14/05/2018

RAPPORT FINAL – PROJET ELECTRONIQUE

Masson J.

Raemdonck S.

Hancquet B.

Groupe

Groupe 4

Masson Julien 2TL2

Raemdonck Sébastien 2TL2

Hancquet Brian 2TL1

Cahier des Charges

L'objectif du projet est de fabriquer un télémètre à ultrason. Celui-ci doit pouvoir calculer une distance entre lui-même et la direction vers laquelle il pointe grâce à sa sonde.

Pour ceci nous utilisons un PIC18F458 qui sera codé en C et relié à une application Java afin de faire sa représentation visuelle.

Premièrement il nous était demandé de crée une simulation du projet avec le logicielle Proteus. Après les tests de celle-ci, nous devions créer une board avec le logicielle Eagle afin de pouvoir en récupérer une plaquette physique pour la représentation réelle du projet.

Les composants doivent être proprement soudé sur la plaquette. Le télémètre doit pouvoir signaler une alerte par le clignotement d'une led rouge lorsque la distance dépasse un seuil défini par l'utilisateur. Si tout se passe bien une led verte allumé en continu indique le bon fonctionnement de la sonde.

L'interface Java doit également afficher un message d'alerte en cas de problème de distance et également afficher la distance calculée en cas de bon fonctionnement. Elle communiquera avec le PIC soit par le port RS232 du pc, soit par le port USB, cela dépendra de ce que le groupe a choisi.

Répartition du travail

Masson Julien

A effectuer le code C, le code Java, le schéma de simulation Proteus, le schéma sur Eagle, le rapport intermédiaire et le rapport final.

Raemdonck Sébastien

A pris une répartition de la soudure.

Hancquet Brian

A pris une répartition de la soudure.

1. Carte

1.1 Description

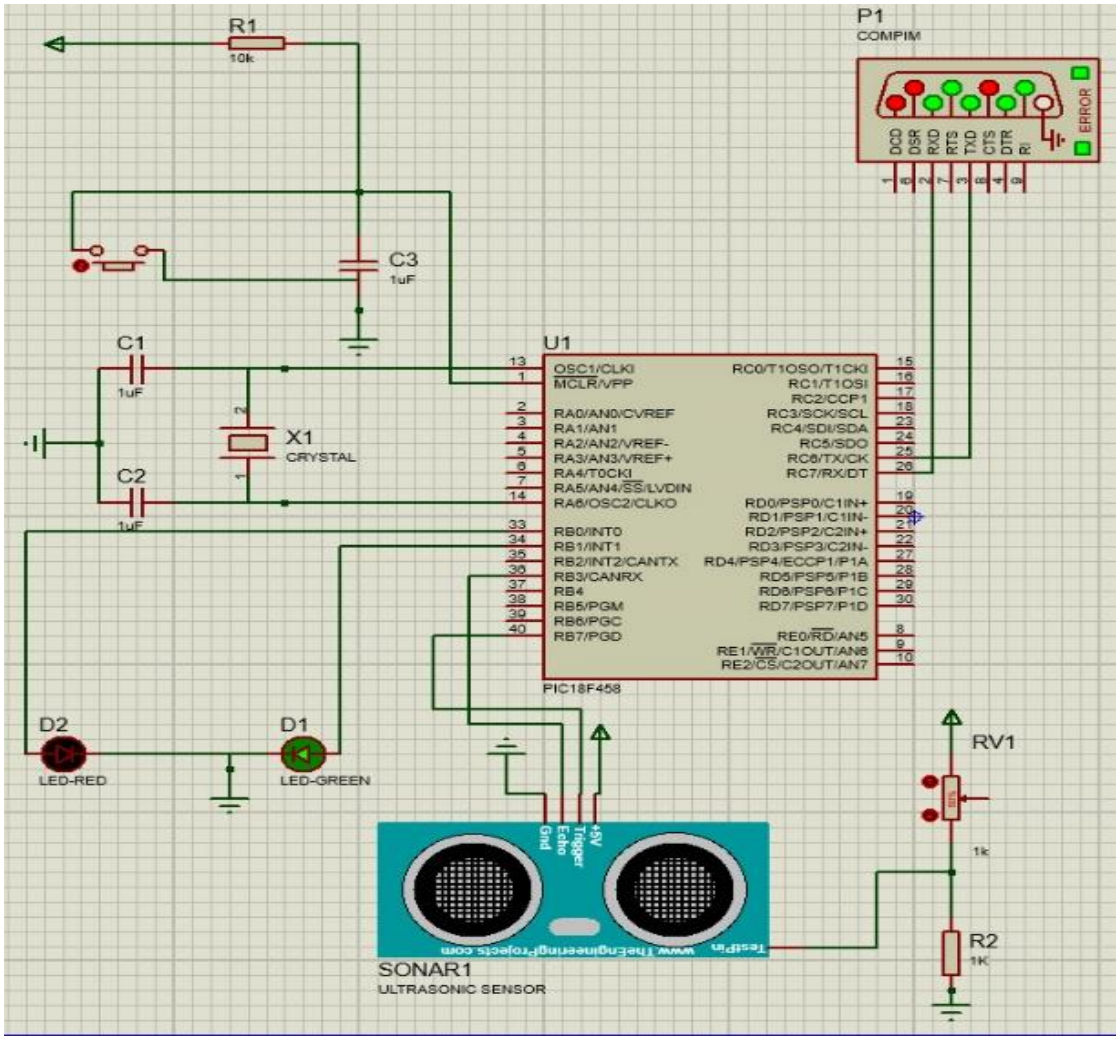
La carte est composée de tous les éléments dont elle a besoin pour fonctionner. Nous avions le choix entre un connecteur RS232 et un DB9-USB-D5-F, nous avons choisis de travailler avec le DB9 qui se connectera au port USB du PC. La carte est également composé d'un bornier d'alimentation, d'un PIC18F458, d'une led rouge et d'une led verte, de la sonde à ultrason, d'un crystal, d'un bouton poussoir, de condensateurs et de résistance. Tout les composants et leurs descriptions sont disponible dans le point « 6. Caractéristiques techniques des éléments. ».

1.2 Fonctionnement

Pour faire fonctionner la carte, il faut la brancher de sa sortie DB9-USB-D5-F à l'entrée USB d'un PC. Elle sera ensuite connectée avec l'application Java. En poussant sur le bouton la carte est censé démarré, si tout se passe bien la led verte reste allumé en continu et la distance mesurée par la sonde à ultrason est affiché par l'application Java. Par contre s'il y a un problème de distance la led rouge s'allumera pour le signaler.

2. Schémas Electronique

2.1 Schéma Protéus

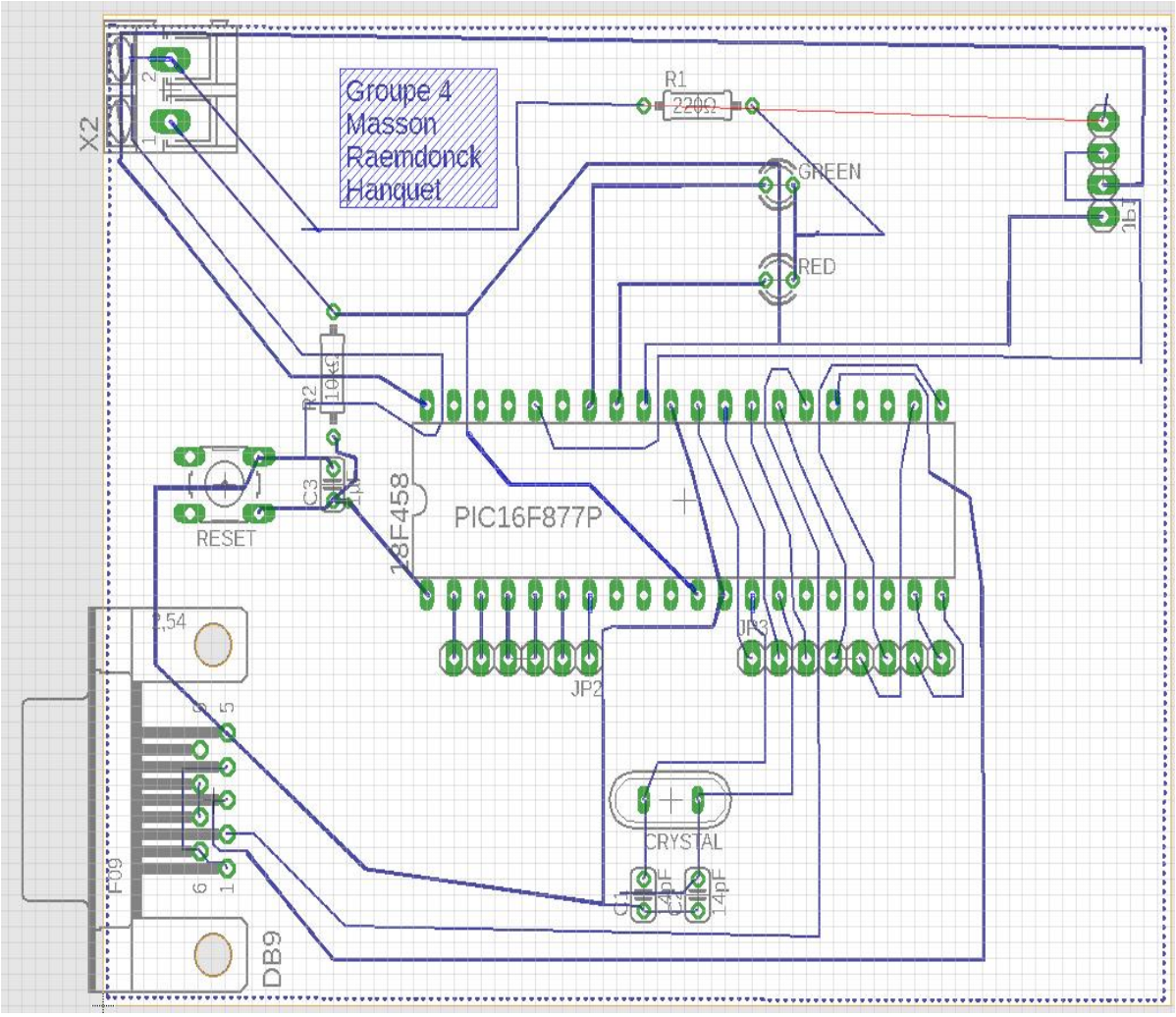


Ci-dessus se trouve l'image du schéma de simulation de Proteus. Il est composé de tout les éléments importants afin de faire fonctionner notre télémètre à ultrason.

Celui-ci à été réalisé grâce aux fiches de cours « Electronique digitale et analyse des signaux partie pratique ».

Vous pourrez trouver plus de détail à propos de ses composants, de leur utilité et de leur configuration dans le point « 6. Caractéristiques techniques des éléments. »

2.2 Schéma Eagle



Ci-dessus se trouve l'image de la board Eagle. Cette image a été réalisé grâce au schéma de Proteus.

Nous avions le choix de travailler avec un connecteur USB et un DB9 et nous avons décidé de travailler avec un DB9 en vue de pouvoir se laisser du RS232 dans notre schéma et sur notre PCB. Les informations à propos du pourquoi le DB9 n'a pas besoin de RS232 se trouvent dans le point « 6. Caractéristiques techniques des éléments. »

Ce schéma a été réalisé en vue de tiré une PCB physique.

3. Les codes

3.1 Le code C (PIC)

Nous allons scinder le code en ses différents points pour une meilleur explication. Le code sera également en fin de partie 3.1 en entier pour une vision plus globale de celui-ci.

3.1.1 Include

```
#include "C:\Users\JULIEN\Desktop\cours\ElectroProj\zlkadmlkaz\elecTrue\elecTrue.h"
```

Nous incluons un seul fichier qui sera nécessaire pour l'application du code pour le PIC. Cette include inclura toutes les données nécessaire pour que le code soit exécuté pour un PIC18F458.

3.1.2 Déclarations

```
int32 dist(void);
int32 limite = 0;
int32 distance;
```

Nous avons que deux déclarations de variables et une déclaration de fonction.

3.1.3 Fonction main

```
void main(){
   printf("Bienvenu dans la simulation proetus\n");
   setup_timer_0(RTCC_INTERNAL|RTCC_DIV_8|RTCC_8_BIT);
   setup_timer_1(T1_INTERNAL|T1_DIV_BY_1);
   enable_interrupts(GLOBAL);
   setup_low_volt_detect(FALSE);
   enable_interrupts(INT_RDA);
   enable_interrupts(GLOBAL);
   distance = 0;
   while(true){
      distance = dist();
      if(distance < limite){
         output_high(PIN_B0);
         output_low(PIN_B1);
         printf("distance trop petite");
         printf("%lu", distance);
      } else {
         output_high(PIN_B1);
         output_low(PIN_B0);
         do{
            delay_ms(1000);
            distance = dist();
            printf("%lu\r\n",distance);
         }while(distance>limite);
      }
   }
}
```

Cette partie est la partie principale du code, celle qui sera exécuté et qui fera appel aux autres fonctions. Après la mise en place de tout les paramètres nécessaires au bon fonctionnement du PIC, nous initialisons la distance à zéro et nous entrons dans une boucle While(true) calculer la distance en continu. Dans cette boucle nous appellerons la fonction dist() l'affecter à notre variable distance. La boucle nous emmène ensuite dans une condition où si la

distance est inférieur à la limite que l'utilisateur à défini (ici 0) le led verte s'éteint et la led rouge s'allume. Si la condition est respectée la led verte s'allume et la led rouge reste éteinte, et la distance calculée s'affiche.

3.1.4 Calcule de la distance

```
int32 dist(){
   float time=0;
   int32 distance=0;
   output_high(pin_b7);
   delay_us(10);
   output_low(pin_b7);
   while(!input(PIN_b3));
   set_timer1(0);
   while(input(PIN_b3));
   time=get_timer1();
   distance = time* 0.00344;
   return distance;
}
```

Dans cette fonction, nous allons utiliser deux variables ; time et distance, toutes deux variable local et initialisée à zéro. Nous allons envoyer une impulsion sur le trigger de la sonde ultrason (PIN_B7) et attendre sa fin, ensuite tant que la PIN_B3 n'a pas reçu de retour un timer calcule le temps qu'il lui faut pour que l'ultrason revienne, quand celui-ci est à l'état haut cela veut dire que l'ultrason est revenu. Il nous suffit alors de récupérer le temps du timer et le calculer pour obtenir une distance en centimètre. Cette fonction return la distance calculée.

3.1.5 Code complet

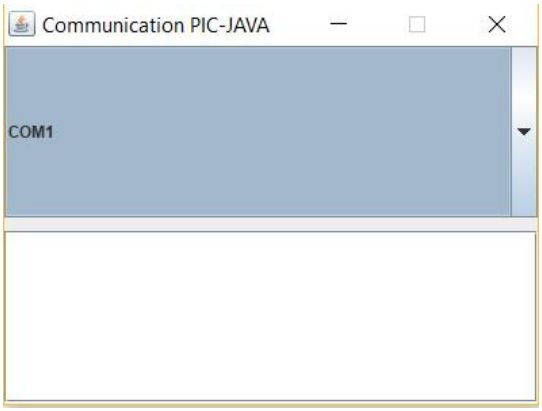
```
#include "C:\Users\JULIEN\Desktop\cours\ElectroProj\zlkadmlkaz\elecTrue\elecTrue.h"

int32 dist(void);
int32 limite = 0;
int32 distance;
void main(){
   printf("Bienvenu dans la simulation proetus\n");
   setup_timer_0(RTCC_INTERNAL|RTCC_DIV_8|RTCC_8_BIT);
   setup_timer_1(T1_INTERNAL|T1_DIV_BY_1);
   enable_interrupts(GLOBAL);
   setup_low_volt_detect(FALSE);
   enable_interrupts(INT_RDA);
   enable_interrupts(GLOBAL);
   distance = 0;
   while(true){
      distance = dist();
      if(distance < limite){
         output_high(PIN_B0);
         output_low(PIN_B1);
         printf("distance trop petite");
         printf("%lu", distance);
      } else {
         output_high(PIN_B1);
         output_low(PIN_B0);
         do{
            delay_ms(1000);
            distance = dist();
            printf("%lu\r\n",distance);
         }while(distance>limite);
      }
   }
}

int32 dist(){
   float time=0;
   int32 distance=0;
   output_high(pin_b7);
   delay_us(10);
   output_low(pin_b7);
   while(!input(PIN_b3));
   set_timer1(0);
   while(input(PIN_b3));
   time=get_timer1();
   distance = time* 0.00344;
   return distance;
}
```

3.2 Le code Java

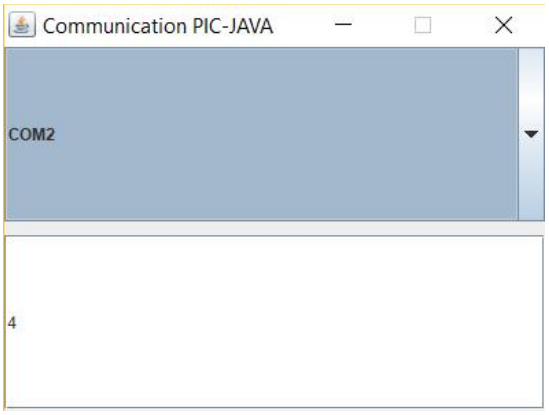
Nous allons scinder le code en ses différents points pour une meilleur explication ainsi qu'un aperçu visuel de l'application en cours de fonctionnement. Le code en sera également mis en fin de partie 3.2 en entier pour une vision plus globale de celui-ci.

3.2.1 Aperçu de l'application

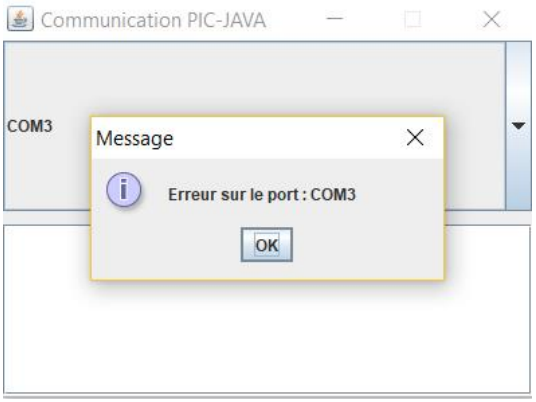
3.2.1.1 à l'arrêt.



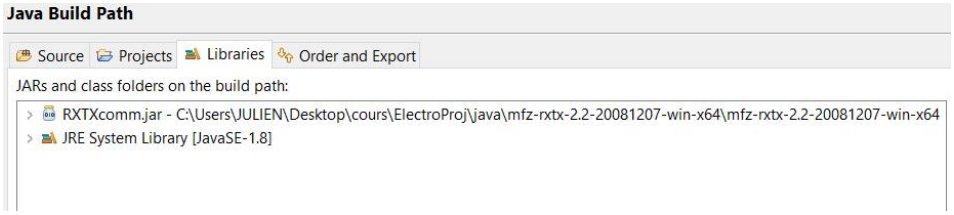
3.2.1.2 en fonctionnement



3.2.1.3 en cas d'erreur



3.2.2 Les librairies



Pour les librairies nous avons utilisé qu'une librairie nommée RXTXcomm.jar qui a servi pour la liaison entre l'application Java et les ports serials de VSPE. Cette librairie sert également à la connexion sur le PCB physique.

3.2.3 Les Imports

```
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JTextField;

import gnu.io.*;
```

Nous avons effectué plusieurs imports dans le code nécessaire au bon fonctionnement de celui-ci envers Proteus et envers notre PCB. Certains de ces imports sont directement importé de librairies internes à Java et d'autres à notre librairie RXTXcomm.jar

3.2.4 Déclaration de la classe principale & déclarations

```
public class Fenetre extends JFrame implements SerialPortEventListener{


    private SerialPort serialPort;
    private JTextField field;
    private BufferedReader BufferIn;
    private CommPortIdentifier comPortId;
    Object[] comList = new Object[] {"COM1", "COM2", "COM3", "COM4", "COM5"};
    private JComboBox<Object> listeCom;
    Object select;
```

Dans la déclaration de la classe nous devons informer celle-ci qu'il y aura une JFrame avec le « exetends JFrame et également des événements impliquant des ports avec le « Implements SerialPortEventListener ». Lors de la déclaration nous déclarons un champ où sera afficher la mesure, un BufferedReader qui lira les données envoyé par Proteus, un CommPortIdentifier qui servira à identifier le port ouvert, une liste avec différents ports dedans, une JComboBox qui servira à l'affichage de la liste dans la JFrame et enfin un Objet.

3.2.5 La classe

```
public Fenetre() {

    field = new JTextField();
    listeCom = new JComboBox<Object>(comList);
    this.setLayout(new GridLayout(2, 2, 10, 10));
    this.getContentPane().add(listeCom);
    this.getContentPane().add(field);

    this.setSize(400, 300);
    this.setTitle("Communication PIC-JAVA");
    this.setResizable(false);
    this.setLocationRelativeTo(null);
    this.setVisible(true);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    listeCom.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
            port();
        }
    });
}
```

La classe servira à construire la JFrame. Cette JFrame constituera la représentation graphique de toutes les fonctions de la Classe. Dans cette partie du code on retrouvera donc tous les éléments qui construiront la fenêtre.

3.2.6 Connexion avec le port Comm

```
public void port() {
    try {
        select = listeCom.getSelectedItem();
        comPortId=CommPortIdentifier.getPortIdentifier(select.toString());
        serialPort=(SerialPort)comPortId.open("",100);
        serialPort.addEventListener(this);
        serialPort.notifyOnDataAvailable(true);

    }catch(Exception e) {
        JOptionPane.showMessageDialog(null, "Erreur sur le port : " + select);}
}
```

Cette fonction permettra l'ouverture de la connexion avec le port Comm. Cette fonction contient un TRY/CATCH qui sera utile au cas où il y aura un problème lors de la connexion au port Comm. Si cela arrive un message d'erreur s'affichera indiquant quel le port n'est pas ouvert.

3.2.7 Reçu des informations

```
public void serialEvent(SerialPortEvent arg0) {
    try {
        BufferIn = new BufferedReader(new InputStreamReader(serialPort.getInputStream()));
        String tes = BufferIn.readLine();
        field.setText(tes);
        BufferIn.close();
    }catch(Exception e) {JOptionPane.showMessageDialog(null, e.toString());}
}
```

Cette fonction servira à recevoir les mesures qui sont envoyées par le PIC ou par Proteus et les affichera dans le zone de texte de l'application.

3.2.8 Code complet

```
package elecTrue;

import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JTextField;

import gnu.io.*;

@SuppressWarnings("serial")

public class Fenetre extends JFrame implements SerialPortEventListener{


    private SerialPort serialPort;
    private JTextField field;
    private BufferedReader BufferIn;
    private CommPortIdentifier comPortId;
    Object[] comList = new Object[] {"COM1", "COM2", "COM3", "COM4", "COM5"};
    private JComboBox<Object> listeCom;
    Object select;


    public Fenetre() {

        field = new JTextField();
        listeCom = new JComboBox<Object>(comList);
        this.setLayout(new GridLayout(2, 2, 10, 10));
        this.getContentPane().add(listeCom);
        this.getContentPane().add(field);

        this.setSize(400, 300);
        this.setTitle("Communication PIC-JAVA");
        this.setResizable(false);
        this.setLocationRelativeTo(null);
        this.setVisible(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


    public void port() {
        try {
            select = listeCom.getSelectedItem();
            comPortId=CommPortIdentifier.getPortIdentifier(select.toString());
            serialPort=(SerialPort)comPortId.open("",100);
            serialPort.addEventListener(this);
            serialPort.notifyOnDataAvailable(true);

        }catch(Exception e) {
            JOptionPane.showMessageDialog(null, "Erreur sur le port : " + select);}
    }

    public void serialEvent(SerialPortEvent arg0) {
        try {
            BufferIn = new BufferedReader(new InputStreamReader(serialPort.getInputStream()));
            String tes = BufferIn.readLine();
            field.setText(tes);
            BufferIn.close();
        }catch(Exception e) {JOptionPane.showMessageDialog(null, e.toString());}
    }
}
```

4. Tests

4.1 Test du PIC

Malheureusement, faute de temps ne nous avons pas eu le temps de mettre le code C dans le PIC donc nous ne savons pas si celle-ci est fonctionne ou pas. Malgré cela nous avons fait tous les tests après la soudure pour voir s'il y avait des éventuels courts-circuits mais tous les tests était correct.

4.2 Test du code Java

L'application se lance correctement sans bug. L'application arrive a identifier et a communiquer avec les ports Comm selon ceux qui sont ouvert bien entendu. Les erreurs s'affichent correctement et sont explicite pour l'utilisateurs.

4.3 Test du code C avec Proteus

Le code C compilé fonctionne bien dans Proteus, aucun message d'erreur ne s'affiche. Dans la simulation le code tourne correctement et nous pouvons apercevoir visuellement son bon fonctionnement grâce à la LED vert allumé en continu.

4.4 Test de la simulation avec tous les codes

La simulation Proteus en combinant le code C et le code Java fonctionne parfaitement, nous pouvons observer l'affichage du nombre de centimètre qui est de quatre centimètre dans l'application Java lorsqu'on la connecte au port Comm de VSPE qui est elle-même relié à la simulation Proteus en train de tourner.

5. Conformité

La conformité du travail par rapport au cahier des charges est relative. La plupart des éléments demandé sont retrouvé mais certains points y manquent. Par exemple il était demandé que la LED rouge clignote lors d'une erreur, elle ne fait que s'allumer et elle ne clignote pas et nous n'avons pas eu le temps de mettre le code C dans le PIC. Pour le reste il semblerait que tous les points demandés ont été respecté.

6. Caractéristiques techniques des éléments.

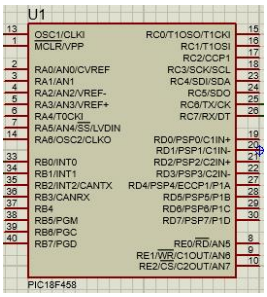
Dans ce point nous allons aborder les composants de la plaque. Nous allons y expliquer leur fonctionnement et nous montrerons leur représentation d'une part dans la simulation Proteus et d'une autre part physiquement sur la plaque.

6.1 PIC18F458

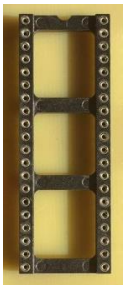
Le PIC18F458 appartient à la famille des PIC18. Cette famille est plus complète que ses précédentes. Elle permet notamment de faire fonctionner du C pour une programmation plus complètes. Une horloge interne permet de cadencer le PIC. Le PIC fonctionne à un voltage de 5V.

Elle est composée de 40 broches qui sont potentiellement des entrée/sorties selon ce que vous voulez faire avec. Ce PIC18F458 peut être configuré de différentes façon pour différentes utilisations comme des sonde à télémètre, des sondes à température, …

Simulation



Physique

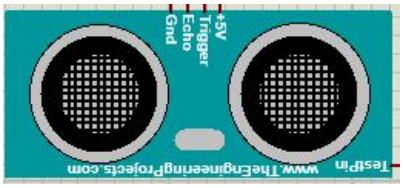


6.2 Sonde à ultrasons

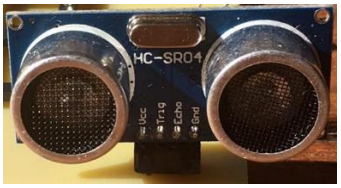
Pour la sonde à ultrason nous avons utilisé un HCSR04. Cette sonde est capable d'évaluer une distance entre 2cm et 400cm. Le module inclus un émetteur qui émettra l'ultrason et un récepteur qui réceptionnera l'ultrason revenant. Voici le principe de fonctionnement du HCSR04 :

1. Le module émet un sonar composé d'une série de 8 impulsions à 40 kHz.
2. En utilisant la broche trigger, il est possible d'envoyer un signal pour activer le déclenchement de l'impulsion sonar.
3. Quand le signal revient la sortie passe au niveau haut durant toute la période où l'onde voyage vers l'objet et revient après avoir réfléchir par ce dernier.

Simulation



Physique



6.3 Bornier d'alimentation

Le bornier d'alimentation sert à alimenter le circuit pour son bon fonctionnent. Il à une entrée et une sortie.

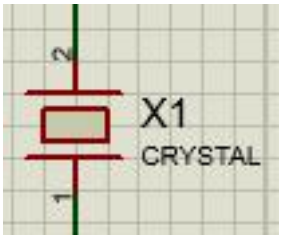
Simulation

Physique

6.4 Crystal

Le Crystal est un composant qui possède comme propriété utile d'osciller à une fréquence stable lorsqu'il est stimulé électriquement.

Simulation



Physique

6.5 LED

Les LED permettront un affichage visuel plus direct. Elles sont allumées en les mettant à l'état haut ou bas.

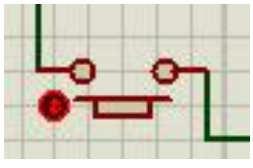
Simulation

Physique

6.6 Le bouton poussoir

Le bouton sera utilisé comme un interrupteur pour déclencher le début d'une action.

Simulation



Physique

7. Commentaires finaux

7.1 Problèmes rencontrés

La charge de travail/recherche du projet pour tout ce qui est code, simulation, compréhension aux composants, … combiné aux autres projets des autres cours n'était clairement pas évident vu l'absence totale de participation au sein du groupe.

Ce qui a pris le plus de temps a été de transformer la simulation Proteus en PCB sur eagle.

Le plus gros problème a été le temps à gérer.

7.2 Futur améliorations

Notre plaquette dispose d'une rangée modulaire pour des améliorations de quelconques modules. J'aurais aimé aller plus loin que le cahier des charges du projet mais je n'ai même pas eu l'occasion de mettre le code C dans mon PIC donc allé plus loin que le projet m'était impossible. La rangée modulaire est donc inutilisée actuellement mais permet de mettre différents modules dessus.