#include <LiquidCrystal.h> //appel de la librairie pour l'ecran LCD

#include <LcdBarGraph.h> //appel de la libraire pour afficher une bargraph sur l'ecran

#include <Wire.h>

LiquidCrystal lcd(6, 7, 10, 11, 12, 13); //l'on définie les pin de l'écran LCD

LcdBarGraph lbg(&lcd, 16, 0, 1); //l'on définie les pin pour le bargraph de l'écran LCD

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#define TIMER1\_TOP (259) //l'on peut modifier la valeur du timertop pour affiner le reglage de la frequence de debordement du timer1

#define USE\_3V3\_AREF (1) //mettre a 1 quand on utilise l'arduino avec un port USB mettre a 0 quand on l'utilise en mode systeme embarquer, il est préférable d'utiliser le 3.3V pour la REF car plus stable.

#define max\_AmpMoyenne 200 // cette variable sera utilisé pour définir la valeur de réference pour le remplissage du bargraph, exemple : max max\_AmpMoyenne est mis à 200 et que la variable à tester atteint la valeur de 100 alors le bargraph fera 50% de la taille de l'écran

//Definission des pin Digital

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//La pin digital 0 n'est pas utilisé, toutefois si l'on utilise le port série pour faire du debuggage elle sera l'entrée série.

const int bouttonCalibrage = 2; //Boutton d'encodage,aussi IN0 pour le reveil du mode veille.

const int Pinbuzzer = 3; //buzzer aka OCR2B pour la géneration de la tonalité.

const int EntrerTimer0 = 4; //bouclage de la pin 4(intput du timer0) avec la pin 9²

const int SortieTimer0 = 9; //bouclage de la pin 9(output du timer0) avec la pin 4

const int BobineTX = 5; //Pin qui alimente la bobine primaire du détecteur de métaux en PWM générer à l'aide du timer0.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//Variable qui servent a piloté l'allumage et l'arret de la carte

const int Boutton\_off = A1;

const int Seuil\_Boutton\_off = 800;

const int AlimMosfetCarte = 8; //la pin D8 peut être connecte a la grille d'un mosfet canal n pour un montage avec pilotage numérique de l'alimentation

int tempsAppui = 0;

int lecture\_Boutton\_off;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Analog pin definitions

const int ReceptionSignal = A0; //Pin qui recois l'information a traiter en sortie de l'ampli op

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Variables utilisé seulement par l'ISR (Interrupt sub-routine)

int16\_t ValeurBrut[4]; // Les ValeurBrut sont utilisées pour accumuler la lecture des données du CAN, une pour chaqune des 4 phases.

uint16\_t Echantillon = 0; //variable utilisé pour gérer les séquences d'échantillonage

const uint16\_t EchantillonageMax = 1024; //Constante qui sert à définir un plafond pour la variable Echantillon

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Variables utilisés par l'ISR(Interrupt sub-routine) et en dehors.

volatile int16\_t Moyenne[4]; // Quand nous avons accumulées suffisament de lecture dans les ValeurBrut, l'ISR les copies dans cette variable et recommence.

volatile uint32\_t ticks = 0; // compteur de tick pour le chronometrage

volatile bool EchantillonPret = false; // Indique que le tableaux des moyennes à été mis a jour.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Variable utilisées uniquement à l'éxterieur de l'ISR.

int16\_t calib[4]; // Valeur (réglés pendant la calibration)que l'on extrait de la variable Moyenne, et qui servira pour recalibré le détecteur.

volatile uint8\_t dernierCompteur; // variable qui stocker la derniere lecture du registre TCNT0 qui stocke la valeur du compteur du Timer0.

volatile uint16\_t echec = 0; // Cette variable compte combiens de temps l'ISR à etais éxecuter trops tard,Elles doit donc rester a 0 si tous marche bien.

const double Racine\_De\_Un\_Demi = 0.707106781; //La constante stocke la valeur la moitié de la racine carrée de 2, cette valeur est sutilisée pour effectuer des transformations de coordonnées (trigonométriques), par exemple pour faire une rotation de 45degrés.

const double QuartDePI = 0.785398163; //Pi/4 correspond a 45 Degrés, cette variable seras utilisé pour des calculs de phase

const double ConvRadianVersDegres = 57.2957795; //conversions des radiant vers les degrés pour calculer le déphasage en degrés.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// L'échantillonnage et maintien du CAN se produit 2 horloges ADC (= 32 horloges système) après que le flag de debordement du Timer1 à etais définie.

const float Ajustement\_De\_Phase = (45.0 \* 32.0)/(float)(TIMER1\_TOP + 1); // Ceci introduit une legere erreur de phase que nous corrigeons par les calculs.

float Seuil = 5.0; //Valeur faible = plus grande sensibilité, 10 par exemple est utilisable que si nous avons une bobine parfaitement balancé.

//L'on pourra ajuster ceci via un potentiometre.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void setup()

{

Serial.begin(19200); //démarrage de la liaison série

pinMode(Boutton\_off,INPUT);

pinMode(AlimMosfetCarte, OUTPUT); //la pin D8 peut être connecte a la grille d'un mosfet canal n pour un montage avec maintien alim

digitalWrite(AlimMosfetCarte,HIGH);

lcd.begin(16, 2); // Lancement de l'écran LCD

pinMode(bouttonCalibrage, INPUT\_PULLUP); //Boutton d'encodage,aussi IN0 pour le reveil du mode veille si nécessaire connecter a une ressistance de pullup interne.

digitalWrite(SortieTimer0, LOW); //L'on définie la pin ouput du Timer1 qui doit aliment le Timer0 a l'état bas

pinMode(SortieTimer0, OUTPUT); // broche d'impulsion du Timer1 utilisée pour alimenter le Timer0.

digitalWrite(BobineTX, LOW); //Pin qui alimente la bobine primaire du détecteur de métaux en PWM générer à l'aide du timer0.

pinMode(BobineTX, OUTPUT); // La sortie du timer0,un signal carré pour piloter la bobine transmettrice.

lcd.setCursor(2, 0);

lcd.print("c'est parti !");

delay(3000);

lcd.clear();

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

cli(); //désactive les interruptions globales, c'est-à-dire toutes les interruptions, à l'exception du watchdog (non utilisé), ceci permet d'éviter que le code sois interrompue et appelé ailleurs avant qu'il est finie d'initialisé tous les registres ce qui pourrait faire planter le code

TCCR0B = 0; // arrete le timer0 qui a etais configurer par defaut par l'arduino.

Serial.println("Registre TCCR0B = " + String(TCCR0B,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

TIMSK0 = 0; // desactive les interruption du timer0

Serial.println("Registre TIMSK0 = " + String(TIMSK0,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

TIFR0 = 0x07; // efface les interruption en attente du timer0

Serial.println("Registre TIFR0 = " + String(TIFR0,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// initialisation du CAN pour lire la broche 0 quand le timer 1 est en debordement

#if USE\_3V3\_AREF //utilise la pin AREF (connecté au 3.3v) comme tension de reference

ADMUX = (1 << ADLAR); // Le registre ADMUX (Analog-to-Digital Multiplexer) contrôle les entrées analogiques qui sont lues par le CAN, 1 << ADLAR" active le bit ADLAR dans le registre ADMUX (ADjust Left ARithmetic),

#else

ADMUX = (1 << REFS0) | (1 << ADLAR); // (1 << REFS0) configure le bit REFS0 pour sélectionner la source de tension de référence pour la conversion ADC. Dans ce cas, le bit est réglé à 1, ce qui sélectionne la tension de référence AVCC. (1 << ADLAR) active la lecture alignée à gauche des données ADC. Cela signifie que les 8 bits les plus significatifs de la conversion ADC sont stockés dans le registre ADCH, tandis que les 2 bits les moins significatifs sont stockés dans le registre ADCL.

#endif

Serial.println("Registre ADMUX = " + String(ADMUX,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

ADCSRB = (1 << ADTS2) | (1 << ADTS1); // Auto-déclenchement du CAN sur débordement du Timer/counter1.

Serial.println("Registre ADCSRB = " + String(ADCSRB,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

ADCSRA = (1 << ADEN) | (1 << ADSC) | (1 << ADATE) | (1 << ADPS2); // Active le CAN, Active l'auto-trigger,prescaler = 16 (1Mhz CAN clock).

Serial.println("Registre ADCSRA = " + String(ADCSRA,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

DIDR0 = 1; // la broche A0 ne sera pas utilisée comme entrée numérique et sera réservée pour une utilisation analogique avec le CAN.

Serial.println("Registre DIDR0 = " + String(DIDR0,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

// Configuration du Timer1

// Prescaler = 1, phase correct PWM mode, TOP = ICR1A

TCCR1A = (1 << COM1A1) | (1 << WGM11); //écriture de deux bits dans le registre du timer1 (TCCR1A), qui le configure en utilisant les bits COM1A1 et WGM11.

Serial.println("Registre TCCR1A = " + String(TCCR1A,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

TCCR1B = (1 << WGM12) | (1 << WGM13) | (1 << CS10); // CTC (clear time counter) mode, prescaler = 1.

Serial.println("Registre TCCR1B = " + String(TCCR1B,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

TCCR1C = 0; //initialise le registre TCCR1C à zéro, ce qui configure le Timer1 en mode normal. En mode normal, le Timer1 compte jusqu'a 65535 avant de redémarrer à zéro.

Serial.println("Registre TCCR1C = " + String(TCCR1C,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

OCR1AH = (TIMER1\_TOP/2 >> 8); //OCR1AH est utilisé pour stocker la valeur de la partie supérieure (MSB) de la valeur de comparaison du TIMER1. OCR1AH est définie en prenant la moitié de la valeur de TIMER1\_TOP, qui est une constante définie dans le code, puis en décalant cette valeur de 8 bits vers la droite. Le décalage de bits vers la droite de 8 bits signifie que la valeur est divisée par 256. pour ensuite etre stocke dans OCR1AH pour être utilisée par le Timer1

Serial.println("Registre OCR1AH = " + String(OCR1AH,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

OCR1AL = (TIMER1\_TOP/2 & 0xFF); //OCR1AL seras égals a la moitié de la valeur maximale du Timer1 (TIMER1\_TOP) dans un format de 8 bits. "OCR1AL" est un registre de comparaison de sortie pour le Timer1.

Serial.println("Registre OCR1AL = " + String(OCR1AL,BIN)); //ce registre nous permet de déterminer à quel moment une action doit être déclenchée dans le code lorsque le compteur de temps atteint cette valeur.La valeur à affecter à OCR1AL est calculée en divisant TIMER1\_TOP par 2, puis en appliquant un masque binaire pour ne conserver que les 8 bits les moins significatifs de la valeur résultante. Cela garantit que la valeur écrite dans OCR1AL est une valeur de 8 bits valide.

ICR1H = (TIMER1\_TOP >> 8); // ICR1 est utilisé pour définir la valeur de comparaison pour le mode de compteur ICR1, qui est utilisé pour mesurer la durée entre deux événements externes.

Serial.println("Registre ICR1H = " + String(ICR1H,BIN)); //la partie haute de la valeur de comparaison est définie en utilisant le décalage de la constante TIMER1\_TOP de 8 bits vers la droite.nous obtenons les 8 bits les plus significatifs de cette valeur, qui sont ensuite stockés dans le registre ICR1H. Les 8 bits les moins significatifs de TIMER1\_TOP sont stockés dans le registre ICR1L.

ICR1L = (TIMER1\_TOP & 0xFF); //écrit la valeur basse du registre ICR1 en appliquant un masque bit à bit pour extraire les 8 bits de poids faible du registre TIMER1\_TOP, En utilisant un masque, cela garantit que seuls les 8 bits de poids faible sont écrits dans le registre ICR1L, tandis que les 8 bits de poids fort sont ignorés.

Serial.println("Registre ICR1L = " + String(ICR1L,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

TCNT1H = 0; //réinitialisation du compteur 8bit du timer1

Serial.println("Registre TCNT1H = " + String(TCNT1H,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

TCNT1L = 0; //réinitialisation du compteur 8bit du timer1

Serial.println("Registre TCNT1L = " + String(TCNT1L,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

TIFR1 = 0x07; // efface toute interruption en attente du timer1.

Serial.println("Registre TIFR1 = " + String(TIFR1,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

TIMSK1 = (1 << TOIE1); //active l'interruption de dépassement de temps pour le timer1.

Serial.println("Registre TIMSK1 = " + String(TIMSK1,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

// Configuration du Timer0

// Clock source = T0, fast PWM mode, TOP (OCR0A) = 7, PWM output on OC0B

TCCR0A = (1 << COM0B1) | (1 << WGM01) | (1 << WGM00); //affecte au registre TCCR0A qui contrôle le fonctionnement du timer0 ,COM0B1 : définit la configuration de la sortie OC0B (broche B de comparaison de sortie) pour la forme d'onde de modulation de largeur d'impulsion (PWM) du mode non inverseur.

Serial.println("Registre TCCR0A = " + String(TCCR0A,BIN)); //WGM01 et WGM00 : définissent le mode de fonctionnement du compteur-timer 0 comme un compteur-timer PWM en mode de comptage jusqu'à 255 (Fast PWM), qui peut être utilisé pour générer des signaux de sortie avec une résolution de 8 bits.

TCCR0B = (1 << CS00) | (1 << CS01) | (1 << CS02) | (1 << WGM02); //configure le timer0 pour qu'il utilise un prescaler de 1024 et génère une sortie PWM avec une résolution de 8 bits.

Serial.println("Registre TCCR0B = " + String(TCCR0B,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

OCR0A = 7; //configue le signal PWM qui sera généré avec une valeur de comparaison de sortie de 7, ce qui affecte la durée de la période active du signal et donc la quantité d'énergie délivrée à la charge connectée.

Serial.println("Registre OCR0A = " + String(OCR0A,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

OCR0B = 3; //En assignant la valeur 3 à OCR0B, cela signifie que le signal PWM généré aura un rapport cyclique de 11,72% (3/255), ce qui correspond à un signal avec une largeur d'impulsion de 11,72% de la période du signal.

Serial.println("Registre OCR0B = " + String(OCR0B,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

TCNT0 = 0; //mets à zéro le Timer0. Cette instruction permet de démarrer le compteur à partir de zéro et de compter à partir de ce point.

Serial.println("Registre TCNT0 = " + String(TCNT0,BIN)); //Ecriture du registre en binaire sur le moniteur serie pour controler sa valeur

Serial.println ();

Serial.println("---------------------------------------------------------------------------");

Serial.println ();

sei(); //maintenant que tous les registres essentielle au bon fonctionnement du code sont activé nous réactivons les interruptions

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

while (!EchantillonPret) {} // efface le premier échantillons

echec = 0; // variable qui compte combiens de temps l'ISR à etais éxecuter trops tard,Elles doit donc rester a 0 si tous marche bien.

EchantillonPret = false; //définie l'opération booléene de la variable EchantillonPret comme fausse

}

// L'Interruption de débordement du Timer0 sert 2 objectifs:

// 1. Il efface l'indicateur de débordement du Timer0. Si nous ne le faisons pas, le CAN ne verra plus de débordements du Timer0 et nous n'obtiendrons pas de conversions.

// 2. Il incrémente le compteur de ticks, ce qui permet de faire le chronométrage. Nous obtenons 62500 ticks/seconde.

// Nous lisons maintenant le CAN dans la routine d'interruption du temporisateur au lieu d'avoir une interruption complète de conversion séparée.

ISR(TIMER1\_OVF\_vect) //interruption de débordement du Timer1

{

++ticks; //incrémentation du compteur de ticks

uint8\_t compteur = TCNT0; //le registre TCNT0 stocke la valeur du compteur de temporisation du Timer0.

int16\_t val = (int16\_t)(uint16\_t)ADCH; //Nous avons besoin uniquement de lire les 8bits les plus significatis.

if (compteur != ((dernierCompteur + 1) & 7)) //sert à vérifier si la valeur de "compteur" a changé d'une unité depuis la dernière itération de la boucle.

{

++echec; //variable compte combiens de temps l'ISR à etais éxecuter trops tard,Elles doit donc rester a 0 si tous marche bien.

}

dernierCompteur = compteur; //copie la valeur de compteur dans dernierCompteur pour pouvoir faire la comparaison lors de la porchaine itération de boucle

int16\_t \*p = &ValeurBrut[compteur & 3]; //crée un pointeur "p" qui pointe vers "ValeurBrut". le masque " & " (ET binaire) permet de fixer ke nombre d'opération jusqu'a compteur = 3.

if (compteur < 4) //éxecution du code lors ce que compteur ne dépasse pas 4

{

\*p += (val);

if (\*p > 15000) \*p = 15000;

}

else //éxecution du code lors ce que compteur dépasse 4

{

\*p -= val;

if (\*p < -15000) \*p = -15000;

}

if (compteur == 7) //éxecution du code lors ce que compteur est parfaitement égal a 7

{

++Echantillon;

if (Echantillon == EchantillonageMax) //Si l'échantillonage a atteint le plafond définie par la constante EchantillonageMax (1024) alors la variable "Echantillon" est remise a zero pour la suite

{

Echantillon = 0;

if (!EchantillonPret) // Si l'echantillon précedent à etais consommé.

{

memcpy((void\*)Moyenne, ValeurBrut, sizeof(Moyenne)); //copier les données à partir de l'emplacement mémoire ValeurBrut vers l'emplacement mémoire Moyenne. memcpy() est une fonction qui permet de copier une séquence d'octets d'un emplacement mémoire à un autre.

EchantillonPret = true;

}

memset(ValeurBrut, 0, sizeof(ValeurBrut)); //initialise à zéro tous les éléments du tableau "ValeurBrut". La fonction "memset()" est une fonction qui permet de remplir une zone de mémoire avec une valeur donnée.

}

}

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void loop()

{

//Section de controle de l'alimentation

lecture\_Boutton\_off = analogRead(Boutton\_off); //Variable ou l'on stock la valeur relevé sur la pin de lecture du boutton off

if(lecture\_Boutton\_off >= Seuil\_Boutton\_off) //test de condition pour voire si le boutton est enfoncé

{

tempsAppui++;

delay(1000); //Variable que l'on incrémente pour faire couper l'alimentation

}

if(tempsAppui >= 3){ //Si la variable atteint la valeur définie alors on coupe le maintient de l'alimentation

digitalWrite(AlimMosfetCarte,LOW); //coupure de l'alimentation

}

if(tempsAppui > 0 && lecture\_Boutton\_off < Seuil\_Boutton\_off) //Test de condition pour voire si le compteur du temps d'appuie est supérieur à 0 et que le boutton a etais relaché, alors il faut réinitialisé le compteur.

{

tempsAppui=0; //réinitialisation du compteur.

}

//Fin de la séction de pilotage de l'alimentation

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

while (!EchantillonPret) {} //boucle qui s'exécute tant que la variable EchantillonPret est fausse. "!"" en face de EchantillonPret est un opérateur de négation qui inverse la valeur booléenne de EchantillonPret. Ainsi, tant que EchantillonPret est faux, le résultat de l'opération de négation sera vrai et la boucle continuera à s'exécuté

uint32\_t oldTicks = ticks;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

if (digitalRead(bouttonCalibrage) == LOW)

{

// lors ce que nous appuyons sur le Bouton d'étalonnage Nous sauvegardons les sorties du détecteur de phase actuel et les soustrayons des résultats futurs.

// Il est préférable de prendre plusieurs échantillons au lieu de n'en prendre qu'un seul pour plus de précision.

for (int i = 0; i < 4; ++i)

{

calib[i] = Moyenne[i];

}

EchantillonPret = false;

Serial.print("Calibré: ");

lcd.setCursor(0,0);

lcd.print("Calibration... ");

for (int i = 0; i < 4; ++i)

{

Serial.write(' ');

Serial.print(calib[i]);

lcd.setCursor(0,1);

lcd.print(' ');

lcd.print(calib[4]);

lcd.print(" ");

}

Serial.println();

}

else

{

for (int i = 0; i < 4; ++i)

{

Moyenne[i] -= calib[i];

}

const double f = 200.0;

// Lisse les résultats pour éliminer la sensibilité à la 3ème harmonique, et divisez par 200

double ValeurAffiner0 = (Moyenne[0] + Racine\_De\_Un\_Demi \* (Moyenne[1] - Moyenne[3]))/f;

double ValeurAffiner1 = (Moyenne[1] + Racine\_De\_Un\_Demi \* (Moyenne[0] + Moyenne[2]))/f;

double ValeurAffiner2 = (Moyenne[2] + Racine\_De\_Un\_Demi \* (Moyenne[1] + Moyenne[3]))/f;

double ValeurAffiner3 = (Moyenne[3] + Racine\_De\_Un\_Demi \* (Moyenne[2] - Moyenne[0]))/f;

EchantillonPret = false; // les moyennes ont étais traité à l'intérieur de nouvelles variables l'ISR est donc libre d'écraser les anciennes pour les prochains échantillonnage .

double amp1 = sqrt((ValeurAffiner0 \* ValeurAffiner0) + (ValeurAffiner2 \* ValeurAffiner2));

double amp2 = sqrt((ValeurAffiner1 \* ValeurAffiner1) + (ValeurAffiner3 \* ValeurAffiner3));

double AmpMoyenne = (amp1 + amp2)/2.0; //Nous réalisons le calcul de la moyenne pour les variables amp1 et amp2

// L'échantillonnage/blocage du CAN a lieu 2 horloges après le débordement du Timer.

double phase1 = atan2(ValeurAffiner0, ValeurAffiner2) \* ConvRadianVersDegres + 45.0;

double phase2 = atan2(ValeurAffiner1, ValeurAffiner3) \* ConvRadianVersDegres;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

if (phase1 > phase2) // si phase1 est supérieur à phase 2, alors nous inversion leur valeur, phase 1 prend la valeur de phase 2 et invérsement.

//cette oprétion permet de garder une valeur de phase1 inférieur a celle de phase2 cela permet de ne pas perturbé les calculs de moyennes car dans les calcul suivant nous allons retrouvé une opération du type "phase2 - phase1"

{

double temp = phase1; //inversement des valeur phase1 et phase2

phase1 = phase2;

phase2 = temp;

}

double phaseMoyenne = ((phase1 + phase2)/2.0) - Ajustement\_De\_Phase; //nous éffectuons le calcul de la moyenne des phase tous en soustrayant le déphasage

if (phase2 - phase1 > 180.0)

{

if (phaseMoyenne < 0.0)

{

phaseMoyenne += 180.0;

}

else

{

phaseMoyenne -= 180.0;

}

}

// À des fins de diagnostic, imprimez les comptages individuels des bacs et les 2 gains et phases calculés indépendamment.

Serial.println("echec : " +String(echec));

Serial.write(' ');

if (ValeurAffiner0 >= 0.0) Serial.write(' ');

Serial.println("ValeurAffiner0 : " +String(ValeurAffiner0, 2)); //Affiche la valeur de ValeurAffiner0 avec une précision de 2décimals

Serial.write(' '); //envoie un espace au port série pour aligné l'affichage des valeurs

if (ValeurAffiner1 >= 0.0) Serial.write(' ');

Serial.println("ValeurAffiner1 : " +String (ValeurAffiner1, 2)); //Affiche la valeur de ValeurAffiner1 avec une précision de 2décimals

Serial.write(' '); //envoie un espace au port série pour aligné l'affichage des valeurs

if (ValeurAffiner2 >= 0.0) Serial.write(' ');

Serial.println("ValeurAffiner2 :" +String (ValeurAffiner2, 2)); //Affiche la valeur de ValeurAffiner2 avec une précision de 2décimals

Serial.write(' '); //envoie un espace au port série pour aligné l'affichage des valeurs

if (ValeurAffiner3 >= 0.0) Serial.write(' ');

Serial.println("ValeurAffiner3 : " +String(ValeurAffiner3, 2)); //Affiche la valeur de ValeurAffiner3 avec une précision de 2décimals

Serial.print(" "); //envoie un espace au port série pour aligné l'affichage des valeurs

Serial.println("Amp1 : " +String(amp1, 2)); //Affiche la valeur de amp1 avec une précision de 2décimals

Serial.write(' '); //envoie un espace au port série pour aligné l'affichage des valeurs

Serial.println("Amp2 : " +String(amp2, 2)); //Affiche la valeur de amp2 avec une précision de 2décimals

Serial.write(' '); //envoie un espace au port série pour aligné l'affichage des valeurs

if (phase1 >= 0.0) Serial.write(' ');

Serial.println("Phase1 : " +String(phase1, 2)); //Affiche la valeur de phase1 avec une précision de 2décimals

Serial.write(' '); //envoie un espace au port série pour aligné l'affichage des valeurs

if (phase2 >= 0.0) Serial.write(' ');

Serial.println("Phase2 : " +String(phase2, 2)); //Affiche la valeur de phase2 avec une précision de 2décimals

// Imprimez l'amplitude et la phase finales, que nous utilisons pour décider de ce que (le cas échéant) nous avons trouvé).

if (AmpMoyenne >= 0.0) Serial.write(' ');

Serial.println("AmpMoyenne : " +String(AmpMoyenne, 1)); //Affiche la valeur de AmpMoyenne avec une précision de 1décimal

Serial.write(' '); //envoie un espace au port série pour aligné l'affichage des valeurs

lcd.setCursor(0,0);

lcd.print(" ");

lcd.print(AmpMoyenne);

lcd.setCursor(0,1);

lbg.drawValue(AmpMoyenne, max\_AmpMoyenne); //affichage de la bar graph selon la valeur de AmpMoyenne

if (phaseMoyenne >= 0.0) Serial.write(' ');

Serial.println("Phase Moyenne : " +String((int)phaseMoyenne));

if (AmpMoyenne >= Seuil) //Si la variable AmpMoyenne est supérieur a notre constante seuil alors nous pourrons faire les test de conditions pour la détéction

//donc si nous augmentons la valeur de la constante seuil il faudra une valeur plus importante a AmpMoyenne, donc plus la valeur de seuil est élevé moin le detecteur sera sensible

{

// Lorsqu'il est maintenu aligné avec le centre de la bobine:

// - les métaux non ferreux donnent un déphasage négatif, par ex. -90deg pour le cuivre ou l'aluminium épais, une olive en cuivre, -30deg pour l'aluminium fin..

// les métaux ferreux donnent un déphasage nul ou un petit déphasage positif.

// Nous dirons donc que tout ce qui a un déphasage inférieur à -20 degrés est non ferreux.

if (phaseMoyenne < -20.0) //si phaseMoyenne est inférieur a -20 alors nous sommes face a un métal non-ferreux

{

Serial.println("Non-ferreux"); // Décidez ce que nous avons trouvé et dites le à l'utilisateur via l'ecran LCD.

lcd.setCursor(0,0);

lcd.print("Non-ferreux ");

Serial.println ();

Serial.println("---------------------------------------------------------------------------");

Serial.println ();

}

else

{

Serial.println("ferreux"); //si phaseMoyenne est supérieur a 20 alors nous sommes face a un métal ferreux

lcd.setCursor(0,0);

lcd.print("ferreux ");

Serial.println ();

Serial.println("---------------------------------------------------------------------------");

Serial.println ();

}

float temp = AmpMoyenne; //stockage de la valeur AmpMoyenne qui nous servait a la detection dans la variable temp qui est une variable flotante

int PilotageBuzzer = map (temp, 10, 200, 100, 1500); //Pilotage du buzzer en fonction de temp (valeur de AmpMoyenne en flotant)

tone(Pinbuzzer, PilotageBuzzer,120);

while (temp > Seuil)

{

temp -= (Seuil/2);

}

}

}

while (ticks - oldTicks < 8000)

{

}

}