RAPPORT DE SAE

STYLO-VOLTMÈTRE

ALTech

ALTech
03/06/2024
ADAM Théo-Félix
LANDREAU Alexandre
SAAD-DJABALLAH Séif-Din

Université de Poitiers

SOMMAIRE

A. CONCEVOIR

Parmi les compétences du Bachelor Universitaire de Technologie en Génie Électrique et Informatique Industrielle, nous trouvons la partie concevoir. Pour notre Situation d'Apprentissage et d'évaluation sur le stylo-voltmètre, nous avons conçu la partie transmission.

SOUS - SOMMAIRE

1. Objectif

L'objectif de notre Situation d'Apprentissage et d'évaluation est de créer un stylo-voltmètre, pouvant être utilisé pour connaître une tension comprise entre 0 et 14 V en continu. Pour mesurer la tension il faut utiliser un convertisseur analogique numérique pour transformer les données de tensions en données comprises par le microprocesseur.

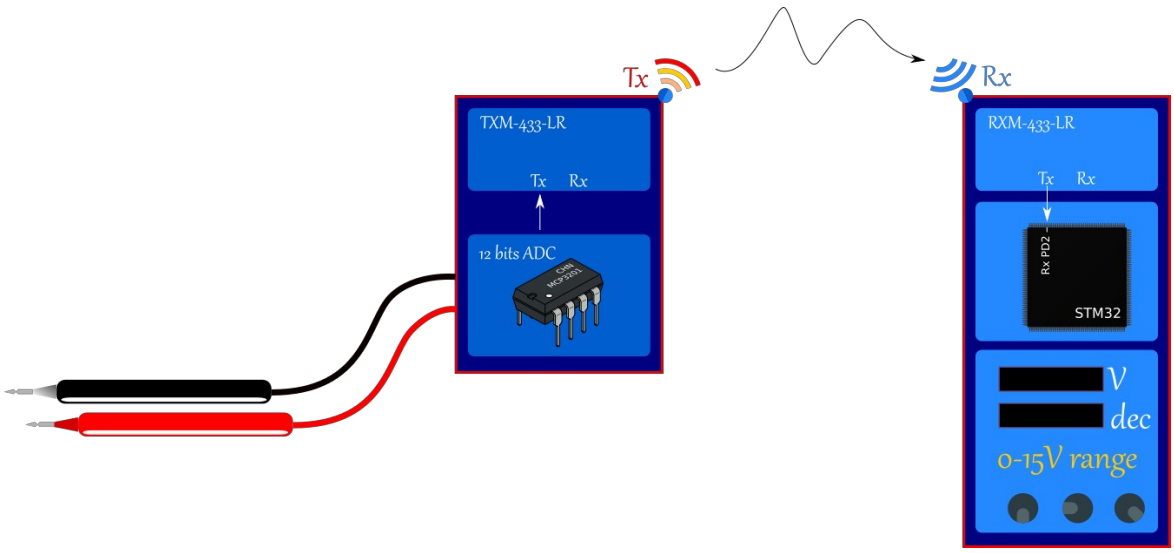


Figure I - Synoptique Du Stylo

2. Schéma Fonctionnel

3. Chronogrammes

A. Signaux Émis

Le stylo-voltmètre communique avec le STM32, via le protocole UART (Universal Asynchronous Receiver / Transmiter). Or, la trame UART est composée, dans l'ordre : d'un bit de start, des bits de données, d'une parité (que nous avons choisi paire) et d'un bit de stop. Nous avons alors créé la trame suivante :

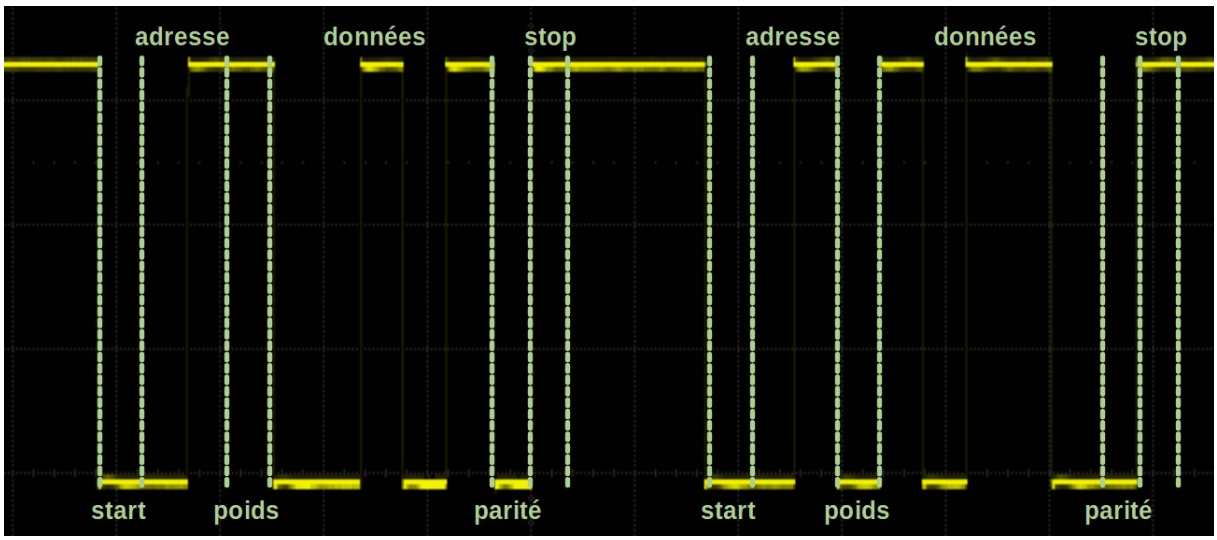


Figure II - Exemple De Trame UART Émise

B. Signaux Reçus

4. Circuits Imprimés

A. Carte De Programmation AtTiny85

Pour programmer le microcontrôleur Tiny 85 du fabricant Atmel, nous avons créé un circuit imprimé pouvant accueillir l'AtTiny85. Ce denier a été conçu avec le logiciel Altium Designer.

Le schéma est le suivant :

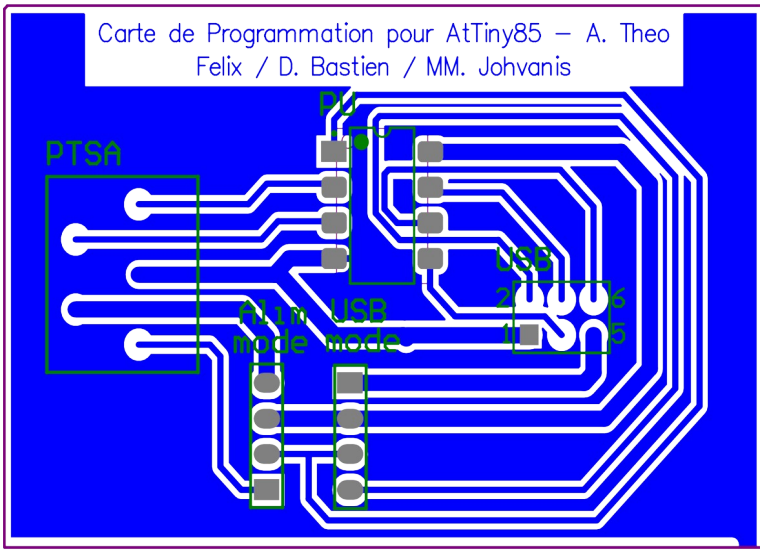


Figure III - Circuit Imprimée Pour Programmation AtTiny85

Attribution des broches :

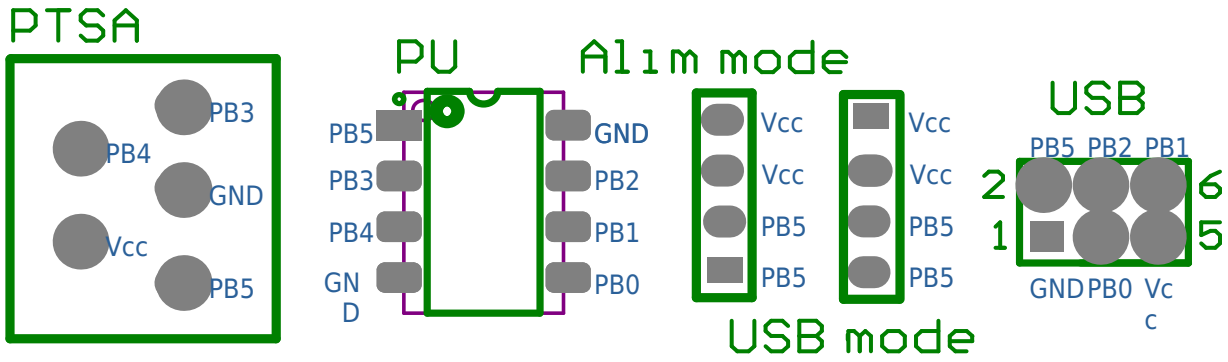


Figure IV - Broches Des Composants Du Circuit Imprimé

Nom	Composant	Fonction	Description
Vcc	-	-	Tension d'alimentation.
GND	-	-	Terre.
PB0	USBASP	MOSI	Sortie maître et entrée esclave.
PB1	USBASP	MISO	Entrée maître et sortie esclave.
PB2	USBASP	SCK	Horloge.
PB3	ATTINY85	DATA	Sortie des données traitées.
PB4	ATTINY85	ADC2	Entrée CAN 2.
PB5	ATTINY85	RST	Remise à zéro.

- Vcc : La tension d'alimentation est de 3,3 V, en utilisant une alimentation stabilisée. La tension est 5 V quand on utilise le USBASP et donc l'ordinateur.
- GND : C'est la masse commune.
- PB0 : Cette broche permet une liaison SPI entre l'ordinateur (le maître) et le microprocesseur AtTiny85 (l'esclave), via le USBASP. C'est sur cette broche que l'esclave reçoit les données du maître.
- PB1 : Cette broche permet une liaison SPI entre l'ordinateur (le maître) et le microprocesseur AtTiny85 (l'esclave), via le USBASP. C'est sur cette broche que le maître reçoit les données de l'esclave.
- PB2 : Cette broche permet au maître de cadencer l'envoi des bits à l'esclave.
- PB3 : Après l'implémentation du programme dans le microprocesseur, cette broche permet la visualisation du signal de sortie créer par l'AtTiny85. Selon le protocole utilisé, cela peut-être du SPI (semestre 3) ou de l'UART (semestre 4).
- PB4 : Après implémentation du programme dans le microprocesseur, cette broche permet l'entrée d'un signal analogique compris entre 0 et 3,3 V continu. Alors, l'AtTiny85 va pouvoir le convertir en données numériques.
- PB5 : Cette broche permet au maître de remettre à zéro le programme contenu dans le microprocesseur.

B. Prototype Du Stylo-Voltmètre

Avant d'imprimer notre stylo-voltmètre sur un circuit imprimé, nous avons réalisé un prototype pour tester nos programmes, comprenant la conversion analogique numérique, le clignotement des DEL, l'utilisation d'un bouton... Sur cette carte, la CAN peut se faire avec le MCP3201, qui était notre premier choix.

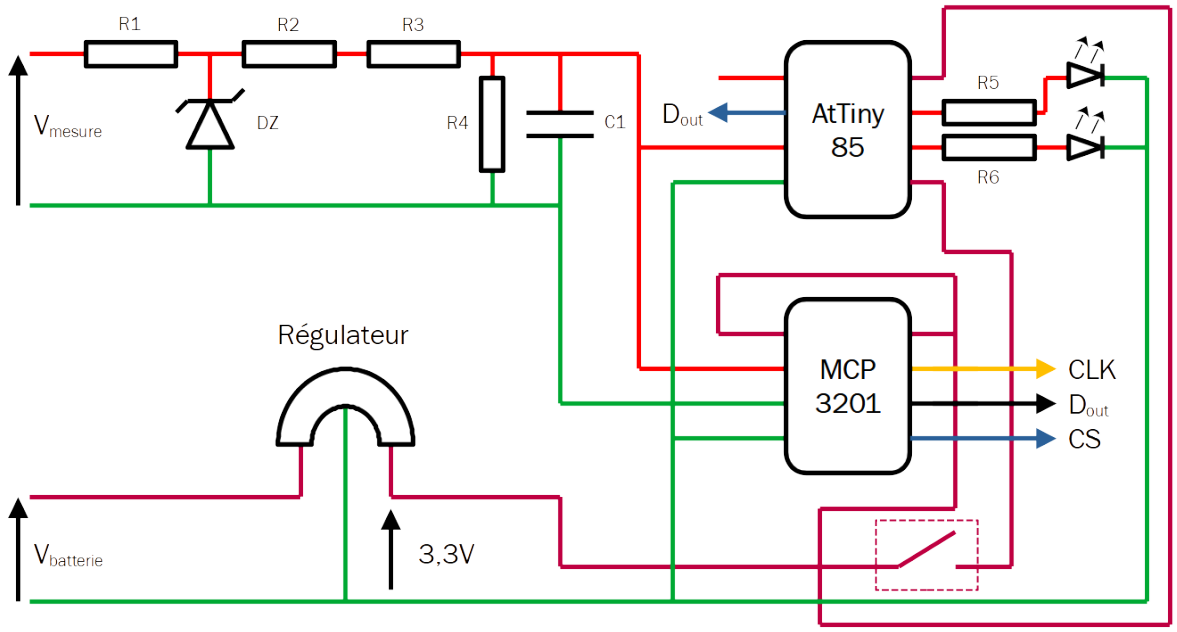


Figure V - Schéma Du Prototype Du Stylo-Voltmètre

Avec les composants suivant :

Nom	Valeur	Fonction	Fabricant
R1	470 Ω	Résistor	TE Connectivity
R2	390 kΩ	Résistor	TE Connectivity
R3	330 kΩ	Résistor	TE Connectivity
R4	220 kΩ	Résistor	TE Connectivity
R5/R6	330 Ω	Résistor	TE Connectivity
C1	100 nF	Condensateur	EPCOS

C. Carte De Transmission

Le module de transmission est le XX du fabricant Linx. Il est déjà intégré à un circuit imprimé assemblé, avec huit broches. Attention toute fois, il ne correspond pas exactement à un circuit intégré type DIP 8, puisque l'écart entre les broches est plus grand. Sur notre stylo-voltmètre, notre Linx porte la référence RT4254.

D. Carte De Réception

La réception s'effectue directement sur le STM32. En effet, certaines cartes de programmation STM32 intègrent directement le module de réception Linx, ainsi que l'antenne. En analysant les cartes déjà imprimées, nous trouvons le schéma suivant :

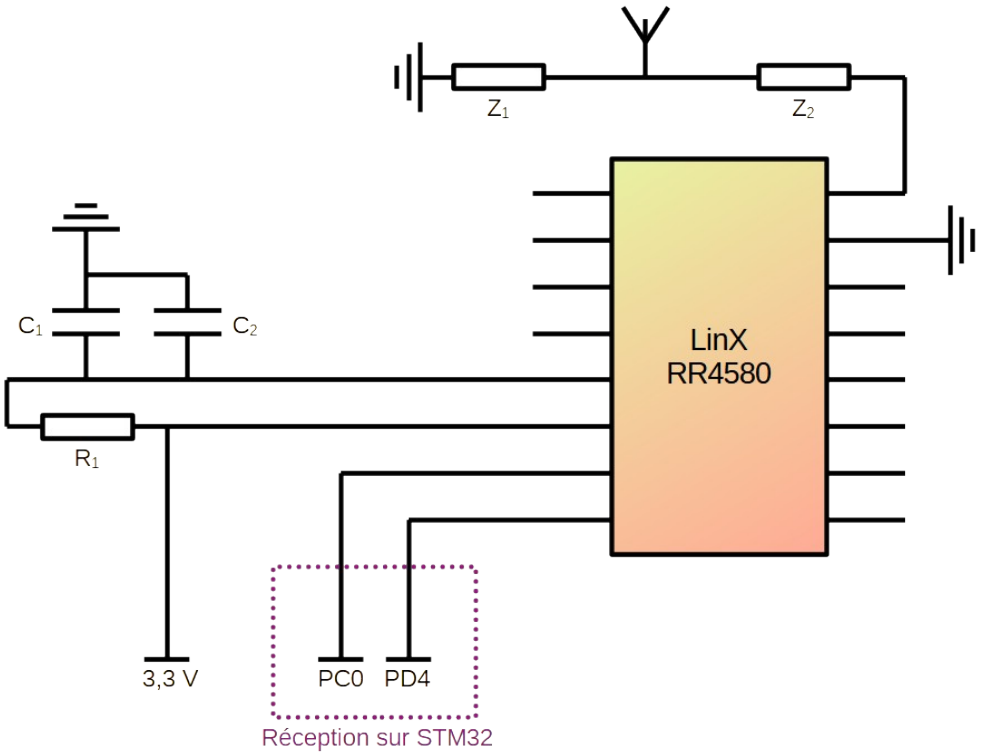


Figure VI - Schéma Du Module De Réception Sur STM32

Or, le fabricant Linx nous donne l'attribution des broches suivant :

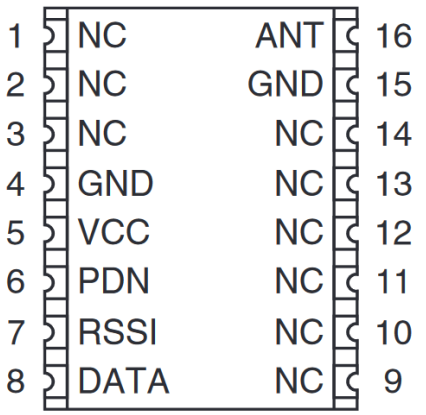


Figure VII - Broches Du Linx Rx

Nom	E/S	Broche	Description
V_{CC}	E	5	Tension d'alimentation.
GND	-	4/15	Terre.
PDN	E	6	« Power Down », mettre à zéro cette broche met le récepteur dans un état de courant faible et le module ne pourra pas recevoir de signal.
RSSI	S	7	« Received Signal Strength Indicator », cette broche fournit une tension analogique proportionnelle à l'intensité du signal reçu.
DATA	S	8	Sortie du signal numérique démodulé.
ANT	E	16	Antenne.

E. Carte Du Stylo-Voltmètre

À l'aide de toutes nos connaissances des précédents essais pour le stylo-voltmètre, nous avons fait le schéma ci-dessous, à l'aide du logiciel KiCad, qui ne nécessite aucune licence.

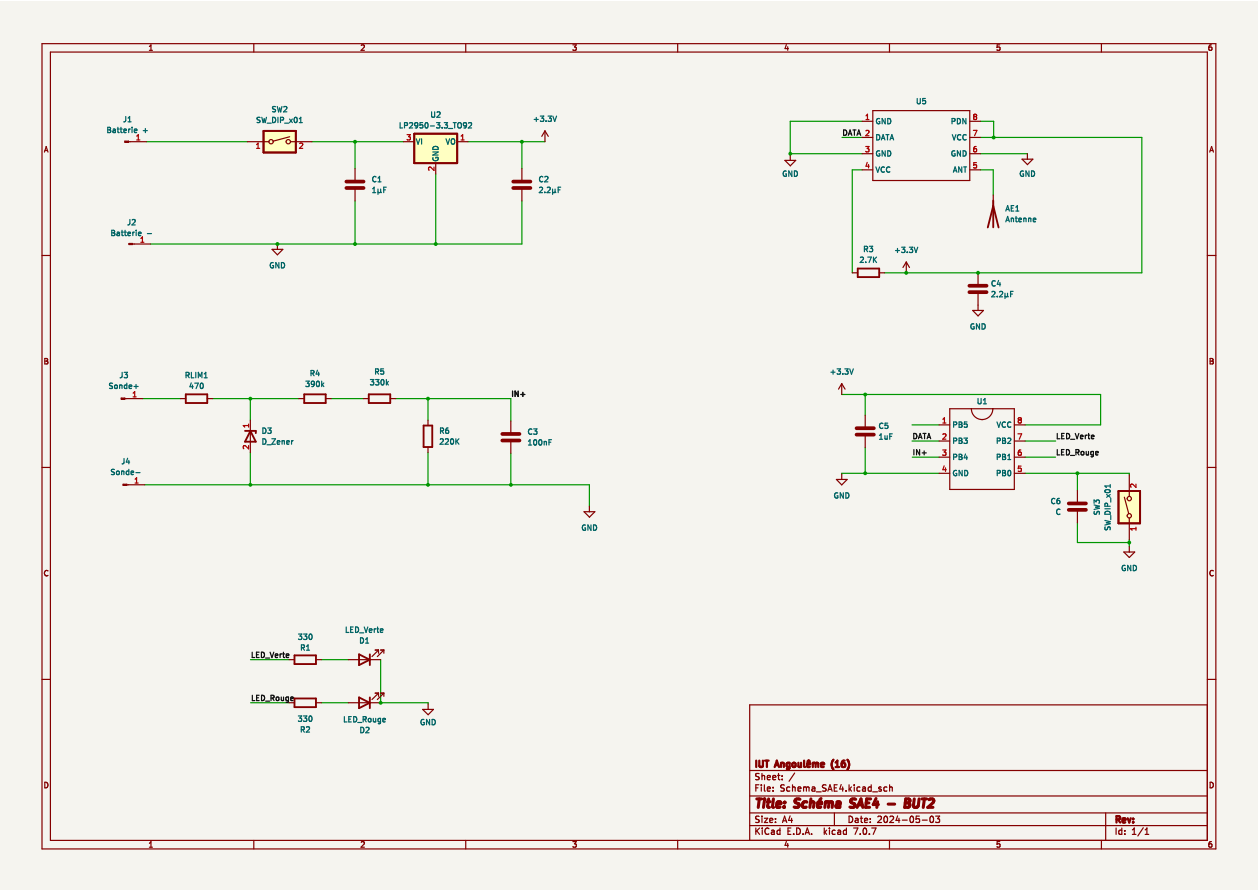


Figure VIII - Schéma Finale Du Stylo-Voltmètre

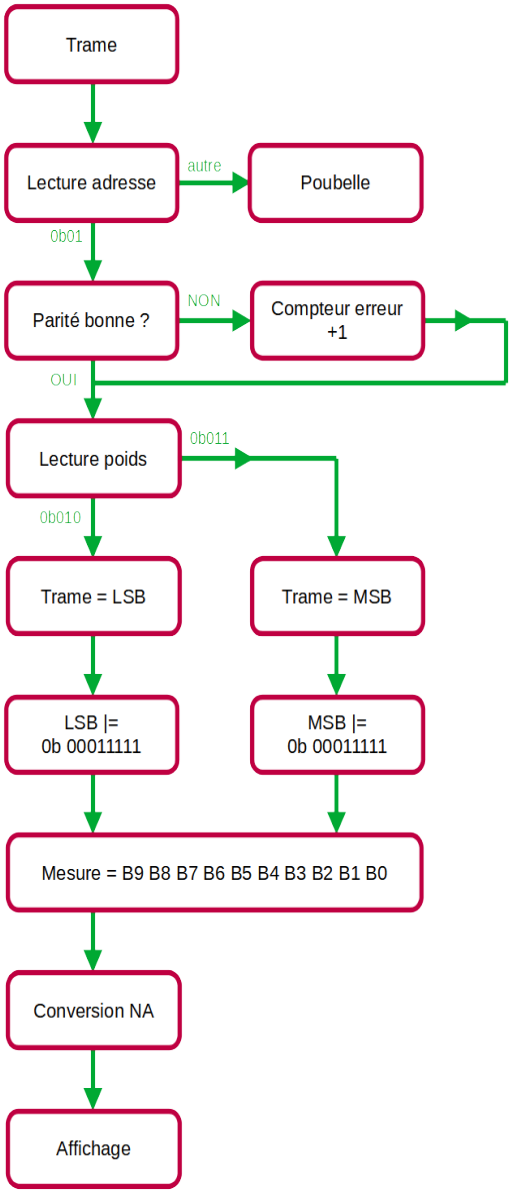
B. IMPLÉMENTER

Parmi les compétences du Bachelor Universitaire de Technologie en Génie Électrique et Informatique Industrielle, nous trouvons la partie implémenter. Pour notre Situation d'Apprentissage et d'évaluation sur le stylo-voltmètre, nous avons programmé l'AtTiny85 pour la transmission et le STM32 pour la réception.

SOUS – SOMMAIRE

1. Algorithmes

A. Réception Des Trames Sous STM32


Trame
Lecture adresse
autre
Poubelle
0b01
Parité bonne ?
NON
Compteur erreur
+1
OUI
Lecture poids
0b011
0b010
Trame = LSB
Trame = MSB
LSB |=
0b 00011111
MSB |=
0b 00011111
Mesure = B9 B8 B7 B6 B5 B4 B3 B2 B1 B0
Conversion NA
Affichage

2. Briques De Code AtTiny85

Pour pouvoir programmer notre AtTiny85, nous avons utilisé plusieurs fonctions en langage C.

- **Convertisseur Analogique Numérique** : Cela permet de convertir nos données analogiques en données numériques exploitable pour les microprocesseurs. Ici, nous avons besoin d'une lecture sur la broche 3. Nous utilisons alors le CAN 2.
- **Interruptions** : Cela permet de faire une pause dans le programme, pendant une période de temps bien précise.
- **Manipulation de bits** : Nous avons manipulé les bits.

3. Programme AtTiny85

```
// I/O Registers definitions
#include <tiny85.h>
#include <delay.h>

                        ////////// INITIALISATION DES VARIABLES //////////

unsigned char interupt = 0;
unsigned char CycleNb = 0;
unsigned char Compteur = 0;
unsigned char trameH;
unsigned char trameB;
unsigned short lecture;
void SetBit(unsigned int non);
void ClearBit(unsigned int oui);

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

// Voltage Reference: AVCC pin
#define ADC_VREF_TYPE ((0<<REFS1) | (0<<REFS0) | (0<<REFS2) | (0<<ADLAR))

// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=(adc_input & 0x0f) | ADC_VREF_TYPE;
// Delay needed for the stabilization of the ADC input voltage
delay_us(10);
// Start the AD conversion
ADCSRA|=(1<<ADSC);
// Wait for the AD conversion to complete
while ((ADCSRA & (1<<ADIF))==0);
ADCSRA|=(1<<ADIF);
return ADCW;
}

                                ////////// INTERRUPTION //////////

interrupt [TIM0_COMPA] void timer0_compa_isr(void)
{
    if (interupt == 1)
    {
        CycleNb++;

                                ////////// BIT_START MSB //////////
        if(CycleNb == 1)
        {
            ClearBit(PORTB3);
        }
    }
                                  ////////// Trame_MSB //////////
     if((CycleNb >= 1) & (CycleNb < 10))
     {

       if((trameH & (1<<(9 - CycleNb))) == (1<<(9-CycleNb)))
       {
            SetBit(PORTB3);
            Compteur++;
       }
       else
       {
            ClearBit(PORTB3);
       }
     }
                             ////////// BIT DE PARITE MSB //////////
     if(CycleNb == 10) //Parité MSB
     {
        if((Compteur %2) == 1)
        {
            SetBit(PORTB3);
        }
```

```
      else
        {
             ClearBit(PORTB3);
        }
            Compteur = 0;
        }
                          ////////// TEMPS ENTRE DEUX TRAMES //////////
        if((CycleNb >= 11) & (CycleNb < 15))
        {

            SetBit(PORTB3);

        }
                                ////////// BIT_START LSB //////////
         if(CycleNb == 15)
        {
            ClearBit(PORTB3);
        }
                                  ////////// Trame_LSB //////////
        if((CycleNb >= 16) & (CycleNb <24))
        {

            if((trameB & (1<<(23 - CycleNb))) == (1<<(23 - CycleNb)))
            {

                SetBit(PORTB3);
                Compteur++;
            }
            else
            {
                ClearBit(PORTB3);
            }
        }
                               ////////// BIT PARITE LSB //////////
        if(CycleNb == 24) //Parité LSB
        {

            if((Compteur %2) == 1)
            {

                SetBit(PORTB3);

            }
            else{

                ClearBit(PORTB3);

            }
        }
                                  ////////// BIT_STOP //////////
        if(CycleNb == 25)
        {
            SetBit(PORTB3);
            CycleNb = 0;
            Compteur = 0;
            interupt = 0;
        }
}
                                ////////////INTERRUPTION 2 //////////

// Timer1 overflow interrupt service routine
interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
// Reinitialize Timer1 value
TCNT1=0x00;
// Place your code here

    if(lecture <= 0b1111100000)
    {
        TCNT1 = 10+(lecture >> 2);
        PORTB ^= 0x2;
    }

}
```

```
void main(void)
{
//Paramètres de l'oscillateur crystal
#pragma optsize-
CLKPR=(1<<CLKPCE);
CLKPR=(0<<CLKPCE) | (0<<CLKPS3) | (0<<CLKPS2) | (0<<CLKPS1) | (0<<CLKPS0);
#ifdef _OPTIMIZE_SIZE_
#pragma optsize+
#endif

//Définition des ports
DDRB=(0<<DDB5) | (0<<DDB4) | (1<<DDB3) | (1<<DDB2) | (1<<DDB1) | (0<<DDB0);
PORTB=(0<<PORTB5) | (0<<PORTB4) | (0<<PORTB3) | (0<<PORTB2) | (0<<PORTB1) | (1<<PORTB0);

TCCR0A=(0<<COM0A1) | (0<<COM0A0) | (0<<COM0B1) | (0<<COM0B0) | (1<<WGM01) | (0<<WGM00);
TCCR0B=(0<<WGM02) | (0<<CS02) | (1<<CS01) | (0<<CS00);
TCNT0=0x00;
OCR0A=0xCF;
OCR0B=0x00;

//Initialisation du timer 1
PLLCSR=(0<<PCKE) | (0<<PLLE) | (0<<PLOCK);
TCCR1=(0<<CTC1) | (0<<PWM1A) | (0<<COM1A1) | (0<<COM1A0) | (1<<CS13) | (1<<CS12) | (1<<CS11) |
(1<<CS10);
GTCCR=(0<<TSM) | (0<<PWM1B) | (0<<COM1B1) | (0<<COM1B0) | (0<<PSR1) | (0<<PSR0);
TCNT1=0x00;
OCR1A=0x00;
OCR1B=0x00;
OCR1C=0x00;

//Initialisation des interruptions
TIMSK=(0<<OCIE1A) | (0<<OCIE1B) | (1<<OCIE0A) | (0<<OCIE0B) | (1<<TOIE1) | (0<<TOIE0);

ACSR=(1<<ACD) | (0<<ACBG) | (0<<ACO) | (0<<ACI) | (0<<ACIE) | (0<<ACIS1) | (0<<ACIS0);
ADCSRB=(0<<ACME);

DIDR0=(0<<AIN0D) | (0<<AIN1D);
ADCSRA=(1<<ADEN) | (0<<ADSC) | (0<<ADATE) | (0<<ADIF) | (0<<ADIE) | (0<<ADPS2) | (0<<ADPS1) |
(0<<ADPS0);
ADCSRB=(0<<BIN) | (0<<IPR) | (0<<ADTS2) | (0<<ADTS1) | (0<<ADTS0);

//Activation globale
#asm("sei")

//Initialisation du CAN
DIDR0|=(1<<ADC0D) | (0<<ADC2D) | (1<<ADC3D) | (1<<ADC1D);
ADMUX=ADC_VREF_TYPE;

  while (1)
  {
        lecture = read_adc(0b00000011); //Lecture CAN
        trameB = (lecture & 0b0000011111) | 0b01000000; //Séparation trame basse
        trameH = (lecture >> 5) | 0b01100000; //Séparation trame haute
        interupt = 1; //Interruption
        while (interupt ==1);
        delay_ms(20); //Délais
  }

}

void ClearBit(unsigned int oui) //Fonction mise à 0 broche
{ PORTB &= ~(1<<oui);}

void SetBit(unsigned int non) //Fonction mise à 1 broche
{ PORTB |= (1<<non);}
```

4. Briques De Code STM32

Pour pouvoir programmer notre STM32, nous avons utilisé plusieurs fonctions en langage C++. Nous les avons répertorié ci-dessous :

- TS (Touch Screen) : Cette fonction permet de savoir si l'écran est touché du doigt ou non.

5. Programme STM32

```
//////////// INITIALISATION DES VARIABLES /////////

uint8_t partieA = 0b00000000;
uint8_t partieB = 0b00000000;
uint8_t partieAz = 0b00000000;
uint8_t partieBz = 0b00000000;
uint16_t partieC = 0b00000000;
uint16_t buffer1[10];
float Q = 0;
float TrameTraitee;
uint16_t TrameBrute1;
uint16_t TrameBrute2;
uint8_t TrameBruteTable1[] = {0b00000000};

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

int main(void)
{
  /* MCU Configuration--------------------------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_CRC_Init();
  MX_DMA2D_Init();
  MX_FMC_Init();
  MX_I2C3_Init();
  MX_LTDC_Init();
  MX_SPI5_Init();
  MX_TIM1_Init();
  MX_TIM10_Init();
  MX_UART5_Init();
  MX_USART1_UART_Init();
  MX_USB_OTG_HS_HCD_Init();

  /* DÉBUT DU CODE 2 */

  Q = 14/pow(2,10); //Calcul du quantum
   HAL_UART_Receive_IT(&huart5, TrameBruteTable1, 1); //Vous devez basculer un point d'arrêt sur
cette ligne !
  BSP_LCD_Init();
  BSP_LCD_LayerDefaultInit(LCD_FOREGROUND_LAYER, (LCD_FRAME_BUFFER ));
  BSP_LCD_SelectLayer(LCD_FOREGROUND_LAYER);
  BSP_LCD_Clear(LCD_COLOR_WHITE);
  BSP_LCD_SetBackColor(color);
  BSP_LCD_DisplayOn();

  //Configure le Touch Screen (TS)
  BSP_TS_Init(240,320);

  /* Obtenir la largeur et la hauteur de l'écran LCD */
  LCD_X_Size = BSP_LCD_GetXSize();
  LCD_Y_Size = BSP_LCD_GetYSize();

  /* Définir l'événement de ligne LTDC  */
  HAL_LTDC_ProgramLineEvent(&hltdc, 0);

  //Affichage de l'ecran d'accueil
  HAL_LTDC_ProgramLineEvent(&hltdc, 0);
  BSP_LCD_SetFont(&Font24);
  CopyBuffer((uint32_t  *)Background.data, (uint32_t  *)LCD_FRAME_BUFFER, 0  , 0, Background.width,
Background.height);

  /* FIN DU CODE 2 */
```

```
  /* Boucle infinie */
  /* DÉBUT DU CODE WHILE */
  while (1)
{
    /* DÉBUT DU CODE 3 */

                ////////// PARTIE TRAITEMENT DE LA TRAME MSB //////////

  if ((TrameBruteTable1[0] & 0b000000110) == 0b000000110){ //Détermination de la trame Haute (MSB).
        partieA = (TrameBruteTable1[0] >> 3) & (0b11111000 >> 3);    //On supprime l'adresse et le
bit MSB puis on la multiplie par un masque pour avoir que notre data.
        partieAz = (partieA & 0b10000) >> 4;  //On ne garde que le premier bit que l'on décale tout
à droite, car la trame vient à l'envers, ce qui donne 0b00001.
        partieAz = partieAz | ((partieA & 0b01000) >> 2);    //On prend le deuxième bit reçu, on le
décale de deux puis on assemble ce qui donne 0b00011.
        partieAz = partieAz | ((partieA & 0b00100));  //On ne touche pas à ce bit on le met juste
avec les autres ce qui donne 0b00111.
        partieAz = partieAz | ((partieA & 0b00010) << 2);    //Cette fois-ci on décale de deux vers
la gauche.
        partieAz = partieAz | ((partieA & 0b00001) << 4);
         }

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

             ////////// PARTIE TRAITEMENT DE LA TRAME LSB //////////

  if ((TrameBruteTable1[0] & 0b000000010) == 0b000000010){   //Determination de la trame Basse
(LSB).
        partieB = (TrameBruteTable1[0] >> 3) & (0b11111000 >> 3);    //On supprime l'adresse et le
bit LSB puis on la multiplie par un masque pour avoir que notre data.
        partieBz = (partieB & 0b10000) >> 4;  //On ne garde que le premier bit que l'on décale tout
à droite car la trame vient à l'envers, ce qui donne 0b00001.
        partieBz = partieBz | ((partieB & 0b01000) >> 2);    //On prends le deuxième bit reçu, on
le décaler de deux puis on assemble ce qui donne 0b00011.
        partieBz = partieBz | ((partieB & 0b00100));  //On ne touche pas à ce bit on le met juste
avec les autres ce qui donne 0b00111.
        partieBz = partieBz | ((partieB & 0b00010) << 2);    //Cette fois ci on décale de deux vers
la gauche.
        partieBz = partieBz | ((partieB & 0b00001) << 4);    //On décale de 4 vers la gauche.

        }      //Toutes les précédente fonctions sont nécéssaire car lors de l'envoie des bit nous
recevons la trame mais avec chaque bits inversés

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

            ////////// PARTIE ASSEMBLAGE DES DEUX TRAMES //////////

             partieC = (partieAz << 5) | partieBz; //On décale la trame MSB afin de laisser la
place à la trame LSB afin de les assembler.
         TrameTraitee = partieC * Q;  //On multiplie notre trame par le quantum pour avoir une
valeur décimale.
          BSP_LCD_SetTextColor(LCD_COLOR_BLACK);      //Met le texte en noir.
          itoa(TrameTraitee,buffer1, 10);     //Cette fonction permet de convertir les chiffres de
la valeur donnée en une chaîne de caractères en les mettant dans un buffer.

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

              ////////// PARTIE AFFICHAGE DE LA VALEUR //////////

  if (TrameTraitee >= 10){     //Si la tension est superieur ou égale à 10 V.
          BSP_LCD_DisplayStringAt(Xp-10,Yp,(uint16_t *)buffer1, LEFT_MODE);  //Affichage          et
positionnement des unités.
          BSP_LCD_DisplayStringAt(Xp+22,Yp, (char *)",", LEFT_MODE); //Affichage  et  positionnement
de la virgule.
          } else if (TrameTraitee < 10 & TrameTraitee>=1){    //Si la tension est inférieure à 10 V
et supérieure ou égale à 1 V.
          BSP_LCD_DisplayStringAt(Xp-12,Yp, (char *)"0", LEFT_MODE); //Affichage  et  positionnement
du 0.
          BSP_LCD_DisplayStringAt(Xp+6,Yp,(uint16_t *)buffer1, LEFT_MODE);   //Affichage          et
positionnement de la valeurs sans virgules.
```

```
          BSP_LCD_DisplayStringAt(Xp+22,Yp, (char *)",", LEFT_MODE); //Affichage  et  positionnement
de la virgule.
          } else if(TrameTraitee < 1 & TrameTraitee>=0.1) {   //Si  la  tension  est  inférieure  à  1 V
et supérieure ou égale à 0.1 V.
          BSP_LCD_DisplayStringAt(Xp-12,Yp, (char *)"0", LEFT_MODE); //Affichage  et  positionnement
du 0.
          BSP_LCD_DisplayStringAt(Xp+6,Yp,(uint16_t *)buffer1, LEFT_MODE);   //Affichage          et
positionnement de la valeur sans virgules.
          BSP_LCD_DisplayStringAt(Xp+22,Yp, (char *)",", LEFT_MODE); //Affichage  et  positionnement
de la virgule.
          } else {
          BSP_LCD_DisplayStringAt(Xp-12,Yp, (char *)"0", LEFT_MODE);
          BSP_LCD_DisplayStringAt(Xp+6,Yp,(uint16_t *)buffer1, LEFT_MODE);
          BSP_LCD_DisplayStringAt(Xp+22,Yp, (char *)",", LEFT_MODE); //Affichage  et  positionnement
de la virgule.
          BSP_LCD_DisplayStringAt(Xp+56,Yp, (char *)"0", LEFT_MODE);
         }

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

          TrameTraitee = 10*(TrameTraitee - abs(TrameTraitee));              //
          itoa(TrameTraitee,buffer1, 10);      //Met la trame traitée dans le buffer.
          BSP_LCD_DisplayStringAt(Xp + 40,Yp, (uint16_t *)buffer1, LEFT_MODE);
        //Affiche la valeur avec un certain positionnement
 }
  /* FIN DU CODE 3 */
}
  /* FIN DU CODE WHILE */
```

C. VÉRIFIER

Pour pouvoir programmer ce microcontrôleur, il est nécessaire de concevoir une carte pouvant loger le microcontrôleur, un plot de cinq broches, un switch pour une alimentation via l'ordinateur ou via une alimentation stabilisée et un header de 2x

Pour la Conversion Analogique Numérique (CAN), nous utilisons le CAN de l'AtTiny85 du fabricant Atmel. C'est un CAN 10 bits, dont la lecture se fait sur la pin 3, soit PB4, soit l'ADC 2.

SOUS - SOMMAIRE

1. Conversion Analogique Numérique

Nous avons testé le CAN, associé au protocole UART. Nous trouvons le tableau des valeurs suivant :

Trame Complète	$Trame_{(2)}$	$Trame_{(10)}$	$V_{prat\_in\_stylo}$	$V_{prat\_in\_attiny}$	$V_{théo\_in\_stylo}$	$V_{théo\_in\_attiny}$
0011000000111100100000011 1...	0	0	0	0	0	0
0011000011111001000011111...	100011	35	0	0,179	0,5126953125	0,11279296875
0011001010111100101011001 1...	10110110	182	2,59	0,63	2,666015625	0,5865234375
0011010100111100101110001 1...	101011100	348	4,68	0,862	5,09765625	1,121484375
0011100100111100100111001 1...	1001001110	590	9,24	1,98	8,642578125	1,9013671875
0011110011111001011100011...	1100111100	828	12,75	2,74	12,12890625	2,668359375
0011111010111100100100001 1...	1110101000	936	17,2	3,08	13,7109375	3,01640625

Oscilloscope :

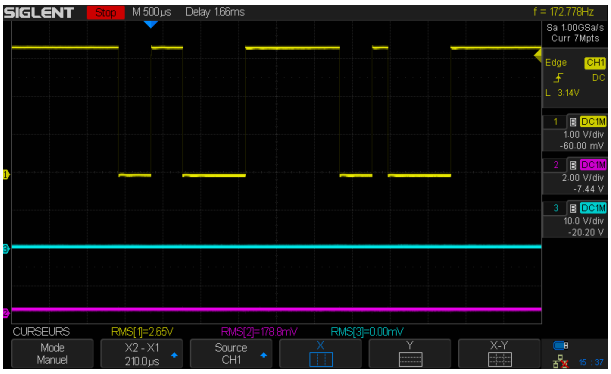


Figure IX - Trame À 0,180 V

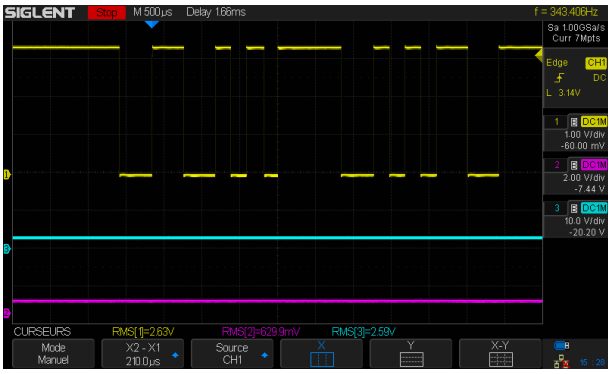


Figure X - Trame À 0,630 V

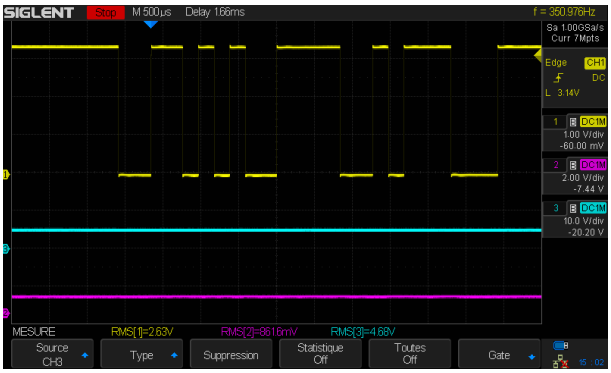


Figure XI - Trame À 0,862 V

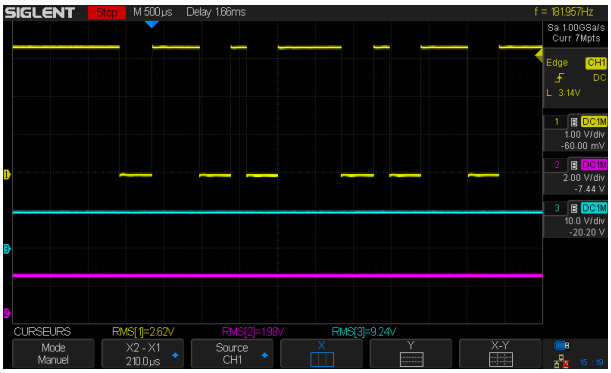


Figure XII - Trame À 1,980 V

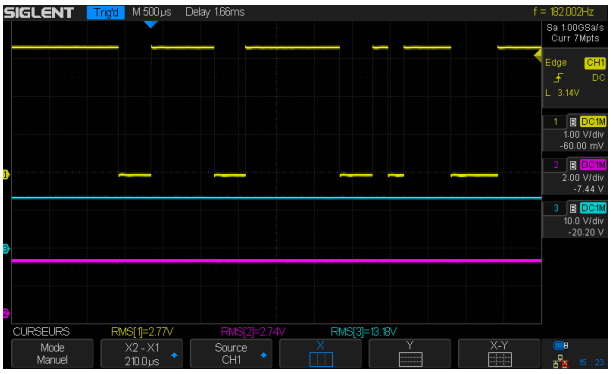


Figure XIII - Trame À 2,740 V

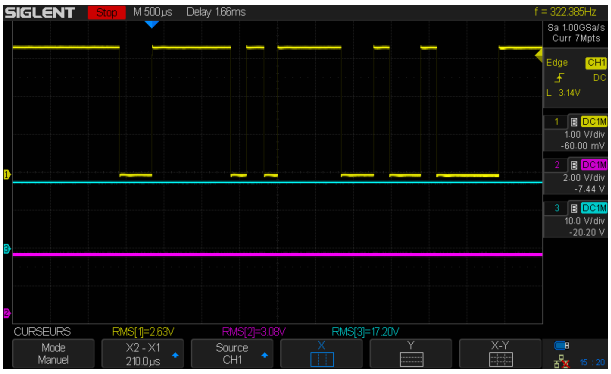


Figure XIV - Trame À 3,080 V

Valeur au GBF :

Figure XV - Tension À 0,5V

Figure XVI - Tension À 2,9V

Figure XVII - Tension À 5,4V

Figure XVIII - Tension À 9,5V

Figure XIX - Tension À 13,4V

Figure XX - Tension À 17,4V

En utilisant nos valeurs du tableau, nous trouvons les courbes suivantes :

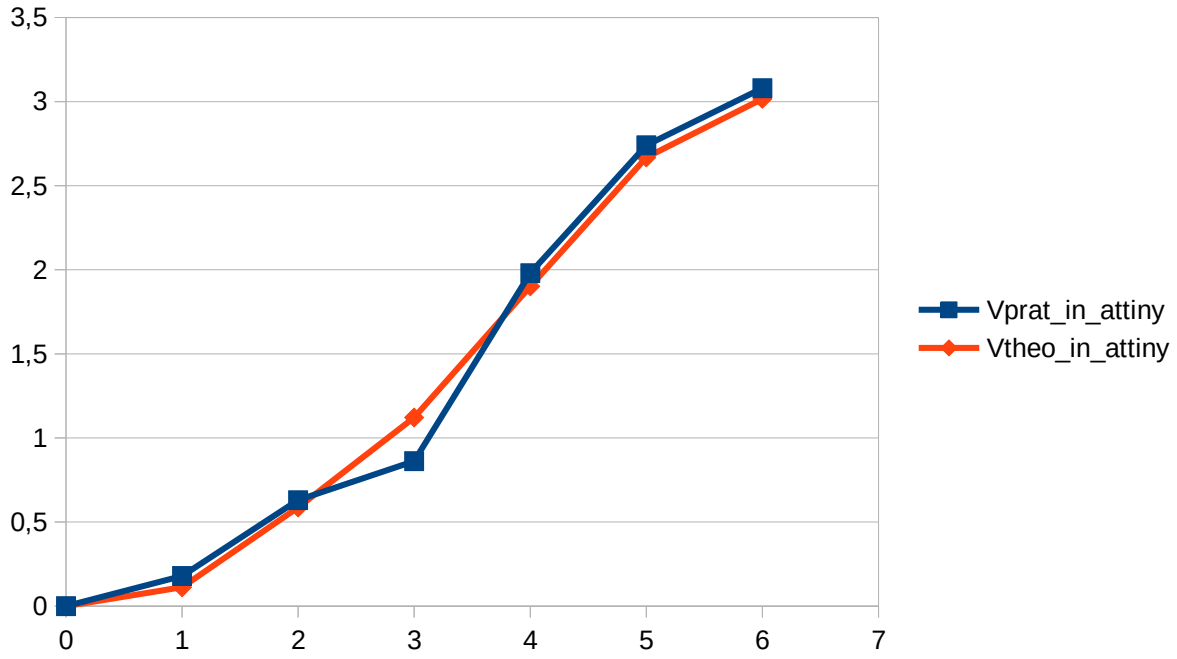


Figure XXI - Courbes Théorique Et Pratique De La Tension En Entrée De L'AtTiny85

Nous observons que la courbe pratique suit la même allure que la courbe théorique, sauf à la troisième valeur, ce qui est peut-être dû à une mauvaise manipulation.

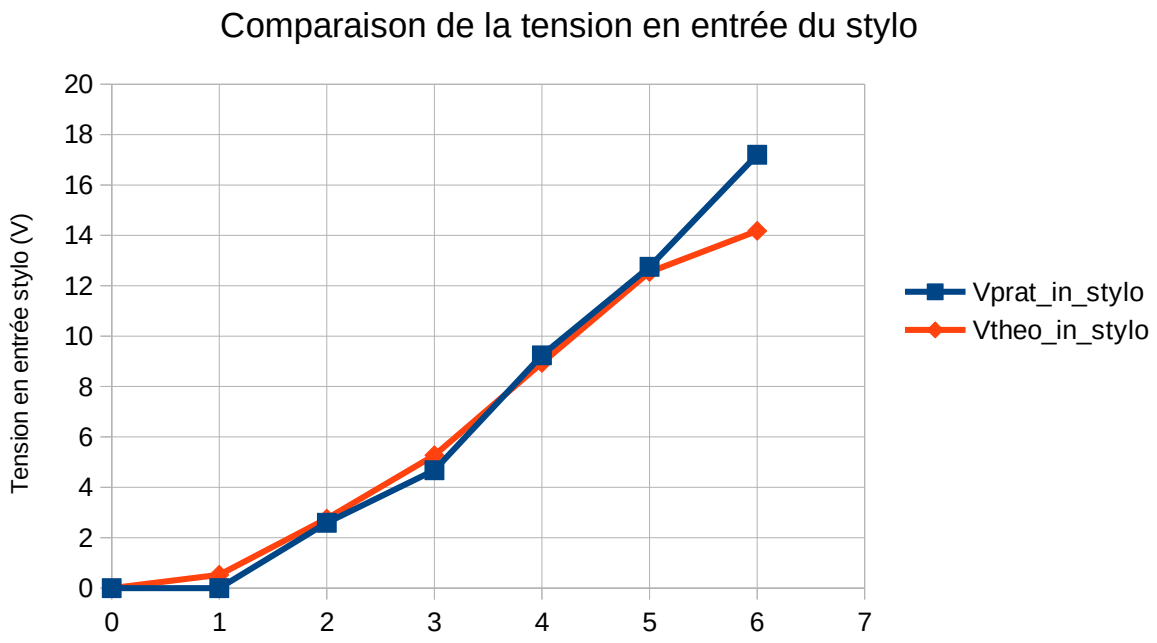


Figure XXII - Courbes Théorique Et Pratique De La Tension En Entrée Du Stylo

Nous observons une cassure au-delà de 14 V, puisque la courbe orange ne dépasse pas cette valeur, alors qu'en entrée nous avons du 17 V. Cela montre bien qu'une tension supérieure à 14 V ne peut être lu et que notre diode Zener fonctionne comme voulu.

2. Transmission

3. Protocole UART

Nous envoyons nos données via le protocole UART.

A. Rappels

Pour rappel, le protocole UART (Universal Asynchronous Receiver/Transmiter) fonctionne sans horloge, avec qu'un seul fil de données.

B. Trame

Notre trame est construite ainsi :

Start	Adresse		Poids	Bits de mesure					Parité
0	0	1	0	B4	B3	B2	B1	B0	X
0	0	1	1	B9	B8	B7	B6	B5	X

4. Réception

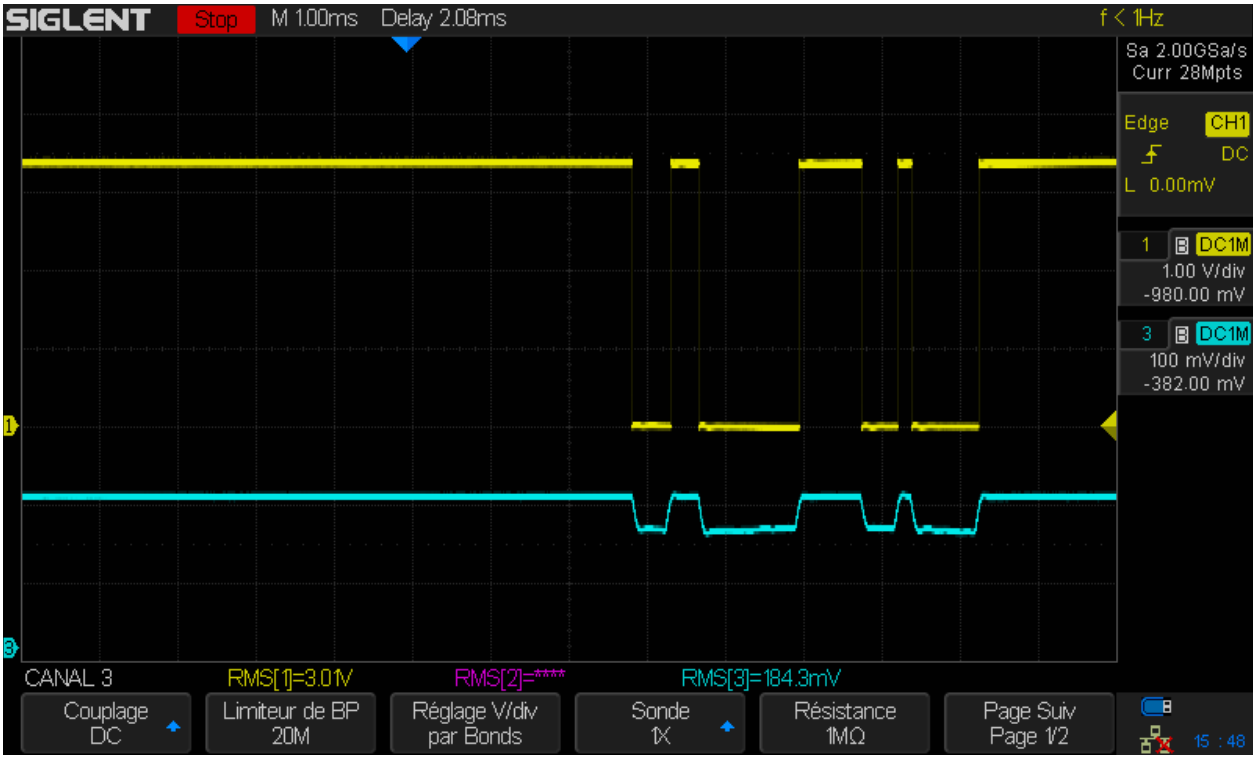


Figure XXIII - Signal Démodulé En Sortie Du RR4594

Nous observons que le module de réception RR4594 reçoit bien notre signal émis du stylo voltmètre (signal jaune) et qu'il le démodule. Le signal bleu correspond au signal de la broche RSSI, qui pour rappel est donne une intensité proportionnelle au signal reçu.

D. MAINTENIR

Dans cette partie nous allons nous intéresser aux différents cas de panne qui puissent arriver. Il est fort probable que plusieurs personnes nous renvoient leur stylo-voltmètre, qui n'est plus opératif, ou alors que nous nous rendons compte d'un défaut de fabrication.

SOUS - SOMMAIRE

1. Coûts

A. Pièces Fournis

Dans le kit du stylo-voltmètre, nous retrouvons les éléments suivants :

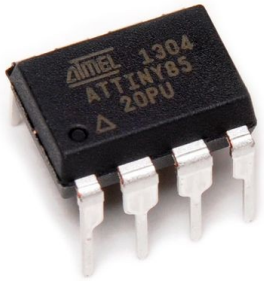
B. Composants

Ci-dessous, un tableau récapitulatif des différents composants utilisées pour la création du stylo-voltmètre :

Nomination	Fonction	Quantité	Valeur	Tension	Température	Fabricant	Prix
AtTiny85	Microprocesseur	1	–	1,8 V – 5,5 V	−40 °C – 85 °C	Atmel	1,80 €
MCP3201	CAN 12 bits	1	–	2,7 V – 5,2 V	−65 °C – 125 °C	Microchip	3,74 €
RT4254	Émetteur	1		2,1 V – 3,6 V	−40 °C – 85 °C	Linx	10,74 €
LP	Régulateur 3,3 V	1					
R1	Ja-Mau-Mar-Au	1	470 Ω			–	
R2	Or-Bl-Ja-Au (	1				–	
R3	Or-Or-Ja-Au	1				–	
R4	Ro-Ro-Ja-Au	1				–	
R5/R6	Or-Or-Mar-Au	2				–	
R7	Ro-Mau-Ro-Au	1				–	
C1	10nK63	1				–	
C2	1J63	2				–	
C3	1K63	1				–	
C4	1J63	0				–	
C5/C6	225K5C	2				–	
Diode Zener	BZX	1				–	
						–	
TOTAL	–	**1**	–	**2,7 V – 3,6 V**	**−40 °C – 85 °C**	–	**16,28 €**

2. Entretien Des Composants

A. AtTiny85



L'AtTiny85 peut se réinitialiser, quand le port reset est mis à 1. Cela pourrait mener à l'inutilité du stylo. Il est également possible qu'il casse, mais cela est très peu probable. Le fabricant nous annonce une durée de vie de 70 ans, ce qui est largement suffisant.

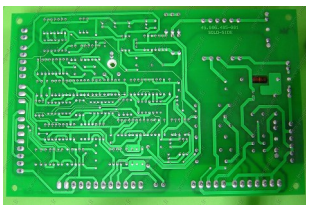
B. Batterie

La batterie peut se décharger au fil du temps. C'est alors au client de remplacer lui-même la batterie par une autre. Sachant qu'une pile de 9 V avec une capacité de 1,2 Ah et une utilisation de 30 mA, nous donne un fonctionnement de 40 heures.

C. Bouton

Le bouton a un certain nombre d'appui maximale. Au fil du temps, il peut se détériorer et le contact peut ne plus fonctionner.

D. Carte



Il est possible que la carte, à cause de problème d'usinage, s'oxyde. Alors, elle peut ne plus être utilisable et causer des pannes.

E. Condensateurs

Les condensateurs chimiques peuvent exploser à cause d'une chaleur intense ou bien d'un trop fort courant. Or, nous utilisons seulement des courants faibles et aucun condensateur chimique, que des condensateurs céramiques. Donc, il est très peu probable qu'ils lâchent.

F. DEL

Les Diodes Électros Luminescentes ont une durée de vie 100 000 heures. Elles claqueront après la fin de vie du stylo-voltmètre. Il est donc peu probable que nous ayont à les changer, sauf si elles sont défaillantes.

G. Diode Zener

Notre stylo-voltmètre est fait pour fonctionner sur une plage de 0 à 14 V. Il est possible que des personnes l'utilisent au-dessus de 15 V. Notre diode Zener sert à limiter cette tension. Mais en augmentant trop la tension, il est fort probable que notre diode grille.

H. On

I. Linx

La transmission peut poser problème, notamment avec la superposition de plusieurs signaux sur la même fréquence d'envoi (4 kHz).

J. Régulateur De Tension

K. Résistors

L. Switch

M. Récapitulatif

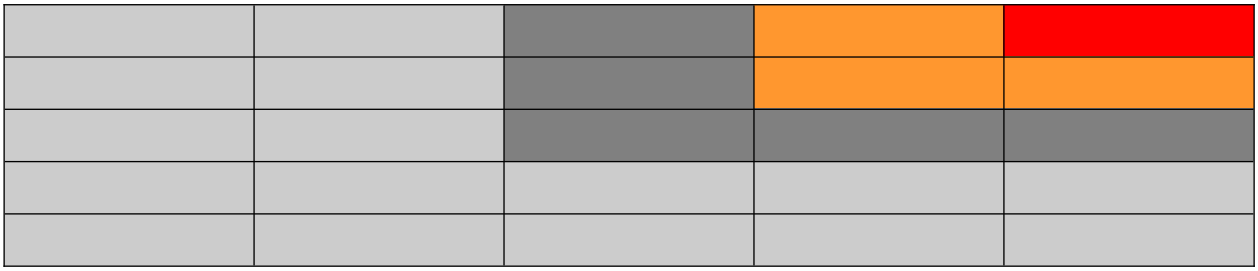
Nous pouvons alors mettre les composants dans un tableau, le taux de casse en fonction de leur durée de vie.

3. Entretien Régulier

4. Maintenance Préventive

A. Boîte À Neuf États

f(Taux d'occurrence) = taux de gravité



N°	Description
1	
2	
3	

B. Field Failure Rate (FFT)

La réception s’effectue à l’aide d’une carte STM32.

5. Service De Maintenance

6. Notice Du Produit

7. Simulation De Dépannage

TABLE DES FIGURES