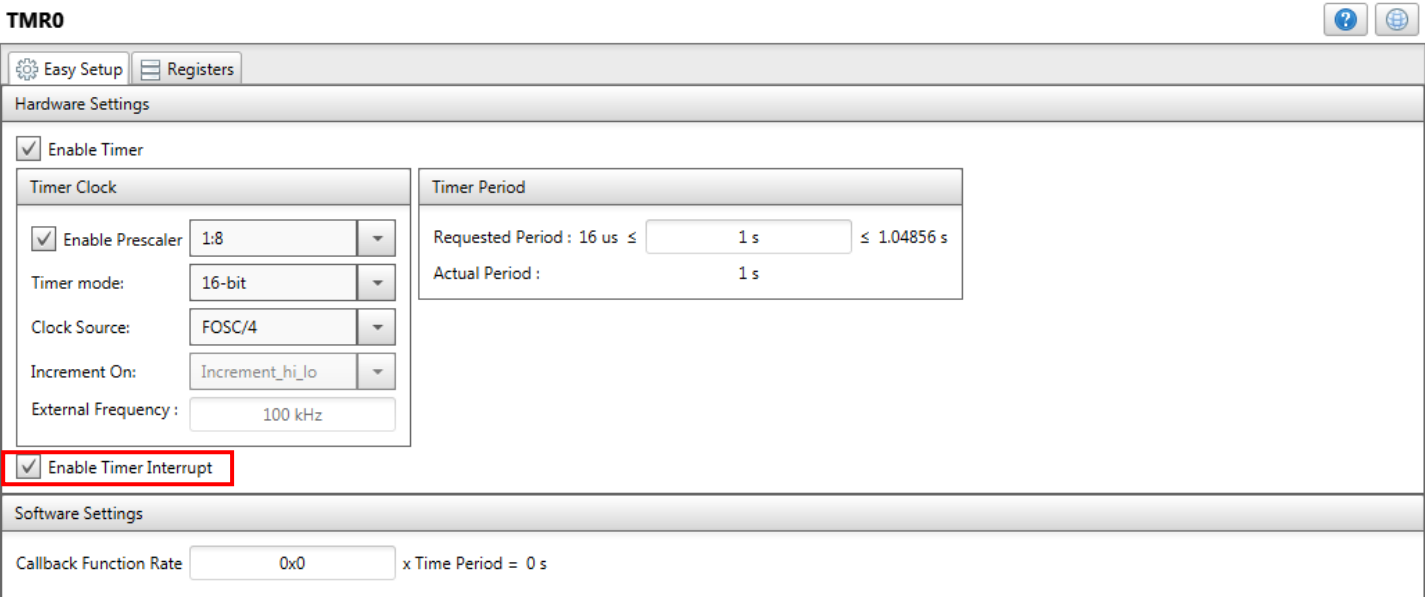
J'ai également codé un switch qui permet la gestion d'un débranchement brutal de la carte ou de lui mettre un état par défaut qui est le déchargement.

Etat avant le débranchement	Etat après mise sous tension
PAR	PAR
VEIL	VEI
ACQ	DEC
DEC	DEC

Ensuite il y a l'implémentation de l'interruption bouton pour pouvoir lancer les acquisitions.

Puis il y a la création d'une unité de compilation permettant l'interaction avec la mémoire EEPROM externe.

Ensuite il y a la configuration du timer :



Pour pouvoir utiliser le timer il faut activer l'interruption.

Comment fonctionne une interruption ?

le programme principal fonctionne normalement. Un événement extérieur survient(dans notre cas un tick du timer). Le micro contrôleur termine l'instruction qu'il exécutait et mémorise l'adresse en cours(elle lui permettra de revenir à l'endroit où il s'est arrêté avant l'interruption). Le microcontrôleur saute ensuite à l'adresse contenant le programme qui correspond à l'interruption(le programme ci-dessous dans notre cas). Il exécute le programme de l'interruption puis il revient à l'adresse suivante à celle qu'il avait précédemment mémorisée.

Ce qui permettra d'effectuer le code qui correspond aux acquisitions qui se trouve dans le gestionnaire d'interruption par défaut du timer. Il faut également ne pas oublier d'activer les interruptions dans le main :

```
// Enable the Global Interrupts
INTERRUPT_GlobalInterruptEnable();
```

Dans ce code, on récupère une première fois les paramètres (temps d'acquisition, temps d'intervalle et paramétrage du relais).

```

if (first){
    getEEPROM(ADR_DUREEACQ,2,tacq);
    tempsAcq = (uint16_t) (tacq[0]<<8);
    tempsAcq += tacq[1];
    //tempsAcq = 3600;

    getEEPROM(ADR_DUREEINT,2,tinter);
    tempsInter = (uint16_t) (tinter[0]<<8);
    tempsInter += tinter[1];
    //tempsInter = 1;

    mode = getDeclenchement(&valDec,&numMesure,&confSens);
    mode = 0;
}

```

Ensuite il y a la partie qui fait les acquisitions et le relais :

```

if(cpt <= tempsAcq){
    //teste relais condition durée
    if (mode == RTEMP){
        if (valDec == cpt){
            IO_RB4_RELAIS_SetHigh();
        }
    }
    //if mode

    if ((cptinter == tempsInter) || first){

        first = false;

        //acq mesure temp
        val = ADC_GetConversion(channel_AN0_lm35);
        tabMesure[0] = (uint8_t) (val >>2);
        //tabMesure[0] = 0x55;
        //acq mesure ANA
        for(i = channel_AN20_ana0;i <= channel_AN27_ana7;i++){
            val = ADC_GetConversion(i);
            tabMesure[(uint8_t) (i-channel_AN20_ana0+1)] = (uint8_t) (val >>2);
            //tabMesure[(uint8_t) (i-channel_AN20_ana0+1)] = i;
        }

        if (mode == RMESURE){
            if ( (sens==confSens) && (sens==DESCENDANT) && (tabMesure[numMesure]<(valDec)) ){
                IO_RB4_RELAIS_SetHigh();
                tabMesure[9] |= 2;
            }
            //if sens 0
            if ( (sens==confSens) && (sens==MONTANT) && (tabMesure[numMesure]>(valDec)) ){
                IO_RB4_RELAIS_SetHigh();
                tabMesure[9] |= 2;
            }
            //if sens 1
            if (memVal > tabMesure[numMesure]) sens = 0; //sens descendant
            else sens = 1; //sens montant
            memVal = tabMesure[numMesure];
        }
    }
}

```

```

    } //if mode
    //acq secteur
    tabMesure[9] += getSecteur();
    if (mode == RSTENS) {
        if (tabMesure[9] & 0x01) {
            IO_RB4_RELAIS_SetHigh();
        } //if tab
    } //if mode
    writeLotMEM(addr, tabMesure);
    nbLots ++;
    setNbLots (nbLots);
    addr += TAILLE_LOT;
    cptinter = 1;

    } //if cptinter
    else {
        cptinter ++;
    } //else
    cpt ++;
} else {
    setEtat (DEC);
    TMRO_StopTimer();
    first = true;
} //else temps acq
} //interrupt handler

```

Récupération de la valeur du secteur

Paramétrage par rapport à la présence secteur

Ecriture du lot dans la mémoire, incrémentation du nombre de lot, incrémentation de l'adresse.

Fin de l'acquisition (passage en état déchargement, arrêt du timer.)

Travail effectué après la revue 2 :

Création d'une fonction permettant l'initialisation de la carte :

```
void iniCarte() {
    TMRO_StopTimer();
    switch(getEtat()) {
        case PAR:
            setEtat(PAR);
            IO_RB4_RELAIS_SetLow();
            break;
        case VEI:
            setEtat(VEI);
            break;
        case ACQ:
            setEtat(DEC);
            break;
        case DEC:
            setEtat(DEC);
            break;
        default:
            setEtat(PAR);
            IO_RB4_RELAIS_SetLow();
            break;
    }
}
```

← Arret du timer

← Si la carte était en état PAR, on redéfinit l'état PAR et on met à 0 la broche du relais.

← Si la carte était en état VEI, on redéfinit l'état VEI

← Si la carte était en état ACQ, elle passe en état DEC

← Si la carte était en état DEC, on redéfinit l'état DEC

← Si l'état est autre que PAR,VEI,ACQ et DEC, on l'a met en état PAR et on met à 0 la broche du relais

Modification de la ZDC :

Fiche

Zone de Mémoire Commune (EEPROM du MC)

Contenue de la zone mémoire :

Adresse	Nombre d'octets	Désignation
0x00	1	Etat de la carte
0x01	4	Nombre de lots *
0x05	2	Durée d'acquisition *
0x07	2	Intervalle d'acquisition *
0x09	3	Paramétrage du relais
0x0C	1	Valeur instantanée de la présence secteur
0x0D	1	Choix de l'oscillateur

* Poids fort en premier

La valeur du relais ne se trouve plus dans la ZDC mais simplement dans une variable dans l'interruption timer.

Création du bouton de début d'acquisition :

Tout d'abord, j'ai configuré la broche qui va accueillir le bouton :

Pin Name ▲	Module	Function	Custom Name	Start High	Analog	Output	WPU	OD	IOC
RB4	Pin Module	GPIO	IO_RB4_RELAIS	☐	☒	☒	☐		none ▼
RB5	Pin Module	GPIO	IO_RB5_PreSec	☐	☐	☐	☐		any ▼
RB6	Pin Module	GPIO	IO_RB6_ProgCa	☐		☒	☐		none ▼
RB7	Pin Module	GPIO	IO_RB7_ProgCa	☐		☒	☐		none ▼
RC0	Pin Module	GPIO	IO_RC0_BPGO	☐		☐			any ▼

Activation de l'interruption

Puis, dans le gestionnaire d'interruption par défaut correspondant, j'ai implémenté le bouton de lancement d'acquisition.

```

void IOCC0_DefaultInterruptHandler(void) {
    // add your IOCC0 interrupt custom code
    // or set custom function using IOCC0_SetInterruptHandler()
    uint8_t etat;

    etat = getEtat();
    if (etat == VEI) { //si la carte est en veille
        setEtat(ACQ); //changer l'état en "En cours d'acquisition"
        TMRO_StartTimer(); //démarrer le timer
    } //if etat
}

```

Création de l'interruption présence secteur.

Tout d'abord, j'ai configuré la broche de la présence secteur.

Pin Name ▲	Module	Function	Custom Name	Start High	Analog	Output	WPU	OD	IOC
RB4	Pin Module	GPIO	IO_RB4_RELAIS	☐	☒	☒	☐		none ▼
RB5	Pin Module	GPIO	IO_RB5_PreSec	☐	☐	☐	☐		any ▼
RB6	Pin Module	GPIO	IO_RB6_ProgCa	☐		☒	☐		none ▼
RB7	Pin Module	GPIO	IO_RB7_ProgCa	☐		☒	☐		none ▼
RC0	Pin Module	GPIO	IO_RC0_BPGO	☐		☐			any ▼

Suite a cela j'ai codé le gestionnaire d'interruption correspondant.

Ce code permet que si il y a un changement d'état au niveau de la présence secteur, l'état est stocké dans l'EEPROM interne du micro contrôleur.

```

void IOCB5_DefaultInterruptHandler(void) {
    // add your IOCB5 interrupt custom code
    // or set custom function using IOCB5_SetInterruptHandler()

    if (IO_RB5_PreSec_GetValue()) {
        setSecteur(1);
    } else {
        setSecteur(0);
    }
}

```

Création d'une méthode d'initialisation de la carte :

A chaque mise sous tension, on fait une initialisation de la carte suivant ce qu'elle avait en mémoire.

```

void iniCarte() {
    TMRO_StopTimer();

    switch(getEtat()) {
        case PAR:
            setEtat(PAR);
            IO_RB4_RELAIS_SetLow();
            break;
        case VEI:
            setEtat(VEI);
            break;
        case ACQ:
            setEtat(DEC);
            break;
        case DEC:
            setEtat(DEC);
            break;
        default:
            setEtat(PAR);
            IO_RB4_RELAIS_SetLow();
            break;
    }
}

```

Tout d'abord on arrête le timer.

Si l'état est PAR, on remet l'état par et on désactive le relais.

Si l'état est VEI, on remet l'état VEI.

Si l'état est ACQ, on met l'état DEC.

Si l'état est DEC, on remet l'état DEC.

Par défaut, on met l'état PAR et on désactive le relais.

Remettre l'état permet d'allumer la led.

Problème de transfert de donnée :

Suite à un déchargement des données, on s'aperçoit qu'il y a des erreurs dans les données.

54	49	20,75195 C	170	170	170	170	170	170	170	170
55	50	20,75195 C	170	170	170	170	170	170	170	170
56	51	20,75195 C	170	170	170	170	170	170	170	170
57	52	20,75195 C	170	170	170	8	9	10	0	1
58	53	20,75195 C	170	170	170	170	170	170	170	170
59	54	20,75195 C	170	170	170	170	170	170	170	170

Suite aux recherches effectuées, on s'est rendu compte que l'écriture de page de donnée dans la mémoire est limitée : les bits A9 à A18 de l'adresse doivent être identique. De ce fait on a rajouté une méthode permettant d'écrire octet par octet.

```

void writeLotMEM(uint32_t addr, uint8_t *data){
    uint8_t tabw[14]={0x00}; // Initialisation du tableau.

    int i;

    for(i=0;i<TAILLE_LOT;i++) {
        IO_RA5_ChipSelect_SetLow();
        SPI1_Exchange8bit(0x06); //write enabled // Activation de l'écriture dans la mémoire
        IO_RA5_ChipSelect_SetHigh();
        // delay ms(5);
        tabw[0]=0x02; // commande pour
        tabw[1]=(uint8_t)(addr>>16); // adresse
        tabw[2]=(uint8_t)(addr>>8); // adresse
        tabw[3]=(uint8_t)addr; // adresse
        // tabw[(uint8_t)(i+4)]=data[i]; // donnée à écrire dans la mémoire
        tabw[4]=data[i]; // donnée à écrire dans la mémoire
        IO_RA5_ChipSelect_SetLow();
        SPI1_Exchange8bitBuffer(tabw,5,NULL); // env // Envoie des données sur le bus SPI
        IO_RA5_ChipSelect_SetHigh();
        addr++;
        __delay_ms(5);
        //while (getStatusReg() & 0x01); // attente fin écriture
    } // for
} // writeLotMEM

```

Après la modification du programme, les données sont devenues correctes.

54	49	62,25586 C	255	255	255	255	255	255	255	255
55	50	62,25586 C	255	255	255	255	255	255	255	255
56	51	62,25586 C	255	255	255	255	255	255	255	255
57	52	62,25586 C	255	255	255	255	255	255	255	255
58	53	62,25586 C	255	255	255	255	255	255	255	255

Physique :

Présentation du bus SPI :

Le SPI (Serial Peripheral Interface) est un bus de données série synchrone et opère en mode full-duplex. La communication se fait selon un schéma maître-esclaves où le maître contrôle la communication. Il peut y avoir plusieurs esclaves car il y a un fil dédié qui permet de sélectionner l'esclave appelé Slave Select (SS).

Il y a 4 fils au total : le Slave Select, « SS », qui permet de sélectionner l'esclave, le Serial Clock, « SCLK », qui est l'horloge générée par le maître, le Master Output/Slave Input, « MOSI », qui est généré par le maître et le Master Input/Slave Output, « MISO », qui est généré par l'esclave.

Il existe d'autres noms souvent utilisés :

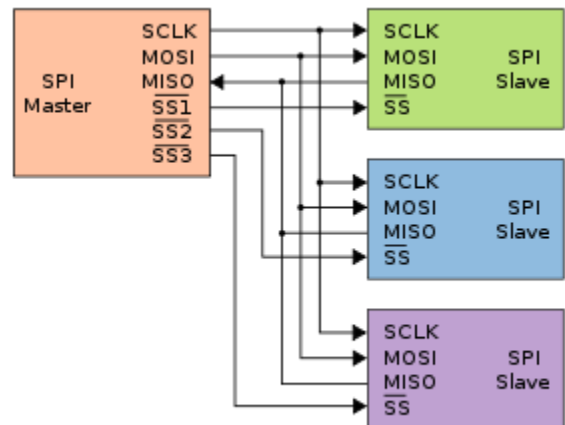
- SCK, SCL Horloge
- SDI, DI, SI Serial Data IN, MISO
- SDO, SDA, DO, SO Serial Data OUT, MOSI
- nCS, CS, nSS, STE, CSN SS

SCLK (serial clock) :
Horloge du bus (produite par le maître)

MOSI (Master Out Slave In) :
Données du maître vers l'esclave actif

MISO (Master In Slave Out) :
Données de l'esclave actif vers le maître

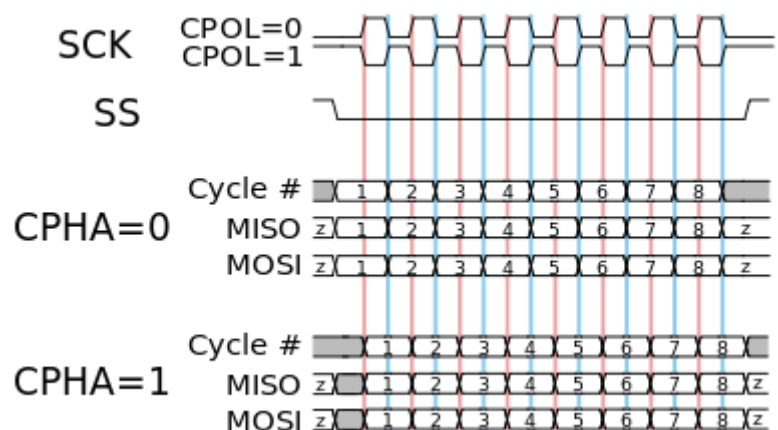
SSn (Slave Select n) :
Sélection de l'esclave n à destination de la transmission



La configuration de la SPI est souvent nommée sous forme de nombre où le bit de poids fort est CPOL (Clock Polarity) et le bit de poids faible CPHA (Clock Phase).

- Mode 0 : L'horloge est à 0 au repos et la donnée est récupérée sur un front montant
- Mode 3 : L'horloge est à 1 au repos et la donnée est récupérée sur un front descendant
- Mode 1 : L'horloge est à 0 au repos et la donnée est récupérée sur un front descendant
- Mode 2 : L'horloge est à 1 au repos et la donnée est récupérée sur un front montant

Mode	CPOL	CPHA
0 (0,0)	0	0
1 (0,1)	0	1
2 (1,0)	1	0
3 (1,1)	1	1



Le bus SPI à plusieurs avantages :

- Il communique en full duplex
- A un débit plus important qu'un bus I²C
- Flexibilité du nombre de bits à transmettre ainsi que du protocole en lui-même
- Simplicité de l'interface matérielle
 - Aucun arbitre nécessaire car aucune collision possible
 - Les esclaves utilisent l'horloge du maître et n'ont donc pas besoin d'oscillateur propre
- Partage d'un bus commun pour l'horloge, MISO et MOSI entre les périphériques

Mais il y a des inconvénients :

- Monopolise plus de broches que l'I²C ou l'UART qui en utilise 2.
- Il n'y a pas d'adressage possible, il faut un fil de sélection par esclave.
- Le protocole n'a pas d'acquittement.
- Le bus ne s'utilise que sur de courtes distances.

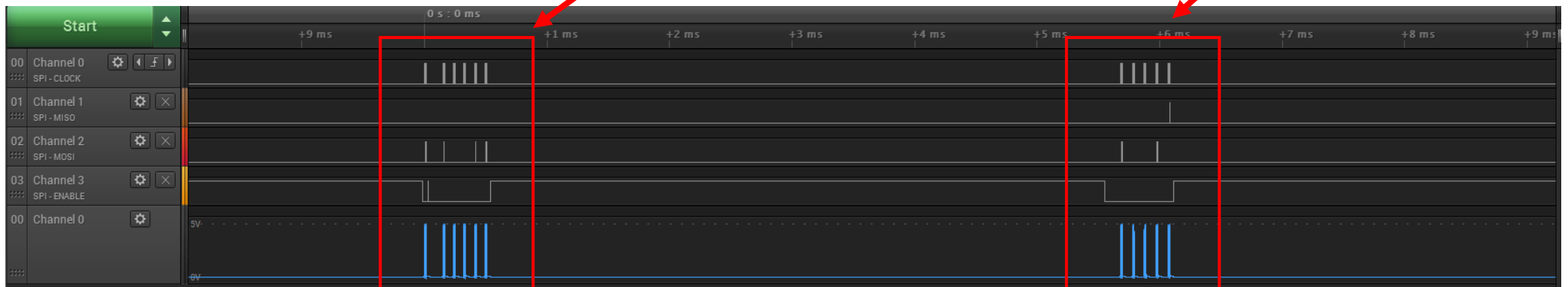
Exemple de frame :

Cette frame correspond à la communication entre le micro contrôleur et la mémoire EEPROM externe (AT25M02).

On peut voir sur cette capture plusieurs frame :

Trame d'écriture

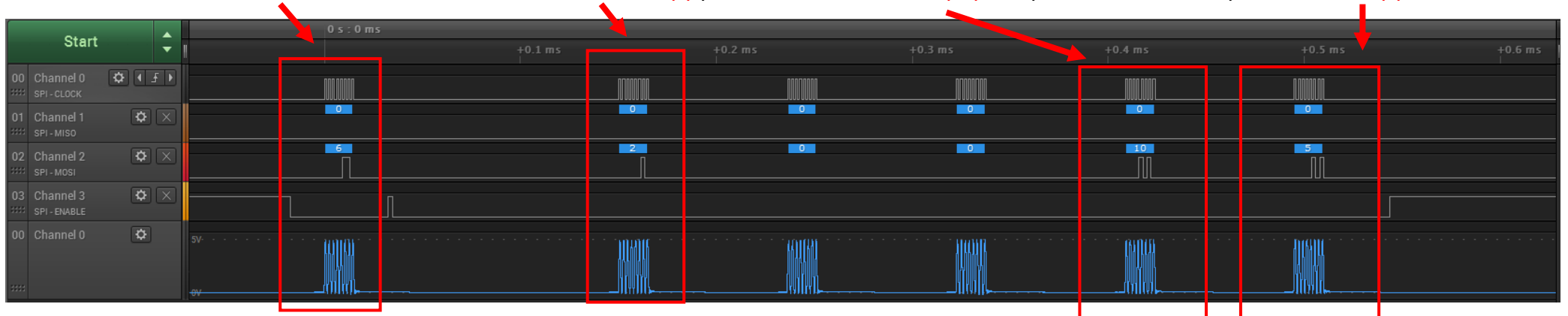
Trame de lecture



La trame d'écriture :

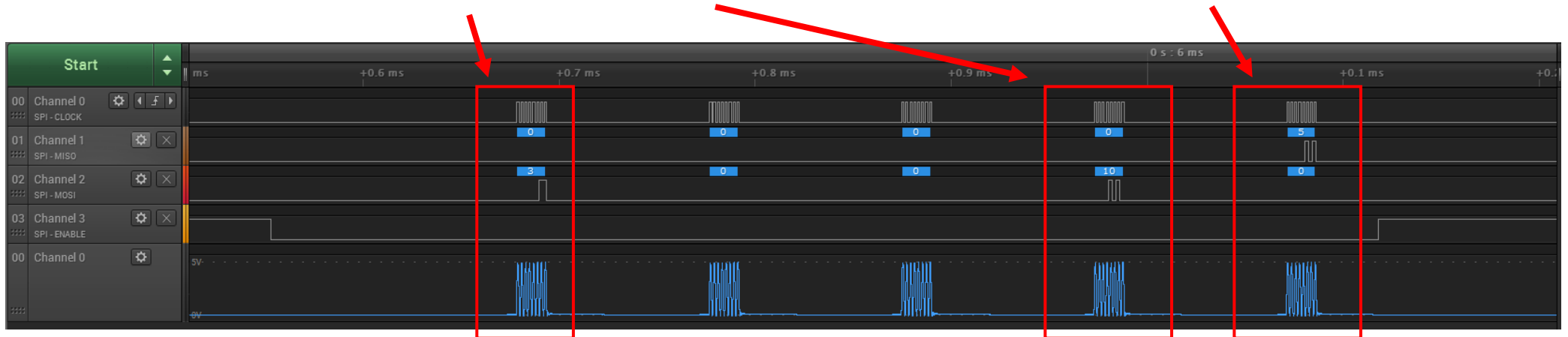
L'écriture est en fait composée de 2 trames. La première permet d'activer l'écriture et la deuxième d'écrire.

Tout d'abord on **active l'écriture** ensuite on envoie **la commande (2)** puis on **envoie l'adresse (10)** à laquelle on veut écrire puis **la donnée (5)**.



La trame de lecture :

Pour la lecture on envoie tout d'abord **la commande (3)** puis **l'adresse (10)**. Suite à cela, la mémoire va renvoyer **la donnée (5)** contenue dans l'adresse.



Partie personnelle Dimier Benjamin

I. Présentation de l'entreprise.

Étudiant en deuxième année en BTS Systèmes Numériques, j'ai été affecté à la réalisation du projet SALIA pour le compte de l'entreprise 2SP ELECTRONIC.

Dans ce projet, je travaille en collaboration avec Alois BRUCCOLERI et Corentin HERY en IR ainsi que Yanis FOUNAS et Nicolas CHARRIER en EC.



Il s'agit pour ce projet de concevoir un matériel facilement transportable qui permettra de réaliser une acquisition cyclique de tensions analogiques, d'une température et enfin de stocker ces informations en mémoire après numérisation. En deuxième lieu, lors de la connexion de cet outil sur le port USB d'un ordinateur, les données stockées devront être récupérées pour ensuite être sauveées et exploitées dans un logiciel tableur.

Le logiciel PC permettra de configurer l'appareil pour l'acquisition lors d'une première connexion USB.

Le but étant de pouvoir apporter une solution de laboratoire comme matériel d'acquisition autonome et possédant une mémoire.

II. Présentation de mes tâches.

Dans ma partie, je dois penser et développer une application visant à exploiter les entrées

Étudiant 3 IR 3	<i>Liste des tâches assurées par l'étudiant</i> Assurer le développement du logiciel PC Linux permettant : - Le paramétrage de la carte SAILA, - La sauvegarde du paramétrage dans une base de données et dans un fichier CSV. - Le rapatriement des données acquises et leur sauvegarde dans une base de données et dans un fichier CSV. - La communication avec la carte SAILA (en collaboration avec l'étudiant IR2).	Installation : Système d'exploitation Linux , EDI, driver USB. Mise en œuvre : Communication USB, BDD, microcontrôleur. Configuration : EDI, Réalisation : Logiciel Linux de gestion de la carte SAILA. Documentation : Manuel d'installation, de fonctionnement.
-------------------------------	---	---

analogiques, les capteurs de la carte SAILA et de pouvoir la paramétrer entièrement.

Lors d'une dernière discussion avec le client, la solution de stockage avec une BDD à été écartée.

Diagrammes de Gantt.

13		Activités IR1 et IR3	32 h	Ven 11/01/19
14		Spécification des informations à paramétrer	2 h	Ven 11/01/19
15		Spécification du protocole de communication entre la carte SAILA et le PC	4 h	Mer 16/01/19
16		Spécification du format du fichier CSV	1 h	Mer 23/01/19
17		Tâche de mise en oeuvre de la liaison série	6 h	Mer 23/01/19
18		Prototype de l'interface graphique	6 h	Jeu 24/01/19
19		Mise en oeuvre de la génération d'un fichier CSV	6 h	Mer 30/01/19
20		Analyse générale Sysml	7 h	Jeu 31/01/19
21		Revue 1	0 h	Mer 06/02/19

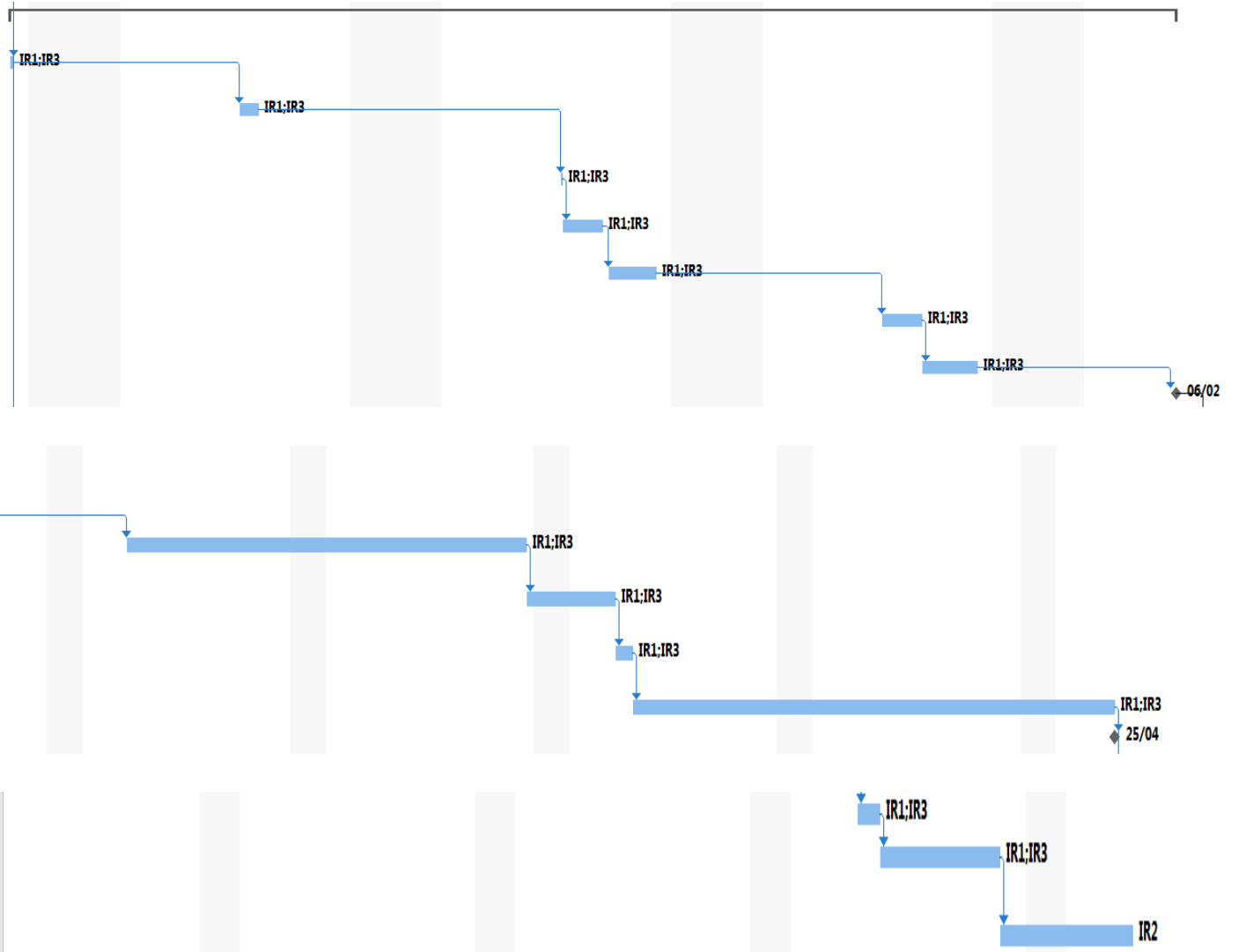
Revue 1.

52		Gérer le paramétrage de la carte SAILA via l'application
53		Analyse SysML
54		Codage du protocole de communication
55		Ajout de la fonctionnalité sauver mission
56		Ajout de la fonctionnalité charger mission
57		Intégration de la carte SAILA
58		Revue 2

Revue 2.

78		Mise à jour du dossier	6 h	Jeu 25/04/19
79		Manuel d'installation et de mise en oeuvre	6 h	Ven 26/04/19
80		Notice de maintenance	6 h	Jeu 02/05/19

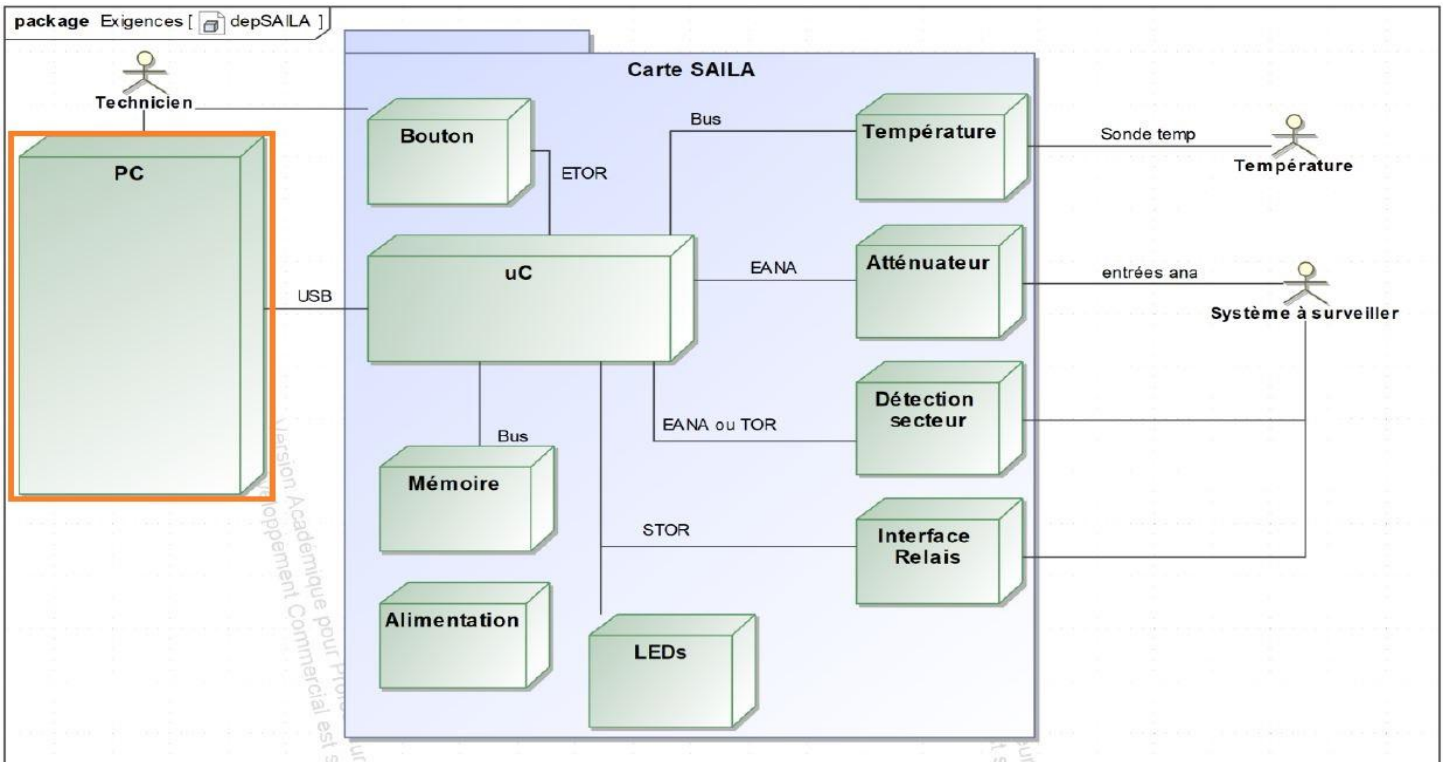
Revue 3



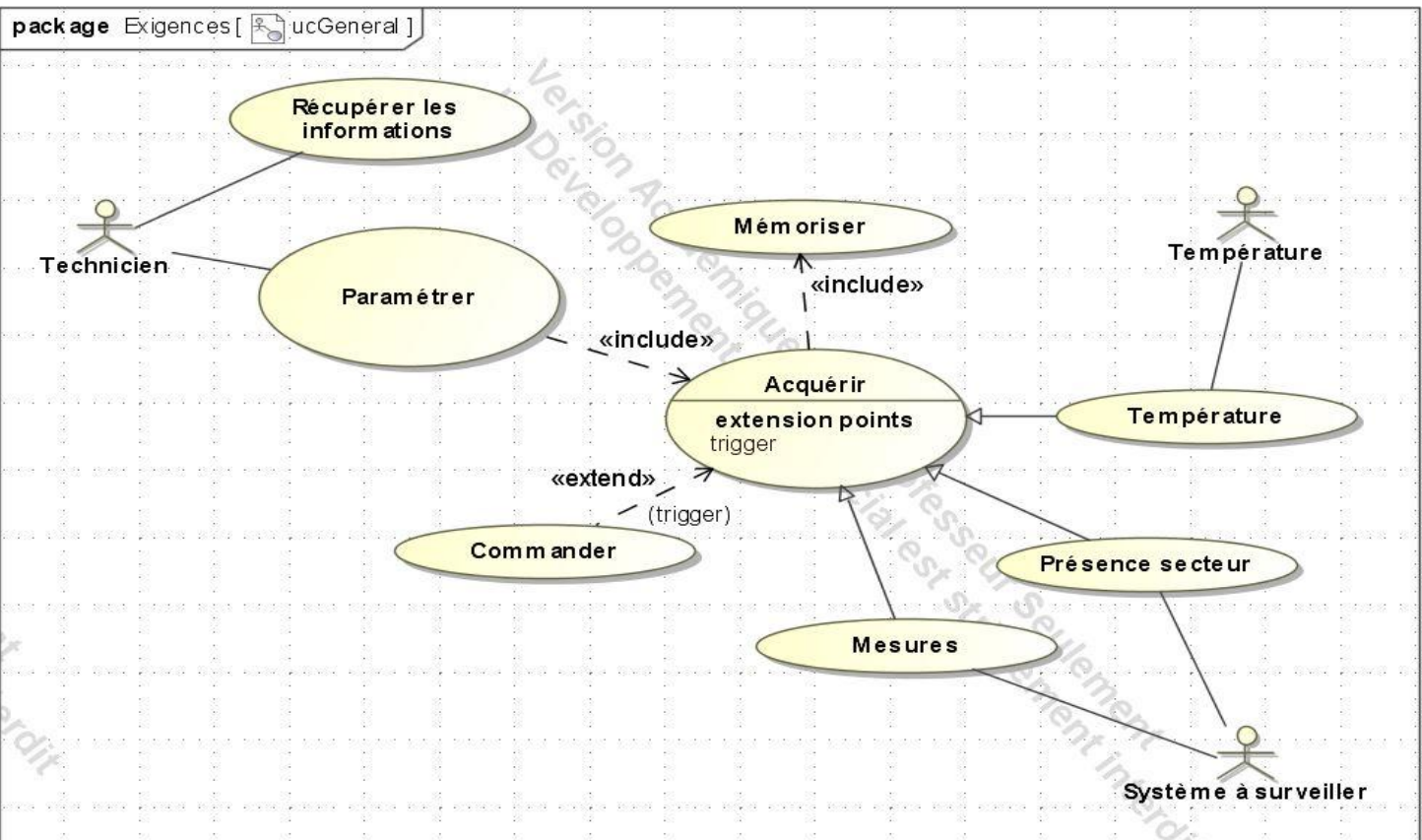
Plan de développement.

N° itération	Activités	Justification
1	Mise en œuvre de tâche de plusieurs spécifications : 1. Définition du protocole de communication carte SAILA/PC. 2. Format du fichier CSV. 3. La liaison série avec Qt. 4. Prototype graphique de l'application SAILA. 5. Analyse SysML.	Apprentissage de l'utilisation d'un fichier CSV, définition du protocole de communication, activité sur un TP de liaison série pour préparation à la communication série et première aperçu et constitution d'un prototype de l'application graphique Qt ainsi qu'une simple analyse SysML.
2	Développement de l'application SAILA : 1. Gérer le paramétrage de la carte SAILA. 2. Codage du protocole de communication. 3. Ajout des fonctionnalités "sauver mission" et "charger mission". 4. Intégration de la carte SAILA.	Avoir un aperçu et constitution de l'interface graphique de l'application SAILA.
3	Finalisation du projet : 1. Création des notices et des manuels d'utilisation. 2. Intégration de toutes les parties du projet.	Aider le client à utiliser de manière claire et simple le matériel et l'application et présentation de l'ensemble du produit.

III. Présentation des Diagrammes.



Comme exposé en orange, ma partie consiste à gérer l'échange des entrées analogiques, le capteur de température, la mémoire et l'interface relais entre le PC et la carte via l'application.



Mon travail dans ce diagramme consiste à gérer et paramétrer l'application qui va récupérer les mesures de températures, et les enregistrer.

IV. ACTIVITÉS

Réalisé pour la revue 1.

Spécification des informations à paramétrer sur l'interface graphique

Ce protocole présente la communication entre le PC et la carte SAILA par une connexion USB.

HERY Corentin et DIMIER Benjamin

15/01/2019

Mis à jour le 05/04/2019

- Paramétrage du bouton poussoir.
- Gérer le paramétrage des données.
 - Temps entre chaque acquisition en seconde (amélioration en hh/mm/ss).
 - Durée totale d'acquisition en seconde (amélioration en hh/mm/ss).
 - Suppression/Reset des données auparavant stockées.
 - Paramétrage des sorties analogiques à utiliser.
- Gérer la récupération des données pour les mettre dans un fichier CSV.
- Le port série.
- Paramétrage du relais en 0 à 255 (Si on le met en place ou pas. Si oui quand le déclencher (si on dépasse le seuil mais si on passe en dessous aussi)).
- Les 9 entrées analogiques dont le capteur de température et une surveillance secteur.
- Paramétrer le nom de la mission.
- **Ajout d'un oscillateur avec une utilisation interne et externe suivant le degré de précision voulu.**
- Charger un fichier CSV existant.

PROTOCOLE SAILA

Ce protocole présente la communication entre le PC et la carte SAILA par une connexion USB.

HERY Corentin et DIMIER Benjamin

16/01/2019

Ce protocole permet la communication point à point entre un PC et la carte d'acquisition SAILA par une connexion USB. Il consiste en un échange de trames à l'initiative du maître et l'esclave acquitte la bonne réception des informations par une trame de réponse. Le protocole est dit maître/esclave, le maître étant le PC, l'esclave la carte SAILA.

Format des commandes

Source	Trame	Désignation	Etat
Maître	:Z	Reset de la carte SAILA. Retour à l'état PAR.	*
Esclave	Etat	Etat est renvoyé au format ASCII ('1' ou '2' ou '3' ou '4').	
Maître	:E	Demande de l'état de la carte SAILA.	*
Esclave	Etat	Etat est renvoyé au format ASCII ('1' ou '2' ou '3' ou '4').	
Maître	:P<durée16 bits><inter16 bits>	Paramétrage de la durée (s) d'acquisition et de l'intervalle entre 2 acquisitions (s).	PAR
Esclave	0		
Maître	:F	Fin du paramétrage. SAILA passe à l'état VEI, attente de départ d'acquisition (bouton poussoir).	PAR
Esclave	Etat	Etat est renvoyé au format ASCII ('1' ou '2' ou '3' ou '4').	
Maître	:D	Commande de déchargement des mesures dans le PC.	DEC
Esclave	<durée16bits><inter16bits> <nbLotMesures32bits> <ParRelais24bits> Pour chaque lot mémorisé : <noLot32bits> <mesure0_8bits>...<mesureN_8bits> <secteur&relais8bits> Après tous les lots : /	Rappel de la configuration enregistrée dans SAILA. Envoi du nombre de lots de mesures effectué en réalité. Chaque lot est précédé de son numéro d'ordre. La fin de la transmission est repérée par le caractère /. Le bit b0 de l'octet <secteur&relais8bits> correspond au bit de l'état secteur. Le bit b1 de l'octet <secteur&relais8bits> correspond au bit de l'état du relais.	
Maître	:C<11XXXXXX><16bits à X> :C<10XXXXXX><temps16bits> :C<01XXXXMM><consigne8bits> <0000000S>	Déclenchement du relais sur la surveillance tension. Déclenchement du relais après un temps défini (sec) sur 16 bits. Déclenchement du relais sur une valeur de mesure. MMMM : Numéro de la mesure (0-8). La consigne est la valeur du seuil de la mesure. S : Bit indiquant le sens montant (S=1) ou descendant (S=0).	PAR

Esclave	0		
Maître	:0 I E	Choix de l'oscillateur : Externe 'E' ou Interne 'I'.	PAR
Esclave	0		

Pour les mots de 16 bits, le poids fort est émis en premier.

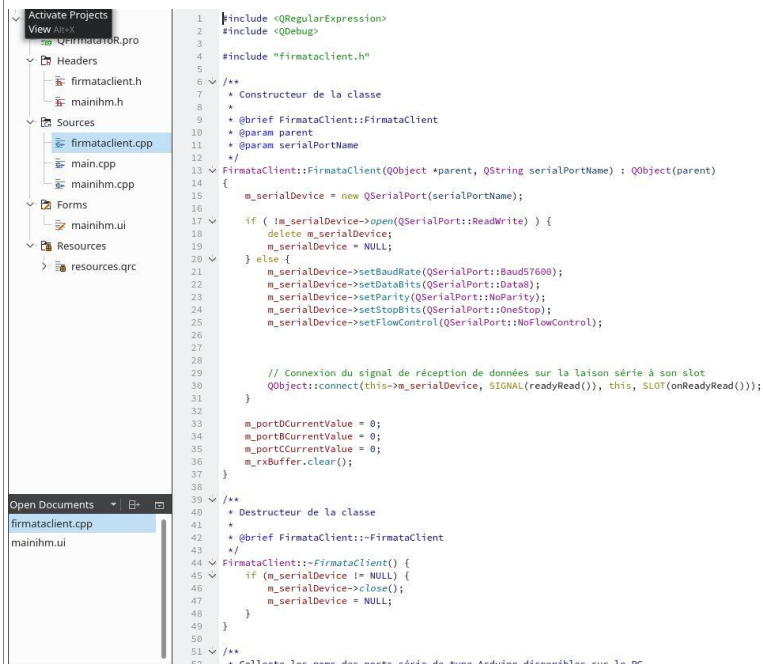
Spécification du fichier CSV.

Réalisé par Hery Corentin et DIMIER Benjamin pour la revue 1.

Mis à jour le 10/052019.

1	Nom de la mission :												
2	Duree total :												
3	Intervalle :												
4	Nombre de lot :												
5	Surveillance présence secteur/Non surveillance présence secteur												
6	Relais non activé / Paramétrage relais activé :												
7	Après une durée de X secondes												
8	Mesure Ana1/.../Ana8/Temp seuil : ex:126 M/D												
9	Coupure secteur												
10	Lot id :	1ère sortie analogique	2ème sortie analogique	3ème sortie analogique	4ème sortie analogique	5ème sortie analogique	6ème sortie analogique	7ème sortie analogique	8ème sortie analogique	Capteur temp	Présence secteur	Déclenchement relais	
11	1												
12	2												
13	3												
14	4												
15	5												
16	6												
17	7												
18	8												
19	9												
20	10												
21	11												
22	12												
23	13												
24	14												
25	15												
26	16												
27	17												
28	18												
29	19												
30													

Mise en œuvre d'une liaison série réalisée pour la revue 1.

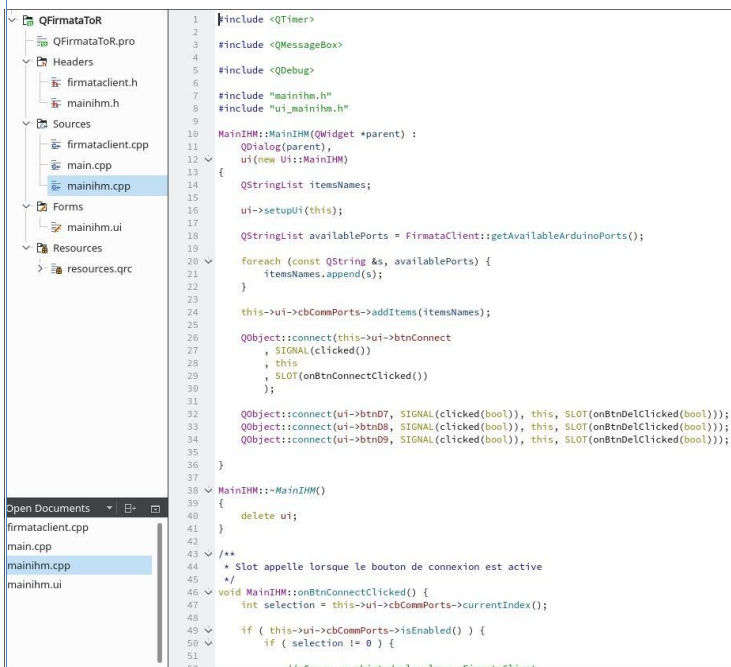


```

1 #include <QRegularExpression>
2 #include <QDebug>
3
4 #include "firmataclient.h"
5
6 /**
7  * Constructeur de la classe
8  *
9  * @brief FirmataClient::FirmataClient
10  * @param parent
11  * @param serialPortName
12  */
13 FirmataClient::FirmataClient(QObject *parent, QString serialPortName) : QObject(parent)
14 {
15     m_serialDevice = new QSerialPort(serialPortName);
16
17     if ( !m_serialDevice->open(QSerialPort::ReadWrite) ) {
18         delete m_serialDevice;
19         m_serialDevice = NULL;
20     } else {
21         m_serialDevice->setBaudRate(QSerialPort::Baud57600);
22         m_serialDevice->setDataBits(QSerialPort::Data8);
23         m_serialDevice->setParity(QSerialPort::NoParity);
24         m_serialDevice->setStopBits(QSerialPort::OneStop);
25         m_serialDevice->setFlowControl(QSerialPort::NoFlowControl);
26
27         // Connexion du signal de réception de données sur la liaison série à son slot
28         QObject::connect(this->m_serialDevice, SIGNAL(readyRead()), this, SLOT(onReadyRead()));
29     }
30
31     m_portDCurrentValue = 0;
32     m_portBCurrentValue = 0;
33     m_portCCurrentValue = 0;
34     m_rxBuffer.clear();
35 }
36
37 /**
38  * Destructeur de la classe
39  *
40  * @brief FirmataClient::~FirmataClient
41  */
42 FirmataClient::~FirmataClient() {
43     if (m_serialDevice != NULL) {
44         m_serialDevice->close();
45         m_serialDevice = NULL;
46     }
47 }
48
49 // Collecte les noms des ports série de tous les périphériques sur le PC

```

Code source du firmataclient.cpp qui définit les spécificités de la liaison série.



```

1 #include <QTimer>
2 #include <QMessageBox>
3 #include <QDebug>
4 #include "mainihm.h"
5 #include "ui_mainihm.h"
6
7 MainIHM::MainIHM(QWidget *parent) :
8     QDialog(parent),
9     ui(new Ui::MainIHM)
10 {
11     QStringList itemsNames;
12     ui->setupUi(this);
13     QStringList availablePorts = FirmataClient::getAvailableArduinoPorts();
14     foreach (const QString &s, availablePorts) {
15         itemsNames.append(s);
16     }
17     this->ui->comboBoxPorts->addItems(itemsNames);
18     QObject::connect(this->ui->btnConnect,
19         SIGNAL(clicked()),
20         this,
21         SLOT(onBtnConnectClicked()));
22
23     QObject::connect(ui->btnD7, SIGNAL(clicked(bool)), this, SLOT(onBtnDeClicked(bool)));
24     QObject::connect(ui->btnD8, SIGNAL(clicked(bool)), this, SLOT(onBtnDeClicked(bool)));
25     QObject::connect(ui->btnD9, SIGNAL(clicked(bool)), this, SLOT(onBtnDeClicked(bool)));
26 }
27
28 void MainIHM::MainIHM()
29 {
30     delete ui;
31 }
32
33 /**
34  * Slot appelle lorsque le bouton de connexion est active
35  */
36 void MainIHM::onBtnConnectClicked() {
37     int selection = this->ui->comboBoxPorts->currentIndex();
38     if ( this->ui->comboBoxPorts->isEnabled() ) {
39         if ( selection != 0 ) {
40             // Créer un objet de la classe FirmataClient

```

Code source du mainihm.cpp qui définit les informations et les actions des boutons sur la partie graphique de l'application.



Sur l'application graphique lancée, on sélectionne le port de l'arduino.



Ensuite la carte est connectée et lorsqu'on clique sur les LEDs rouge, verte et orange, elles s'allument.

Prototype graphique de l'application SAILA :

Interface de programmation SAILA

Sélectionner le port : Connexion à la carte

Mission

Nom de la mission :

Sauver la mission Charger mission

Réinitialiser carte

RESET



Paramétrage des entrées analogiques et d'une mission

Entrée 1 :	☐ Activer
Entrée 2 :	☐ Activer
Entrée 3 :	☐ Activer
Entrée 4 :	☐ Activer
Entrée 5 :	☐ Activer
Entrée 6 :	☐ Activer
Entrée 7 :	☐ Activer
Entrée 8 :	☐ Activer
Entrée 9 :	☐ Activer

Capteur de température : ☐ Activer

Réinitialiser les paramètres

Paramètres d'acquisition

Définir la durée de l'acquisition : 00:00

Intervalle d'acquisition : 0 Secondes

☐ Activer Paramètres du relais et seuil

En seconde : ☐ 00:00

Seuil mesures : ☐ ☐ Montée ☐ Descente

Surveillance secteur : ☐

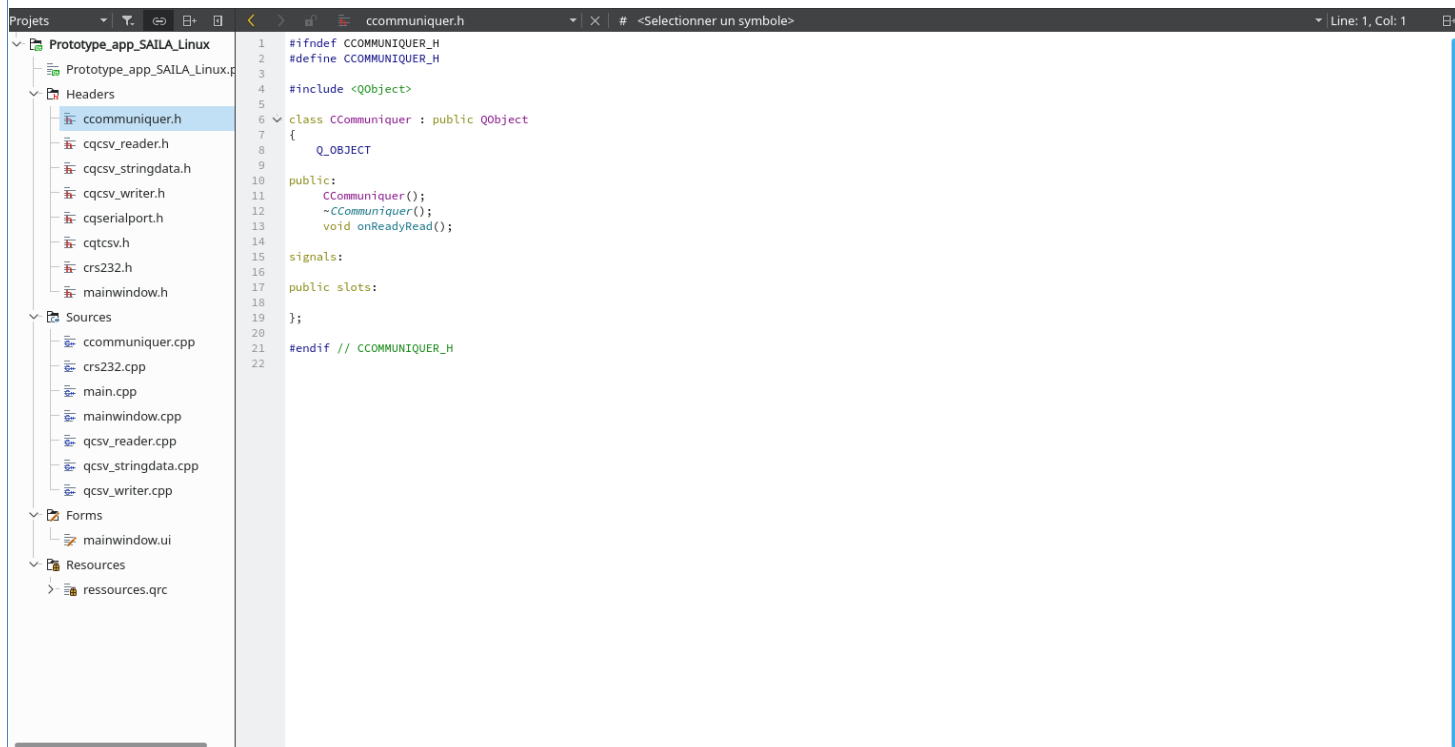
Valider paramètres

Application graphique : à rajouter dans la prochaine version la commande de l'oscillateur.

```

Projets  ccommuniquer.cpp  Line: 18, Col: 1
1 #include "ccommuniquer.h"
2 #include <QDebug>
3
4
5 CCommunique::CCommunique()
6 {
7
8     void CCommunique::onReadyRead()
9     {
10         QByteArray car;
11         int nb = mPs->bytesAvailable();
12         for(int i=0 ; i<nb ; i++)
13         {
14             car = mPs->read(1); // exemple de lecture d'un caractère
15             qDebug() << "CCommunique::onReadyRead: Lecture d'un caractère : " << car;
16         }
17     }
18 }
  
```

Code de ccommuniquer.cpp qui gère la communication de la liaison série.

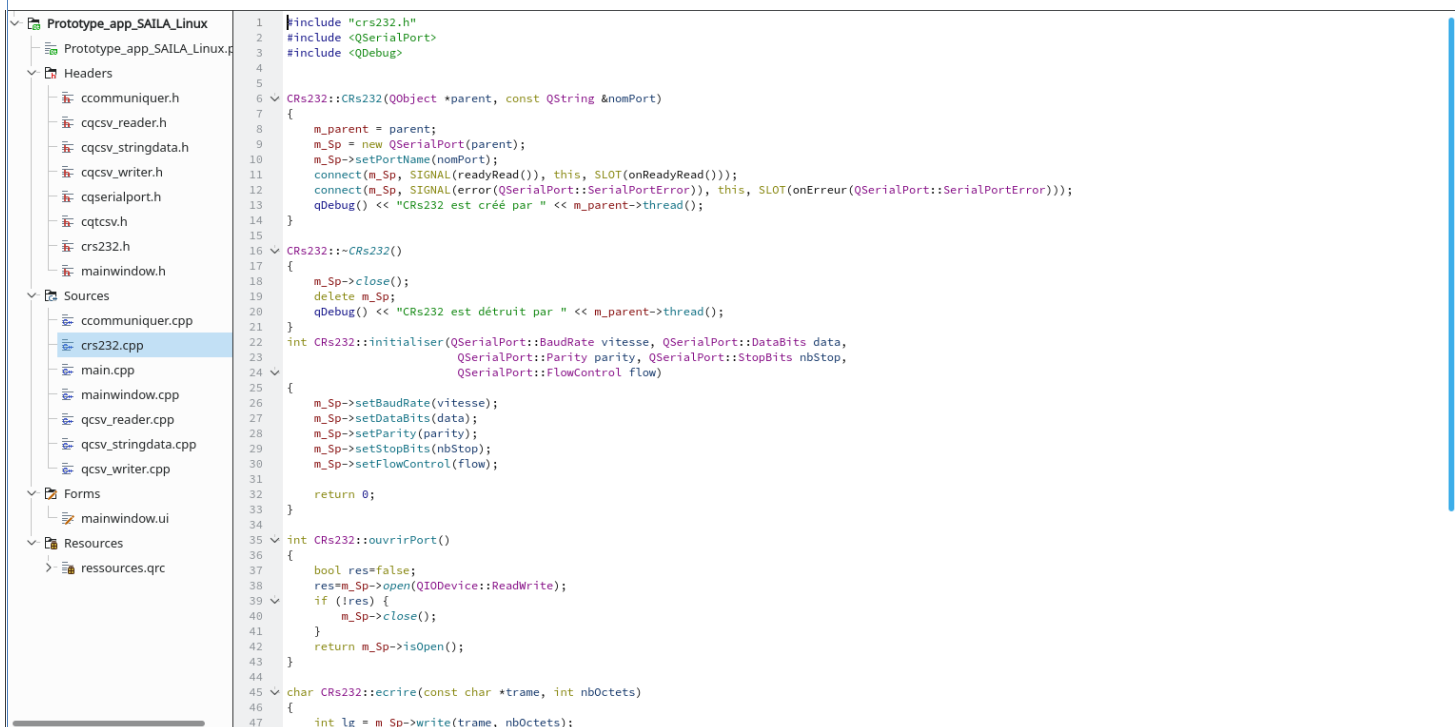


```

1  #ifndef CCOMMUNIQUE_H
2  #define CCOMMUNIQUE_H
3
4  #include <QObject>
5
6  class CCommuniquer : public QObject
7  {
8      Q_OBJECT
9
10     public:
11         CCommuniquer();
12         ~CCommuniquer();
13         void onReadyRead();
14
15     signals:
16
17     public slots:
18
19 };
20
21 #endif // CCOMMUNIQUE_H
22

```

Classe de ccommuniquer.h qui définit les attributs de la classe.



```

1  #include "crs232.h"
2  #include <QSerialPort>
3  #include <QDebug>
4
5  CRs232::CRs232(QObject *parent, const QString &nomPort)
6  {
7      m_parent = parent;
8      m_Sp = new QSerialPort(parent);
9      m_Sp->setPortName(nomPort);
10     connect(m_Sp, SIGNAL(readyRead()), this, SLOT(onReadyRead()));
11     connect(m_Sp, SIGNAL(error(QSerialPort::SerialPortError)), this, SLOT(onErreur(QSerialPort::SerialPortError)));
12     qDebug() << "CRs232 est créé par " << m_parent->thread();
13 }
14
15 CRs232::~CRs232()
16 {
17     m_Sp->close();
18     delete m_Sp;
19     qDebug() << "CRs232 est détruit par " << m_parent->thread();
20 }
21
22 int CRs232::initialiser(QSerialPort::BaudRate vitesse, QSerialPort::DataBits data,
23                       QSerialPort::Parity parity, QSerialPort::StopBits nbStop,
24                       QSerialPort::FlowControl flow)
25 {
26     m_Sp->setBaudRate(vitesse);
27     m_Sp->setDataBits(data);
28     m_Sp->setParity(parity);
29     m_Sp->setStopBits(nbStop);
30     m_Sp->setFlowControl(flow);
31
32     return 0;
33 }
34
35 int CRs232::ouvrirPort()
36 {
37     bool res=false;
38     res=m_Sp->open(QIODevice::ReadWrite);
39     if (!res) {
40         m_Sp->close();
41     }
42     return m_Sp->isOpen();
43 }
44
45 char CRs232::ecrire(const char *trame, int nbOctets)
46 {
47     int lg = m_Sp->write(trame, nbOctets);
48 }
49

```

Code du crs232.cpp qui utilise sa classe crs232.h pour initialiser les données physiques de la liaison série et permet de configurer son port.

V. Journal de bord.

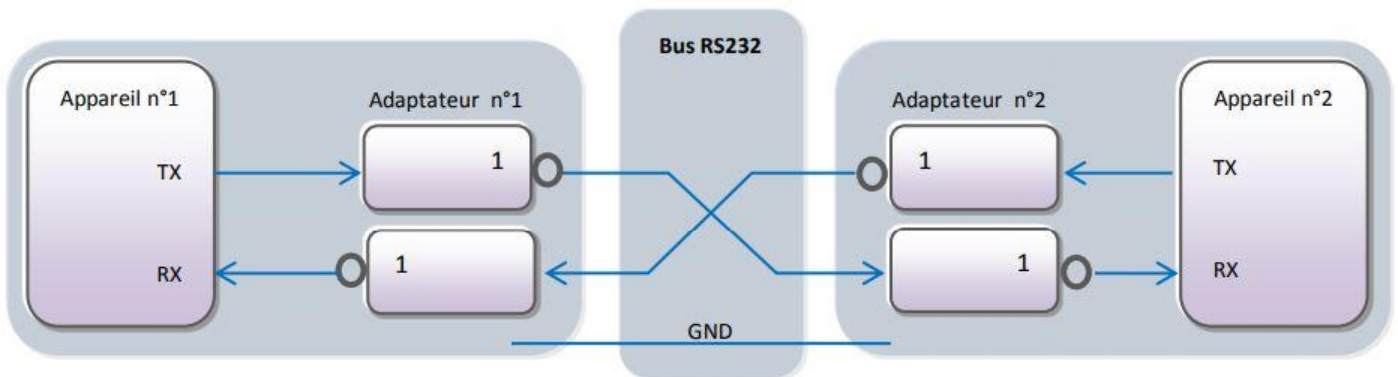
<i>Date</i>	<i>Intervenant</i>	<i>Objet</i>
16/01/2019	• DIMIER Benjamin • HERY Corentin	• Élaboration de la fiche des spécifications de l'interface graphique à réaliser.
16/01/2019	• DIMIER Benjamin • HERY Corentin	• Élaboration du protocole SAILA.
• 10/03/2019	• DIMIER Benjamin	• Ajout de l'oscillateur suivant les préférences du client.

Partie Physique

La liaison série RS-232

Il s'agit d'une liaison simple à mettre œuvre, utilisable sur la quasi-totalité des machines jusqu'au milieu des années 2000. Habituellement désignée « port série », elle a rendu possible les échanges du PC avec quasiment tous ses périphériques mais est supplantée à présent par la liaison USB, nettement plus rapide.

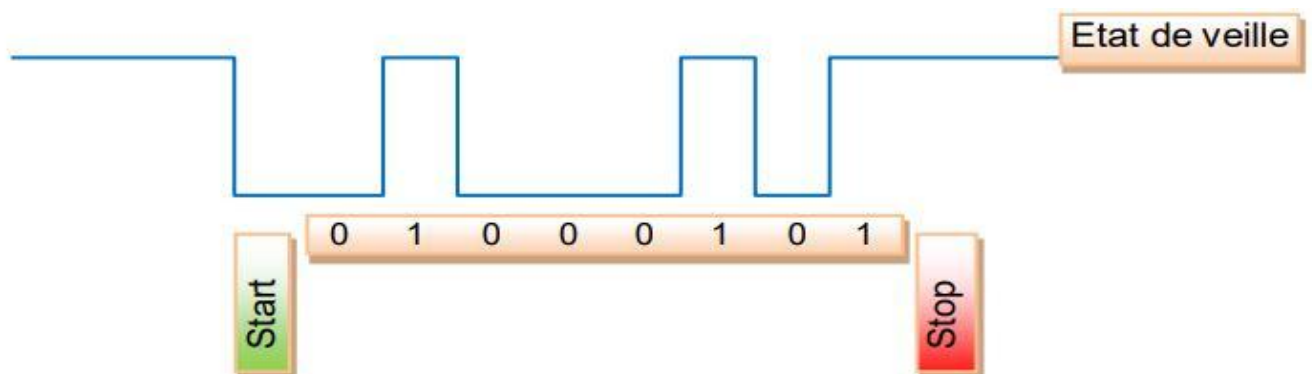
Caractéristiques :



- Liaison série numérique asynchrone : Les bits d'information sont transmis en chaîne les uns après les autres et il n'y a pas d'horloge.
- C'est un bus série qui ne peut relier que deux appareils.
- Il est composé de trois fils au minimum. Un dialogue est créé, lorsque l'un parle, l'autre écoute et inversement. Le full duplex est supporté.
- Les fils du bus série sont blindés ce qui améliore la distance d'une dizaine de mètres.
- Il possède un niveau bas qui correspond à un potentiel compris entre +5V et +30V alors que lorsqu'il est au niveau haut, le potentiel est compris entre -5V et -30V.
- Amélioration de l'immunité aux parasites et accroissent ainsi la distance de transmission grâce au niveau de tension.

Protocole de transmission :

Dans le protocole de la liaison série, un bit de départ au niveau bas et transmis. Ensuite, le bit de poids faible de l'octet transmis est envoyé en premier lieu. Pour l'arrêt de la communication, un bit d'arrêt est transmis au niveau haut. Enfin la liaison est en « état de veille » jusqu'à une durée non déterminée.



La rapidité ou débit de propagation s'exprime en Bauds ou bits par seconde. Nous pouvons choisir une vitesse d'autant plus haute que la distance de la liaison est courte. On dispose de plusieurs vitesses : 2400, 4800, 9600 et 19200 bauds.

Cependant, on peut configurer d'une autre manière le bus : avec une transmission à 7 bits à la place de l'octet avec 2 bits d'arrêt et un bit de parité pour la correction d'erreurs.

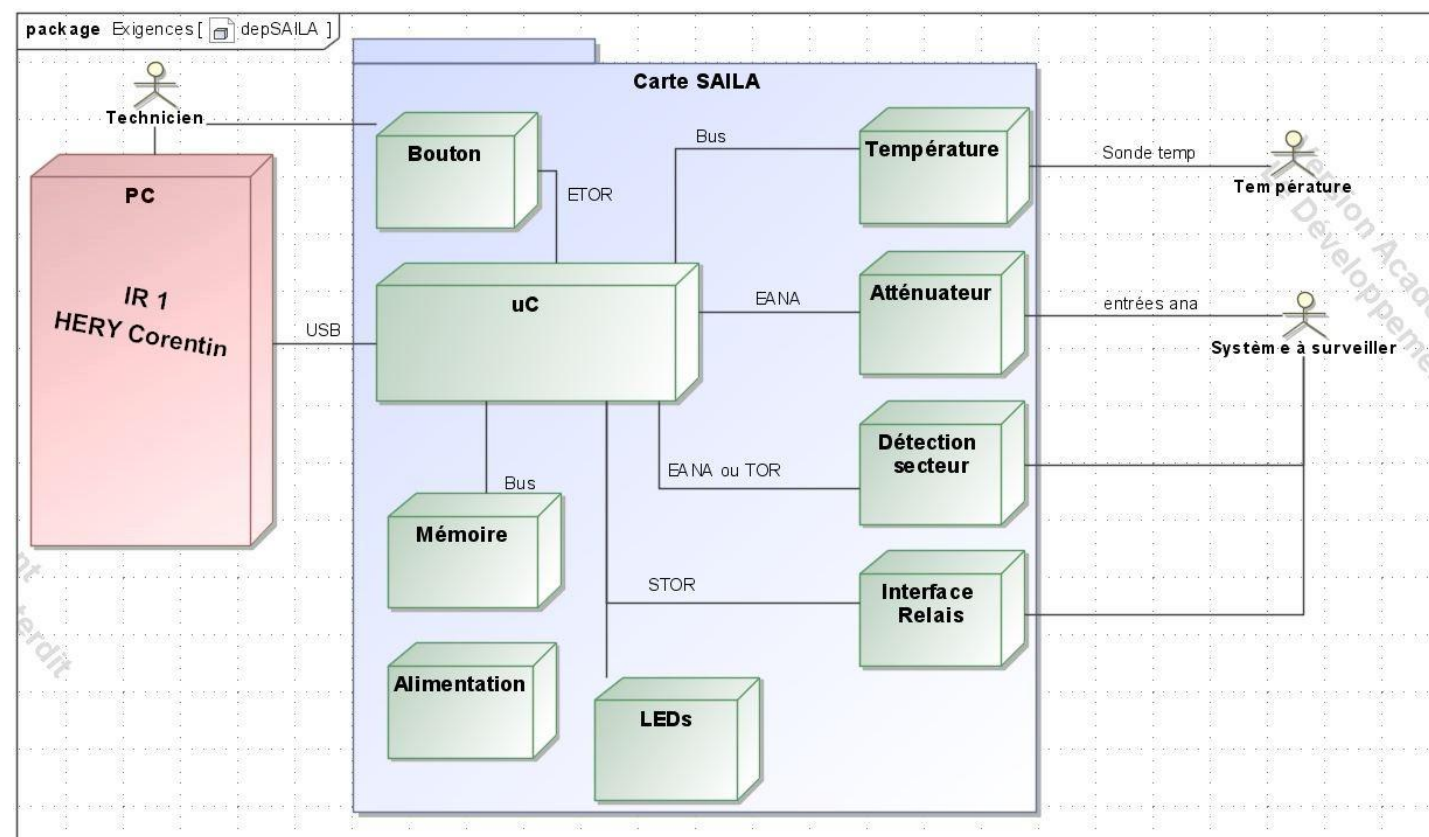
A la synchronisation du récepteur à l'émetteur, un bit de départ de d'arrêt sont nécessaires Les équipements doivent se mettre d'accord également sur la même vitesse et avoir une configuration identique pour pouvoir dialoguer.

Un contrôle de validité par le bit de parité est possible. Ce dernier paramétré par l'expéditeur, est dédié et utilisé comme renseignement par le récepteur sur la validité ou pas du mot binaire reçu. Concernant le paramétrage en parité paire, ce bit passe au niveau haut pour que la quantité complète de bits transmis soit pair. Donc, si trois bits sont hauts pendant que celui de parité est haut alors il y a eu parasitage et le récepteur pourra exiger à l'expéditeur de renvoyer la trame. La protection reste cependant sommaire car, elle ne permet seulement une correction concernant le parasitage d'un seul bit. En effet, une deuxième erreur ne serait pas visible.

Partie personnelle Hery Corentin

Présentation de la partie IR1

J'ai pour objectif de réaliser, dans ce projet, une interface graphique réalisée en C# sous Windows à l'aide de Visual Studio pour gérer le paramétrage de la carte SAILA, mais aussi, d'effectuer le rapatriement des données acquises dans un fichier CSV et la communication avec la carte SAILA (en collaboration avec l'étudiant IR2).



Plan des itérations :

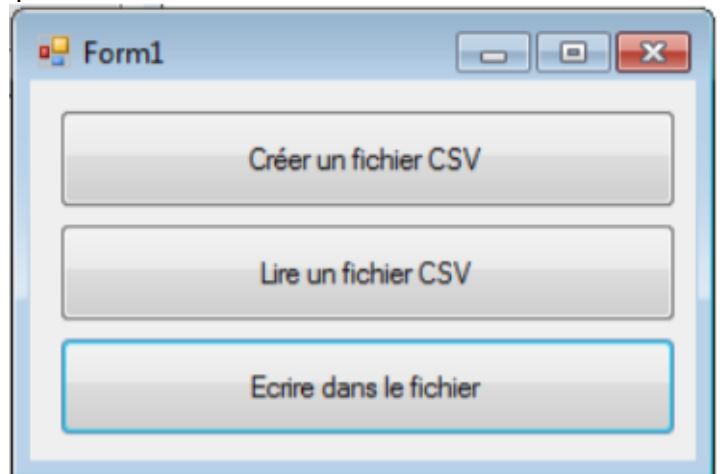
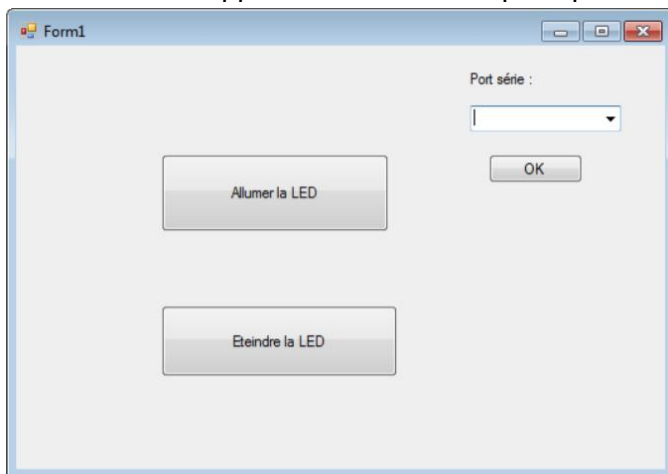
N° itération	Activités	Justification
1	Mise en œuvre de tâche de plusieurs spécifications : 1. Définition du protocole de communication carte SAILA/PC. 2. Format du fichier CSV. 3. La liaison série avec Qt. 4. Prototype graphique de l'application SAILA. 5. Analyse SysML.	Apprentissage de l'utilisation d'un fichier CSV, définition du protocole de communication, activité sur un TP de liaison série pour préparation à la communication série et premier aperçu et constitution d'un prototype de l'application Visual Studio Code ainsi qu'une simple analyse SysML.

2	Développement de l'application SAILA : 1. Gérer le paramétrage de la carte SAILA. 2. Codage du protocole de communication. 3. Ajout des fonctionnalités "sauver mission" et "charger mission". 4. Intégration de la carte SAILA.	Avoir un aperçu et constitution de l'interface graphique de l'application SAILA.
3	Finalisation du projet : 1. Création des notices et des manuels d'utilisation. 2. Intégration de toutes les parties du projet.	Aider le client à utiliser de manière claire et simple le matériel et l'application et présentation de l'ensemble du produit.

Travail effectué :

Pour réaliser mon projet, j'ai dû effectuer différentes tâches de mise en œuvre pour m'approprier l'utilisation de la liaison série, visual studio code, le C#, la création d'un fichier CSV...

J'ai réalisé des applications toutes simples, par exemple :



Ce qui m'a permis de m'exercer à réaliser des IHM avec Visual Studio pour allumer une LED avec le protocole Firmata et créer un fichier CSV.

Une fois que j'ai compris l'utilisation des différentes fonctionnalités, j'ai pu commencer à réaliser des prototypes d'IHM.

Pour pouvoir simuler la carte SAILA, j'ai utilisé un deuxième PC avec le logiciel Realterm qui va me permettre de lire ce que je vais envoyer dans la liaison série et écrire aussi dans celle-ci. J'ai connecté les 2 PC à l'aide d'une liaison série, et j'ai pu tester l'envoi des trames. J'ai créé un protocole communication à partir d'une bonne base, donnée par notre professeur.

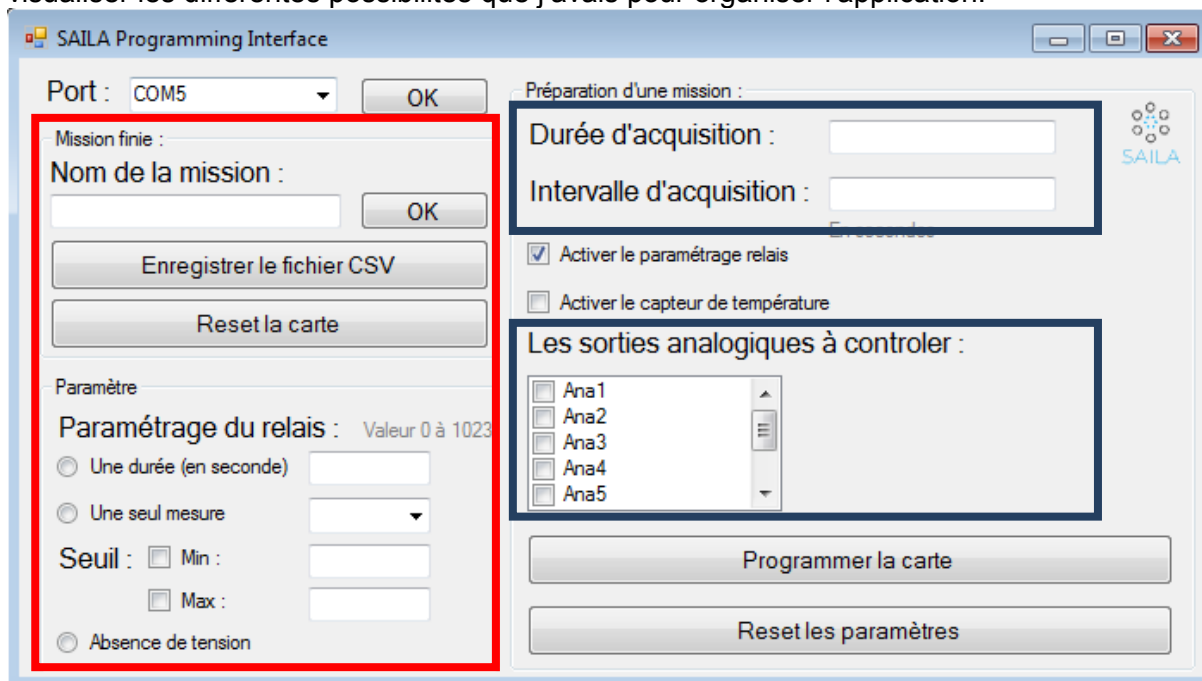
Format des commandes

Source	Trame	Désignation	Etat
Maître	:Z	Reset de la carte SAILA. Retour à l'état PAR.	*
Esclave	Etat	Etat est renvoyé au format ASCII ('1' ou '2' ou '3' ou '4').	
Maître	:E	Demande de l'état de la carte SAILA.	*
Esclave	Etat	Etat est renvoyé au format ASCII ('1' ou '2' ou '3' ou '4').	
Maître	:P<durée16 bits><inter16 bits>	Paramétrage de la durée (s) d'acquisition et de l'intervalle entre 2 acquisitions (s).	PAR
Esclave	O		
Maître	:F	Fin du paramétrage. SAILA passe à l'état VEI, attente de départ d'acquisition (bouton poussoir).	PAR
Esclave	Etat	Etat est renvoyé au format ASCII ('1' ou '2' ou '3' ou '4').	
Maître	:D	Commande de déchargement des mesures dans le PC.	DEC
Esclave	<durée16bits><inter16bits> <nbLotMesures32bits> <ParRelais24bits> <Oscillateur8bits> Pour chaque lot mémorisé : <noLot32bits> <mesure0_8bits>...<mesureN_8bits> <secteur&relais8bits> Après tous les lots : / <secteur&relais8bits> Correspond au bit de l'état secteur. Le bit b1 de l'octet <secteur&relais8bits> Correspond au bit de l'état du relais.	Rappel de la configuration enregistrée dans SAILA. Envoi du nombre de lots de mesures effectué en réalité. Chaque lot est précédé de son numéro d'ordre. La fin de la transmission est repérée par le caractère /. Le bit b0 de l'octet <secteur&relais8bits> Correspond au bit de l'état secteur. Le bit b1 de l'octet <secteur&relais8bits> Correspond au bit de l'état du relais.	
Maître	:C<11XXXXXX><16bits à X> :C<10XXXXXX><temps16bits> :C<01XXMMM><consigne8bits> <0000000S> :C<24bitsàX>	Déclenchement du relais sur la surveillance tension. Déclenchement du relais après un temps défini (sec) sur 16 bits. Déclenchement du relais sur une valeur de mesure. MMM : Numéro de la mesure (0-8). La consigne est la valeur du seuil de la mesure. S : Bit indiquant le sens montant (S=1) ou descendant (S=0).	PAR

		Lorsque le relais n'est pas à paramétrer, on envoie ' :C' suivi de 24 bits à 0	
Esclave	☐		
Maître	☐ : ☐ I ☐ E	Choix de l'oscillateur : Externe 'E' ou Interne 'I'.	PAR
Esclave	☐		

Pour les mots de 16 bits, le poids fort est émis en premier.

Je me suis créé une première interface graphique en essayant d'y mettre le nécessaire pour répondre au besoin de l'utilisateur. Encore aucune interaction n'était possible, cette interface de test m'a juste permis de visualiser les différentes possibilités que j'avais pour organiser l'application.

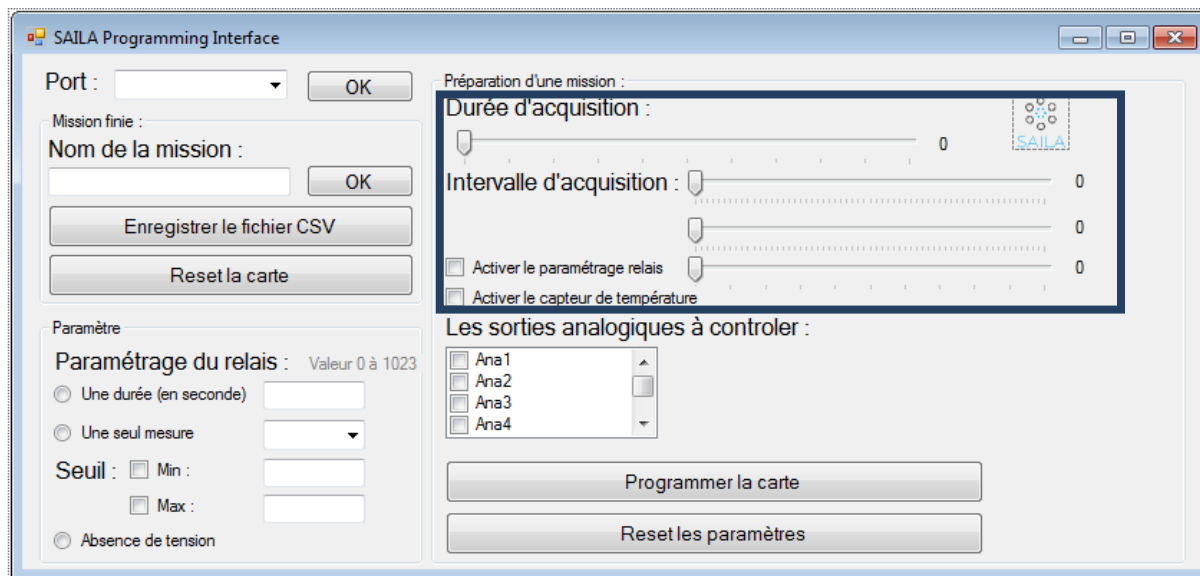


On peut voir ci-dessus la première IHM. Mon objectif était de pouvoir laisser la possibilité à l'utilisateur de mettre une durée et l'intervalle qu'il souhaite appliquer, de contrôler les sorties à utiliser comme on peut le voir dans les rectangles bleus.

La partie dans le rectangle rouge avait pour objectif de gérer tout ce qui était enregistrement d'une mission et aussi l'activation du paramétrage du relais.

Les problèmes qu'aurait rencontré l'utilisateur avec cette première interface aurait été au niveau du paramétrage de la durée et de l'intervalle. Nous avons une contrainte qui est celle de la mémoire. En effet, celle-ci n'est pas illimitée (environ 262144 octets). On sait qu'une acquisition fait 10 octets, donc si l'utilisateur souhaite mettre un intervalle de 1 seconde entre chaque acquisition, il serait contraint de mettre une durée de mission inférieure à 8h pour ne pas dépasser la capacité de la mémoire.

Donc il fallait pouvoir limiter les possibilités de l'utilisateur pour optimiser l'utilisation de l'application.



Voici la nouvelle IHM que j'ai développée par la suite. Comme vous pouvez le constater dans l'encadré bleu, j'ai remplacé ce qu'on appelle dans visual studio code les « textbox » par des « trackbar ».

Il y avait une interaction qui commençait à prendre place. Lorsque l'utilisateur démarrait l'application aucune fonctionnalité n'était disponible sur l'IHM (Pour forcer l'utilisateur à se connecter à la carte avant de faire quoi que ce soit). Une fois le port de communication qui correspondait à la carte SAILA choisit, et qu'il appuyait sur le bouton « OK », l'application rendait disponible le reste de l'IHM.

L'utilité de mettre des « trackbar » était de pouvoir forcer l'utilisateur à ne pas dépasser la mémoire et aussi de contrôler qu'il ne mette pas un intervalle supérieur à la durée de la mission. Pour l'intervalle, j'avais mis trois « trackbar », une pour les secondes, une pour les minutes et une pour les heures.

Le problème rencontré avec celle-ci c'était surtout au niveau de l'optimisation graphique de l'IHM. Mais aussi il y avait une possibilité de simplifier l'utilisation des « trackbars ».

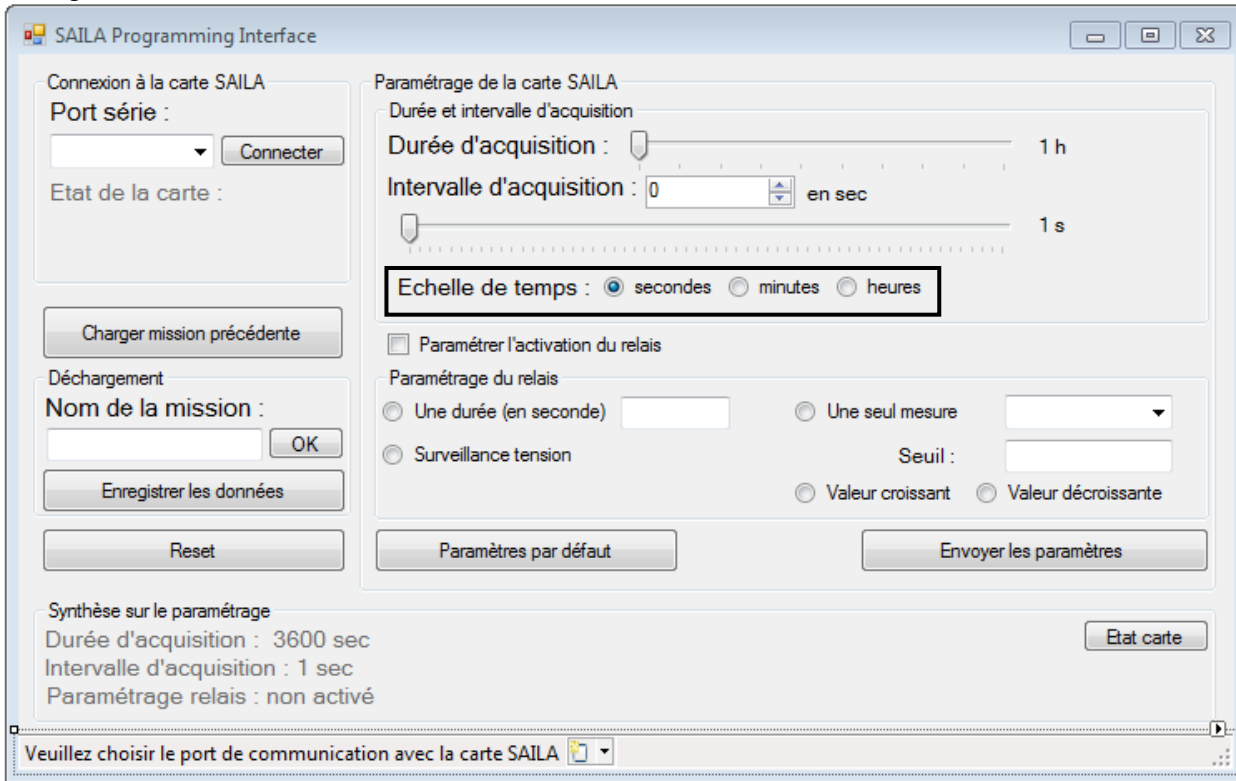
Le code utilisé pour la connexion à la carte est celui-ci :

```
private void btnOK_Click(object sender, EventArgs e)
{
    if (!serialPort.IsOpen)
    {
        String port = comboBox1.Text;
        serialPort.PortName = port;
        serialPort.BaudRate = 19200; // Vitesse de transmission
        serialPort.DataBits = 8; // 8 bits de données
        serialPort.Parity = Parity.None;
        serialPort.Open();

        if (serialPort.IsOpen)
        {
            comboBox1.Enabled = false;
            btnOK.Text = "Stop";
        }
        else MessageBox.Show("Impossible d'ouvrir le port" + port);
    } //if serial port
}

private void Form1_Load(object sender, EventArgs e)
{
    foreach (string SerialName in System.IO.Ports.SerialPort.GetPortNames())
    {
        comboBox1.Items.Add(SerialName); // Ajout du nom des ports
        // série dans la comboBox
    }
}
```

On déclare une variable port de type chaîne de caractères et qu'on assigne au texte de la comboBox. Ensuite la longueur des bits de données est de 8bits ici.



On peut voir ci-dessous la nouvelle interface. J'ai enlevé deux « Trackbar » de l'intervalle d'acquisition et rajouté trois « RadioButon » dans l'encadré noir. Lorsqu'on clique sur « heures » l'échelle de la « trackbar » passe de 1 à 18 h (car la durée est codée sur 16 bits et lors de l'envoi des paramètres, l'intervalle est converti en seconde donc 18h est l'intervalle max), et pour les « minutes » de 1 à 59 min... Les « trackbar » prennent en compte le stockage de la mémoire et s'adaptent pour ne pas la dépasser.

J'ai choisi d'enlever ce qu'on appelle une « groupbox » (l'utilité était de permettre à l'utilisateur de choisir les entrées à utiliser (voir les IHM précédentes)) dans Visual Studio après avoir vu avec notre professeur, car dans tous les cas la carte SAILA effectue les acquisitions de toutes les entrées. L'utilisateur aura juste à brancher les entrées voulu et les autres entrées auront des valeurs aléatoires.

J'ai voulu mettre tout ce qui concerne le paramétrage de la carte dans la partie droite de l'application et le reste à gauche pour optimiser graphiquement l'application.

Voici le code rentré pour la gestion des Radiobutton et des Trackbar :

```
private void TBduree_Scroll(object sender, EventArgs e) //Méthode pour gérer la trackbar de
la durée
{
    UInt16 DureeMax = 0;
    ValDuree = (UInt16)(TBduree.Value * 3600);

    LbdureeTB.Text = TBduree.Value.ToString() + " " + "h";

    DureeMax = (UInt16)(CSaila.Duree((UInt16)TBinter.Value));
    TBduree.Maximum = DureeMax;

    LBValueDuree.Text = "Durée d'acquisition : " + " " +(TBduree.Value * 3600).ToString() + " "
+ "sec";
}
```

```

private void TBinter_Scroll(object sender, EventArgs e) //Méthode pour gérer la taille de
la trackbar de l'intervalle
{
    TB();
    ValInt = (UInt16)(NumInt.Value); //C'est la valeur finale de l'intervalle
    ValDuree = (UInt16)(TBduree.Value * 3600); //C'est la valeur finale de la durée

    if (RBsec.Checked) //Si le radiobuton des secondes est coché
    {
        ValInt = SiRBs();
    }

    if (RBh.Checked) //Si le radiobuton des heures est coché
    {
        ValInt = SiRBh((UInt16)(TBinter.Value));
    }

    if (RBmin.Checked) //Si le radiobuton des minutes est coché
    {
        ValInt = SiRBm((UInt16)(TBinter.Value));
    }
    LbdureeTB.Text = TBduree.Value.ToString() + " " + "h"; //On affiche la durée finale sur l'IHM
    LBValueInter.Text = "Intervalle d'acquisition :" + " " + ValInt + " " + "sec";
} //On affiche l'intervalle final sur l'IHM

public UInt16 Duree(UInt16 InterAcqui)
{
    UInt16 DureeMax = (UInt16)((InterAcqui * 262144) / 36000); //Voici le calcul en question
    return DureeMax; //InterAcqui correspond à l'intervalle
}

public UInt16 InterH(UInt16 InterAcqui)
{
    UInt16 ValueInter = 0;

    ValueInter = (UInt16)(InterAcqui * 3600);
    return ValueInter; //Méthode appelée lorsque les radiosbuton changent
//d'unité pour actualiser l'affichage
}

public UInt16 InterM(UInt16 InterAcqui)
{
    UInt16 ValueInter = 0;

    ValueInter = (UInt16)(InterAcqui * 60);
    return ValueInter;
}

```

Gestion des radiobuton :

```

private void RBsec_CheckedChanged(object sender, EventArgs e)
{
    if (RBsec.Checked)
    {
        ValInt = SiRBs();
        LbdureeTB.Text = TBduree.Value.ToString() + " " + "h";
        LbinterTB.Text = TBinter.Value.ToString() + " " + "secs";
        LBValueInter.Text = "Intervalle d'acquisition :" + " " + ValInt + " " + "secs";
    }
} //Gérer l'affichage de la durée et de l'intervalle

private void RBmin_CheckedChanged(object sender, EventArgs e)
{
    if (RBmin.Checked)
    {
        ValInt = SiRBm((UInt16)(TBinter.Value));
    }
}

```

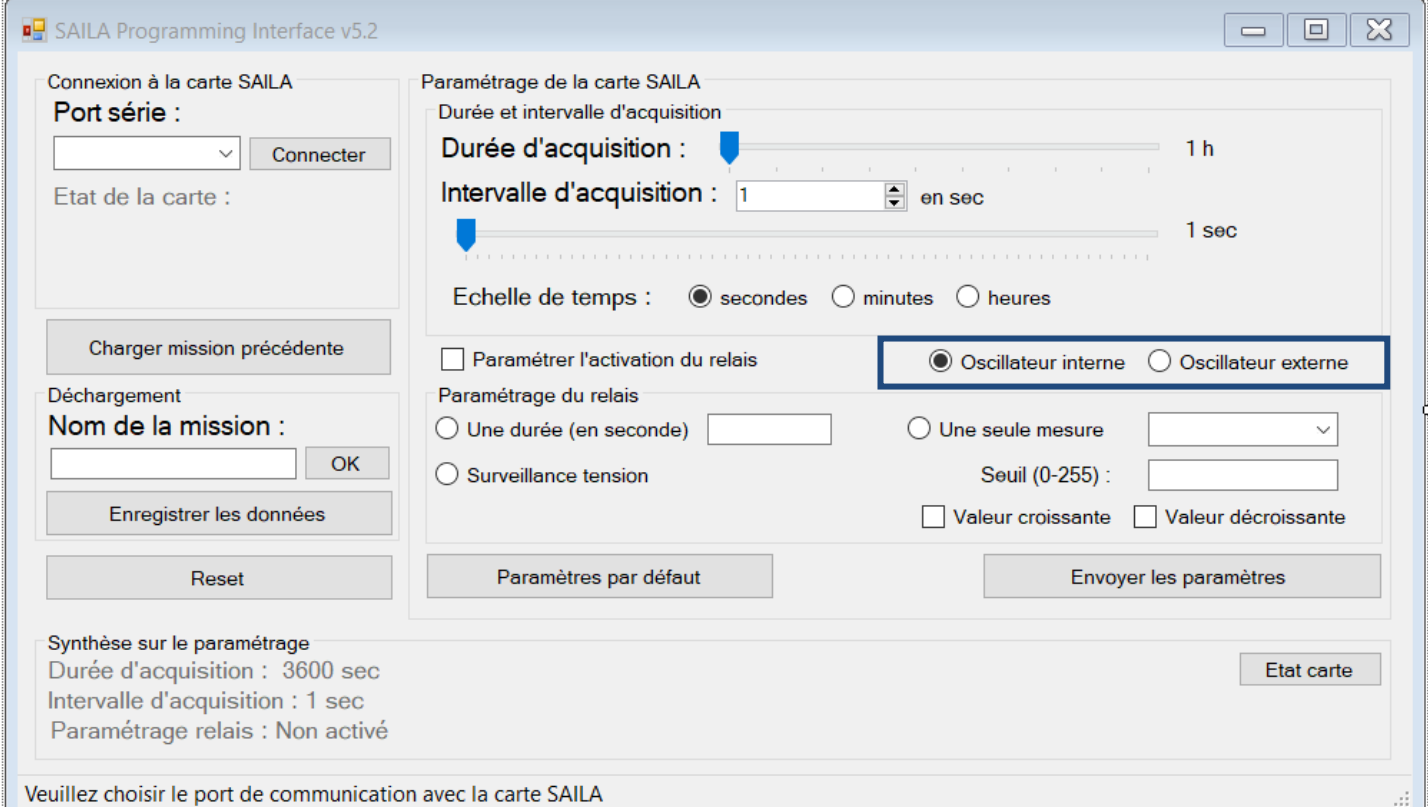
```

LbdureeTB.Text = TBduree.Value.ToString() + " " + "h";
LbInterTB.Text = TBinter.Value.ToString() + " " + "min";
LBValueInter.Text = "Intervalle d'acquisition :" + " " + ValInt + " " + "secs";
    }
} //Gérer l'affichage de la durée et de l'intervalle

private void RBh_CheckedChanged(object sender, EventArgs e)
{
    if (RBh.Checked)
    {
        ValInt = SiRBh((UInt16)(TBinter.Value));
        LbdureeTB.Text = TBduree.Value.ToString() + " " + "h"; //Gérer l'affichage de la
        LbInterTB.Text = TBinter.Value.ToString() + " " + "h"; //durée et de l'intervalle
        LBValueInter.Text = "Intervalle d'acquisition :" + " " + ValInt + " " + "secs";
    }
}

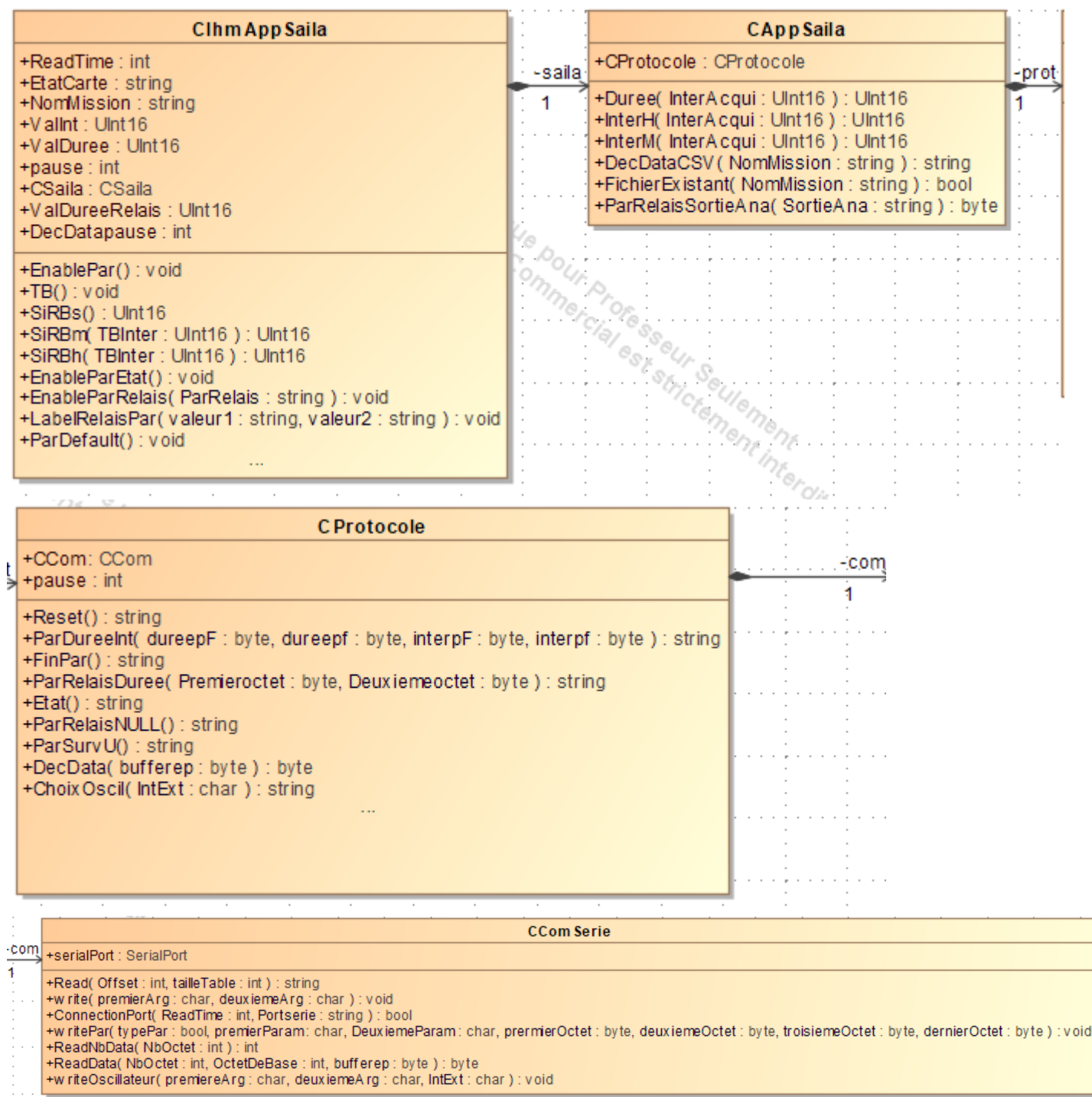
```

On a voulu rajouter un oscillateur à la carte SAILA car on voulait gagner en précision au niveau de l'horloge, donc il fallait pouvoir permettre à l'utilisateur d'utiliser soit l'oscillateur interne à la carte ou celui ajouter à celle-ci. Donc voici la dernière interface graphique avec l'oscillateur et sans faute d'orthographe :



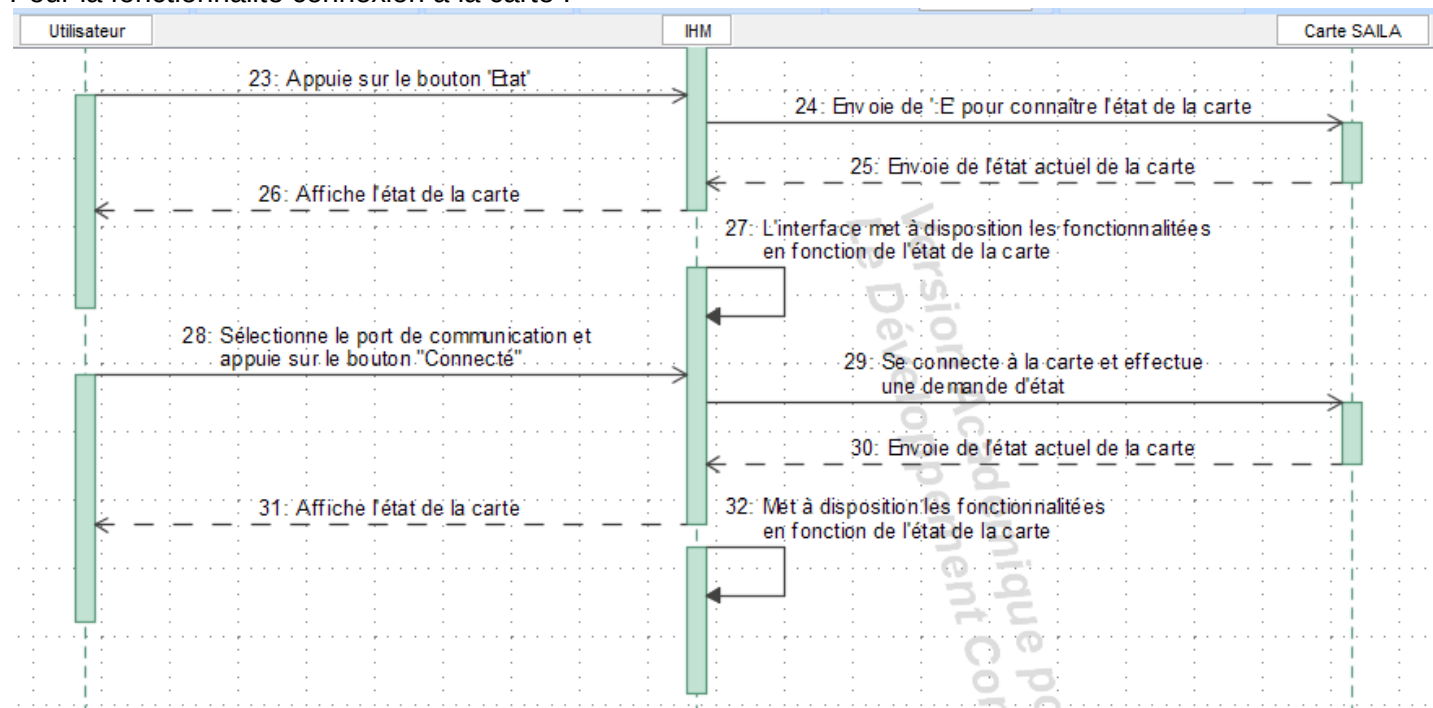
On peut voir l'encadré en bleu les « Radiobuton » pour gérer l'oscillateur de la carte. Grâce aux tâches de mise en œuvre mis en place au début du projet et de la création du protocole de communication, j'ai pu ajouter les fonctionnalités comme par exemple l'envoi du paramétrage que l'utilisateur aura choisi à la carte SAILA.

Avant de vous présenter mon code, je vais vous mettre ci-dessous mon diagramme de classe terminé de mon IHM final.

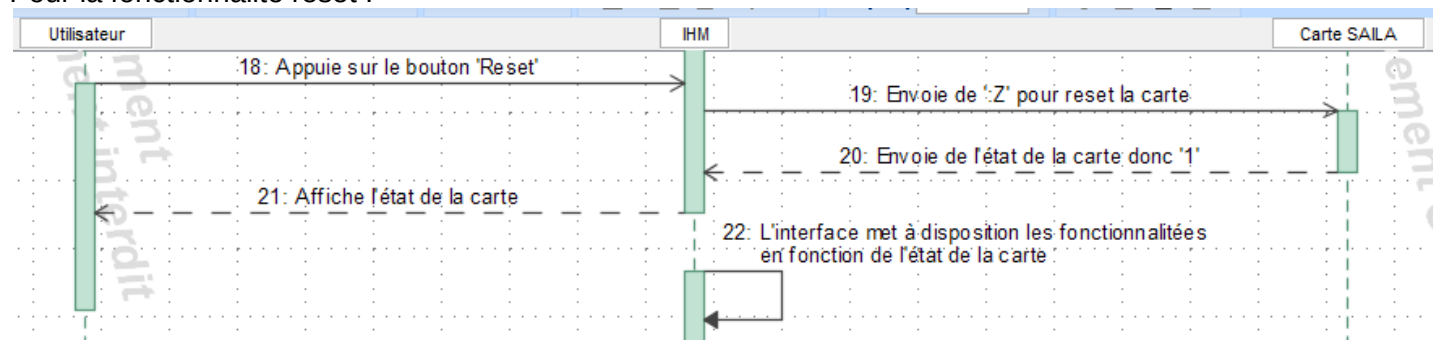


Voici les diagrammes de séquences de chaque fonctionnalité de la carte

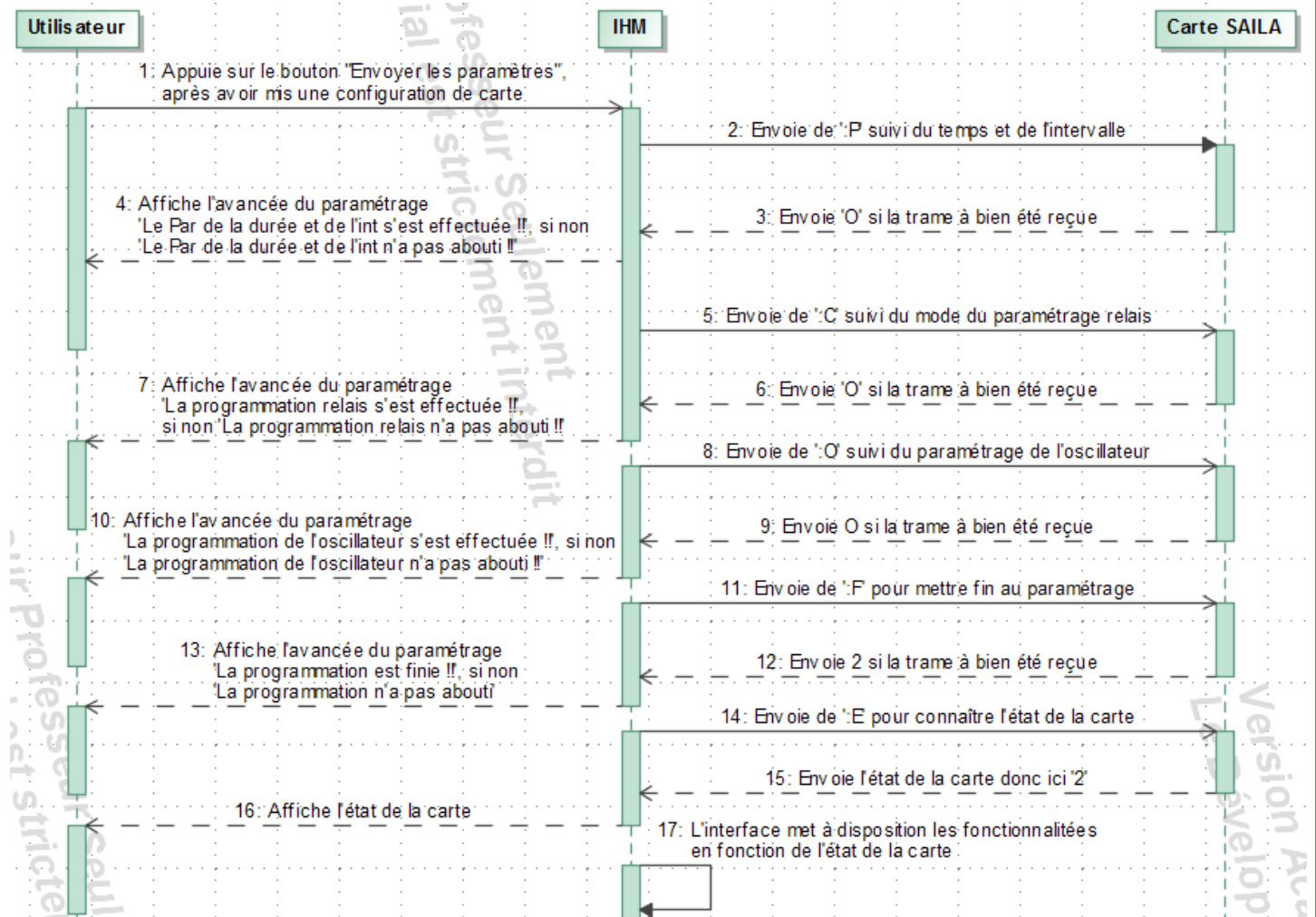
Pour la fonctionnalité connexion à la carte :



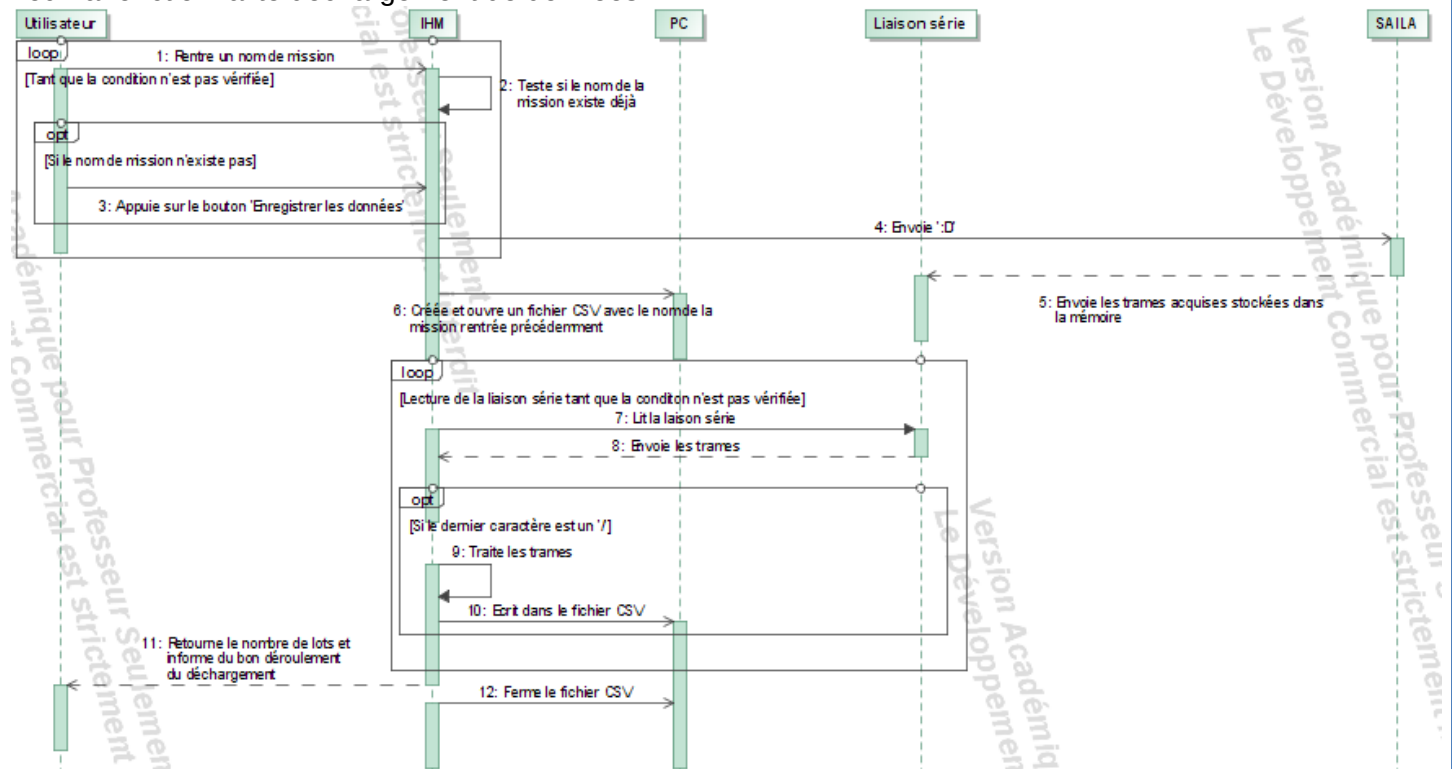
Pour la fonctionnalité reset :



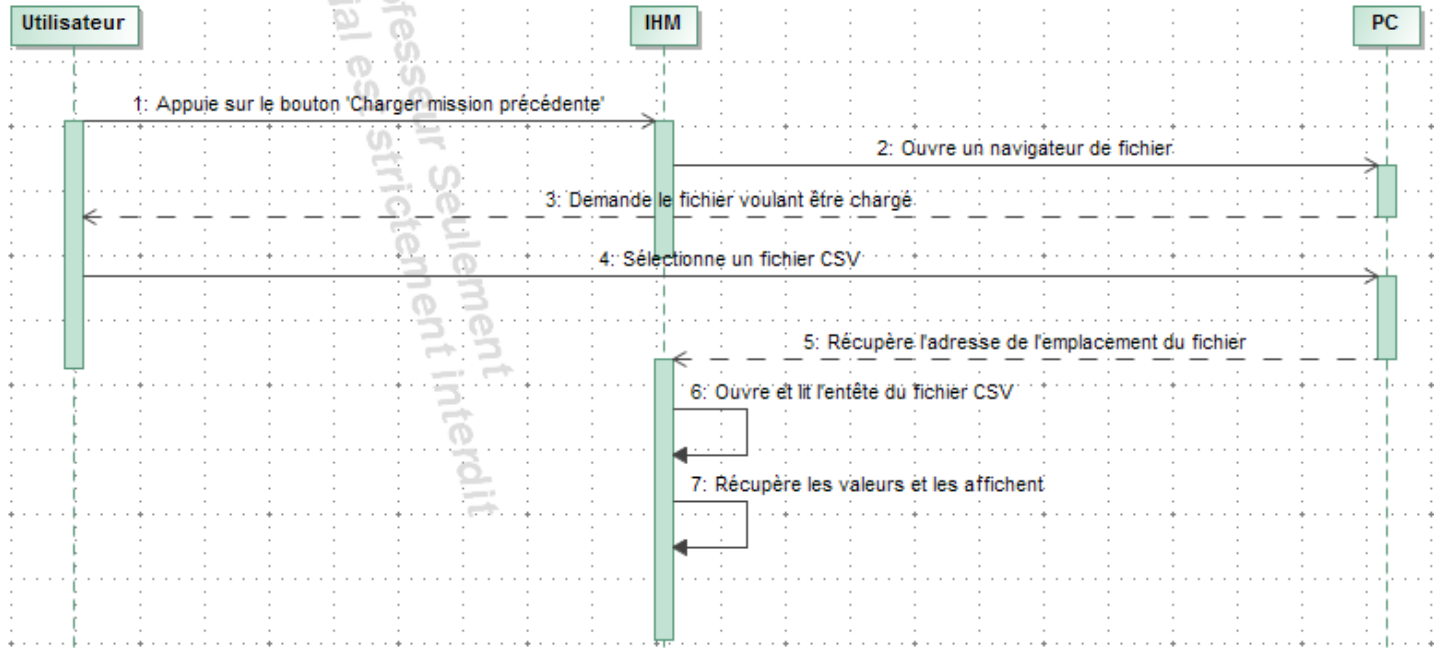
Pour la fonctionnalité envoi des données :



Pour la fonctionnalité déchargement de données :



Pour la fonctionnalité charger mission précédentes :



Voici le code pour réaliser cette fonctionnalité :

```

private void BPFInPar_Click(object sender, EventArgs e) //Fonctionnalité fin de paramétrage
{
    int dureepF = 0;
    int dureepf = 0;
    int interpF = 0;
    int interpf = 0;
    UInt16 ValueDuree = ValDuree;
    UInt16 ValueInt = ValInt;

    byte Premieroctet = 0;
    byte Deuxiemeoctet = 0;
    byte troisiemeoctet = 0;
    byte dernieroctet = 0;

    dureepF = ValueDuree >> 8;
    dureepf = ValueDuree - (dureepF << 8);
    interpF = ValueInt >> 8;
    interpf = ValueInt - (interpF << 8);

    Premieroctet = Convert.ToByte(dureepF);
    Deuxiemeoctet = Convert.ToByte(dureepf);
    troisiemeoctet = Convert.ToByte(interpF);
    dernieroctet = Convert.ToByte(interpf);

    EtatCarte = CSaila.CProtocole.ParDureeInt(Premieroctet, Deuxiemeoctet, troisiemeoctet,
dernieroctet);
    EtatSAILA.Text = "Etat de la carte :\n" + EtatCarte; //Envoie de la durée et de
l'intervalle
    Application.DoEvents();

    Thread.Sleep(pause);
    if (CBParRelais.Checked)
    {
        if (RBParRelaisDuree.Checked)
        {
            ValueDuree = Convert.ToUInt16(tBParRelaisDuree.Text);

            dureepF = (UInt16)(ValueDuree >> 8);

```

```

dureepf = (UInt16)(ValueDuree - (dureepF << 8));

Premieroctet = Convert.ToByte(dureepF);
Deuxiemeoctet = Convert.ToByte(dureepf);
EtatCarte = CSaila.CProtocole.ParRelaisDuree(Premieroctet, Deuxiemeoctet);
//Envoie du paramétrage relais si la durée est sélectionnée
EtatSAILA.Text = "Etat de la carte :\n" + EtatCarte;
Application.DoEvents();
Thread.Sleep(pause);
}

if (RBParRelaisSurvU.Checked)
{
    EtatCarte = CSaila.CProtocole.ParSurvU();
    EtatSAILA.Text = "Etat de la carte :\n" + " " + EtatCarte;      //Envoie du
paramétrage relais si la surveillance tension est sélectionnée
    Application.DoEvents();
    Thread.Sleep(pause);
}

if (RBParRelaisValue.Checked)
{
    byte SortieAna;
    string ValueAna = cBParRelaisValue.Text;
    byte ValueConsigne = Convert.ToByte(tBParRelaisValue.Text);
    byte CroiOUDecroi = 0;

    if(CBParRelaisCroi.Checked)
    {
        CroiOUDecroi = 1;
    }
    else
    {
        CroiOUDecroi = 0;
    }

    SortieAna = Convert.ToByte(64 + CSaila.ParRelaisSortieAna(ValueAna));

    EtatCarte = CSaila.CProtocole.ParRelaisValue(SortieAna, ValueConsigne,
CroiOUDecroi);
    //Envoie du paramétrage relais si c'est sur une valeur
    EtatSAILA.Text = "Etat de la carte :\n" + EtatCarte;
    Application.DoEvents();
    Thread.Sleep(pause);
}
}
else
{
    EtatCarte = CSaila.CProtocole.ParRelaisNULL();      //Envoie du paramétrage relais si
rien n'est sélectionné
    EtatSAILA.Text = "Etat de la carte :\n" + EtatCarte;
    Application.DoEvents();
    Thread.Sleep(pause);
}

if (RBOscilI.Checked)
{
    EtatCarte = CSaila.CProtocole.ChoixOscil('I');
    EtatSAILA.Text = "Etat de la carte :\n" + EtatCarte;      //Envoie du paramétrage
Oscillateur Interne
    Application.DoEvents();
    Thread.Sleep(pause);
}
else

```

```

    {
        EtatCarte = CSaila.CProtocole.ChoixOscil('E');           //Envoie du paramétrage
Oscillateur Externe
        EtatSAILA.Text = "Etat de la carte :\n" + EtatCarte;
        Application.DoEvents();
        Thread.Sleep(pause);
    }

    EtatCarte = CSaila.CProtocole.FinPar();           //Envoie du :F pour la fin de paramétrage
    EtatSAILA.Text = "Etat de la carte :\n" + EtatCarte;
    Application.DoEvents();
    Thread.Sleep(pause);
    EtatCarte = CSaila.CProtocole.Etat().ToString();           //Envoie du :E pour effectuer un
getEtat
    EtatSAILA.Text = "Etat de la carte :\n" + EtatCarte;

    EnableParEtat(); //Appel de la méthode pour faire le bonne affichage en fonction de l'état
}

```

Je récupère les valeurs de chaque Trackbar, pour mettre les bits de poids fort du bon côté et les envoyer à la liaison série, précédé par le « : P ». Ensuite récupérer le paramétrage du relais, qui fonctionne comme le paramétrage pour la durée et l'intervalle (c'est-à-dire sur une trame de 24 à envoyer selon le mode coché, voir le protocole de communication) et envoyer la bonne trame selon le mode sélectionnée, précédée par le « : C ». Si le mode sélectionné pour le relais est sur une durée, on va récupérer la valeur de la « TextBox » et la mettre sur 16 bits avec les bits de poids fort au bon endroit. Si le mode sélectionné est l'activation sur la surveillance d'une mesure, on va récupérer l'entrée à surveiller (exemple : Ana1 = 1, Ana2 = 2...) le coder sur les 4 derniers bits du premier octet, ensuite récupérer la valeur de la consigne sur 8 bits et le dernier octet sera utilisé pour coder si l'activation relais se fait sur la montée (donc 1) ou sur la descente (donc 0).

Une fois les trames envoyées, lire l'état de la carte et afficher les fonctionnalités utilisables en fonction de l'état de celle-ci.

Une fois le paramétrage effectué, l'utilisateur peut aller faire ses acquisitions. Une fois celles-ci terminées, il peut passer au déchargement des données. La tâche de mise en œuvre que j'ai réalisé pour la création d'un fichier CSV m'a permis de me familiariser au codage de cette fonctionnalité.

```

private void BPDecData_Click(object sender, EventArgs e)
{
    EtatCarte = CSaila.DecDataCSV(NomMission);

    EtatSAILA.Text = "Etat de la carte :\n" + EtatCarte;
    Application.DoEvents();
    Thread.Sleep(DecDatapause);
    EtatCarte = CSaila.CProtocole.Etat().ToString();
    EtatSAILA.Text = "Etat de la carte :\n" + EtatCarte;
    EtatIHM.Text = "Les données sont déchargées ! Vous pouvez faire un reset de la carte !";

    EnableParEtat();
    EtatIHM.Text = "Les données sont déchargées ! Vous pouvez faire un reset de la carte !";
}

```

Lorsque qu'on appuie sur le bouton « Enregistrer les données », on fait appel à la méthode DecDataCSV, qui va récupérer la totalité des trames de la carte SAILA pour les traiter, créer le fichier CSV avec le nom défini par l'utilisateur, et rentrer toutes les informations dans le fichier en respectant la spécification du fichier CSV

ci-dessous :

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Nom de la mission :												
2	Duree total :												
3	Intervalle d acquisition:												
4	Nombre de lots :												
5	Parametrage relais active Surveillance tension												
6	Parametrage relais act Duree :												
7	Parametrage relais act Mesure :	Temperature/Ana(Suivi du numero de mesur	seuil :			montant/descendant							
8	Parametrage oscilateur :	Interne/Externe											
9	Lot id	Temperature	Ana1	Ana2	Ana3	Ana4	Ana5	Ana6	Ana7	Ana8	Présence secteur	Déclenchement relais	
10	1												
11	2												
12	3												
13	4												
14	5												
15	6												
16	7												
17	8												
18	9												
19	10												
20	11												
21	12												

Voici la méthode qui va traiter les trames, les trier et les écrire dans le fichier CSV :

```
public string DecDataCSV(string NomMission)
{
    byte[] bufferep = new byte[2500000];
    UInt32 numlot = 0;
    UInt16 duree = 0;
    UInt16 inter = 0;
    UInt32 nbLot = 0;
    UInt32 ParRelais = 0;
    string Message;

    byte Premieremesure = 0;
    byte Deuxiememesure = 0;
    byte Troisiememesure = 0;
    byte Quatriememesure = 0;
    byte Cinquiememesure = 0;
    byte Sixiememesure = 0;
    byte Septiememesure = 0;
    byte Huitiememesure = 0;
    float CaptTemp = 0;
    byte SectRelais = 0;
    int lot = 12;
    UInt16 dureeRel = 0;
    int testRelMesure = 0;
    int numeroMesureRel = 0;
    int seuil = 0;
    int MouD = 0;
    string IntExt = "";
    StreamWriter writer;

    CProtocole.DecData(bufferep); //Utilisation d'une méthode pour lire les données

    duree = (UInt16)((bufferep[0] << 8) + bufferep[1]); //J'écris dans la durée de la mission,
    récupérer dans les 16 premiers bits
    inter = (UInt16)((bufferep[2] << 8) + bufferep[3]); //J'écris dans l'intervalle de la
    mission, récupérer dans les 16 bits d'après
    nbLot = (UInt32)((bufferep[4] << 24) + (bufferep[5] << 16) + (bufferep[6] << 8) +
    bufferep[7]); //J'écris le nombre de lots total de la mission, récupérer dans les 32 bits d'après
    ParRelais = (UInt32)((0x00 << 24) + (bufferep[8] << 16) + (bufferep[9] << 8) +
    bufferep[10]); //Correspond au paramétrage relais

    testRelMesure = bufferep[8];
    seuil = bufferep[9];
    MouD = bufferep[10];
    IntExt = bufferep[11].ToString();

    writer = new StreamWriter(@"C:\testcsv\" + NomMission + ".csv"); //Je crée le fichier CSV
    writer.WriteLine("Nom de la mission :" + ";" + NomMission);
    writer.WriteLine("Duree de la mission :" + ";" + duree + " secs"); //J'écris la première
    ligne
```

```

writer.WriteLine("Intervalle d acquisition :" + ";" + inter + " secs"); //J'écris la
deuxième ligne
writer.WriteLine("Nombre de lots :" + ";" + (nbLot - 1)); //J'écris la troisième ligne

if (ParRelais == 12582912) //1100 0000 16bitsà0 = 12582912 donc le paramétrage relais est
sur la surveillance tension
{
    writer.WriteLine("Parametrage relais active :" + ";" + "Surveillance tension");
}

if (testRelMesure == 128) //Si le premier octet des 24 bits du relais est = 128 ça veut
dire que le paramétrage est activé sur la durée (voir le protocole)
{
    dureeRel = (UInt16)((bufferrep[9] << 8) + bufferrep[10]);
    writer.WriteLine("Parametrage relais active :" + ";" + "Duree :" + ";" + dureeRel);
}

if (testRelMesure >= 64 & testRelMesure < 73 & MouD <= 1) //Si le premier octet se situe
entre [64, 73] (voir protocole) 64 car c'est le minimum et 72 le max car on additionne le 64 et le
numéro de mesure (le min = 0 pour la température et max = 8 pour Ana8)
{ //Le dernier octet ne peut pas dépasser 1 (si c'est sur la monté MouD = 1 et sur la
descente MouD = 0)
    numeroMesureRel = testRelMesure - 64;

    if (MouD == 1)
    {
        if (numeroMesureRel == 0)
        {
            writer.WriteLine("Parametrage relais active :" + ";" + "Mesure :" + ";" +
"Temperature" + ";" + "seuil :" + ";" + seuil + ";" + "montant");
        }
        else
        {
            writer.WriteLine("Parametrage relais active :" + ";" + "Mesure :" + ";" +
"Ana" + numeroMesureRel + ";" + "seuil :" + ";" + seuil + ";" + "montant");
        }
    }
    else
    {
        if (numeroMesureRel == 0)
        {
            writer.WriteLine("Parametrage relais active :" + ";" + "Mesure :" + ";" +
"Temperature" + ";" + "seuil :" + ";" + seuil + ";" + "descendant");
        }
        else
        {
            writer.WriteLine("Parametrage relais active :" + ";" + "Mesure :" + ";" +
"Ana" + numeroMesureRel + ";" + "seuil :" + ";" + seuil + ";" + "descendant");
        }
    }
}

if (ParRelais == 0) //S'il y a aucun paramétrage relais
{
    writer.WriteLine("Parametrage relais active :" + ";" + "Desactive");
}

if (IntExt == "E") //Ecriture du choix de l'oscillateur
{
    writer.WriteLine("Parametrage oscillateur :" + ";" + "Externe");
}
else

```

```

{
    writer.WriteLine("Parametrage oscillateur :" + ";" + "Interne");
}

writer.WriteLine("Lot id" + ";" + "Temperature" + ";" + "Ana1" + ";" + "Ana2" + ";" +
"Ana3" + ";" + "Ana4" + ";" + "Ana5" + ";" + "Ana6" + ";" + "Ana7" + ";" + "Ana8" + ";" + "Presence
secteur" + ";" + "Declenchement relais");

for (int i = 0; i != nbLot; i++) //Traitement des données en fonction du nombre de lot
{
    numlot = (UInt32)((bufferep[lot] << 24) + (bufferep[lot + 1] << 16) + (bufferep[lot +
2] << 8) + bufferep[lot + 3]);
    if (numlot != nbLot)//On souhaite ne pas écrire le dernier lot car il est souvent
incomplet donc avec des valeurs aléatoires
    {
        CaptTemp = Convert.ToSingle((bufferep[lot + 4] * 4) * (5.0 / 1024) * 100);
        Premieremesure = bufferep[lot + 5];
        Deuxiememesure = bufferep[lot + 6];
        Troisiememesure = bufferep[lot + 7];
        Quatriememesure = bufferep[lot + 8];
        Cinquiememesure = bufferep[lot + 9];
        Sixiememesure = bufferep[lot + 10];
        Septiememesure = bufferep[lot + 11];
        Huitiememesure = bufferep[lot + 12];
        SectRelais = bufferep[lot + 13];

        switch (SectRelais)
        {
            case 0:
                writer.WriteLine(numlot + ";" + CaptTemp + " C" + ";" + Premieremesure +
";" + Deuxiememesure + ";" + Troisiememesure + ";" + Quatriememesure + ";" + Cinquiememesure + ";" +
Sixiememesure + ";" + Septiememesure + ";" + Huitiememesure + ";" + 0 + ";" + 0);
                break;
            case 1:
                writer.WriteLine(numlot + ";" + CaptTemp + " C" + ";" + Premieremesure +
";" + Deuxiememesure + ";" + Troisiememesure + ";" + Quatriememesure + ";" + Cinquiememesure + ";" +
Sixiememesure + ";" + Septiememesure + ";" + Huitiememesure + ";" + 1 + ";" + 0);
                break;
            case 2:
                writer.WriteLine(numlot + ";" + CaptTemp + " C" + ";" + Premieremesure +
";" + Deuxiememesure + ";" + Troisiememesure + ";" + Quatriememesure + ";" + Cinquiememesure + ";" +
Sixiememesure + ";" + Septiememesure + ";" + Huitiememesure + ";" + 0 + ";" + 1);
                break;
            case 3:
                writer.WriteLine(numlot + ";" + CaptTemp + " C" + ";" + Premieremesure +
";" + Deuxiememesure + ";" + Troisiememesure + ";" + Quatriememesure + ";" + Cinquiememesure + ";" +
Sixiememesure + ";" + Septiememesure + ";" + Huitiememesure + ";" + 1 + ";" + 1);
                break;
            default:
                writer.WriteLine(numlot + ";" + CaptTemp + " C" + ";" + Premieremesure +
";" + Deuxiememesure + ";" + Troisiememesure + ";" + Quatriememesure + ";" + Cinquiememesure + ";" +
Sixiememesure + ";" + Septiememesure + ";" + Huitiememesure + ";" + "?" + ";" + "?");
                break; //Ecriture de chaque lot en fonction de la présence secteur et du
déclenchement relais
        }
        lot = lot + 14;
    }
    else
    { }
};

```

```

        writer.Flush(); //J'efface la mémoire tampon du writer
        writer.Close();

        Message = "Le déchargement de :\n" + (nbLot - 1) + " lots" + " s'est effectué avec
\nsuccès !!";

        return Message;
    }

```

Voici le code de la méthode DecData utilisé dans le code ci-dessus :

```

public byte[] DecData(byte[] bufferep)
{
    int NbOctet = 0;
    char DernierCaractere;
    int OctetDeBase = 0;

    CCom.write(':', 'D'); //Envoie du ":D" dans la liaison série
    do
    {
        NbOctet = CCom.ReadNbData(NbOctet); //Je fais appel à une méthode me permettant de lire
le nombre d'octets à traiter
        bufferep = CCom.ReadData(NbOctet, OctetDeBase, bufferep); //Je lis le nombre d'octets
présent et je les mets dans un tableau
        DernierCaractere = Convert.ToChar(bufferep[NbOctet-1]); //Je lis la dernière case du
tableau
        OctetDeBase = NbOctet; //Et j'attribue à ma variable le nombre d'octets déjà présents
car si je n'ai pas tout lu jusqu'au « / » je vais effectuer une autre lecture et je vais devoir
l'ajouter à la suite de mon tableau
    } while (DernierCaractere != '/'); //J'effectue cette boucle tant que le dernier octet ne
correspond pas à un "/" (voir protocole)

    return bufferep; //Une fois avoir tout lu, on retourne le tableau
}

```

Voici le code des méthodes ReadNbData et ReadData utilisé ci-dessus :

```

public int ReadNbData(int NbOctet) //On lie le nombre d'octets dans la liaison série
{
    Thread.Sleep(1000);
    NbOctet = NbOctet + serialPort.BytesToRead;

    return NbOctet;
}

public byte[] ReadData(int NbOctet, int OctetDeBase, byte[] bufferep)
{
    serialPort.Read(bufferep, OctetDeBase, NbOctet); //On lie la liaison série avec le nombre
d'octets détecté

    return bufferep;
}

```

Une fois les données déchargées, on peut effectuer un reset de la carte en appuyant sur le bouton « Reset » de l'IHM. La carte SAILA va repasser en mode paramétrage.

Voici le code utilisé pour effectuer le reset :

```

public string Reset()
{
    CCom.write(':', 'Z'); //On envoi ":Z" dans la liaison série
    string EtatCarte = CCom.Read(0, 2); //La carte nous renvoi l'état (voir protocole)
    int EtatAffich = Convert.ToInt16(EtatCarte); //On affiche l'état de façon plus
compréhensible pour l'utilisateur
    if (EtatAffich < 2)
    {
        EtatCarte = "Mode paramétrage";
    }
}

```

```

    }

    if (EtatAffich == 2)
    {
        EtatCarte = "Mode en attente\nd'acquisitions";
    }

    if (EtatAffich == 3)
    {
        EtatCarte = "Mode en cours\nd'acquisitions";
    }

    if (EtatAffich == 4)
    {
        EtatCarte = "Mode en attente\nde déchargement";
    }

    return EtatCarte;//On retourne l'état pour l'afficher
}

```

Une fois en mode paramétrage, l'utilisateur peut soit reprogrammer comme auparavant ou alors récupérer les paramètres d'une mission précédentes. Il suffit d'appuyer sur le bouton « Charger une mission précédente » de l'IHM, cela va ouvrir un explorateur de fichier et il suffira de sélectionner le fichier et appuyer sur ouvrir. Les paramètres seront récupérés et affichés dans l'application. Ensuite il suffira d'appuyer sur « Envoyer les paramètres ». Voici pour finir le code pour charger le fichier :

```

private void BPChargerMission_Click(object sender, EventArgs e)
{
    int NumDeLigne = 0;
    string ParRelais;
    string ParRelaisChoisie = "";
    string ligne;
    Char caractere = ';';

    var fileContent = string.Empty;
    var filePath = string.Empty;

    using (OpenFileDialog openFileDialog = new OpenFileDialog())
    {
        openFileDialog.InitialDirectory = "c:\\\\";
        openFileDialog.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
        openFileDialog.FilterIndex = 2;
        openFileDialog.RestoreDirectory = true;

        if (openFileDialog.ShowDialog() == DialogResult.OK)
        {
            //Get the path of specified file
            filePath = openFileDialog.FileName;
        }
    }
    try
    {
        System.IO.StreamReader fichier = new System.IO.StreamReader(filePath);

        while ((ligne = fichier.ReadLine()) != null)//On va lire ligne par ligne et cellule par
cellule
        {
            String[] substrings = ligne.Split(caractere);//Dans un premier temps on sépare les
cellules et on attribue la première cellule à caractère ensuite on effectue la même chose avec la 2ème
etc...

            int Cellule = 0;//Correspond aux lignes, ici ligne 0 est égal à la première ligne
            foreach (var substring in substrings)
            {

```

```

Console.WriteLine(substring);
if (NumDeLigne == 1 & Cellule == 1)//On veut récupérer la deuxième ligne et la
deuxième cellule pour récupérer la durée
{
    ValDuree = Convert.ToUInt16(substring);
    TBduree.Value = ValDuree / 3600;//Ensuite on met à jour la trackbar
    Application.DoEvents();
}

if (NumDeLigne == 2 & Cellule == 1)//On veut récupérer la troisième ligne et la
deuxième cellule pour récupérer l'intervalle
{
    ValInt = Convert.ToUInt16(substring);
    if (ValInt < 60)//On gère l'affichage en sec si l'intervalle est compris entre
0 et 59 sec
    {
        TBinter.Value = ValInt;
        RBh.Checked = false;
        RBmin.Checked = false;
        RBsec.Checked = true;
        LbdureeTB.Text = TBduree.Value.ToString() + " " + "h";
        LbinterTB.Text = TBinter.Value.ToString() + " " + "secs";
    }

    if (ValInt > 59 & ValInt < 3600)//On gère l'affichage en min si l'intervalle
est compris entre 0 et 59 min
    {
        TBinter.Value = ValInt/60;
        RBh.Checked = false;
        RBmin.Checked = true;
        RBsec.Checked = false;
        LbdureeTB.Text = TBduree.Value.ToString() + " " + "h";
        LbinterTB.Text = TBinter.Value.ToString() + " " + "min";
    }

    if (ValInt > 3599)//On gère l'affichage en heure si l'intervalle est supérieur
à 3599 sec
    {
        TBinter.Value = ValInt / 3600;
        RBh.Checked = true;
        RBmin.Checked = false;
        RBsec.Checked = false;
        LbdureeTB.Text = TBduree.Value.ToString() + " " + "h";
        LbinterTB.Text = TBinter.Value.ToString() + " " + "h";
    }
    NumInt.Value = ValInt; //On actualise l'affichage numérique
}

if (NumDeLigne == 4 & Cellule >= 1)//On récupère le paramétrage relais à la 5ème
ligne, on lit la 2ème cellule et on l'affiche
{
    ParRelais = substring;
    if (ParRelais == "Duree :" | ParRelais == "Mesure :" | ParRelais ==
"Surveillance tension")
    {
        EnableParRelais(ParRelais);
        ParRelaisChoisie = ParRelais;
    }

    if (ParRelais == "Surveillance tension")
    {
        RBParRelaisSurvU.Checked = true;

```

```

        LBParRelais.Text = "Paramétrage relais : Activé sur la surveillance
tension";
    }

    if(ParRelaisChoisie == "Duree :" & Cellule == 2) //On récupère la valeur de la
troisième cellule si le relais est paramétré sur une durée
    {
        ValDureeRelais = Convert.ToUInt16(substring);
        RBParRelaisDuree.Checked = true;
        tBParRelaisDuree.Text = ValDureeRelais.ToString();
        LBParRelais.Text = "Paramétrage relais : Activé après une durée de :" + "
" + tBParRelaisDuree.Text + " " + "secs";
    }

    if (ParRelaisChoisie == "Mesure :" & Cellule == 2) //On récupère la valeur de
la troisième cellule si le relais est paramétré sur une mesure pour afficher l'entrée à surveiller
    {
        RBParRelaisValue.Checked = true;
        cBParRelaisValue.Text = substring;
    }

    if (ParRelaisChoisie == "Mesure :" & Cellule == 4)//On récupère la valeur de
la cinquième cellule si le relais est paramétré sur une mesure pour récupérer la consigne (voir
protocole)
    {
        tBParRelaisValue.Text = substring;
    }

    if (ParRelaisChoisie == "Mesure :" & Cellule == 5)//On récupère la valeur de
la sixième cellule si le relais est paramétré sur une mesure pour récupérer le sens (soit
montant/descendant)
    {
        if (substring == "montant")
        {
            CBParRelaisCroi.Checked = true;
            CBParRelaisDecroi.Checked = false;
        }
        else
        {
            CBParRelaisCroi.Checked = false;
            CBParRelaisDecroi.Checked = true;
        }
        if (CBParRelaisCroi.Checked)
        {
            LBParRelais.Text = "Paramétrage relais : Activé sur une valeur
montante de la sortie" + " " + cBParRelaisValue.Text + " " + "La valeur est de :" + " " +
tBParRelaisValue.Text;
        }
        if (CBParRelaisDecroi.Checked)
        {
            LBParRelais.Text = "Paramétrage relais : Activé sur une valeur
descendante de la sortie" + " " + cBParRelaisValue.Text + " " + "La valeur est de :" + " " +
tBParRelaisValue.Text;
        }
    }

    }
    Cellule++; //On passe à la cellule suivante
}
NumDeLigne++; //On passe à la ligne suivante
}

LBValueDuree.Text = "Durée d'acquisition :" + " " + ValDuree + " " + "sec"; //on gère
l'affichage de la durée

```

```

        LBValueInter.Text = "Intervalle d'acquisition :" + " " + ValInt + " " + "sec";//on gère
l'affichage de l'intervalle d'acquisition
        fichier.Close();
    }
    catch { }
}

```

Présentation du RS-232 :

Pour réaliser notre projet nous avons utilisé la communication via une liaison série de type RS-232. RS-232 est une norme standardisant une voie de communication de type série. Il est communément appelé le « port série ». Sur le système d'exploitation Windows, les ports RS-232 sont désignés par les noms COM1, COM2, etc. Il est de plus en plus remplacé par le port USB.

Les liaisons RS-232 sont fréquemment utilisées dans l'industrie pour connecter différents appareils électroniques (automate, appareil de mesure, etc...).

L'objectif était de voir les trames qu'on envoie à la carte SAILA via la liaison série. On va traiter ce que reçoit la carte (RX) et ce qu'elle renvoie (TX).

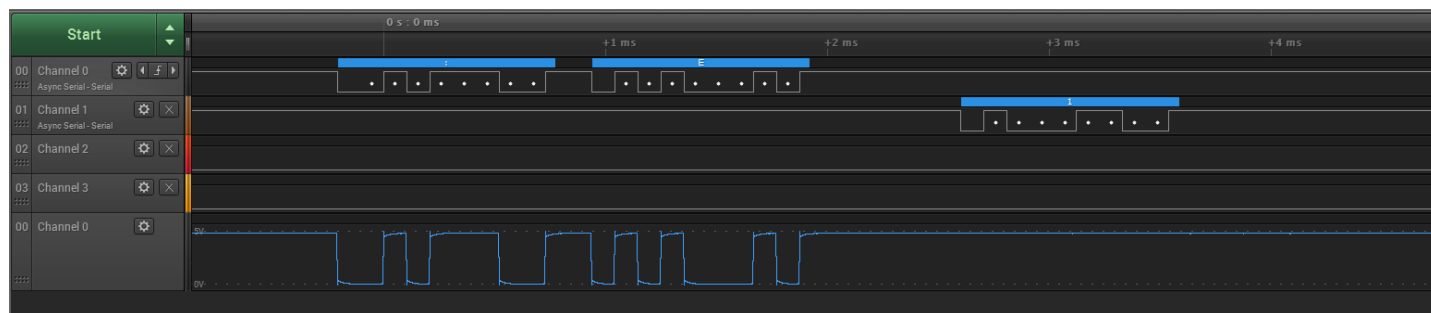
RX est un sigle signifiant : réception, en procédure de télégraphie (morse).

TX est une abréviation qui peut signifier : émission ou transmission, en télécommunication.

Pour les analyser, nous allons utiliser l'appareil Saleae qui est un analyseur logique et qui va nous permettre de voir l'avancé binaire au fil du temps d'un signal.



Après avoir connecté l'appareil à la carte SAILA et mis en route le logiciel Logic avec la bonne configuration, on va pouvoir apercevoir les courbes ci-dessous.



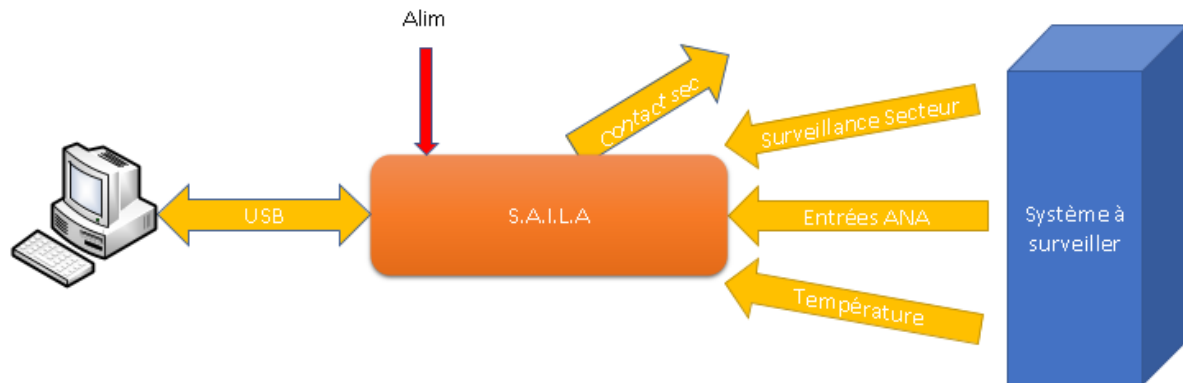
Le signal en haut correspond au RX reçu et celui en-dessous au TX, émis par la carte. Ici j'ai effectué une demande d'état donc j'ai envoyé ':E' et je suis censé recevoir un état donc dans l'exemple ici la carte était dans l'état 1. On peut voir que j'envoie 2 octets, et on remarque que ça correspond bien au ':' et 'E'. et que la carte renvoie bien un octet et ça correspond à '1' codé en Ascii.

Partie personnel Yanis Founas

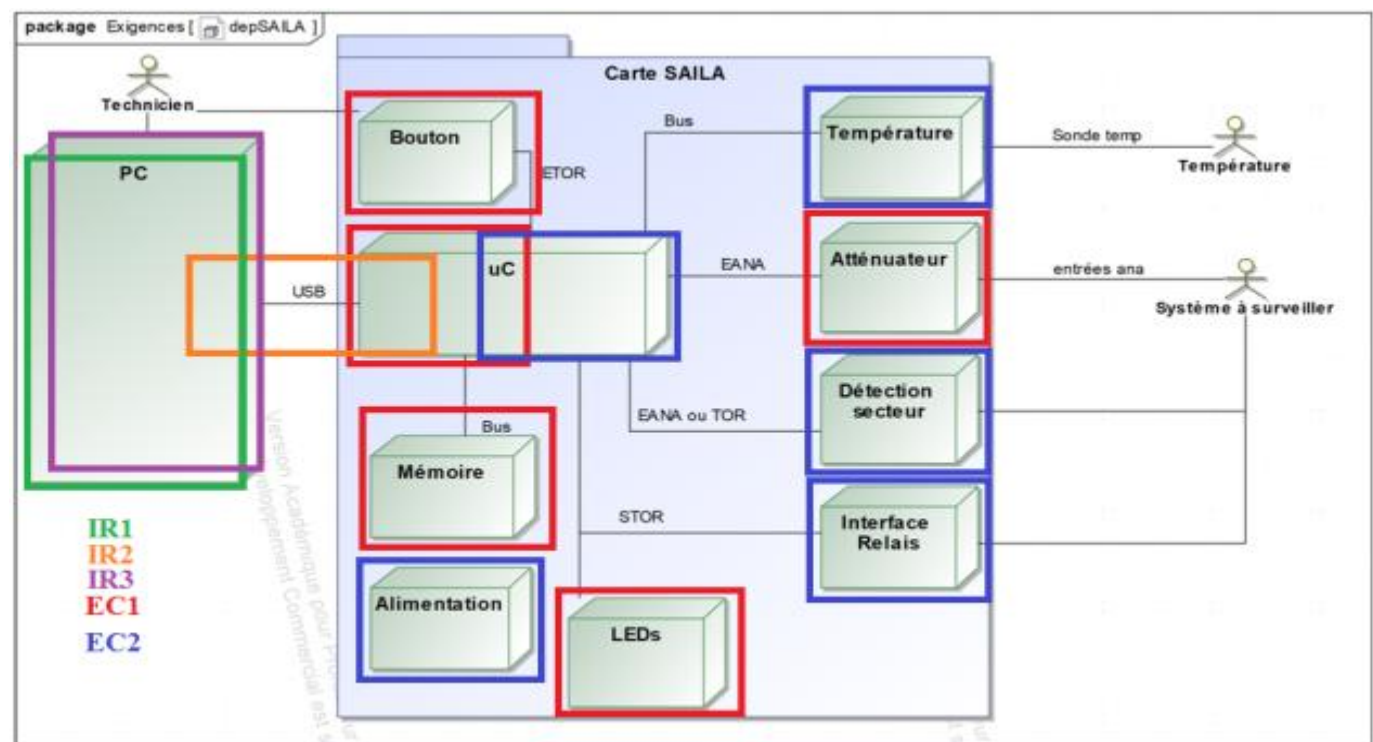


I) Introduction :

Le projet SAILA est un système d'acquisition d'information Logique Analogique. L'objectif est de fournir une carte qui possède 9 entrées analogiques, dont 1 entrée qui devra fournir une température.



Mon travail se concentre, sur la température, l'interface relai, l'alimentation et la présence secteur et l'unité centrale.



PLAN DE DÉVELOPPEMENT

N° itération	Activités	Justification
1	Mise en œuvre de tâche de plusieurs spécifications : 1. Test des leds (Proteus) 2. Teste du relais 3. Programme Mplab x(leds) 4. Programme Mplab x(Relais)	Première revue : Pour commencer mon projet j'ai effectué, teste sur le logiciel Proteus et Mplab X ensuite j'ai fait les premières installations physiques pour la première revue
	Développement de la carte SAILA 1. Création d'un programme pour relever la température 2. Création d'un schéma Kicad (Empreintes, schéma) 3. Création des routes 4. Dernière modification	Deuxième revue : J'ai créé un programme pour relever la température, puis j'ai mis les empreintes sur Kicad et j'ai créé mon schéma, ensuite j'ai créé les routes et pour finir nous avons fait les dernières modifications avant la fabrication.
	Finalisation du projet : 1. Liste des composant 2. Soudure	Troisième revue : J'ai récupéré ma liste des composants Puis j'ai pris un PCB et un stencil puis j'ai fait les soudures des composants dans la carte.

Cahier des charges :

- Nombre d'entrées analogiques : 8 (0-5V).
- Température : Seul un capteur externe sera prévu.
- CAN de 8 bits. La précision : $5v / 255 = 0.02v$
- Surveillance de la tension secteur sur une entrée séparée (option).
- Durée d'enregistrement : de quelques minutes à une dizaine d'heures.
- L'intervalle d'acquisition : de 1s à 15mn. Déclenchement de l'acquisition par un bouton.
- LED de fonctionnement de l'appareil.
- Sortie relais contact sec 2A 2RT (230V). Excitation du relais selon le paramétrage de l'horloge (durée de fonctionnement), un seuil franchi sur une entrée ou la température.
- Le microcontrôleur choisi est PIC 8bits 18F45K50.

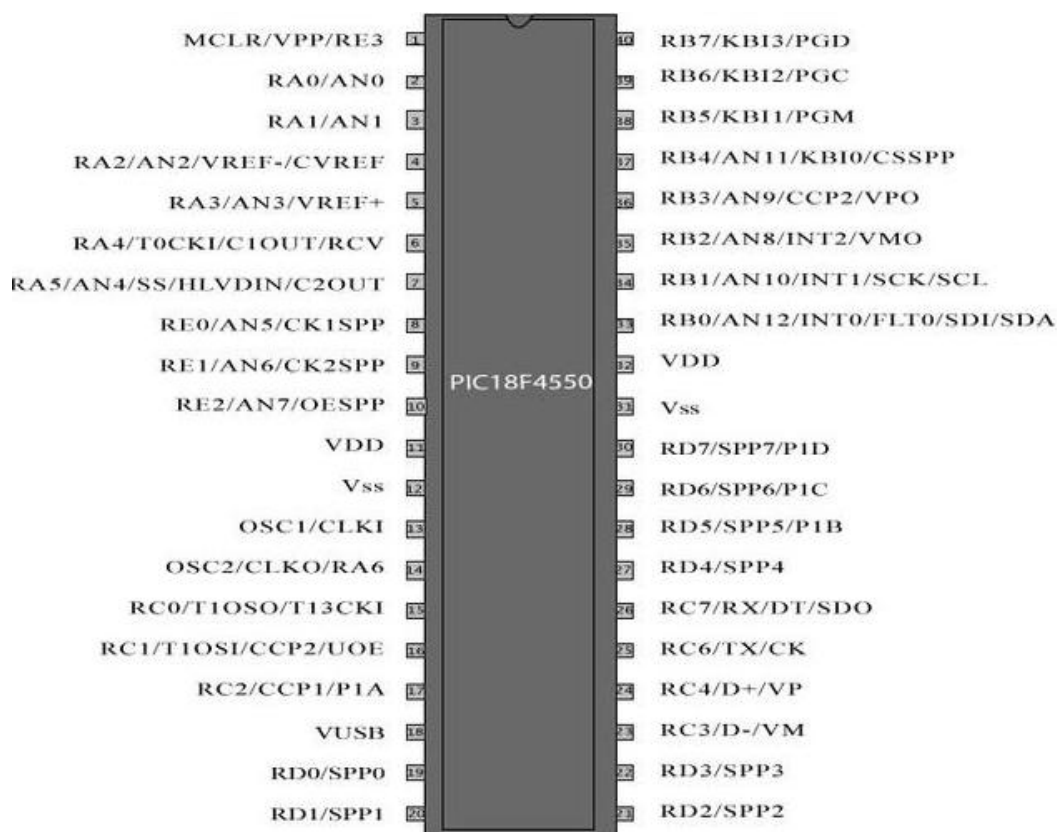
Le microcontrôleur

Le **PIC18F4550** comprend :

- Une mémoire flash de 32 Ko.
- Une mémoire SRAM de 2 Ko.
- Une EEPROM de 256 octets.

Device	Program Memory		Data Memory		Pins	I/O	10-Bit A/D Channels	Comparators	CCP/ ECCP	BOR/LVD	CTMU	MSSP	EUSART	Timers 8-bit/16-bit	USB 2.0
	Flash (bytes)	Single-Word Instructions	SRAM (bytes)	Data EEPROM (bytes)											
PIC18(L)F45K50	32K	16384	2048	256	40/44	36	25-ch	2	1/1	Yes	Yes	1	1	2/2	Yes
PIC18(L)F25K50	32K	16384	2048	256	28	25	14-ch	2	1/1	Yes	Yes	1	1	2/2	Yes
PIC18(L)F24K50	16K	8192	2048	256	28	25	14-ch	2	1/1	Yes	Yes	1	1	2/2	Yes

Caractéristiques des broches :



Le **PIC18F4550** peut fonctionner sur une plage de fréquence allant de 31 KHz à 48 MHz. Il a quatre minuteries intégrées. Ce contrôleur contient divers périphériques intégrés tels qu'un convertisseur analogique-numérique, des comparateurs, etc.

C'est un microcontrôleur avancé doté de protocoles de communication améliorés tels qu'EUSART, SPI, USB.

Documentation du pic :

Pin Number		Pin Name	Pin Type	Buffer Type	Description
SPDIP, SOIC, SSOP	QFN				
2	27	RA0/C12IN0-/AN0			
		RA0 C12IN0- AN0	I/O I I	TTL/DIG Analog Analog	Digital I/O. Comparators C1 and C2 inverting input. Analog input 0.
3	28	RA1/C12IN1-/AN1			
		RA1 C12IN1- AN1	I/O I I	TTL/DIG Analog Analog	Digital I/O. Comparators C1 and C2 inverting input. Analog input 1.
4	1	RA2/C2IN+/AN2/DACOUT/VREF-			
		RA2	I/O	TTL/DIG	Digital I/O.
		C2IN+	I	Analog	Comparator C2 non-inverting input.
		AN2	I	Analog	Analog input 2.
		DACOUT	O	Analog	DAC Reference output.
		VREF-	I	Analog	A/D reference voltage (low) input.
5	2	RA3/C1IN+/AN3/VREF+			
		RA3	I/O	TTL/DIG	Digital I/O.
		C1IN+	I	Analog	Comparator C1 non-inverting input.
		AN3	I	Analog	Analog input 3.
		VREF+	I	Analog	A/D reference voltage (high) input.
6	3	RA4/C1OUT/SRQ/T0CKI			
		RA4	I/O	ST/DIG	Digital I/O.
		C1OUT	O	DIG	Comparator C1 output.
		SRQ	O	DIG	SR latch Q output.
		T0CKI	I	ST	Timer0 external clock input.
7	4	RA5/C2OUT/SRNQ/SS/HLVDIN/AN4			
		RA5	I/O	TTL/DIG	Digital I/O.
		C2OUT	O	DIG	Comparator C2 output.
		SRNQ	O	DIG	SR latch $\bar{Q}$ output.
		$\bar{SS}$	I	TTL	SPI slave select input (MSSP).
		HLVDIN	I	Analog	High/Low-Voltage Detect input.
10	7	RA6/CLKO/OSC2			
		RA6	I/O	TTL/DIG	Digital I/O.
		CLKO	O	DIG	Outputs 1/4 the frequency of OSC1 and denotes the instruction cycle rate.
		OSC2	O	—	Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator modes.

On peut voir par exemple que la broche RA0 est une entrée/sortie digital et une sortie analogique.

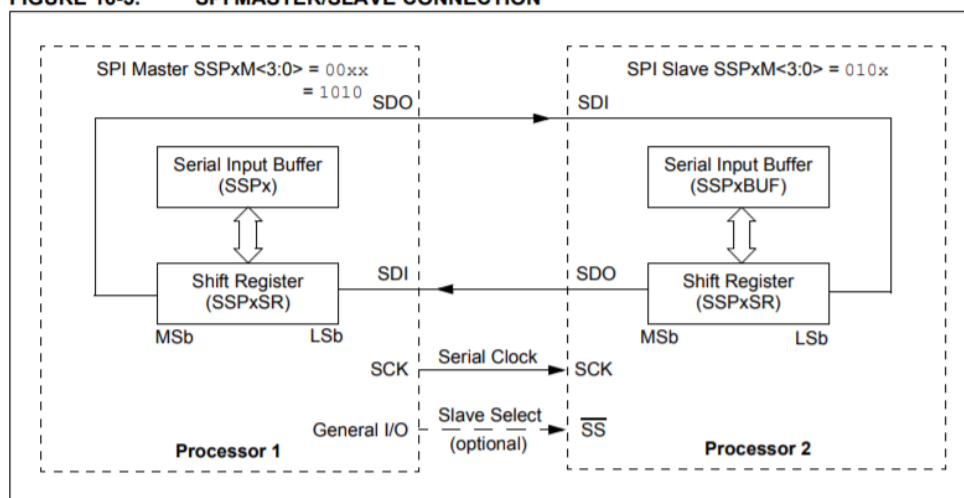
Liaison série

1) Le bus SPI

Le bus SPI (Serial Peripheral Interface) est un bus de communication de données série synchrone qui fonctionne en mode duplex intégral. Les appareils communiquent dans un environnement maître / esclave où le périphérique maître initie la communication. Un appareil esclave est contrôlé par une sélection de puces appelée sélection d'esclave. Le bus SPI spécifie quatre connexions de signal :

- Horloge série (SCK)
- Sortie de données série (SDO)
- Entrée de données série (SDI)
- Slave Select (SS)

FIGURE 16-5: SPI MASTER/SLAVE CONNECTION



Sur la figure, on y voit trois lignes que sont SDO, SDI et SCK et une ligne de sélection de mode de fonctionnement du circuit (maître ou esclave) appelée SS.

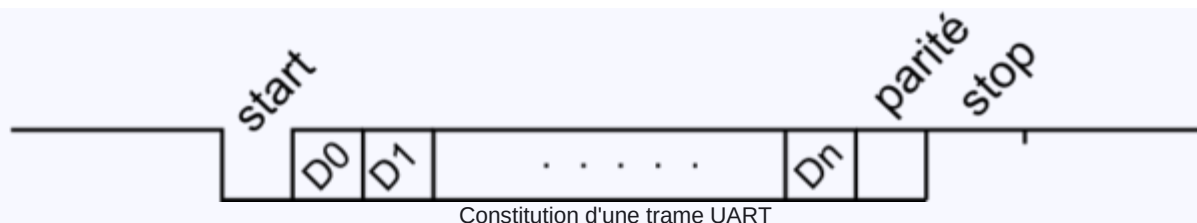
- * La SDO, pour Serial Data Out, est la sortie de données séries.
- * La patte SDI, pour Serial Data In, est l'entrée de données séries.
- * La ligne SCK pour Serial Clock est utilisée pour synchroniser les échanges de données puisque nous sommes en mode série synchrone. Cette ligne est nécessairement une sortie sur le maître et une entrée sur le ou les esclaves.

La ligne SS est la sélection du mode maître ou esclave. Elle est au niveau haut sur un maître et au niveau bas sur un esclave. Pour éviter toute erreur de transmission, cette ligne doit être au niveau bas sur le ou les esclaves avant toute transmission et y rester pendant toute la durée de celle-ci. En outre, il est évident qu'à un instant donné, n'importe quel circuit peut être maître et n'importe quel autre esclave, mais il ne doit y avoir qu'un maître par transaction.

L'EUSRT

Constitution d'une trame UART

Une trame UART est constituée des bits suivants :



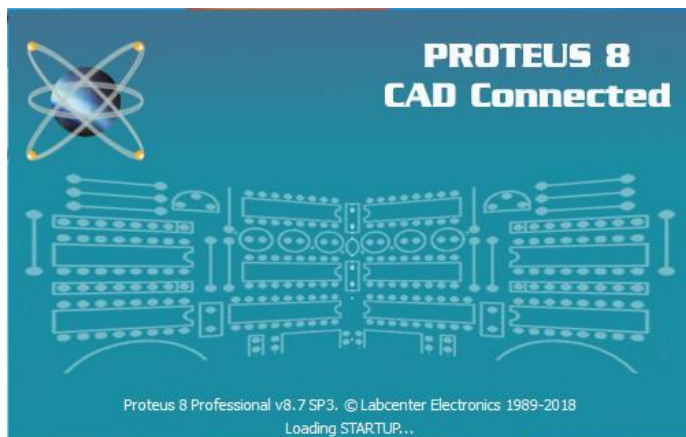
- Un bit de *start* toujours à 0 : servant à la synchronisation du récepteur
- Les données : la taille peut varier (généralement entre 5 et 9 bits)
- Éventuellement un bit de parité paire ou impaire
- Un bit de *stop* toujours à 1 (la durée peut varier entre 1, 1,5 et 2 temps bit)

Le niveau logique de repos est le 1

Le module EUSART du pic comprend les fonctionnalités suivantes :

- Transmission et réception asynchrones en duplex intégral
- Tampon d'entrée à deux caractères
- Tampon de sortie à un caractère
- Longueur de caractère programmable de 8 ou 9 bits
- Détection d'adresse en mode 9 bits
- Détection d'erreur de saturation du tampon d'entrée
- Détection d'erreur de cadrage de caractères reçue
- maître synchrone half-duplex
- Esclave synchrone half-duplex
- polarité d'horloge et de données programmable

Logiciel Proteus



Proteus est composé de deux logiciels principaux : *ISIS*, permettant entre autres la création de schémas et la simulation électrique, et *ARES*, destiné à la création de circuits imprimés.

I) Le logiciel MplabX

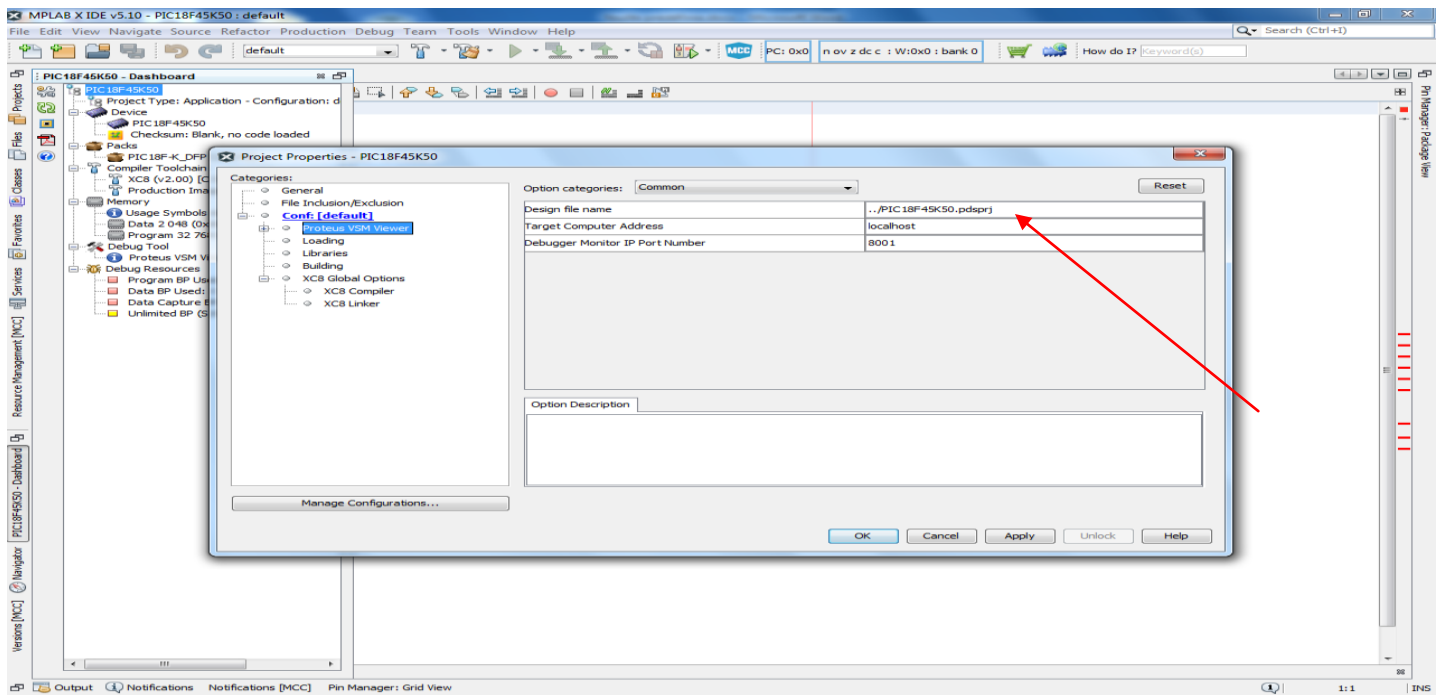


MPLAB est un environnement de développement intégré gratuit et gratuit destiné au développement d' applications intégrées sur les microcontrôleurs PIC et dsPIC . Il est développé par Microchip Technology.

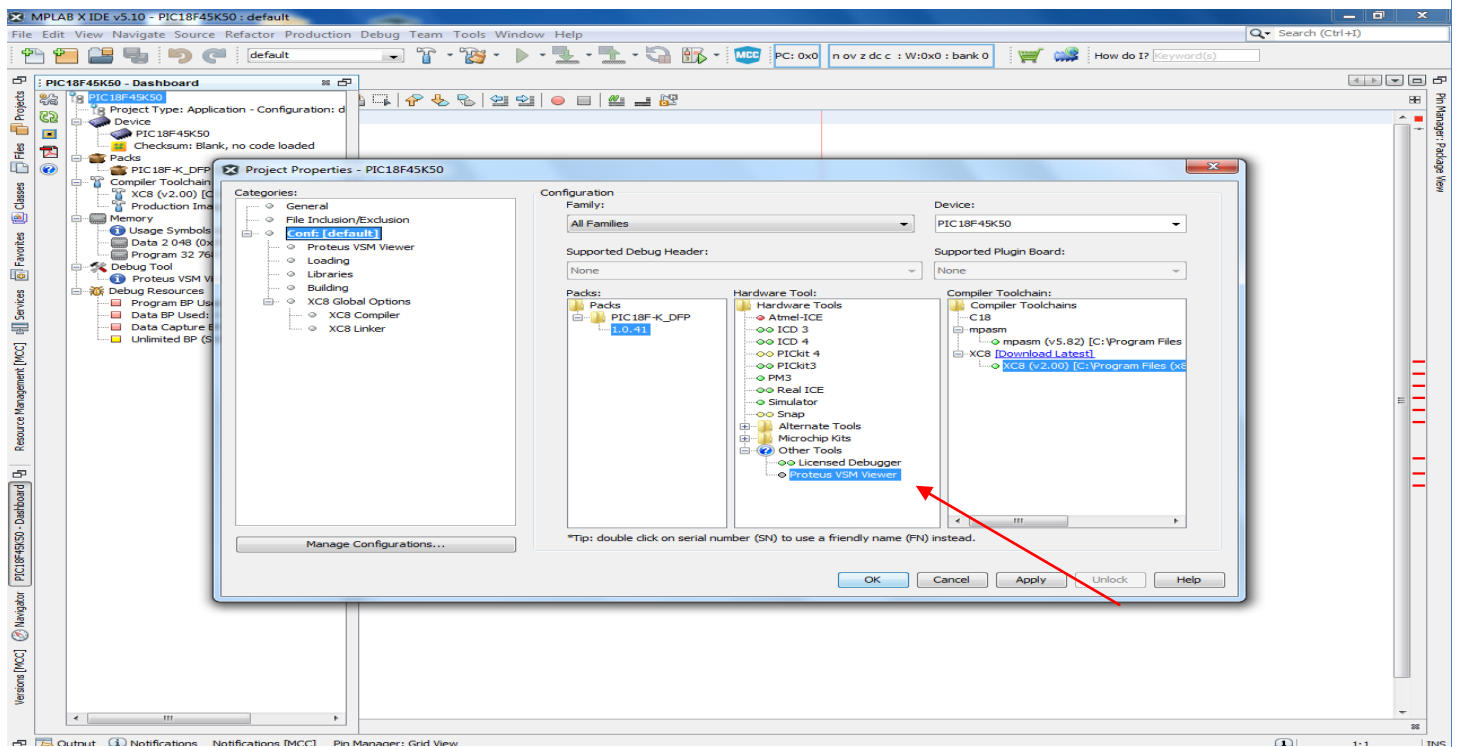
MPLAB 8.X est la dernière version de la technologie MPLAB IDE existante, construite sur mesure par Microchip Technology dans Microsoft Visual C ++ . MPLAB prend en charge la gestion de projet, l'édition, le débogage et la programmation de microcontrôleurs PIC 8 bits, 16 bits et 32 bits de Microchip. MPLAB ne fonctionne que sur Microsoft Windows .

Leds et relais

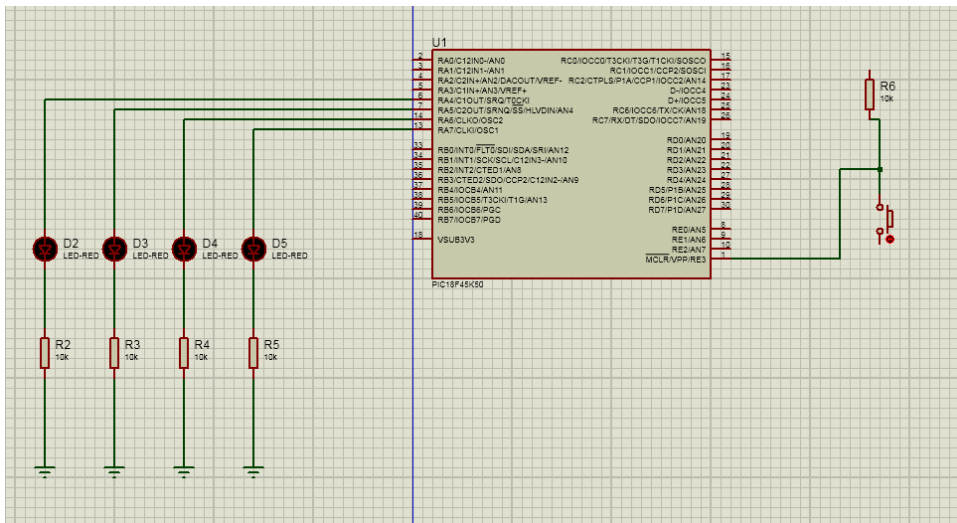
Pour commencer j'ai créé un schéma pour contrôler mes leds :



Pour voir les résultats de mon programme sur mon schéma sur Proteus, je dois effectuer des paramétrages en sélectionnant Proteus VSM Viewer.



Dans un premier temps j'ai fait un schéma sur Proteus afin de pouvoir tester mes leds avec un future programme que j'aurai créé sur un logiciel de dévoilement.

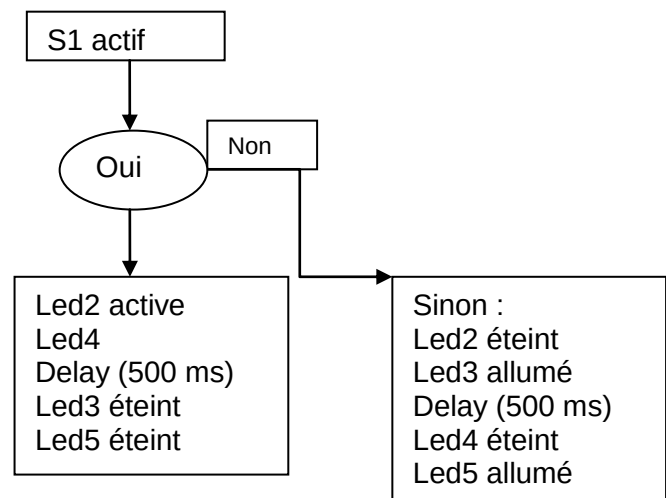


Le but de mon programme est de faire varier l'activation des Leds pour tester mon relai. Si mon bouton poussoir S1 est actif.

```
#include <xc.h>
#include <stdio.h>
#include "mcc_generated_files/mcc.h"

void main(void) {
    SYSTEM_Initialize();

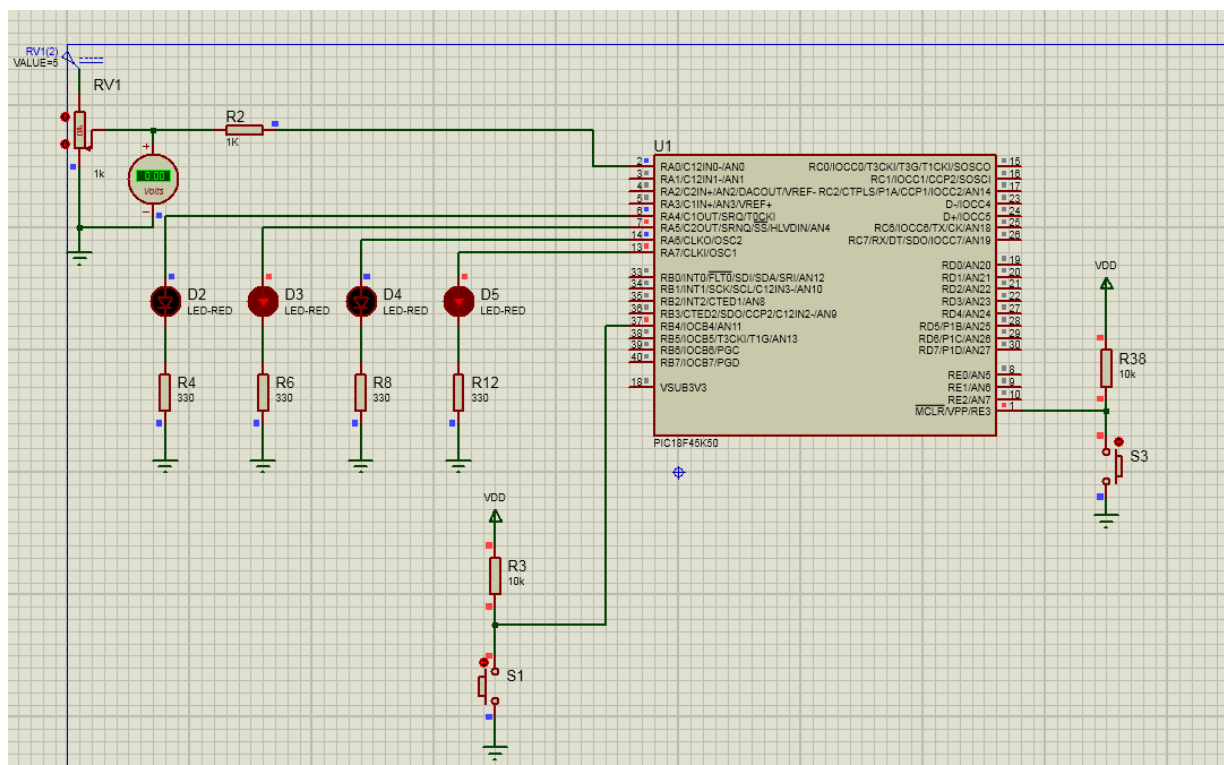
    while(1)
    {
        if (S1_GetValue() == 0)
        {
            D2_Toggle();
            D4_Toggle();
            __delay_ms(500);
            D3_Toggle();
            D5_Toggle();
        }
        else
        {
            D2_SetLow();
            D3_SetLow();
            D4_SetLow();
            D5_SetLow();
        }
    }
}
```



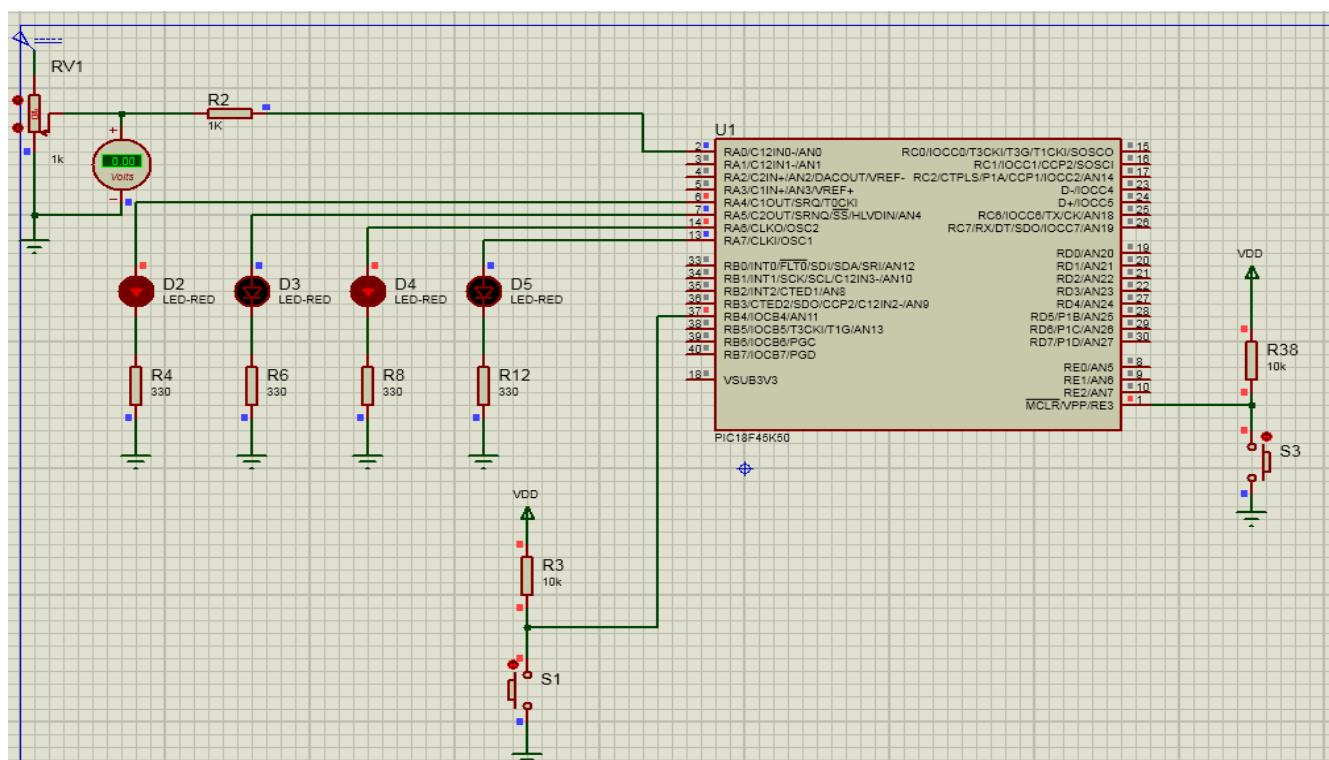
CAS 1 = Si le bouton poussoir S1 est actif, les leds D2 et D4 sont actives, puis il y a un délai de 500 ms et les leds D3 et D5 s'activent à leurs tours.

Cas 2 = Si le bouton poussoir S1 est inactif les leds D3 et D5 sont éteintes, les leds D2 et D4 reste fixe le relais de commute pas.

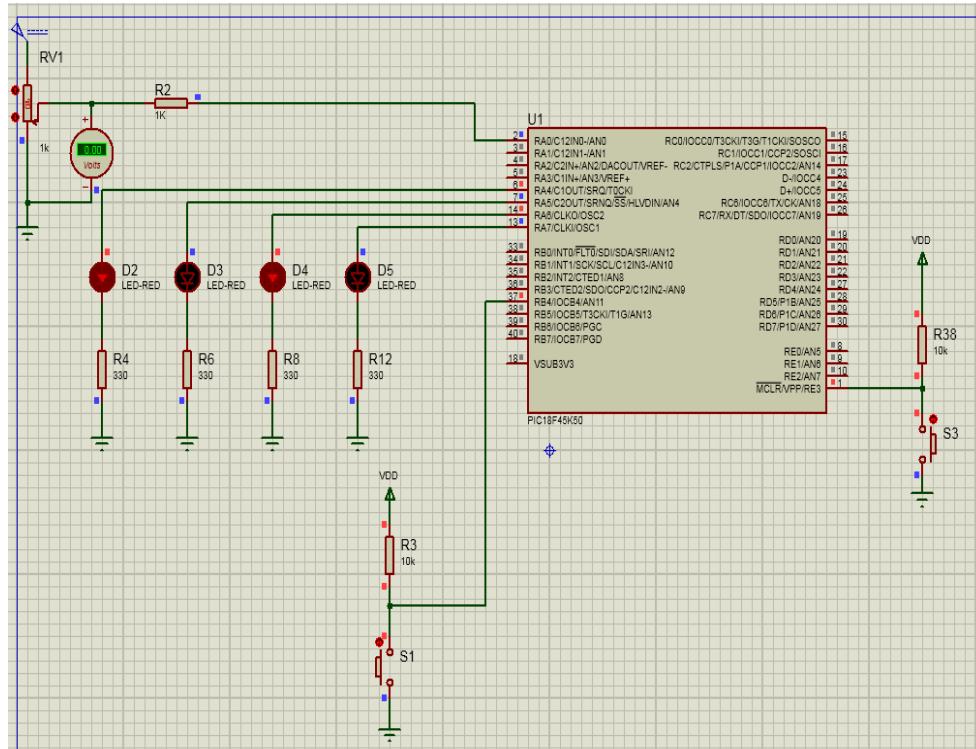
CAS 1 = Si le bouton poussoir S1 est actif, les leds D2 et D4 sont actives, puis il y a un délai de 500 ms et les leds D3 et D5 s'activent à leurs tours.



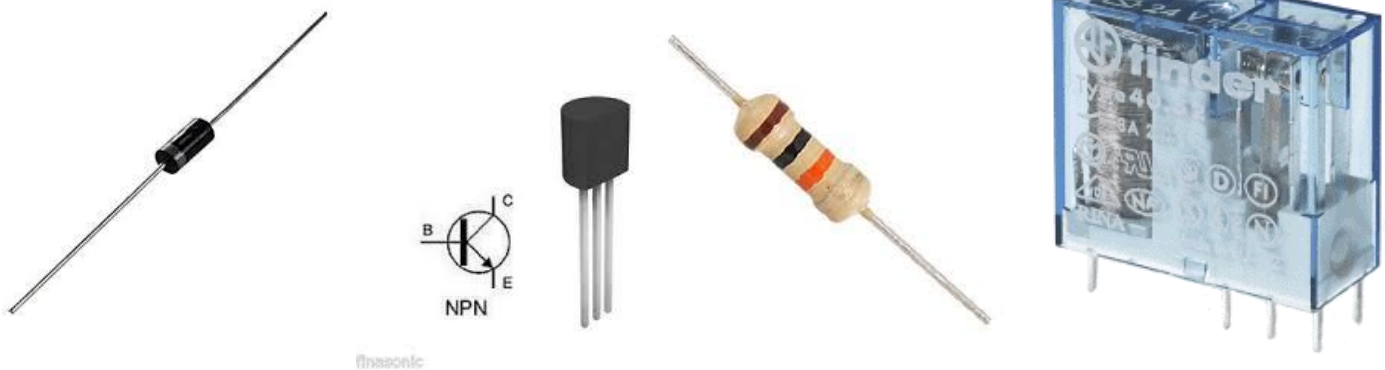
Délais de 500 ms



Cas 2 = Si le bouton poussoir S1 est inactif les leds D3 et D5 sont éteintes, les leds D2 et D4 reste fixe le relais de commute pas.



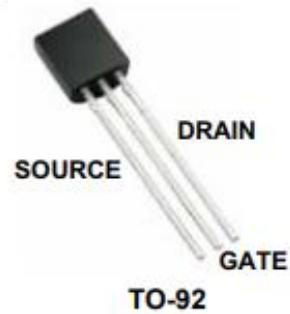
J'ai ensuite intégré un relais 2RT 12V, pour sa il m'a fallu un transistor NPN (actif à l'état bas) une diode et une résistance.



Le transistor

Le transistor est un *tripôle* correspondant à un assemblage de semi-conducteurs N et P. le transistor NPN que je vais inclure dans mon projet est un transistor VN2222

Pin Configuration



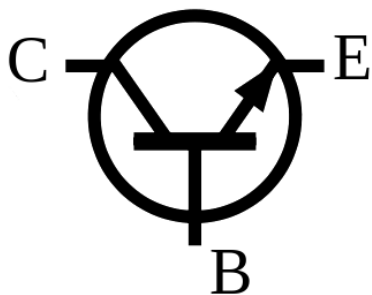
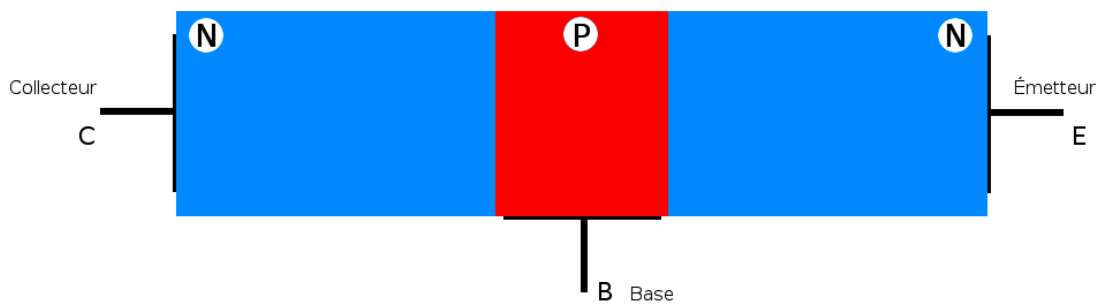
Product Marking



Package may or may not include the following marks: Si or 

TO-92

Schéma et symbole du transistor :

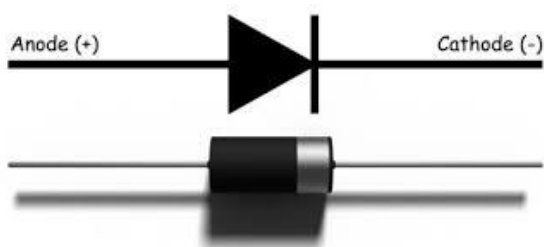


Fonctionnement d'un transistor

Le transistor dispose de trois bornes :

- Le collecteur ;
- La base ;
- L'émetteur.

Le transistor est un composant électronique qui est utilisé dans la plupart des circuits électroniques (circuits logiques, amplificateur, stabilisateur de tension, modulation de signal, etc.) aussi bien en basse qu'en haute tension.

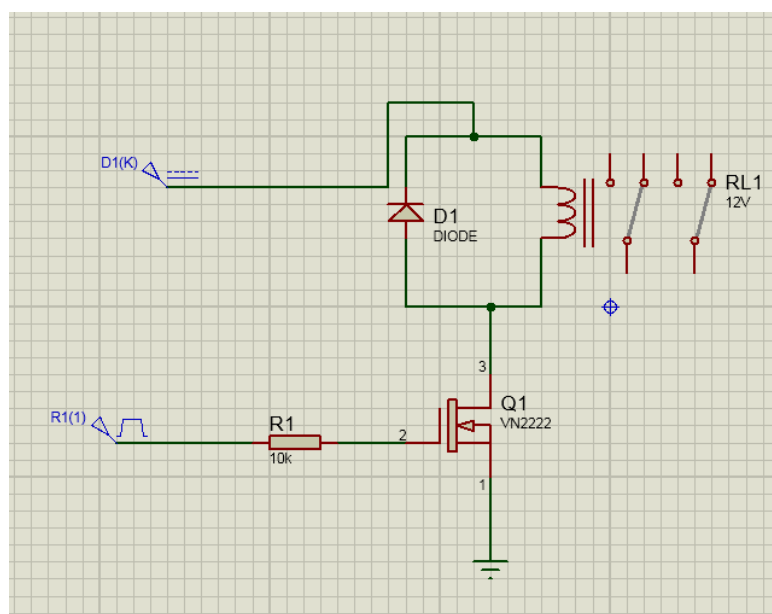


La diode à une polarité la couche noire représente l'Anode (+) et la couche grise représente la Cathode (-)

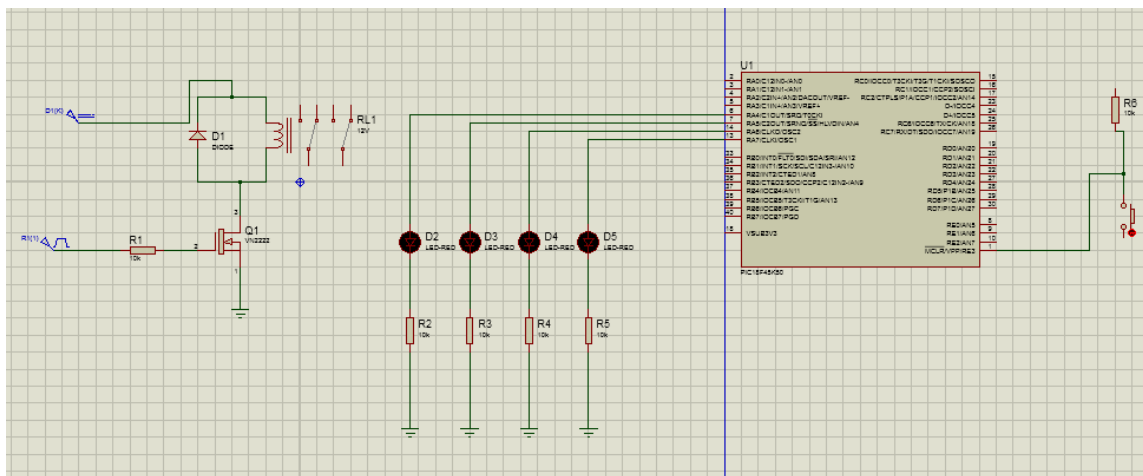
Rôle de la diode de roue libre

Si on ne met pas la diode de roue libre aux bornes d'un relais, à courant continu, le courant ne peut "aller nulle part", va décroître ainsi très vite et créer une surtension qui détruira le transistor Q1. La diode de roue libre enlève cette surtension en offrant un passage pour le courant.

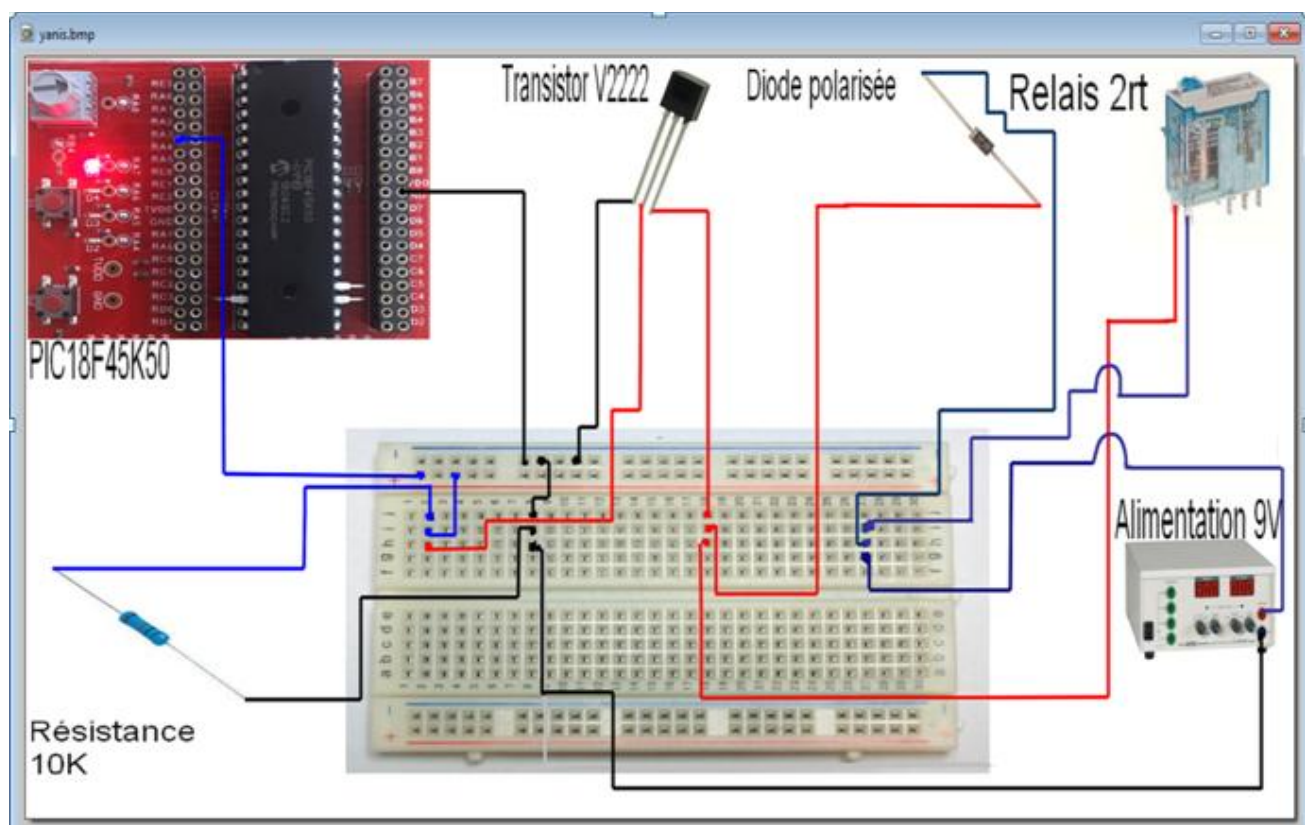
Un relais électromécanique est un organe électrique permettant de dissocier la partie puissance de la partie commande : il permet l'ouverture et la fermeture d'un circuit électrique par un second circuit complètement isolé (isolation galvanique) et pouvant avoir des propriétés différentes.



Ensuite je l'ai inclus dans mon schéma :



Une fois les tests effectués sur Proteus, j'ai réalisé le câblage avec une carte d'essai.



.Capteur Lm35.

Le capteur de température LM35 est un capteur analogique de température fabriqué par Texas Instruments. Il est extrêmement populaire en électronique, car précis, peu coûteux, très simple d'utilisation et d'une fiabilité à toute épreuve.

Le capteur de température LM35 est capable de mesurer des températures allant de -55°C à +150°C dans sa version la plus précise et avec le montage adéquat, de quoi mesurer n'importe quelle température.

La sortie analogique du capteur est proportionnelle à la température. Il suffit de mesurer la tension en sortie du capteur pour en déduire la température. Chaque degré Celsius correspond à une tension de +10mV.

Le LM35 est un capteur à circuit intégré qui peut être utilisé pour mesurer la température avec un signal électrique proportionnel à la température (en ° C)

Le LM35 génère une tension de sortie plus élevée que les Thermocouples et ne peut exiger que la tension de sortie soit amplifiée.

Il a une tension de sortie est proportionnelle à la température Celsius. Le facteur d'échelle est 0.1V/°C

Le LM35 ne nécessite pas de calibration externe ou de taille et

Maintient une précision de + / -0,4 °C à température ambiante et + / - 0,8 °C sur une plage de 0 °C à +100 °C.

Le capteur a une sensibilité de 10 mV /°C

$$\text{Température (° C)} * V_{\text{out}} = (100^{\circ}\text{C} / \text{V})$$

Donc, si Vout est 1V, puis, la température = 100 ° C La tension de sortie varie Linéairement avec la température.

Reférenceur de tension Lm40.

Une référence de tension est un composant qui délivre une tension très stable, qui évolue très peu en fonction de l'âge du composant, de la température ambiante et des conditions d'utilisation (tension d'alimentation et courant consommé par la charge).

On utilise ce type de composant dans des circuits où on a besoin d'une tension de référence très précise, tel que des appareils de mesure ou des convertisseurs analogique/numérique. Selon l'usage, on peut se contenter d'une référence de tension standard (une simple diode peut parfois suffire) ou au contraire opter pour un composant de "qualité supérieure".

J'ai ensuite fait le montage pour relever la température, nous avons une nouvelles fois utilisé MplabX.

PIC18F4550 ADC

- Lorsque nous connectons des capteurs au microcontrôleur, la sortie du capteur est souvent analogique. Mais le microcontrôleur traite les signaux numériques. Nous utilisons donc un ADC entre le capteur et le microcontrôleur. Il convertit un signal analogique en numérique et le transmet au microcontrôleur.

Voyons maintenant ADC dans PIC18F4550.

- Le PIC18f4550 est doté d'un convertisseur analogique-numérique à 10 canaux et 10 bits.
- 13 canaux de l'ADC sont nommés AN0-AN12. Cela signifie que nous pouvons connecter 13 capteurs différents en même temps.
- ADC 10 bits signifie :
- Il donnera un nombre numérique compris entre 0 et 1023 (2^{10}).

Calcul de la valeur de sortie numérique

$$ADCResolution = \frac{V_{ref}}{2^n - 1}$$

$$DigitalOutput = \frac{V_{in}}{Resolution}$$

Pour garder les choses simples, considérons que Vref est 5V,

Pour 0Vin valeur numérique 0 / p = 0

Pour 5Vin valeur numérique 0 / p = 1023 (10 bits)

Pour valeur numérique 0 / p de 2,5Vin = 512 (10 bits)

```
#include <pic18.h>
#include <xc.h>
#include "mcc_generated_files/mcc.h"

int valeurLM35;
float temperature;
char buffer[20];

/*
| Main application
*/
void main(void)
{
    unsigned int result;
    // Initialize the device
    SYSTEM_Initialize();
    ADC_SelectChannel(1);

    // Add your application code
    valeurLM35 = ADC_GetConversion(1);
    temperature = (float)valeurLM35*(2.5/1024)*100;
    Nop();
    // Add your application code
    // printf("La valeur de temperature est : %d \r\n",temperature);
    sprintf(buffer,"%0.3g",temperature);
    printf(buffer);
    __delay_ms(500);
}
```

Mon programme utilise le convertisseur analogique numérique du microcontrôleur, Ensuite il exécute le calcul le calcul $(2.5 / 1024) * 100$ pour transformer la tension en température.

A noter : 2.5 est un la tension de référence Lm4040.

Ensuite Grace à mon programme j'ai pu relever la température.

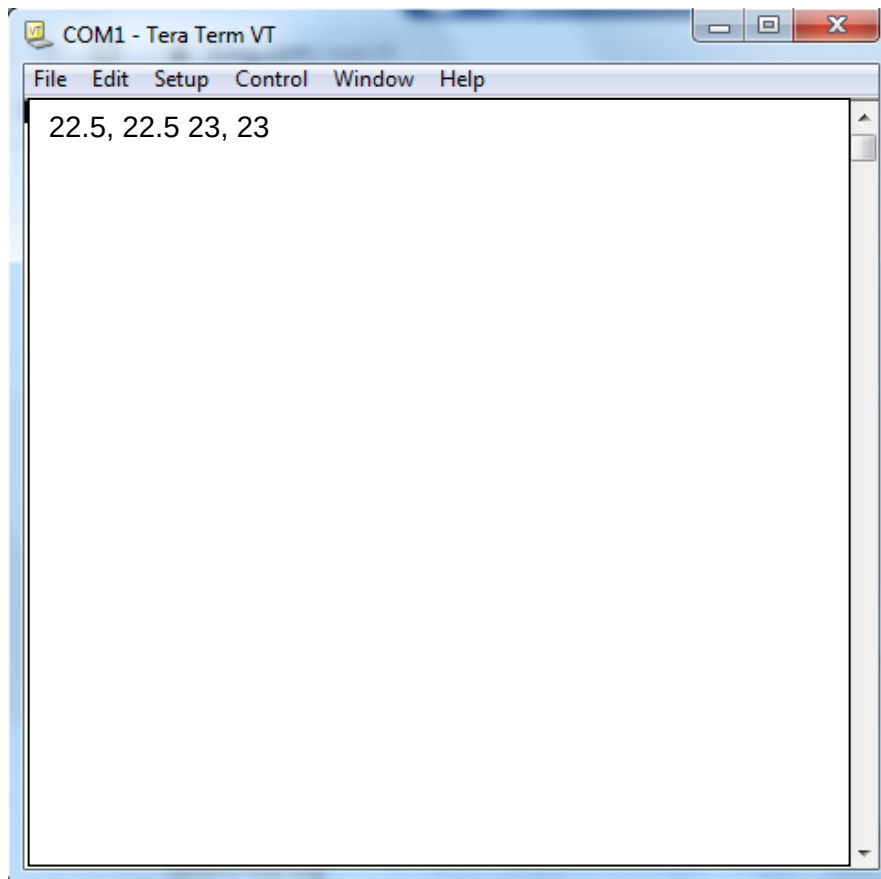


Schéma structurel

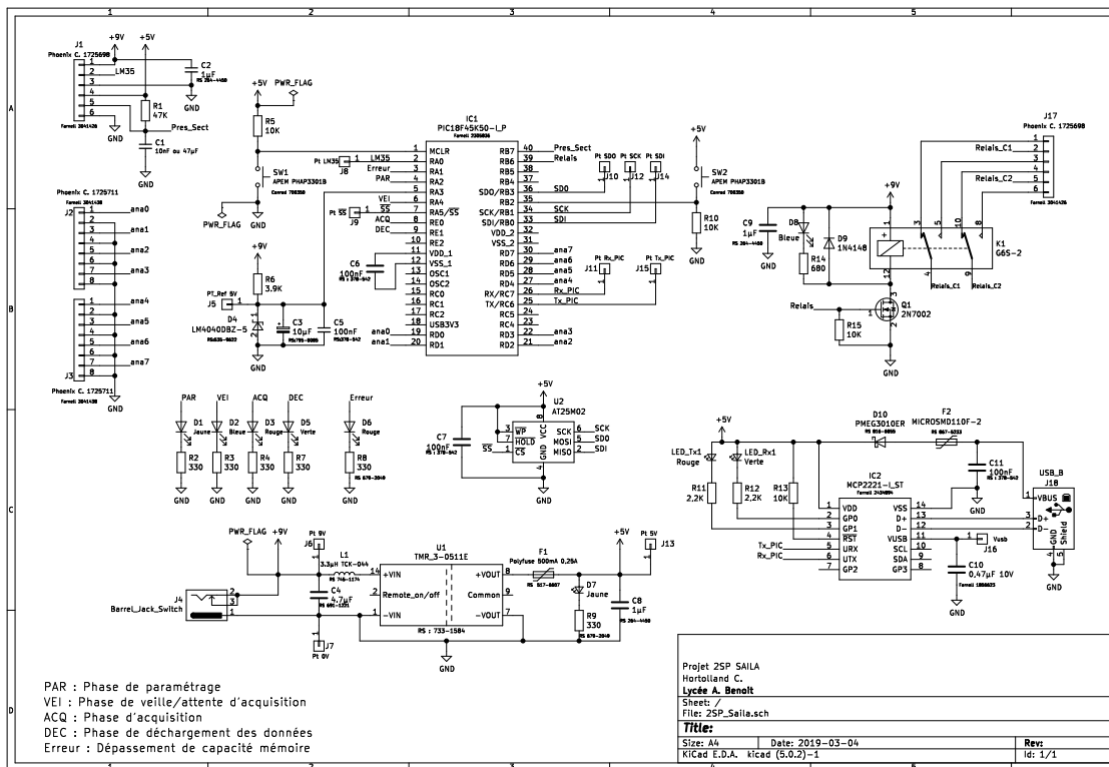
Nous avons ensuite fait un schéma structurel sur le Logiciel Kicad



Kicad est un logiciel multiplateforme. Il utilise la bibliothèque graphique libre wxWidgets. Il permet de réaliser toutes les étapes nécessaires à la conception d'un circuit imprimé : Réalisation du schéma électronique, association des empreintes, routage, export au format gerber.

La visualisation 3D : il est possible de visualiser une réalisation en 3 dimensions.

Schéma structurel Version 1 :

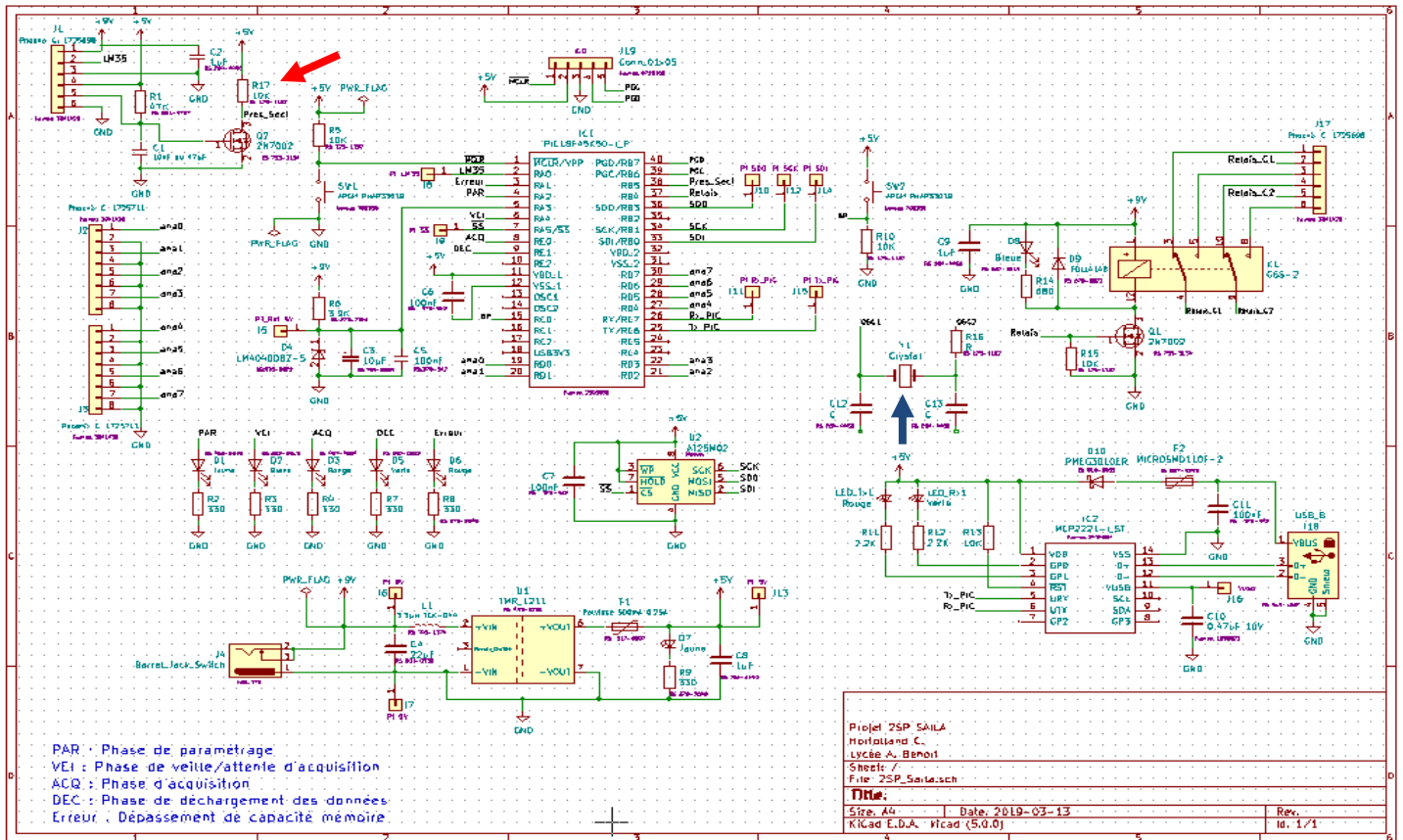


Pour faciliter la recherche d'information j'ai créé une documentation technique qui référence les différentes caractéristiques des broches, qui évoluera en même temps que notre projet.

Désignation	Noms	Broches	Numéros des Broches
Reset	MCLR	MCLR/VPP	1
Capteur de température (pt Im35)	Pt Im35	RA0	2
Capteur (tension de référence)	Pt Im4040	RA3	5
Signal mode Veille	VEI	RA4	6
Signal Phase déchargement des données	DEC	RE1	9
Chip select	Pt SS	RA5	7
Pt SDI	SDI	SDI/RB0	33
Pt SCK	SCK	SCK/RB1	34
Pt SDO	SDO	SDO/RB3	36
Tension	5V	VDD_1	11
Condensateur	100 nF	VSS_1	12
Entrée analogique	ana0	RD0	19
Entrée analogique	ana1	RD1	20
Entrée analogique	ana2	RD2	21
Entrée analogique	ana3	RD3	22
Entrée analogique	ana4	RD4	27
Entrée analogique	Ana5	RD5	28
Entrée analogique	Ana6	RD6	29
Entrée analogique	Ana7	RD7	30
Emission pt TX_pic	TX	TX/RC6	25
Réception pt RX_pic	RX	RX/RC7	26
Le Relais	relais	RB4	37
Présence secteur	Press sec	RB5	38
Liaison série PGC (programmer la carte)	PGC	RB6	39
Liaison série PGD (programmer la carte)	PGD	RB7	40
Bouton poussoir	bpgo	RC0	35

J'ai défini les couleurs par rapport au type de broches pour une lecture beaucoup plus fluide.

Après certaines complications, nous avons décidé d'ajouter quelques éléments dans notre schéma structurel.



Nous avons décidé de rajouter une résistance de pull up pour avoir un état bas du signal.

Nous avons aussi décidé de rajouter un Quartz pour envoyer un signal qui a une fréquence plus précise.

Kicad empreinte et routes

Une fois mon schéma structurel achevé, j'ai inséré les empreintes Kicad.

Kicad fournit des bibliothèques d'empreintes afin que l'on puisse associer facilement une empreinte à la majorité des composants. Il suffit donc de trouver un composant réel ayant la même empreinte que celles proposées par le logiciel. Néanmoins, il est fréquent que l'on ait à utiliser un composant précis dont l'empreinte n'existe pas dans les bibliothèques fournies par défaut avec Kicad. Dans ce cas-là il faudra inclure une nouvelle empreinte dans Kicad.

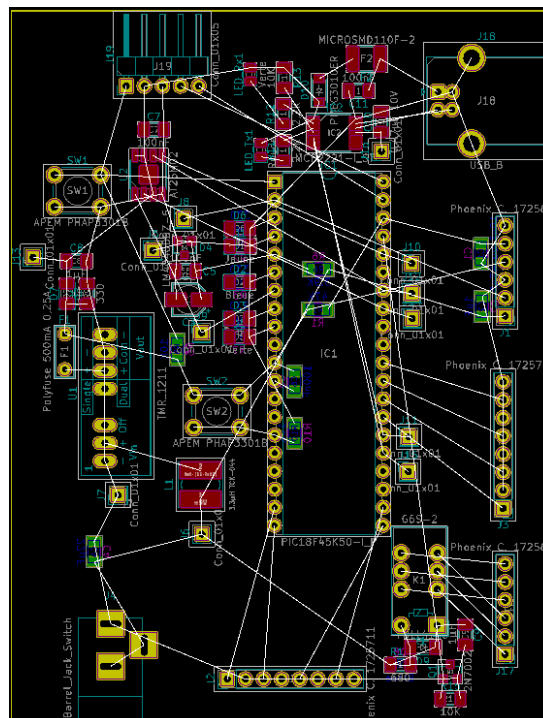
0_ModulesProject	1	C1 -	10nF ou 47pF : Capacitor_SMD:C_1206_3216Metric	1	0_ModulesProject:Arduino_MKR_WAN_Socket
Battery	2	C2 -	1pF : Capacitor_SMD:C_1206_3216Metric	2	0_ModulesProject:Arduino_UNO_R3_2
Button_Switch_Keyboard	3	C3 -	10pF : Capacitor_SMD:CP_Elec_4x5.7	3	0_ModulesProject:Arduino_Uno_Shield
Button_Switch_SMD	4	C4 -	22pF : Capacitor_SMD:C_1206_3216Metric	4	0_ModulesProject:Arduino_Uno_Shield_V2_Stackable
Button_Switch_THT	5	C5 -	100nF : Capacitor_SMD:C_1206_3216Metric	5	0_ModulesProject:BARREL_JACK
Buzzer_Beeper	6	C6 -	100nF : Capacitor_SMD:C_1206_3216Metric	6	0_ModulesProject:Bornier_5-08_2cts_Weidmuller_Noir
Calibration_Scale	7	C7 -	100nF : Capacitor_SMD:C_1206_3216Metric	7	0_ModulesProject:Bornier_5-08_3cts_Weidmuller_Oran
Capacitor_SMD	8	C8 -	1pF : Capacitor_SMD:C_1206_3216Metric	8	0_ModulesProject:Bornier_Phoenix_Contact_x6cts_172
Capacitor_THT	9	C9 -	1pF : Capacitor_SMD:C_1206_3216Metric	9	0_ModulesProject:Bornier_Phoenix_Contact_x8cts_172
Capacitor_Tantalum_SMD	10	C10 -	0,47pF 10V : Capacitor_SMD:C_1206_3216Metric	10	0_ModulesProject:Buzzer_KingstateKPEG242
Connector	11	C11 -	100nF : Capacitor_SMD:C_1206_3216Metric	11	0_ModulesProject:Choke_TCK-062
Connector_Audio	12	D1 -	Jaune : LED_SMD:LED_1206_3216Metric	12	0_ModulesProject:COMBICON_FTDA_5mm_4_way
Connector_BarrelJack	13	D2 -	Bleue : LED_SMD:LED_1206_3216Metric	13	0_ModulesProject:COMBICON_FTDA_5mm_10_way
Connector_Card	14	D3 -	Rouge : LED_SMD:LED_1206_3216Metric	14	0_ModulesProject:DCDC-Conv_TRACO_TMR_2411
Connector_Coaxial	15	D4 -	LM4040DBZ-5 : Package_TO_SOT_SMD:SOT-23	15	0_ModulesProject:DIP-20_XBee
Connector_DIN	16	D5 -	Verte : LED_SMD:LED_1206_3216Metric	16	0_ModulesProject:DSUB-9_Male_Horizontal_P2.77x2.54
Connector_Daub	17	D6 -	Rouge : LED_SMD:LED_1206_3216Metric	17	0_ModulesProject:Embase_AMPMODU_HE14_Angle_droit_4
Connector_FFC-FPC	18	D7 -	Jaune : LED_SMD:LED_1206_3216Metric	18	0_ModulesProject:Embase_femelle_angle-droit_6pts
Connector_HDMI	19	D8 -	Bleue : LED_SMD:LED_1206_3216Metric	19	0_ModulesProject:Embase_Grove_Horizontale
Connector_Harwin	20	D9 -	FDLL4148 : Diode_SMD:D_MiniMELF	20	0_ModulesProject:GSG-SMA-EDGE_BTS_SN
Connector_Hirose	21	D10 -	PMEG3010ER : Diode_SMD:D_SOD-123	21	0_ModulesProject:GSG-SMA-EDGE_BTS_SN_filete
Connector_IDC	22	F1 -	Polyfuse 500mA 0,25A : 0_ModulesProject:Polyswitch_RXE020	22	0_ModulesProject:he10-14c
Connector_JAE	23	F2 -	MICROSMD110F-2 : 0_ModulesProject:Polyswitch_MICROSMD110F-2	23	0_ModulesProject:he10-20c
Connector_JST	24	IC1 -	PIC18F45K50-I_P : Package_DIP:DIP-40_W15.24mm	24	0_ModulesProject:Header_4x1_stackable
Connector_Molex	25	IC2 -	MCP2221-I_ST : Package_SO:TSSOP-14_4.4x5mm_P0.65mm	25	0_ModulesProject:Keystone_1060_1x2032-CoinCell
Connector_Multicomp	26	J1 -	Phoenix C. 1725698 : 0_ModulesProject:Bornier_Phoenix_Contact_x6cts_172	26	0_ModulesProject:Kingbright_KA-4040
Connector_PCBEde	27	J2 -	Phoenix C. 1725711 : 0_ModulesProject:Bornier_Phoenix_Contact_x8cts_17	27	0_ModulesProject:Kingbright_KA-4040_BLUE
Connector_Phoenix_GMSTB	28	J3 -	Phoenix C. 1725711 : 0_ModulesProject:Bornier_Phoenix_Contact_x8cts_17	28	0_ModulesProject:Kingbright_KA-4040_GREEN
Connector_Phoenix_MC	29	J4 -	Barrel_Jack_Switch : 0_ModulesProject:BARREL_JACK	29	0_ModulesProject:Kingbright_KA-4040_RED
Connector_Phoenix_MC_HighVoltage	30	J5 -	Conn_01x01 : Connector_PinHeader_2.54mm:PinHeader_1x01_P2.54mm_Vv	30	0_ModulesProject:Kingbright_KA-4040_YELLOW
Connector_Phoenix_MSTB	31	J6 -	Conn_01x01 : Connector_PinHeader_2.54mm:PinHeader_1x01_P2.54mm_Vv	31	0_ModulesProject:Polyswitch_MICROSMD110F-2
Connector_Pin	32	J7 -	Conn_01x01 : Connector_PinHeader_2.54mm:PinHeader_1x01_P2.54mm_Vv	32	0_ModulesProject:Polyswitch_RXE020
Connector_PinHeader_1.00mm	33	J8 -	Conn_01x01 : Connector_PinHeader_2.54mm:PinHeader_1x01_P2.54mm_Vv	33	0_ModulesProject:Relay_DPDT_Omron_G6S-2_mod
Connector_PinHeader_1.27mm	34	J9 -	Conn_01x01 : Connector_PinHeader_2.54mm:PinHeader_1x01_P2.54mm_Vv	34	0_ModulesProject:SHT21
Connector_PinHeader_2.00mm	35	J10 -	Conn_01x01 : Connector_PinHeader_2.54mm:PinHeader_1x01_P2.54mm_Vv	35	0_ModulesProject:Socket_Strip_Straight_2x20
Connector_PinHeader_2.54mm	36	J11 -	Conn_01x01 : Connector_PinHeader_2.54mm:PinHeader_1x01_P2.54mm_Vv	36	0_ModulesProject:Support_pile_bouton_Keystone_1060
Connector_PinSocket_1.00mm	37	J12 -	Conn_01x01 : Connector_PinHeader_2.54mm:PinHeader_1x01_P2.54mm_Vv	37	0_ModulesProject:transfovlf
Connector_PinSocket_1.27mm	38	J13 -	Conn_01x01 : Connector_PinHeader_2.54mm:PinHeader_1x01_P2.54mm_Vv	38	0_ModulesProject:USB-B_Molex_67068-7041_Horizontal
Connector_PinSocket_2.00mm	39	J14 -	Conn_01x01 : Connector_PinHeader_2.54mm:PinHeader_1x01_P2.54mm_Vv		
Connector_PinSocket_2.54mm	40	J15 -	Conn_01x01 : Connector_PinHeader_2.54mm:PinHeader_1x01_P2.54mm_Vv		
Connector_RJ	41	J16 -	Conn_01x01 : Connector_PinHeader_2.54mm:PinHeader_1x01_P2.54mm_Vv		
Connector_SATA_SAS	42	J17 -	Phoenix C. 1725698 : 0_ModulesProject:Bornier_Phoenix_Contact_x6cts_172		
Connector_Samtec	43	J18 -	USB B : 0_ModulesProject:USB-B_Molex_67068-7041_Horizontal		

Librairie

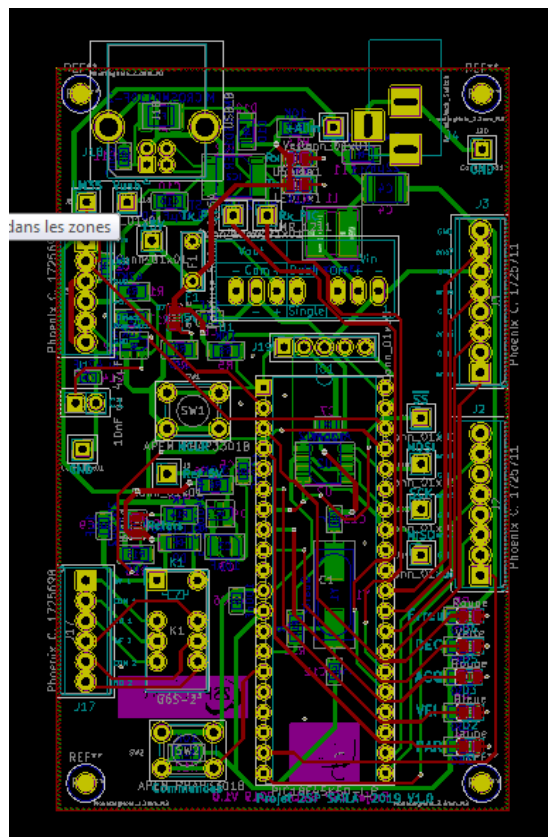
Les empreintes que
j'ai insérées

Librairie

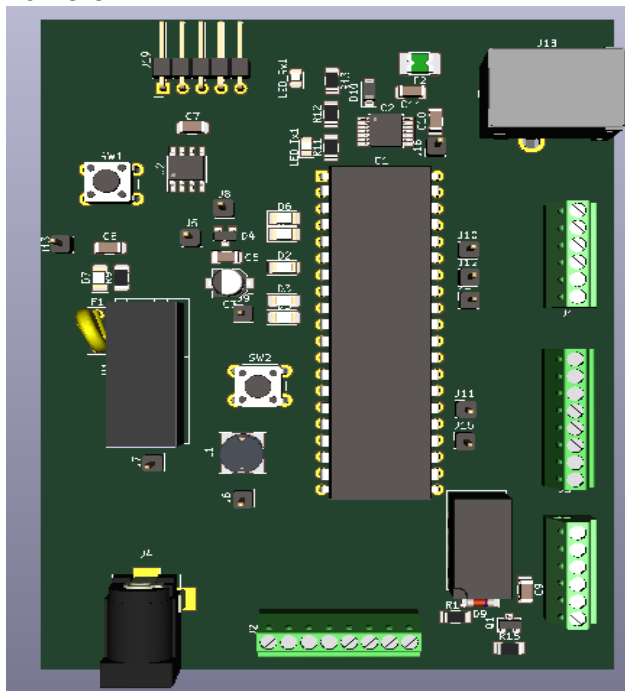
J'ai alors proposé une première version de mon rouage, pour cette première version j'ai mis en avant le routage et essayer de créer une carte où les routes étaient facilement réalisables.



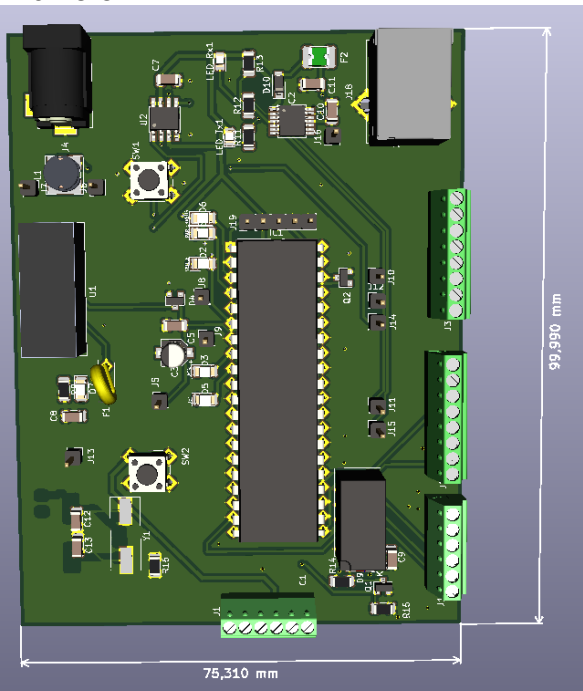
Après une réunion nous avons décidé de changer de place les connecteurs et les borniez pour les mettre du même côté afin d'améliorer l'esthétique, Puis j'ai créé les routes, mon Professeur à ajouter quelques améliorations et nous avons fait fabriquer la carte.



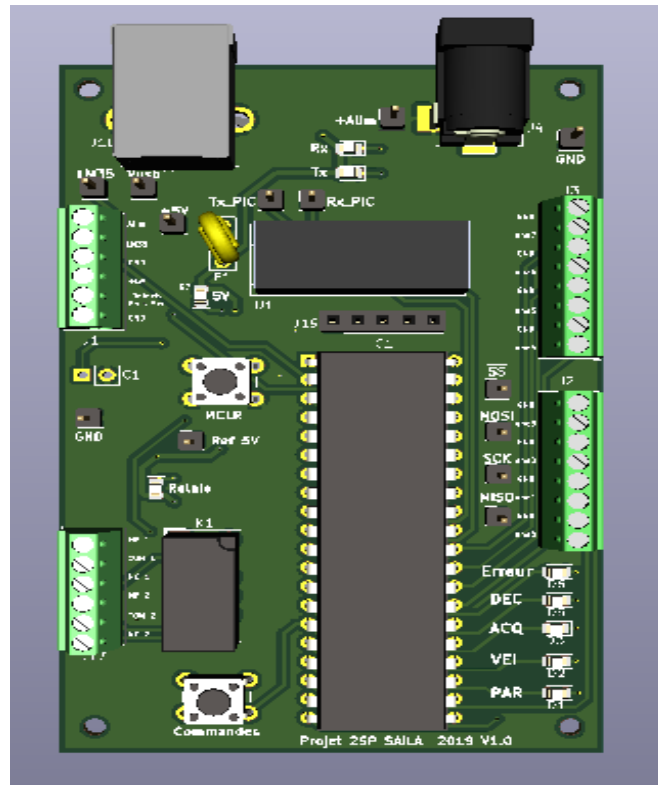
Numéro 1 :



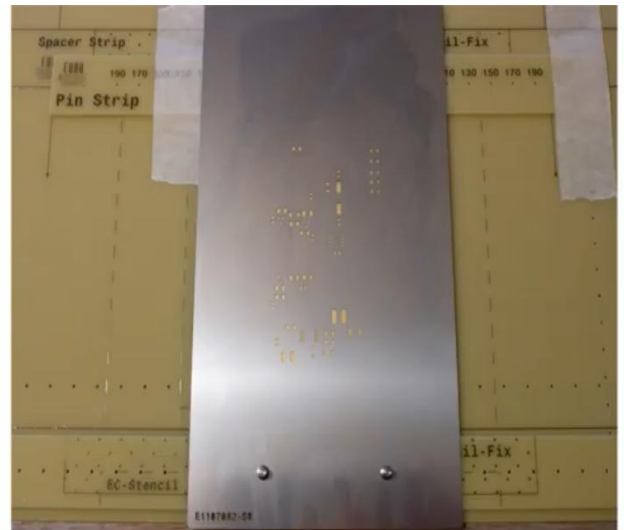
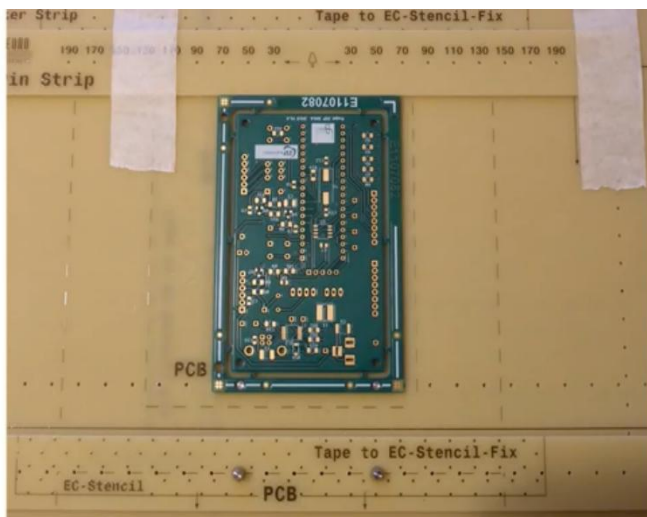
Numéro 2 :



Numéro 3 :



Document de fabrication



Pour Maintenir la Carte j'ai utilisé deux autres PCB et une fois qu'une fois que j'ai les bonnes empreintes, je suis prêt à mettre la patte à braser



J'ai déposé de la pâte à braser Multicore RA10

Réglages effectués (après tests) pour cette crème : Prog 2

- Préchauffage 90°C
- 2 minutes 30 secondes montée à 220°C
- 1 minute montée à 250°C (refusion)
- 1 minute 50 secondes : refroidissement (Prendre tout de même des pinces brucelles pour sortir la carte)

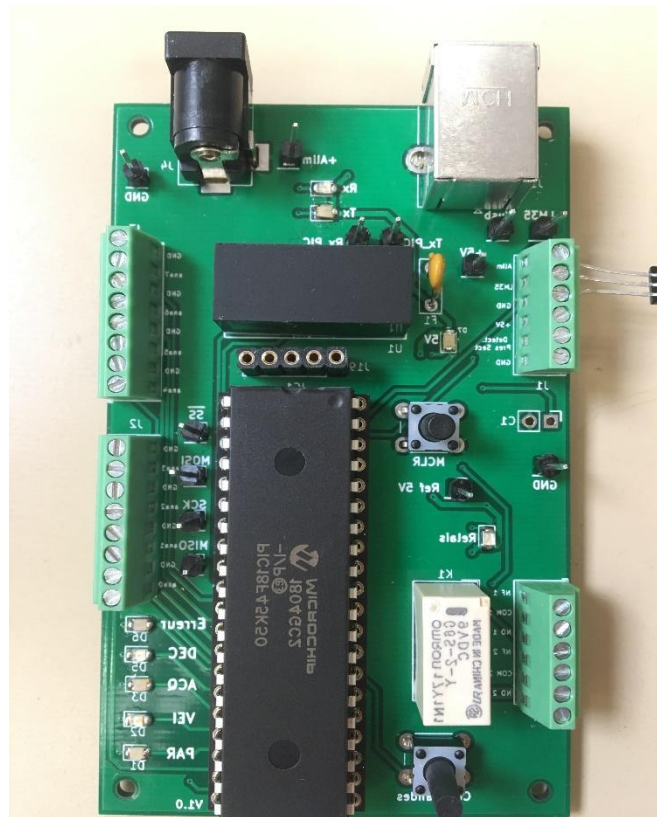
Avec une spatule et nettoyer les soudures "classiques" avec de l'alcool isopropanol.



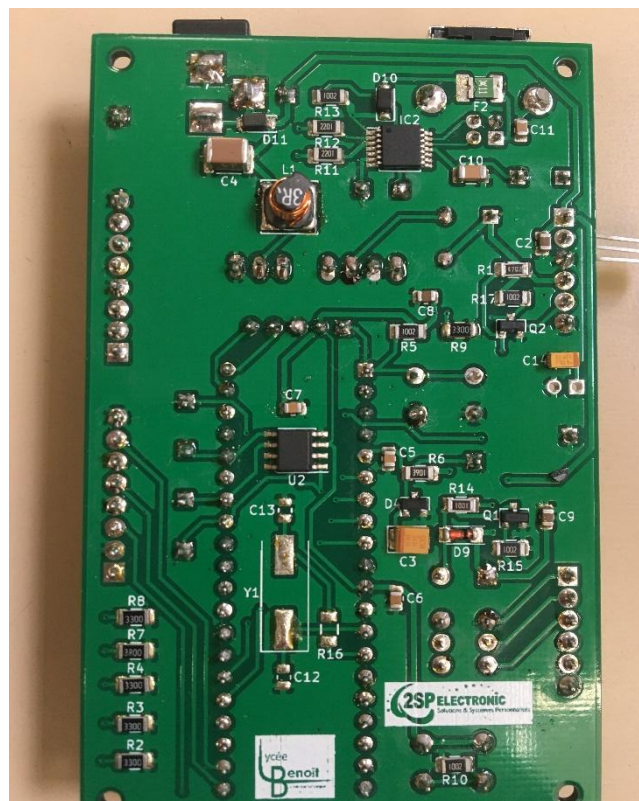
Une fois le travail effectué j'ai utilisé une Loupe binoculaire bi-éclairante pour vérifier l'état de mes soudures.

Carte finale :

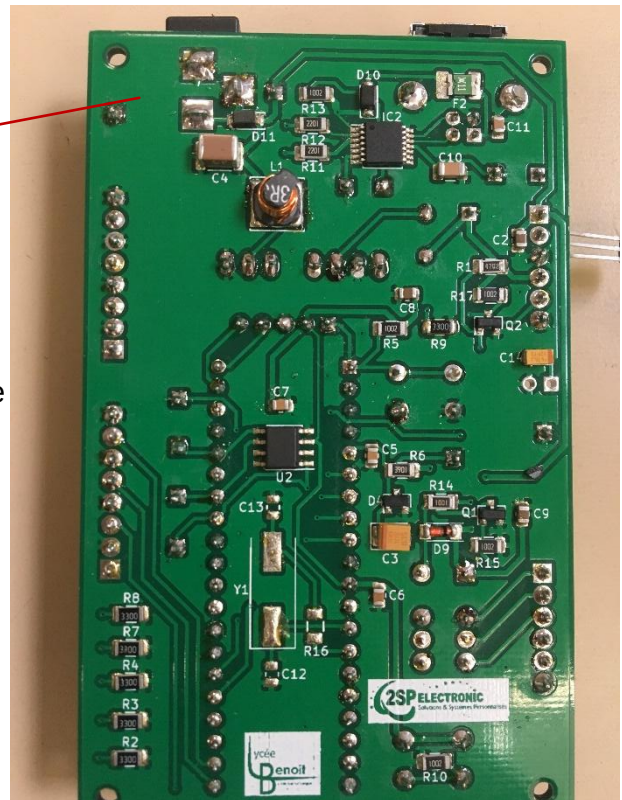
Carte recto :



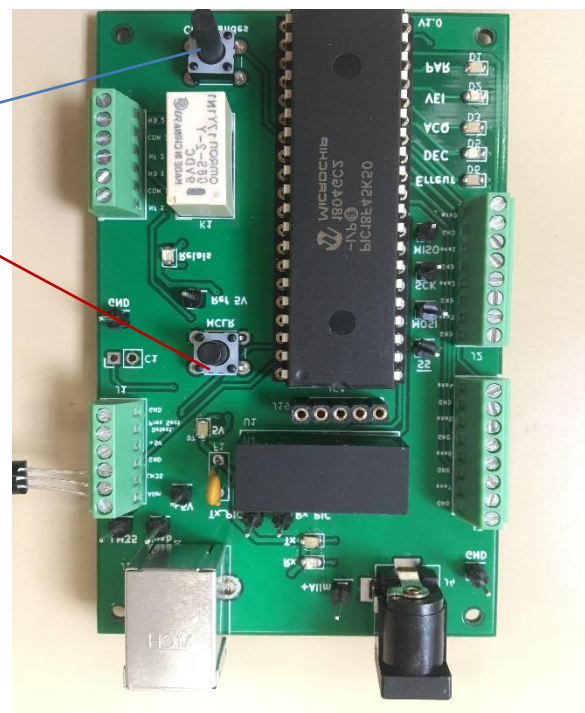
Carte verso :



- Pour mes tests je me suis servie d'un multimètre en Mode tension continue, j'ai dans un premier temps vérifié ma tension alimentation 9 v.
- Après avoir vérifié ma tension d'alimentation J'ai vérifié ma tension 5 v et ensuite avec un fils je l'ai Raccorder à mes leds et mon relai pour simuler un ordre Les leds s'allument et mon relais commute.



- Quand j'appuie sur le bouton Sw2 la tension passe de 0v a 5 V et sur le bouton Mclr la tension passe de 5v a 0v.



Annexes

Contient les fiches recettes et le mode d'emploi.

FICHE RECETTE, QUALIFICATION DU SYSTEME

Nom du système :	Système SAILA	Code de la campagne de test :	2019-05
Technicien 1 :	Aloïs BRUCOLERI	Date :	23/05/2019
Technicien 2 :	Corentin HERY		
Technicien 3 :	Benjamin DIMIER		
Technicien 4 :	Yanis FOUNAS		
Technicien 5 :	Nicolas CHARRIER		

IDENTIFICATION DU SCENARIO

Identification du scénario de recette

Titre :	Récupération des datas d'une mission précédentes
Objectif du scénario :	Récupérer les paramètres d'une mission précédente

CONDITIONS INITIALES NECESSAIRES POUR EFFECTUER LA RECETTE

Carte SAILA connectée au logiciel (voir recette 2019-01).
La carte doit se trouver dans l'état Paramétrage (voir recette 2019-07).

EXECUTION DU TEST

Description courte :	Appui sur le bouton 'Charger mission Précédente'. Ouvrir la mission à l'aide de l'explorateur de fichier mis à disposition.
Résultats attendus :	Les paramètres de cette mission sont directement chargés dans le logiciel.

BILAN

REMARQUES

CONCLUSION

VALIDE		NON VALIDE	
--------	--	------------	--

FICHE RECETTE, QUALIFICATION DU SYSTEME

Nom du système :	Système SAILA	Code de la campagne de test :	2019-03
Technicien 1 :	Aloïs BRUCOLERI	Date :	23/05/2019
Technicien 2 :	Corentin HERY		
Technicien 3 :	Benjamin DIMIER		
Technicien 4 :	Yanis FOUNAS		
Technicien 5 :	Nicolas CHARRIER		

IDENTIFICATION DU SCENARIO

Identification du scénario de recette

Titre :	Acquisition des mesures avec/sans déclenchement du relais
Objectif du scénario :	Réaliser des acquisitions avec/sans déclenchement relais (temps, mesure, tension)

CONDITIONS INITIALES NECESSAIRES POUR EFFECTUER LA RECETTE

La carte doit être alimentée et en état d'une attente d'acquisition (voir recette 2019-02).
Les entrées à acquérir doivent être connectées.

EXECUTION DU TEST

Description courte :	Connexion des entrées analogiques et appui sur le bouton pour lancer les acquisitions. Sans le déclenchement du relais : Les acquisitions se stopperont à la fin de la durée ou alors au débranchement de l'alimentation. Avec le déclenchement du relais sur le temps : Les acquisitions se stopperont à la fin de la durée ou alors au débranchement de l'alimentation ou encore à la fin de la durée du relais. Avec le déclenchement du relais sur une mesure : Les acquisitions se stopperont à la fin de la durée ou au débranchement de l'alimentation ou enfin si la condition du relais paramétré au préalable, sur une mesure, est vérifiée. Avec le déclenchement du relais pour la surveillance tension : Les acquisitions se stopperont à la fin de la durée ou au débranchement de l'alimentation ou alors si la condition du relais paramétré au préalable, sur la surveillance tension, est vérifiée.
Résultats attendus :	Changement d'état de la carte 'Mode en attente de déchargement'.

BILAN

REMARQUES

CONCLUSION

VALIDE		NON VALIDE	
--------	--	------------	--

FICHE RECETTE, QUALIFICATION DU SYSTEME

Nom du système :	Système SAILA	Code de la campagne de test :	2019-07
Technicien 1 :	Aloïs BRUCOLERI	Date :	23/05/2019
Technicien 2 :	Corentin HERY		
Technicien 3 :	Benjamin DIMIER		
Technicien 4 :	Yanis FOUNAS		
Technicien 5 :	Nicolas CHARRIER		

IDENTIFICATION DU SCENARIO

Identification du scénario de recette

Titre :	Reset de la carte
Objectif du scénario :	Reset la carte SAILA

CONDITIONS INITIALES NECESSAIRES POUR EFFECTUER LA RECETTE

Carte SAILA connectée au logiciel (voir recette 2019-01).

EXECUTION DU TEST

Description courte :	Appui sur le bouton 'Reset'.
Résultats attendus :	L'état de la carte passe à l'état Paramétrage.

BILAN

REMARQUES

CONCLUSION

VALIDE		NON VALIDE	
--------	--	------------	--

FICHE RECETTE, QUALIFICATION DU SYSTEME

Nom du système :	Système SAILA	Code de la campagne de test :	2019-01
Technicien 1 :	Aloïs BRUCOLERI	Date :	23/05/2019
Technicien 2 :	Corentin HERY		
Technicien 3 :	Benjamin DIMIER		
Technicien 4 :	Yanis FOUNAS		
Technicien 5 :	Nicolas CHARRIER		

IDENTIFICATION DU SCENARIO

Identification du scénario de recette

Titre :	Connexion avec la carte SAILA
Objectif du scénario :	Se connecter à la carte SAILA via l'IHM

CONDITIONS INITIALES NECESSAIRES POUR EFFECTUER LA RECETTE

Avoir installé l'application.
 Connecter la carte SAILA au port USB du PC.
 Une fois la carte branchée, démarre l'application.

EXECUTION DU TEST

Description courte :	Sélection du port de la carte SAILA. Appui sur le bouton 'Connecter'.
Résultats attendus :	L'IHM affiche 'La carte est connectée !' et ensuite l'état de la carte.

BILAN

RAS

REMARQUES

Faire bien attention de connecter la carte au PC avant de lancer l'application. Et sélectionner le bon port de communication. Regardez au préalable le numéro du port de la carte SAILA dans le gestionnaire de périphérique (section port série).

CONCLUSION

VALIDE		NON VALIDE	
--------	--	------------	--

FICHE RECETTE, QUALIFICATION DU SYSTEME

Nom du système :	Système SAILA	Code de la campagne de test :	2019-04
Technicien 1 :	Aloïs BRUCOLERI	Date :	23/05/2019
Technicien 2 :	Corentin HERY		
Technicien 3 :	Benjamin DIMIER		
Technicien 4 :	Yanis FOUNAS		
Technicien 5 :	Nicolas CHARRIER		

IDENTIFICATION DU SCENARIO

Identification du scénario de recette

Titre :	Déchargement des données
Objectif du scénario :	Déchargement des données acquises

CONDITIONS INITIALES NECESSAIRES POUR EFFECTUER LA RECETTE

Carte SAILA connectée au logiciel (voir recette 2019-01).
La carte doit se trouver dans l'état Déchargement (voir recette 2019-03).

EXECUTION DU TEST

Description courte :	Rentrer un nom de mission inexistant. Appui sur le bouton 'OK'. Appui sur le bouton 'Enregistrer les données'.
Résultats attendus :	Affichage du nombre de lot déchargé. Création d'un fichier CSV avec le nom de mission rentré auparavant et toutes les données triés à l'intérieur. Affichage de l'état déchargement.

BILAN

REMARQUES

CONCLUSION

VALIDE		NON VALIDE	
--------	--	------------	--

FICHE RECETTE, QUALIFICATION DU SYSTEME

Nom du système :	Système SAILA	Code de la campagne de test :	2019-02
Technicien 1 :	Aloïs BRUCOLERI	Date :	23/05/2019
Technicien 2 :	Corentin HERY		
Technicien 3 :	Benjamin DIMIER		
Technicien 4 :	Yanis FOUNAS		
Technicien 5 :	Nicolas CHARRIER		

IDENTIFICATION DU SCENARIO

Identification du scénario de recette

Titre :	Paramétrage de la carte SAILA et transfert
Objectif du scénario :	Paramétrer la carte avec une configuration avec et sans commande du relais.

CONDITIONS INITIALES NECESSAIRES POUR EFFECTUER LA RECETTE

Carte SAILA Connectée au logiciel (voir recette 2019-01).
La carte doit se trouver dans l'état Paramétrage (voir recette 2019-07)

EXECUTION DU TEST

Description courte :	Choisir une durée de mission, un intervalle d'acquisition, choisir l'oscillateur, et paramétrer le relais ou non. Et ensuite appuyer sur le bouton 'Envoyer les paramètres'.
Résultats attendus :	On pourra voir l'avancé du paramétrage au niveau de l'état de la carte et le nouvel état qui passera en 'Mode en attente d'acquisition'.

BILAN

REMARQUES

CONCLUSION

VALIDE		NON VALIDE	
--------	--	------------	--

Mode d'emploi

Ce mode d'emploi va détailler l'installation, l'utilisation et le dépannage du système.

HERY Corentin et BRUCCOLERI Aloïs

05/06/2019

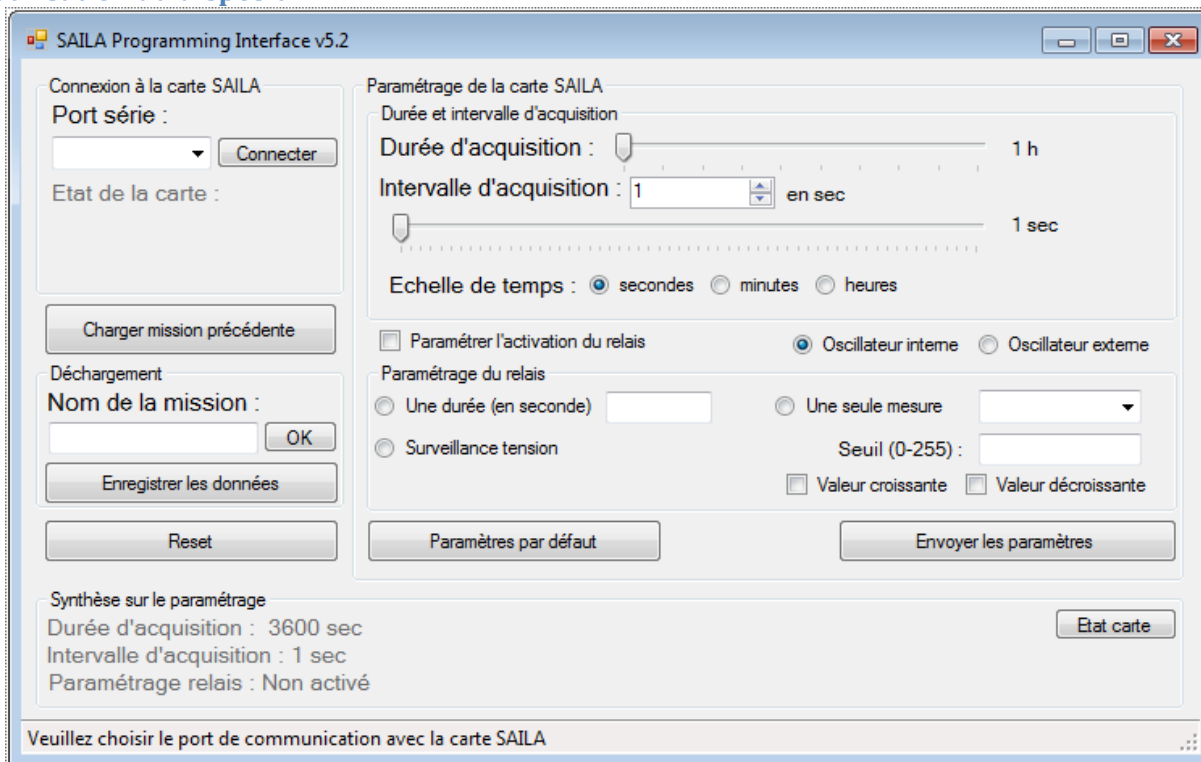
Description du produit :

Le dispositif SAILA va permettre de réaliser des acquisitions cycliques de tension et de température, et de les stocker dans une mémoire pour pouvoir les traiter par la suite dans un fichier Excel. L'application permettra de paramétrer une durée, et un intervalle d'acquisition et aussi d'effectuer le déchargement de ses données dans un fichier.

Installation :

Lancer le logiciel d'installation et vérifier la création du dossier Fichier CSV dans les documents.

Utilisation du dispositif :



Connexion :

Une fois l'application installée, branchez la carte SAILA via le câble USB au PC. Exécuter l'application et se connecter à la carte en sélectionnant le port série (vérifier au préalable le port de communication correspondant à la carte SAILA dans le gestionnaire de périphérique).

Paramétrage d'une mission :

Une fois connecté, vous pouvez procéder au paramétrage de la carte en mettant une durée et un intervalle que vous souhaitez. Faire attention de ne pas mettre un intervalle supérieur à la durée pour le bon fonctionnement du système. Choisir d'activer le relais ou non. Si oui, il suffit de choisir soit la surveillance tension, soit de mettre une durée et la durée voulue, soit de mettre sur une mesure en mettant la mesure en question avec un seuil et choisir si vous voulez le mettre sur le sens montant ou descendant. Une fois le paramétrage choisit, il suffit d'appuyer sur le bouton « Envoyer les paramètres ». Une fois le paramétrage effectué, la carte passe en état « En attente d'acquisition ».

Mettre en route les acquisitions :

Une fois en état « En attente d'acquisition », il suffit de connecter les entrées analogiques à contrôler à la carte SAILA et d'appuyer sur le bouton pour lancer les acquisitions. Une fois les acquisitions terminées ou

alors une fois la carte mis hors tension, elle passe en état « En attente de déchargement ».

Déchargement des données :

Pour récupérer les acquisitions et pouvoir les traiter, il faut reconnecter la carte au PC via le câble USB, exécuter l'application. Une fois connecté et en état « En attente de déchargement », il faut appuyer sur le bouton « Enregistrer les données » en ayant mis au préalable un nom de mission inexistant dans le répertoire et appuyer sur OK (Répertoire par défaut du fichier C:\Documents\FichierCSV).

Reset de la carte :

Une fois connecté à la carte SAILA, appuyer sur le bouton « Reset » et la carte repassera en état « Paramétrage »

Charger une mission :

Une fois en état « Paramétrage », appuyez sur le bouton « Charger mission précédente ». Un explorateur de fichier s'ouvrira et il faudra sélectionner une mission enregistré auparavant pour charger les paramètres de cette dernière. Ensuite vous pouvez appuyer sur le bouton « Envoyer les paramètres » pour configurer la carte avec les paramètres chargés.

Dépannage :

Etat de fonctionnement	Cause possible	Action
LED ERREUR	Problème d'envoi de donnée	Vérifié l'intégrité du câble. Vérifié qu'il n'a pas de perturbation électromagnétique.
Les acquisitions ne se lance pas	Mauvais état de la carte	Vérifié l'état de la carte. Re-paramétrer la carte.
Problème de connexion entre le logiciel et la carte	Mauvaise connexion au niveau du port série.	Fermer le logiciel, déconnecter puis reconnecter la carte et relancer le logiciel.
Fin de vie de la mémoire	La fin de vie de la mémoire est atteinte	Contacteur SAV pour le remplacement.