Ressources pour les étudiants

Brevet de technicien supérieur

« Cybersécurité, Informatique et réseaux, Électronique »

C04	ANALYSER UNE STRUCTURE MATÉRIELLE ET LOGICIELLE

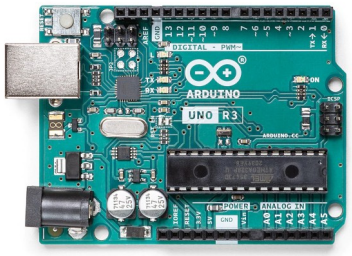
Principales activités mettant en œuvre la compétence :

E1 – Étude et conception de produits électroniques
E2 – Tests et essais
E4 – Intégration matérielle et logicielle
E5 – Maintenance et réparation de produits électroniques
R2 – Installation et qualification

Connaissances associées (et niveaux taxonomiques)

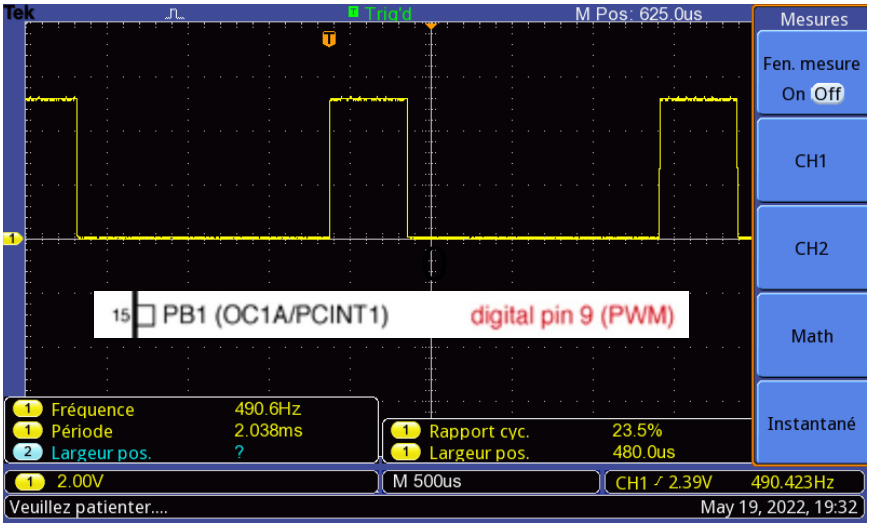
– SysML (Exigences, Cas d'utilisation, Séquence)	*Niveau 3*
– Circuits : microcontrôleurs, mémoires, circuits numériques spécifiques	*Niveau 3*
– Chaîne de traitement du signal	*Niveau 3*
– Concepts fondamentaux des transmissions	*Niveau 2*
– Réseaux locaux industriels et bus de carte : Ethernet, CAN, I²C, SPI, RS485, RS232	*Niveau 3*
– Gestion de l'énergie et alimentation	*Niveau 2*
– Programmation en langage C	*Niveau 3*
– Composants d'électronique analogique (R, L, C, Quartz, Diodes, Transistors bipolaires et MOS, ALI) limitées à la symbolique, au rôle et à la technologie du composant	*Niveau 2*
• Étude de l'environnement et modélisation d'un système électronique	*Niveau 2*
• Fonctions, structures et caractéristiques d'une chaîne de mesure dans un système électronique	*Niveau 3*
• Supports de propagation dans un système électronique ou une carte électronique	*Niveau 3*
• Principes et techniques de transmission en espace libre ou guidée dans les réseaux et les systèmes électroniques	*Niveau 3*
• Fonctions, structures et caractéristiques d'une chaîne d'action dans un système électronique	*Niveau 3*
• Réglages, effets et caractéristiques d'un système en boucle fermée	*Niveau 3*
• Lois générales de l'électricité	*Niveau 3*
• Analyse et caractérisation temporelle et fréquentielle des signaux	*Niveau 3*
• Fonction filtrage, structures de filtres analogiques et numériques	*Niveau 3*

Le PWM
Pulse Width Modulation



```
void setup() {

  pinMode(9, OUTPUT);    // Fréquence PWM: 490 Hz
}

void loop() {

  analogWrite(9, 60);
  delay(500);
  analogWrite(9, 120);
  delay(500);
  analogWrite(9, 200);
  delay(500);

}
```

(Voir documentation)



Calcul du rapport cyclique

AnalogWrite(9, **60**) ;

$$\frac{60}{255}\times 100 = 23{,}5\,\%$$

Résolution 8 bits

Calcul de la largeur d'impulsion

$$\frac{60}{255}\times\frac{1}{490}\times 10^{6} = 480{,}2\,\mu s$$

Durée en µs

Fréquence PWM (490 Hz)

Utilisation des registres

Paramétrage du générateur PWM

```
void setup() {

  pinMode(9, OUTPUT);

// ---------------------------- Initialisation PWM -----------------------------
  // Fast PWM : Freq = 16 MHz / N*(ICR1 + 1)
  // No prescaling -> N = 1
  // ICR1 = 3199 pour une fréquence 5 KHz

  // Set OC1A/OC1B on Compare Match, clear OC1A/OC1B at BOTTOM (inverting mode)
  TCCR1A = (1 << COM1A1) | (1 << COM1B1) | (1<<WGM11);

  // Fast PWM TOP(ICR1) -  No prescaling
  TCCR1B = (1 << WGM13) | (1 << WGM12) | (1<<CS10);

  ICR1 = 3199;       // La fréquence (Voir formule)
  OCR1A = 1600;      // Le rapport cyclique
// -----------------------------------------------------------------------------

}

void loop() {

  OCR1A = 1600;     // Le rapport cyclique
  delay(1000);
  OCR1A = 800;      // Le rapport cyclique
  delay(1000);
  OCR1A = 400;      // Le rapport cyclique
  delay(1000);

}
```

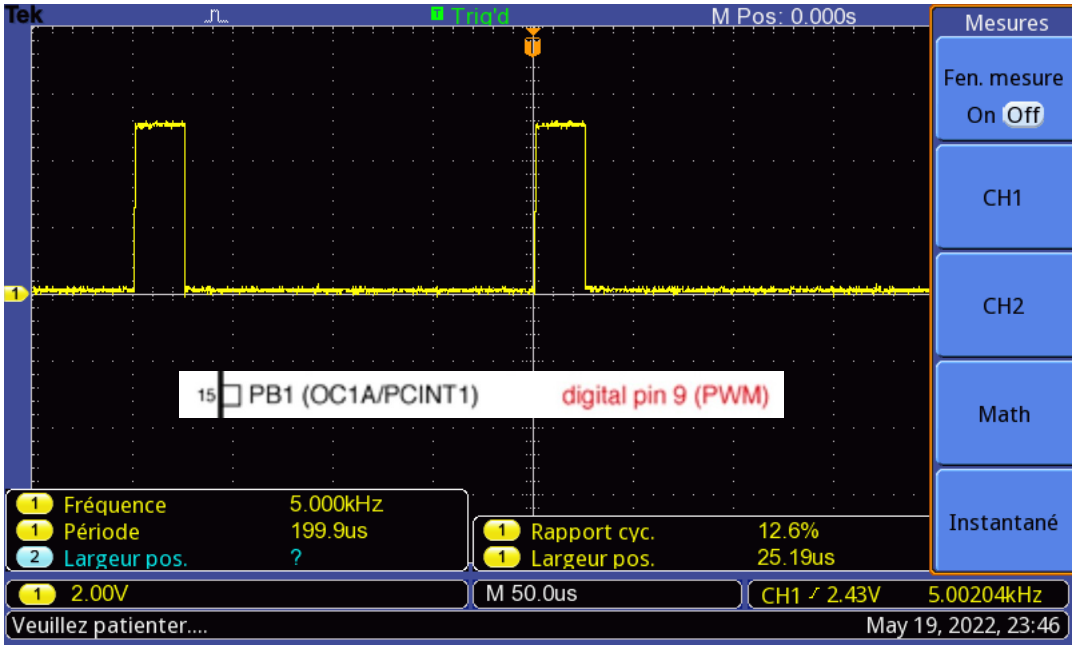
Voir documentation constructeur
DS40002061A - page 133

Waveform Generation Mode 14

Table 16-4. Waveform Generation Mode Bit Description[1] (Continued)

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Note: 1. The CTC1 and PWM11:0 bit definition names are obsolete. Use the WGM12:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.



$$\frac{400}{3199}\times 100 = 12{,}5\,\%$$

$$\frac{400}{3199}\times\frac{1}{5\,KHz}\times 10^{3} = 25\,\mu s$$

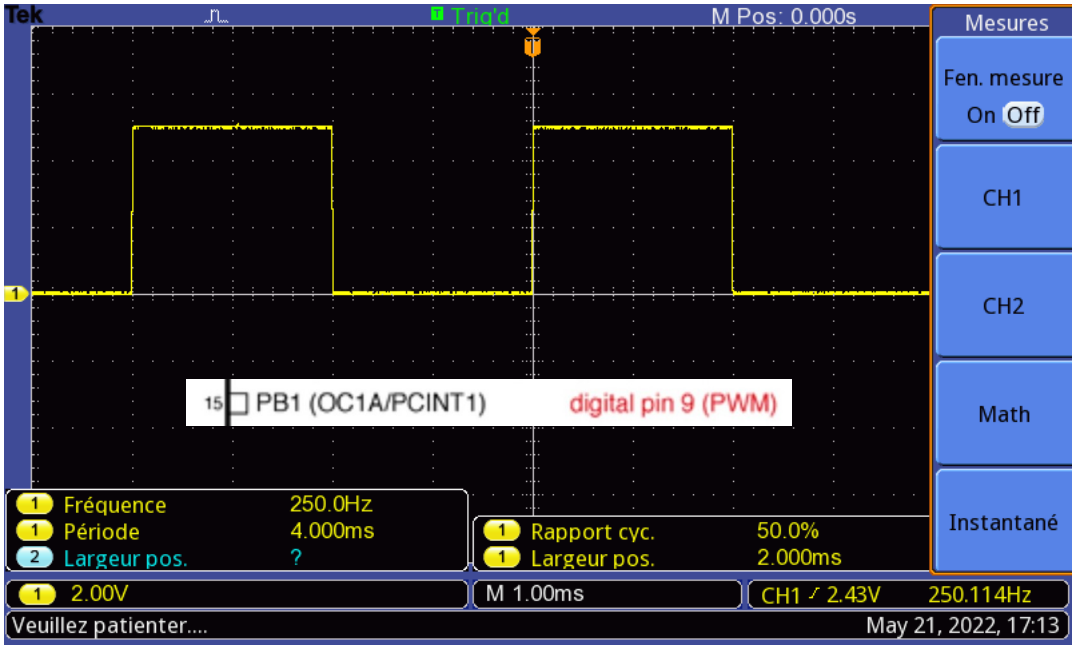
Application : Générateur de signal carré

Paramétrage du générateur

```
// Générateur de signal carré de fréquence (Freq) et de rapport cyclique de 50%
// Version 1.0
unsigned int Val;
void Freq(unsigned int Freq_Hz){

  TCCR1A = (1 << COM1A1) | (1 << COM1B1) | (1<<WGM11);
  TCCR1B = (1 << WGM13) | (1 << WGM12) | (1<<CS10);

  Val = (16000000 / Freq_Hz)-1;
  ICR1 = Val;
  OCR1A = Val >> 1;      // Le rapport cyclique de 50%
}
```

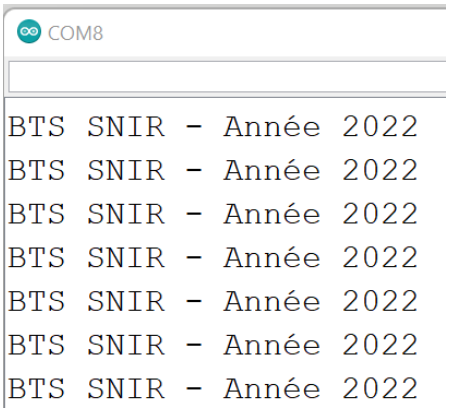
Val >> 1 équivaut à (Val / 2)

```
void setup() {
  pinMode(9, OUTPUT);
}

void loop() {

  Freq(12500);          // 12.5 KHz
  delay(10000);
  Freq(250);            // 250 Hz
  delay(10000);

}
```



USART et transfert de bloc mémoire

```
String Text = "Année 2022";

void setup() {
  Serial.begin(9600); // Baud rate 9600
}

void loop() {

  // Messages à transmettre
  Serial.print("BTS SNIR");
  Serial.print(" - ");
  Serial.println(Text);

  delay(1000);
}
```

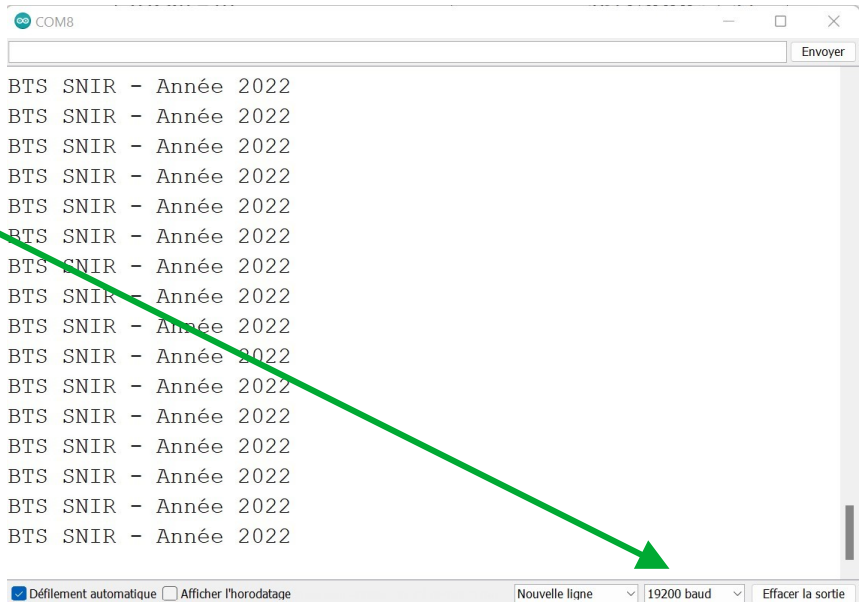


1.0) Utilisation des registres : Initialisation

```
String Text = "Année 2022";

void Init_USART0(unsigned int Baud){

    UBRR0 = (1000000/Baud) - 1;          // Voir formule page 182
    // Reception et Transmission actives
    UCSR0B = (1<<RXEN0) | (1<<TXEN0);
    // Format: 8 bit et 1 bit de Stop
    UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);
}
```

Voir document constructeur

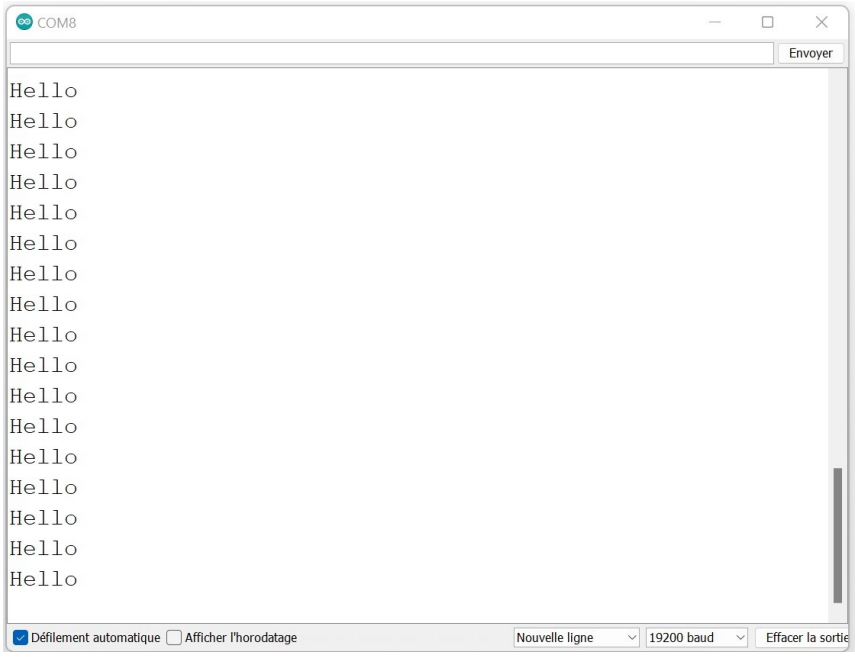
```
void setup() {
  Init_USART0(19200);
}

void loop() {

  // Messages à transmettre
  Serial.print("BTS SNIR");
  Serial.print(" - ");
  Serial.println(Text);

  delay(1000);
}
```



1.1) Utilisation des registres : Transmission d'un caractère

```
String Text = "Année 2022";

void Init_USART0(unsigned int Baud){

    UBRR0 = (1000000/Baud) - 1;
    // Reception et Transmission actives
    UCSR0B = (1<<RXEN0) | (1<<TXEN0);
    // Format: 8 bit et 1 bit de Stop
    UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);
}

void USART0_Char(unsigned char Tx){

    // Transfert n-1 terminé et aucune réception ?
    while( !(UCSR0A & (1<<UDRE0)) ){}
    UDR0 = Tx;           // Transfert du caractere
}

void setup() {
  Init_USART0(19200);
}

void loop() {

  USART0_Char('H');
  USART0_Char('e');
  USART0_Char('l');
  USART0_Char('l');
  USART0_Char('o');
  USART0_Char(0x0D);
  USART0_Char(0x0A);

  delay(1000);
}
```

Voir document constructeur



Voir document constructeur page 200

20.11.2 UCSRnA – USART Control and Status Register n A

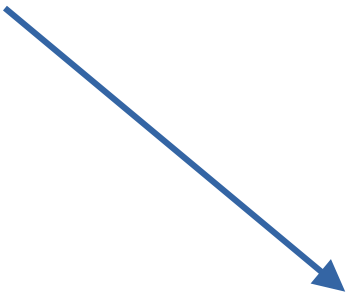
Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- **Bit 5 – UDREn: USART Data Register Empty**

The UDREn Flag indicates if the transmit buffer (UDRn) is ready to receive new data. If UDREn is one, the buffer is empty, and therefore ready to be written. The UDREn Flag can generate a Data Register Empty interrupt (see description of the UDRIEn bit). UDREn is set after a reset to indicate that the Transmitter is ready.

1.2) Utilisation des registres : Transmission d'une chaîne de caractères

```
void Init_USART0(unsigned int Baud){

    UBRR0 = (1000000/Baud) - 1;
    // Reception et Transmission actives
    UCSR0B = (1<<RXEN0) | (1<<TXEN0);
    // Format: 8 bit et 1 bit de Stop
    UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);
}

void USART0_Text(String Text){

    unsigned int n;
    n = 0;

    // Fin de la chaine de caracteres?
    while( Text[n] != 0x00){
        // Transfert n-1 terminé et aucune réception ?
        while( !(UCSR0A & (1<<UDRE0)) ){}
        UDR0 = Text[n];                    // Transfert du caractere
        n++;
    }

    // Retour chariot et début de ligne
    while( !(UCSR0A & (1<<UDRE0)) ){}
    UDR0 = 0x0D;
    while( !(UCSR0A & (1<<UDRE0)) ){}
    UDR0 = 0x0A;
}

void setup() {
  Init_USART0(19200);
}

void loop() {

  USART0_Text("BTS SNIR - Année 2022 - Hello");
  delay(1000);
}
```



```
COM8
BTS SNIR - Année 2022 - Hello
BTS SNIR - Année 2022 - Hello
BTS SNIR - Année 2022 - Hello
BTS SNIR - Année 2022 - Hello
BTS SNIR - Année 2022 - Hello
BTS SNIR - Année 2022 - Hello
BTS SNIR - Année 2022 - Hello
BTS SNIR - Année 2022 - Hello
BTS SNIR - Année 2022 - Hello
BTS SNIR - Année 2022 - Hello
BTS SNIR - Année 2022 - Hello
BTS SNIR - Année 2022 - Hello
BTS SNIR - Année 2022 - Hello
BTS SNIR - Année 2022 - Hello
BTS SNIR - Année 2022 - Hello
BTS SNIR - Année 2022 - Hello
BTS SNIR - Année 2022 - Hello
```

1.3) Utilisation des registres : Réception d'un caractère

```
unsigned int n;
// ------------------------------------------------------
void Init_USART0(unsigned int Baud){
    UBRR0 = (1000000/Baud) - 1;
    UCSR0B = (1<<RXEN0) | (1<<TXEN0);
    UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);
}
// ------------------------------------------------------
void USART0_Char(unsigned char Tx){
    while( !(UCSR0A & (1<<UDRE0)) ){}
    UDR0 = Tx;           // Transfert du caractere
}
// ------------------------------------------------------
void USART0_Text(String Text){
    n = 0;
    while( Text[n] != 0x00){
        // Transfert n-1 terminé et aucune réception ?
        while( !(UCSR0A & (1<<UDRE0)) ){}
        UDR0 = Text[n];
        n++;
    }
    while( !(UCSR0A & (1<<UDRE0)) ){}
    UDR0 = 0x0D;
    while( !(UCSR0A & (1<<UDRE0)) ){}
    UDR0 = 0x0A;
}
// ------------------------------------------------------
unsigned char USART0_Rx(void){
    // Attendre reception d'un caractere
    while ( !(UCSR0A & (1<<RXC0)) ){}
    // renvoie le caractere recu
    return UDR0;
}
// ------------------------------------------------------
void setup() {
  Init_USART0(9600);
}

void loop() {

  while( USART0_Rx() != '?'){}
  USART0_Text("Année 2022");
  delay(1000);
}
```

Voir document constructeur page 189

Attendre la réception du caractère (?)

1.4) Utilisation des registres : Réception d'une chaîne de caractères

```
unsigned int n;
unsigned char Rx_USART0[256];
// ---------------------------------------------------------
void Init_USART0(unsigned int Baud){
    UBRR0 = (1000000/Baud) - 1;
    UCSR0B = (1<<RXEN0) | (1<<TXEN0);
    UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);
}
// ---------------------------------------------------------
void USART0_Char(unsigned char Tx){
    while( !(UCSR0A & (1<<UDRE0)) ){}
    UDR0 = Tx;           // Transfert du caractere
}
// ---------------------------------------------------------
void USART0_Text(String Text){
    n = 0;
    while( Text[n] != 0x00){
        // Transfert n-1 terminé et aucune réception ?
        while( !(UCSR0A & (1<<UDRE0)) ){}
        UDR0 = Text[n];
        n++;
    }
    while( !(UCSR0A & (1<<UDRE0)) ){}
    UDR0 = 0x0D;
    while( !(UCSR0A & (1<<UDRE0)) ){}
    UDR0 = 0x0A;
}
// ---------------------------------------------------------
void USART0_String_Rx(void){
    n = 0;
    do{
      while ( !(UCSR0A & (1<<RXC0))){}
      Rx_USART0[n] = UDR0;
      n++;
    }while(Rx_USART0[n-1] != 0x0A);
}
// ---------------------------------------------------------
void setup() {
  Init_USART0(9600);
}

void loop() {
  USART0_String_Rx();
  USART0_Text(Rx_USART0);
  delay(1000);
}
```

Réception de la chaîne de caractères dans Rx_USART0[ ]

Attendre la réception du caractère 0x0A

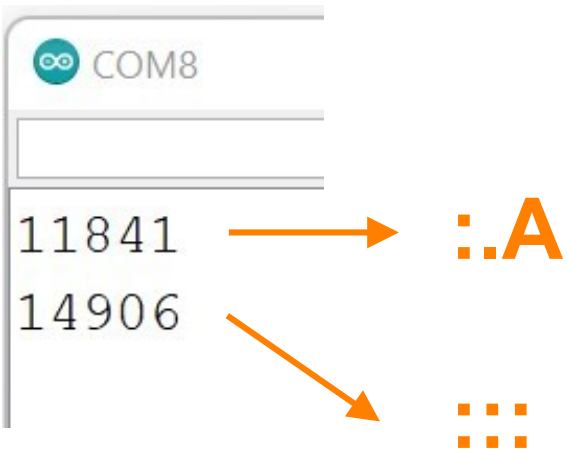
Première chaîne de caractères : Année 2022

Seconde chaîne de caractères : Hello

1.5) Utilisation des registres : Transfert de donnée (INT16)

```
int Rx_INT;

// ------------------------------------------------------
void Init_USART0(unsigned int Baud){
    UBRR0 = (1000000/Baud) - 1;
    UCSR0B = (1<<RXEN0) | (1<<TXEN0);
    UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);
}
// ------------------------------------------------------
void USART0_INT_Rx(void){

    // Attendre le caractere de depart (:)
    do{
      while ( !(UCSR0A & (1<<RXC0))){}
    }while(UDR0 != ':');

    // Reception de l'INT16
    while ( !(UCSR0A & (1<<RXC0))){}
    *((char*)(&Rx_INT) + 1) = UDR0;
    while ( !(UCSR0A & (1<<RXC0))){}
    *(char*)(&Rx_INT) = UDR0;

}
// ------------------------------------------------------
void setup() {
  Init_USART0(9600);
}

void loop() {
  USART0_INT_Rx();
  Serial.print(Rx_INT);
  Serial.print("\n\r");
  delay(1000);
}
```

Réception de l'INT16 (2 octets)

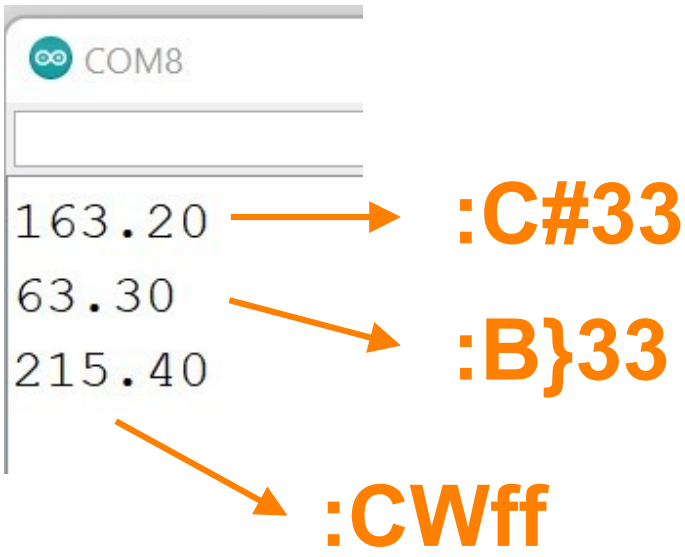


1.6) Utilisation des registres : Transfert de donnée (float)

```
float Rx_Float;

// ------------------------------------------------------
void Init_USART0(unsigned int Baud){
    UBRR0 = (1000000/Baud) - 1;
    UCSR0B = (1<<RXEN0) | (1<<TXEN0);
    UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);
}
// ------------------------------------------------------
void USART0_Float_Rx(void){

    do{
      while ( !(UCSR0A & (1<<RXC0))){}
    }while(UDR0 != ':');

    // Reception du float
    while ( !(UCSR0A & (1<<RXC0))){}
    *((char*)(&Rx_Float) + 3) = UDR0;
    while ( !(UCSR0A & (1<<RXC0))){}
    *((char*)(&Rx_Float) + 2) = UDR0;
    while ( !(UCSR0A & (1<<RXC0))){}
    *((char*)(&Rx_Float) + 1) = UDR0;
    while ( !(UCSR0A & (1<<RXC0))){}
    *((char*)(&Rx_Float)) = UDR0;
}
// ------------------------------------------------------
void setup() {
  Init_USART0(9600);
}

void loop() {
  USART0_Float_Rx();
  Serial.print(Rx_Float);
  Serial.print("\n\r");

  delay(1000);
}
```

Attendre le caractère de départ (:)

Réception du float (4 octets)



1.7) Interfaçage avec Python (Le programme Arduino)

```
// Liaison avec Python
unsigned int n;
// ---------------------------------------------------------
void Init_USART0(unsigned int Baud){
    UBRR0 = (1000000/Baud) - 1;
    UCSR0B = (1<<RXEN0) | (1<<TXEN0);
    UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);
}
// ---------------------------------------------------------
void USART0_Text(String Text){
    n = 0;
    while( Text[n] != 0x00){
        // Transfert n-1 terminé et aucune réception ?
        while( !(UCSR0A & (1<<UDRE0)) ){}
        UDR0 = Text[n];
        n++;
    }
    while( !(UCSR0A & (1<<UDRE0)) ){}
    UDR0 = '\r';
    while( !(UCSR0A & (1<<UDRE0)) ){}
    UDR0 = '\n';
}
// ---------------------------------------------------------
unsigned char USART0_Rx(void){
    // Attendre reception d'un caractere
    while ( !(UCSR0A & (1<<RXC0)) ){}
    // renvoie le caractere recu
    return UDR0;
}
// ---------------------------------------------------------
void setup() {
  Init_USART0(9600);
}

void loop() {

  if(USART0_Rx() == '?'){USART0_Text("Connexion avec le programme Python");}
  if(USART0_Rx() == '>'){USART0_Text("Technologie");}
  delay(200);

}
```

? envoyé par Python

Réception et affichage

1.8) Interfaçage avec Python (Le script Python)


```
import serial
import time

ser = serial.Serial( port='COM8', baudrate=9600, timeout=0 )

def Com_Verif():
    if ser.isOpen():
        time.sleep(2)
        return(True)


def Code_Carte(Code):
    ser.flushOutput()
    ser.flushInput()

    ser.write(Code)
    time.sleep(0.1)
    Temp = ser.readline()
    return(Temp)


# Connexion avec la carte Arduino UNO
if Com_Verif():
    print(Code_Carte(b'?'))
    print(Code_Carte(b'>'))
    ser.close()
```

Voir explication page suivante

Script permettant la lecture d'une trame complète readline() → /n

Après exécution du script

```
*** Console de processus distant Réinitialisée ***
b'Connexion avec le programme Python\r\n'
b'Technologie\r\n'
>>>
```

Rappels du programme Arduino

```
if(USART0_Rx() == '?'){USART0_Text("Connexion avec le programme Python");}
if(USART0_Rx() == '>'){USART0_Text("Technologie");}
```

```
while( !(UCSR0A & (1<<UDRE0)) ){}
UDR0 = '\r';
while( !(UCSR0A & (1<<UDRE0)) ){}
UDR0 = '\n';
```

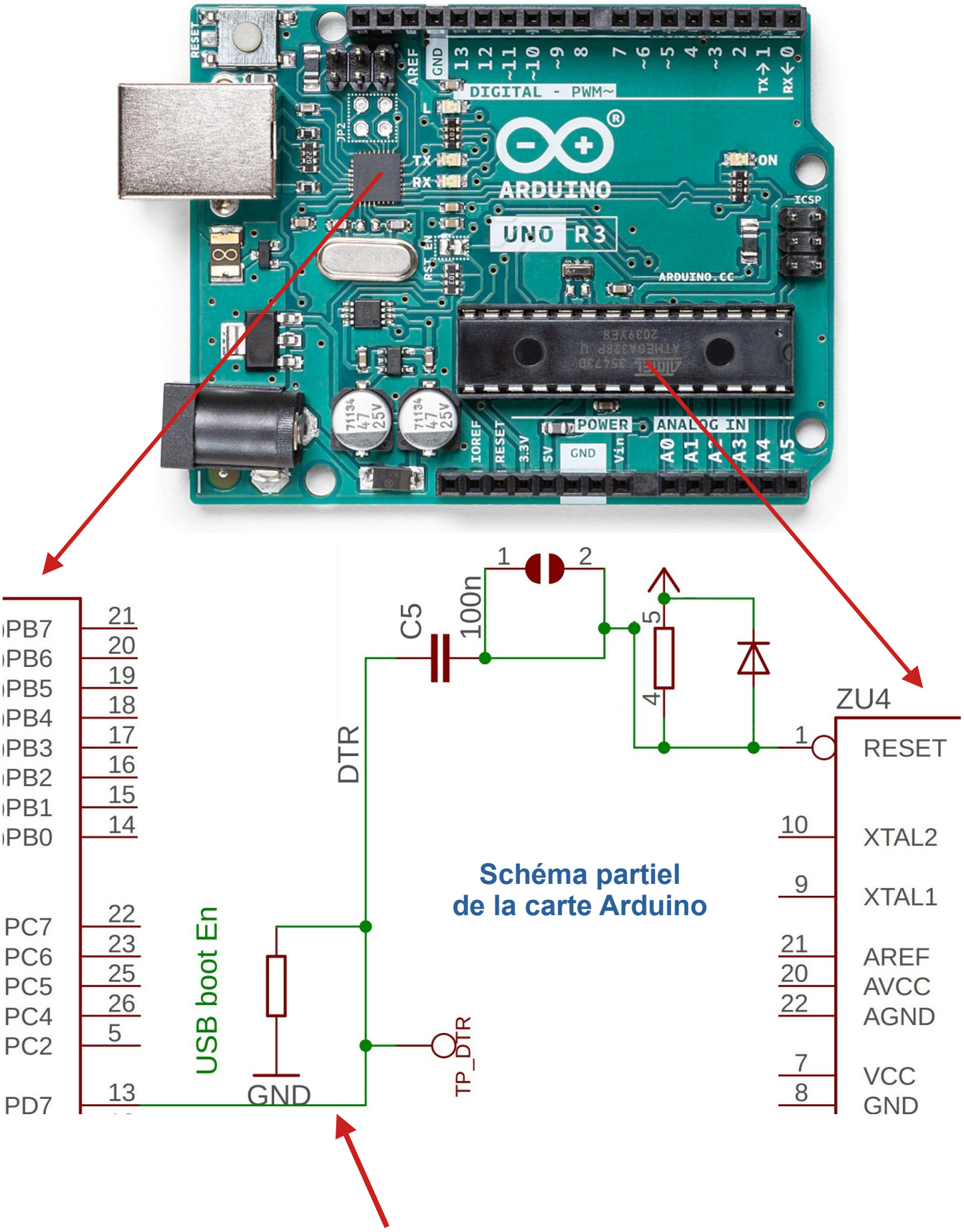


Schéma partiel de la carte Arduino

A l'ouverture du port de communication (Python)
un ***Reset de la carte Arduino est créé***

1.9) Interfaçage avec Python (Le programme Arduino UNO) (Transfert de données → capteur par exemple)

INT_Transfert | Mes_programmes

```
// Mes programmes

void Init_USART0(unsigned int Baud){
    UBRR0 = (1000000/Baud) - 1;
    UCSR0B = (1<<RXEN0) | (1<<TXEN0);
    UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);
}
// ----------------------------------------------------------

unsigned char USART0_Rx(void){
    // Attendre reception d'un caractere
    while ( !(UCSR0A & (1<<RXC0)) ){}
    // renvoie le caractere recu
    return UDR0;
}
// ----------------------------------------------------------

void USART0_Char(unsigned char Tx){
    while( !(UCSR0A & (1<<UDRE0)) ){}
    UDR0 = Tx;
}
// ----------------------------------------------------------

void USART0_INT_Tx(unsigned int INT16){
    while( !(UCSR0A & (1<<UDRE0)) ){}
    UDR0 = *((char*)(&INT16) + 1);
    while( !(UCSR0A & (1<<UDRE0)) ){}
    UDR0 = *(char*)(&INT16);
}
// ----------------------------------------------------------

unsigned int nbr;
void Transfert_Data(void){

  USART0_Char(':');
  USART0_INT_Tx(Buf_INT[0]);
  for(nbr = 0; nbr < Buf_INT[0]; nbr++){
    USART0_INT_Tx(Buf_INT[nbr+1]);
  }
}
```

Transmission de l'INT (2 octets)

Format de transmission

:(Nombre de données)(Les données)

INT_Transfert | Mes_programmes

```
unsigned int Buf_INT[256];

void setup() {
  Init_USART0(9600);
  // Simulation d'acquisition de données
  Buf_INT[0] = 10;      // 1ere donnée -> le nombre de donnée
  Buf_INT[1] = 238; Buf_INT[2] = 190; Buf_INT[3] = 186; Buf_INT[4] = 172; Buf_INT[5] = 168;
  Buf_INT[6] = 10; Buf_INT[7] = 13; Buf_INT[8] = 27; Buf_INT[9] = 35; Buf_INT[10] = 1250;
  // Note 0x0A (\n) = 10 et 0x0D (\r) = 13
}

void loop() {
  if(USART0_Rx() == 'B'){ Transfert_Data(); }
}
```

1.9.1) Interfaçage avec Python (Le script Python) (Transfert de données → capteur par exemple)


```
import serial
import time

ser = serial.Serial( port='COM8', baudrate=9600, timeout=0 )

def Com_Verif():
    if ser.isOpen():
        time.sleep(2)
        return(True)


def Code_Carte(Code):
    ser.flushOutput()
    ser.flushInput()

    ser.write(Code)
    time.sleep(0.1)
    Temp = ser.read_all()

    # -- Les étapes --
    print(Temp)

    Temp1 = Temp[1:]
    print(Temp1)

    List_INT16 = [int.from_bytes(Temp1[2*n:2*n+2], byteorder='big') for n in range( (len(Temp1))//2 )]
    print(List_INT16)

    List_Str = list(map(str, List_INT16))
    print(List_Str)

    file = open( 'Save_Data.txt', 'w' )
    for i in range( len(List_Str)-1 ):
        file.write( List_Str[i] + ',' )
    file.write( List_Str[len(List_Str)-1] )
    file.close()

    return(List_Str[0] + " données")


# Connexion avec la carte Arduino UNO
# Transfert des données vers le script Python
if Com_Verif():
    print(Code_Carte(b'B'))
    ser.close()
```

Voir document Python version 3.10.4

classmethod `int.`**`from_bytes`**`(`*bytes*, *byteorder*, *, *signed=False*`)`

Renvoie le nombre entier représenté par le tableau d'octets fourni.

```
>>> int.from_bytes(b'\x00\x10', byteorder='big')
16
>>> int.from_bytes(b'\x00\x10', byteorder='little')
4096
>>> int.from_bytes(b'\xfc\x00', byteorder='big', signed=True)
-1024
>>> int.from_bytes(b'\xfc\x00', byteorder='big', signed=False)
64512
>>> int.from_bytes([255, 0, 0], byteorder='big')
16711680
```

Les données reçues

Enlève le premier symbole (:)

Transforme les INT en String

Sauvegarde des données dans un fichier

Retourne le nombre de données reçues

Après exécution du script

```
*** Console de processus distant Réinitialisée ***
b':\x00\n\x00\xee\x00\xbe\x00\xba\x00\xac\x00\xa8\x00\n\x00\r\x00\x1b\x00#\x04\xe2'
b'\x00\n\x00\xee\x00\xbe\x00\xba\x00\xac\x00\xa8\x00\n\x00\r\x00\x1b\x00#\x04\xe2'
[10, 238, 190, 186, 172, 168, 10, 13, 27, 35, 1250]
['10', '238', '190', '186', '172', '168', '10', '13', '27', '35', '1250']
10 données
>>>
```

2) Interfaçage avec Python (Commande de la carte Arduino) (Générateur de signal carré commandé avec Python)

Signal_Carre_python | USART0

```
void Init_USART0(unsigned int Baud){
    UBRR0 = (1000000/Baud) - 1;
    UCSR0B = (1<<RXEN0) | (1<<TXEN0);
    UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);
}

void USART0_Tx(unsigned char Tx){

    // Transfert n-1 terminé et aucune réception ?
    while( !(UCSR0A & (1<<UDRE0)) ){}
    UDR0 = Tx;          // Transfert du caractere
}

unsigned char USART0_Rx(void){
    // Attendre reception d'un caractere
    while ( !(UCSR0A & (1<<RXC0)) ){}
    // renvoie le caractere recu
    return UDR0;
}

unsigned int Signal_Freq;
void Signal_Carre_Freq(void){

    // Reception de la fréquence
    while ( !(UCSR0A & (1<<RXC0))){}
    *((char*)(&Signal_Freq) + 1) = UDR0;
    while ( !(UCSR0A & (1<<RXC0))){}
    *(char*)(&Signal_Freq) = UDR0;

    // Programme du générateur de signal carré (duty cycle 50%)
    Carre_Freq(Signal_Freq);
}
```

Suite du programme Arduino

Signal_Carre_python | USART0

```
unsigned int Val;
void Carre_Freq(unsigned int Freq_Hz){

  TCCR1A = (1 << COM1A1) | (1 << COM1B1) | (1<<WGM11);
  TCCR1B = (1 << WGM13) | (1 << WGM12) | (1<<CS10);
  Val = (16000000 / Freq_Hz)-1;
  ICR1 = Val;
  OCR1A = Val >> 1;

  USART0_Tx('A');
  USART0_Tx('C');
  USART0_Tx('K');
  USART0_Tx('\n');
}

//-------------------------------------------------------
void setup() {
  Init_USART0(9600);
  pinMode(9, OUTPUT);
  Carre_Freq(1000);
}

void loop() {
  if(USART0_Rx() == ':'){ Signal_Carre_Freq(); }
  delay(200);
}
```

Acknowledge (Arduino signifie la réception de la donnée)

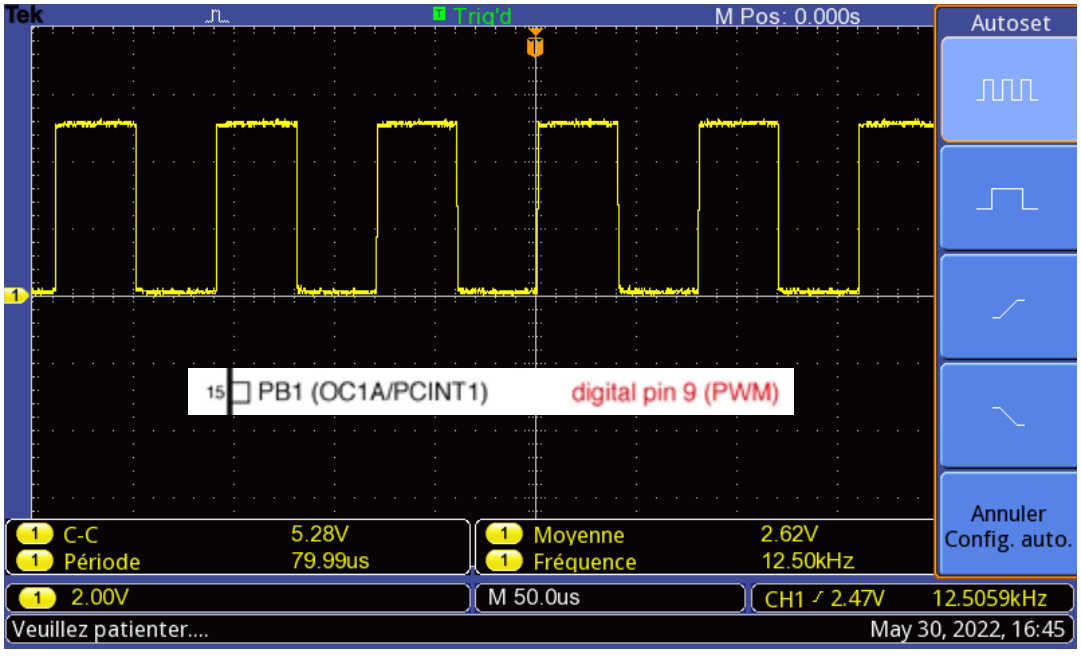
Python : readline() → /n

2.1) Interfaçage avec Python (Commande de la carte Arduino)

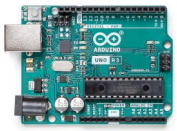
```python
import serial
import time

ser = serial.Serial( port='COM8', baudrate=9600, timeout=0 )

def Com_Verif():
    if ser.isOpen():
        time.sleep(2)
        return(True)


def Code_Carte(Code, Data_Tx):
    ser.flushOutput()
    ser.flushInput()

    ser.write(Code)
    ser.write( Data_Tx.to_bytes(2, byteorder='big') )
    time.sleep(0.1)
    return( ser.readline() )


# Connexion avec la carte Arduino UNO
if Com_Verif():
    print( Code_Carte( b':', 12500) )
    ser.close()
```

Après exécution du script

```
*** Console de processus distant Réinitialisée ***
b'ACK\n'
>>>
```



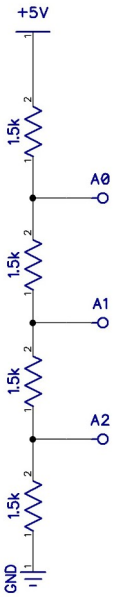
Le Convertisseur Analogique - Numérique

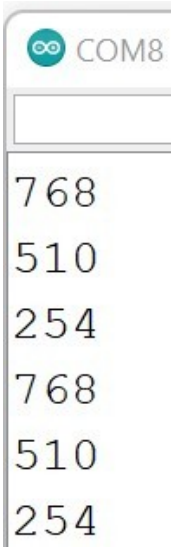


3) Le programme Arduino

```
void Init_CAN(char Factor){

  ADMUX = 0b01000000;              // Vref = +5V, ADLAR = 0, MUX = A0
  ADCSRA = 0b00000000 | Factor;
}
// ----------------------------

// Mode Single Conversion
unsigned int Start_MUX_CAN(char MUX){

  ADMUX = 0b01000000 | MUX;        // Selection de l'entrée (page 258)
  ADCSRA |= 0b11000000;            // ADC enable, Start conversion

  // Attendre conversion complete (ADIF) et update registres
  while( (ADCSRA & 0b00010000) == 0x00){}
  ADCSRA &= 0b00000111;            // ADC disable, Factor
  return(ADCL | (ADCH << 8));
}
// ----------------------------

void setup() {

  Init_CAN(0x05);          // Divisor Factor 32 (voir page 259)
  Serial.begin(9600);
}

void loop() {

  Serial.println( Start_MUX_CAN(0x00));
  delay(1500);
  Serial.println( Start_MUX_CAN(0x01));
  delay(1500);
  Serial.println( Start_MUX_CAN(0x02));
  delay(1500);
}
```

10 bits de résolution
$2^{10} = 1024$

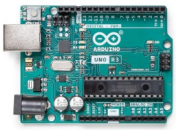




When ADCL is read, the ADC Data Register is not updated until ADCH is read. Consequently, if the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH.

Les Interruptions

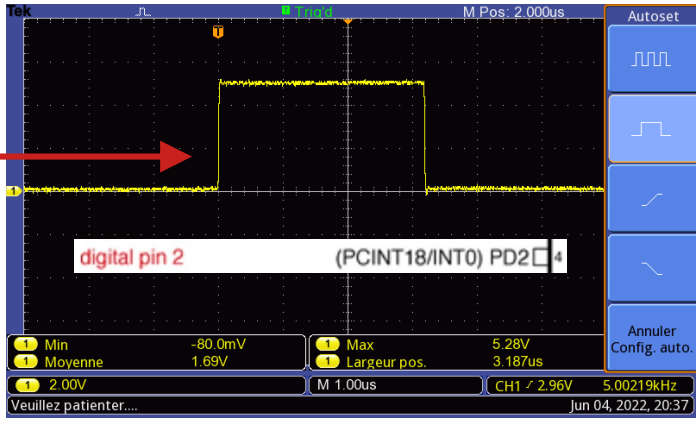
Timer0 - Interruption



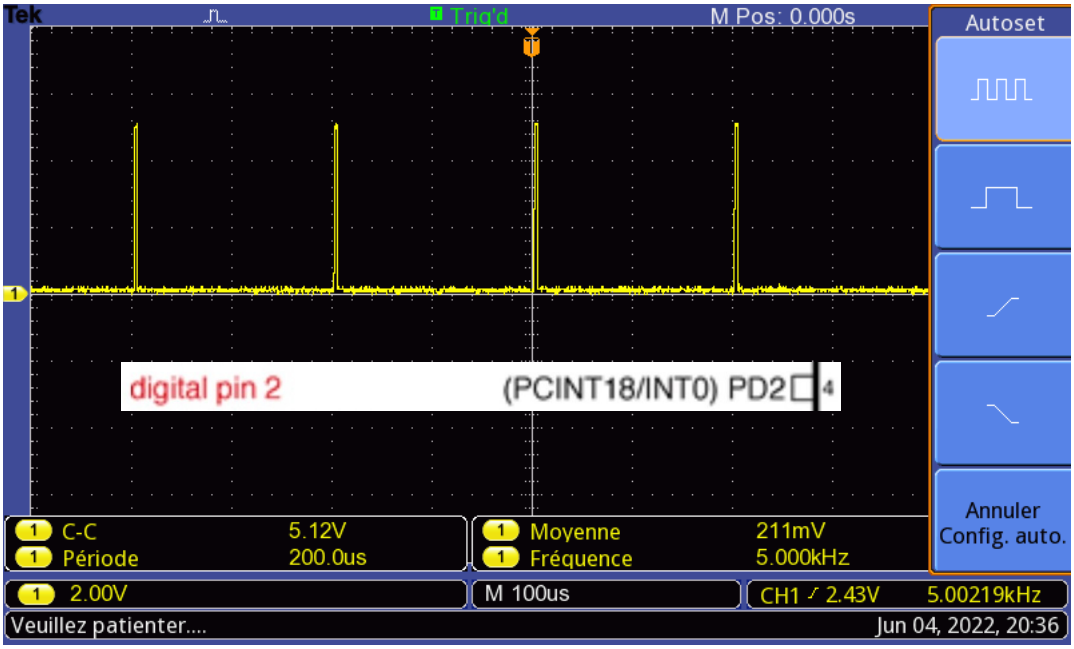
```
#define Test 2
```

digital pin 2 (PCINT18/INT0) PD2 4

```
void Init_Timer0(void){

  TCNT0 = 0x00;
  TCCR0A = 0x00;          // Normal port operation,
                          // OC0A disconnected.
  TCCR0B = 0x03;          // Clock Select Bit /64

  OCR0A = 49;             // OCRA = 49 pour T_ms -> 0.2ms
                          // OCRA = 250*T_ms - 1

  TIMSK0 = 0b00000010;    // Compare Match A,
                          // Interrupt Enable.
}
// ----------------------------------------
ISR(TIMER0_COMPA_vect){

  TCNT0 = 0x00;
  digitalWrite(Test, HIGH);
  digitalWrite(Test, LOW);
}
// ----------------------------------------
void setup() {
  cli();
  pinMode(Test, OUTPUT);
  digitalWrite(Test, LOW);
  Init_Timer0();
  sei();
}
// ----------------------------------------
void loop() {
    // Voir interruption
}
```

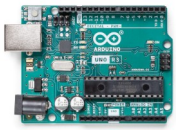


SEI		Global Interrupt Enable
CLI		Global Interrupt Disable



Traitement de données

Moyenne flottante sur 10 valeurs



Le programme test Arduino

```
int n;
int Tab_Val[12] = {0,0,0,0,0,0,0,0,0,0,0,0};
int Somme;

// Renvoie la moyenne sur 10 valeurs
float Sum_Tab(int In_Val){

  // Décalage de la table
  for(n = 9; n >= 0; n--){ Tab_Val[n+1] = Tab_Val[n]; }

  // Insertion de la nouvelle valeur
  Tab_Val[0] = In_Val;

  Somme += In_Val - Tab_Val[10];
  //Somme = 0;
  //for(n = 0; n < 10; n++){Somme += Tab_Val[n];}

  return(float(Somme)/10.0);
}

void setup() {

  Somme = 0;
  Serial.begin(9600);
}

void loop() {

  // Test
  Serial.println( Sum_Tab(0) );

  Serial.println( Sum_Tab(10) ); Serial.println( Sum_Tab(20) );
  Serial.println( Sum_Tab(30) ); Serial.println( Sum_Tab(40) );
  Serial.println( Sum_Tab(50) ); Serial.println( Sum_Tab(60) );
  Serial.println( Sum_Tab(70) ); Serial.println( Sum_Tab(80) );
  Serial.println( Sum_Tab(90) ); Serial.println( Sum_Tab(100) );

  Serial.println( Sum_Tab(0) ); Serial.println( Sum_Tab(0) );
  Serial.println( Sum_Tab(0) ); Serial.println( Sum_Tab(0) );
  Serial.println( Sum_Tab(0) ); Serial.println( Sum_Tab(0) );
  Serial.println( Sum_Tab(0) ); Serial.println( Sum_Tab(0) );
  Serial.println( Sum_Tab(0) ); Serial.println( Sum_Tab(0) );

  while(true){}
}
```

Initialisation de la table

2 méthodes

```
COM8
0.00
1.00
3.00
6.00
10.00
15.00
21.00
28.00
36.00
45.00
55.00
54.00
52.00
49.00
45.00
40.00
34.00
27.00
19.00
10.00
0.00
```

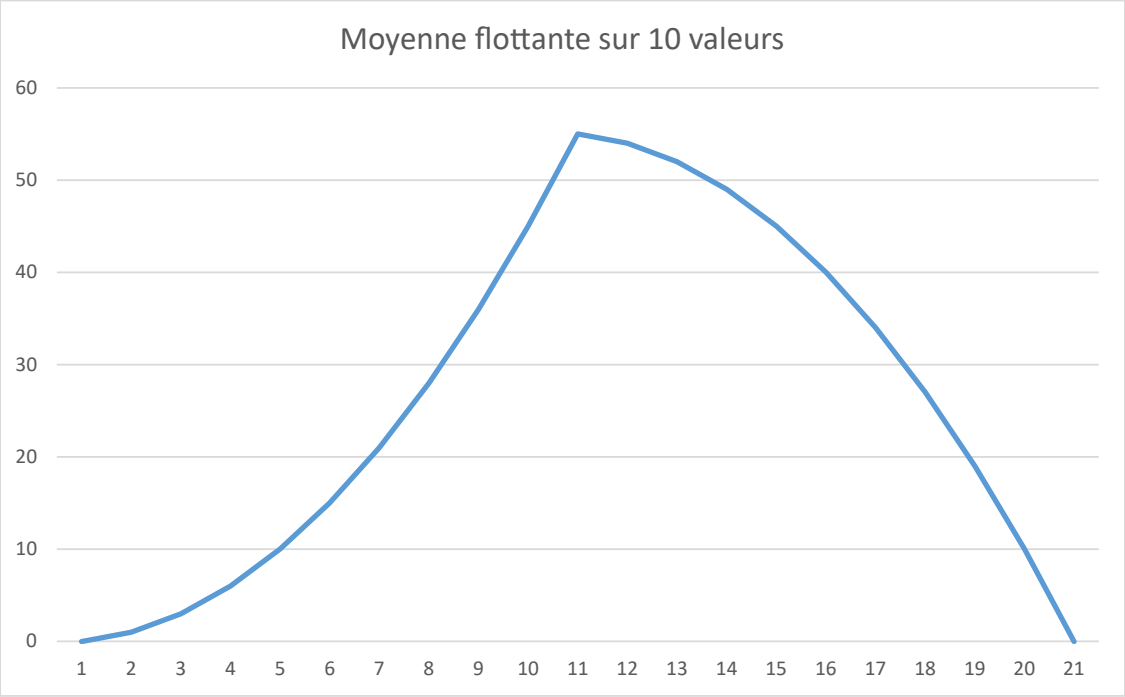
Version tableur

	A	B	C
1	Données	Somme flottante	Moyenne flottante
2	0		
3	0		
4	0		
5	0		
6	0		
7	0		
8	0		
9	0		
10	0		
11	0	0	0
12	10	10	1
13	20	30	3
14	30	60	6
15	40	100	10
16	50	150	15
17	60	210	21
18	70	280	28
19	80	360	36
20	90	450	45
21	100	550	55
22	0	540	54
23	0	520	52
24	0	490	49
25	0	450	45
26	0	400	40
27	0	340	34
28	0	270	27
29	0	190	19
30	0	100	10
31	0	0	0

Initialisation de la table

COM8

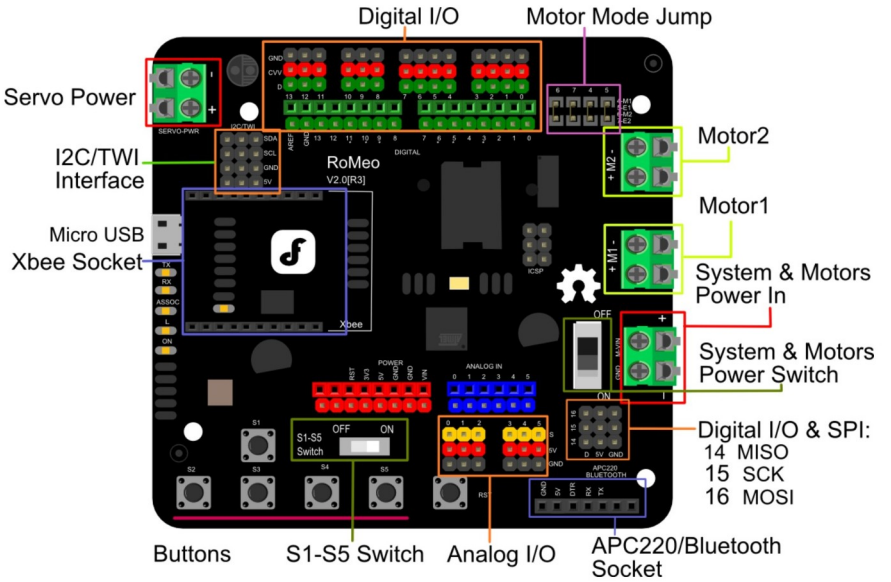
```
0.00
1.00
3.00
6.00
10.00
15.00
21.00
28.00
36.00
45.00
55.00
54.00
52.00
49.00
45.00
40.00
34.00
27.00
19.00
10.00
0.00
```



Applications avec la carte

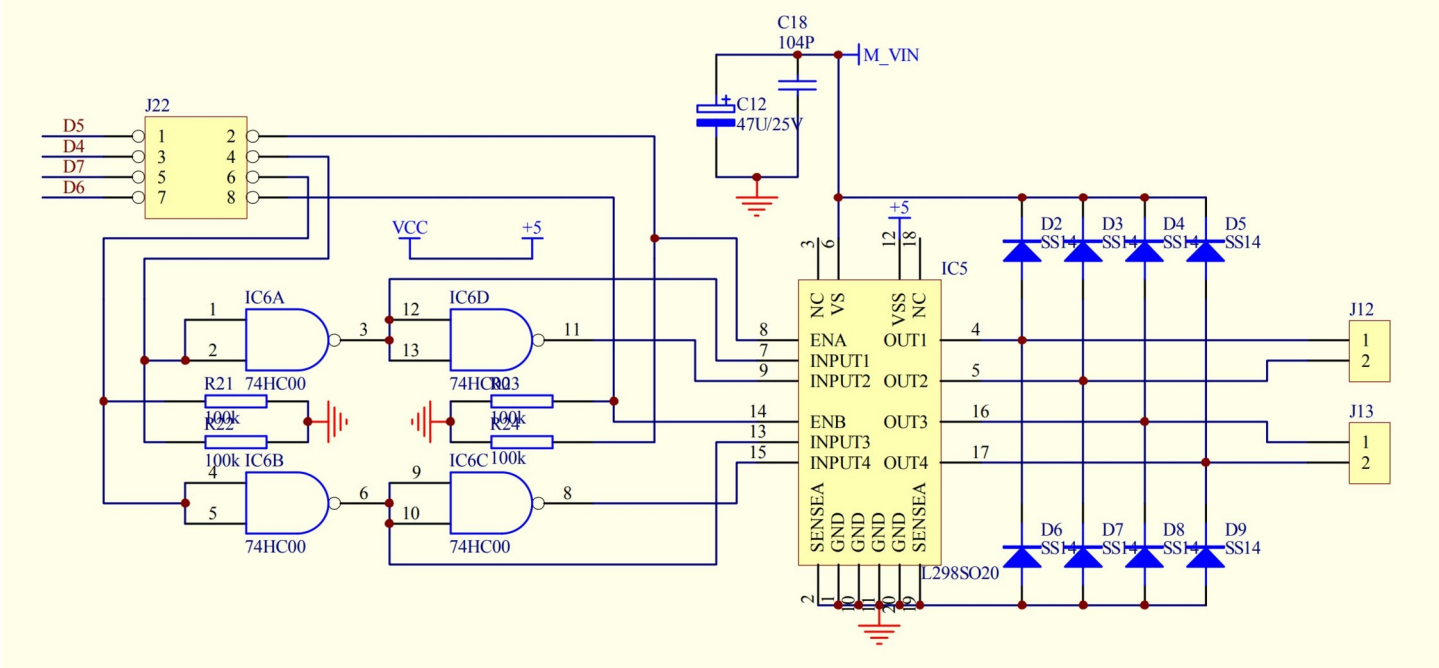
ROMEO V.2

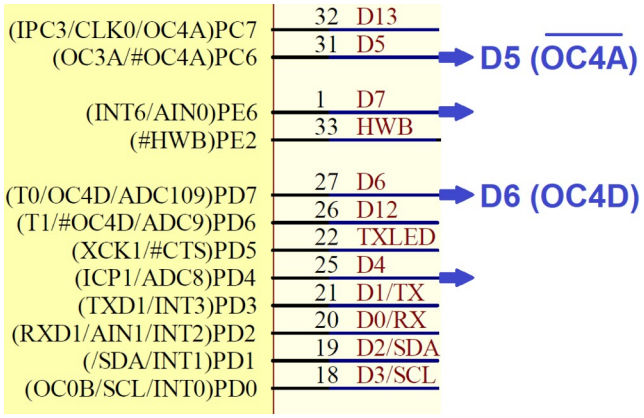
RoMeo V2 Pinout



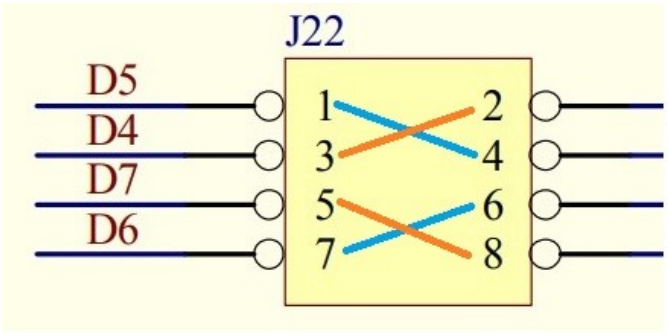
Microcontrôleur : ATmega 32U4

1) Le driver moteurs (L298)





ATmega 32U4



Configuration
Lock Anti Phase Drive

15. 10-bit High Speed Timer/Counter4

15.1 Features

- Up to 10-Bit Accuracy
- Three Independent Output Compare Units
- Clear Timer on Compare Match (Auto Reload)
- Glitch Free, Phase and Frequency Correct Pulse Width Modulator (PWM)
- Enhanced PWM mode: one optional additional accuracy bit without effect on output frequency
- Variable PWM Period
- Independent Dead Time Generators for each PWM channels
- Synchronous update of PWM registers
- Five Independent Interrupt Sources (TOV4, OCF4A, OCF4B, OCF4D, FPF4)
- High Speed Asynchronous and Synchronous Clocking Modes
- Separate Prescaler Unit

1.1) Programme d'initialisation – Le Timer 4

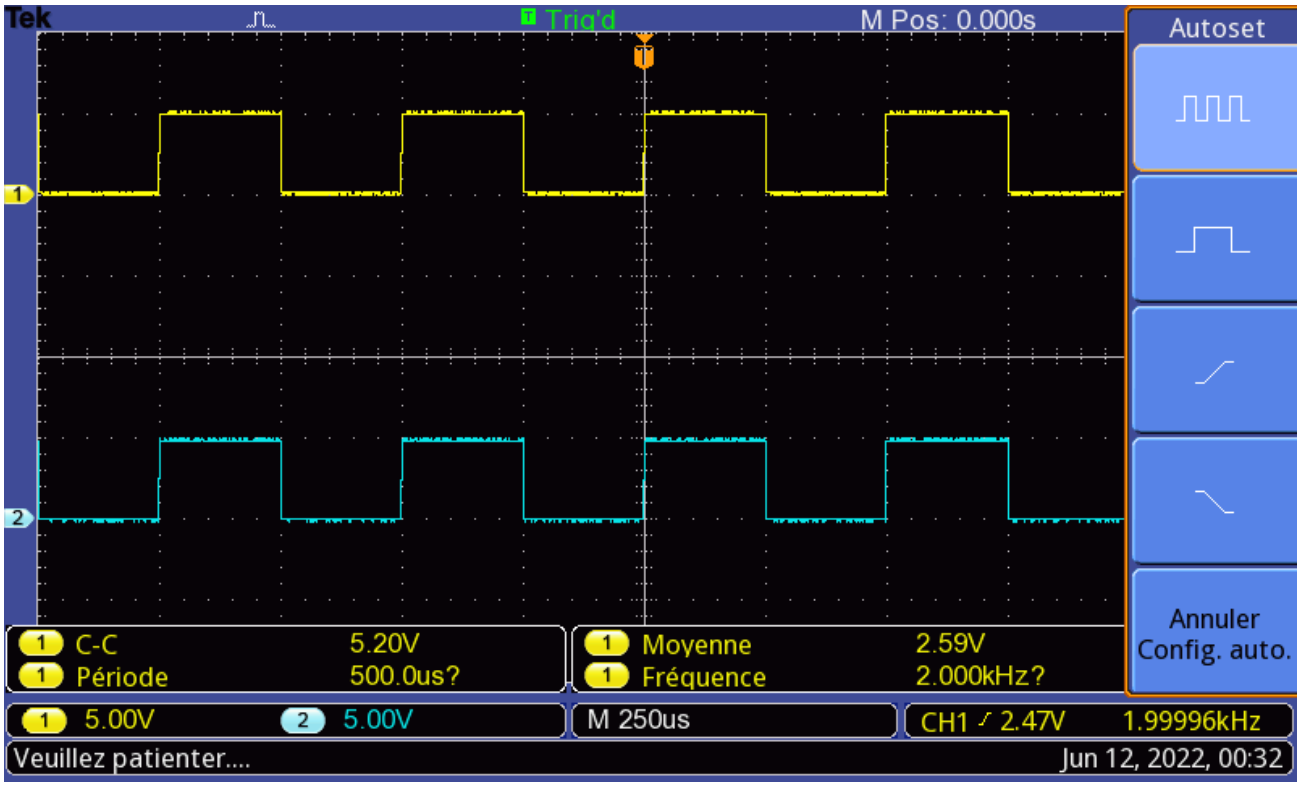
Romeo_V2_prog § | PWM_Timer4 §

```
// PWM résolution de 8 bits (Timer 4)

// Broche D7: PE6 (Enable)
// Broche D6: PD7 (OC4D)
// Broche D5: PC6 (OC4A)
// Broche D4: PD4 (Enable)

unsigned char Freq_Timer4;

// Freq_PWM en Hz
// Renvoie la valeur de OCR4C
unsigned char Init_PWM_Timer4(unsigned int Freq_PWM){

  DDRD |= 0b10010000;     // PD7 (OC4D) et PD4 (Enable) -> Output
  DDRC |= 0b01000000;     // PC6 (OC4A) -> Output
  DDRE |= 0b01000000;     // PE6 (Enable) -> Output

  // Freq_T4 = 250000 Hz
  TCCR4B = 0b10000111;    // Prescaler /64

  // Page 169: Bit 0 - PWM4D: Pulse Width Modulator D Enable
  // When set (one) this bit enables PWM mode based on comparator OCR4D.
  TCCR4C = 0b00001101;
  TCCR4D = 0x00;

  // OCR4C = (250000/Freq_PWM) - 1
  // OCR4C = 82 donnera une fréquence PWM de 3KHz
  Freq_Timer4 = (250000/Freq_PWM) - 1;
  OCR4C = Freq_Timer4;

  // Moteur à l'arrêt (Lock Anti Phase Drive)
  OCR4A = Freq_Timer4 >> 1;
  OCR4D = OCR4A;

  // Page 167: Bit 1 - PWM4A: Pulse Width Modulator A Enable
  // When set (one) this bit enables PWM mode based on comparator OCR4A.
  TCCR4A = 0b01000010;
  TCNT4 = 0x00;

  PORTE |= 0b01000000;    // Broche D7: PE6 (Enable)
  PORTD |= 0b00010000;    // Broche D4: PD4 (Enable)
  return(Freq_Timer4);
}
```

Suite (programme d'initialisation – Le Timer 4)

Romeo_V2_prog § | PWM_Timer4 §

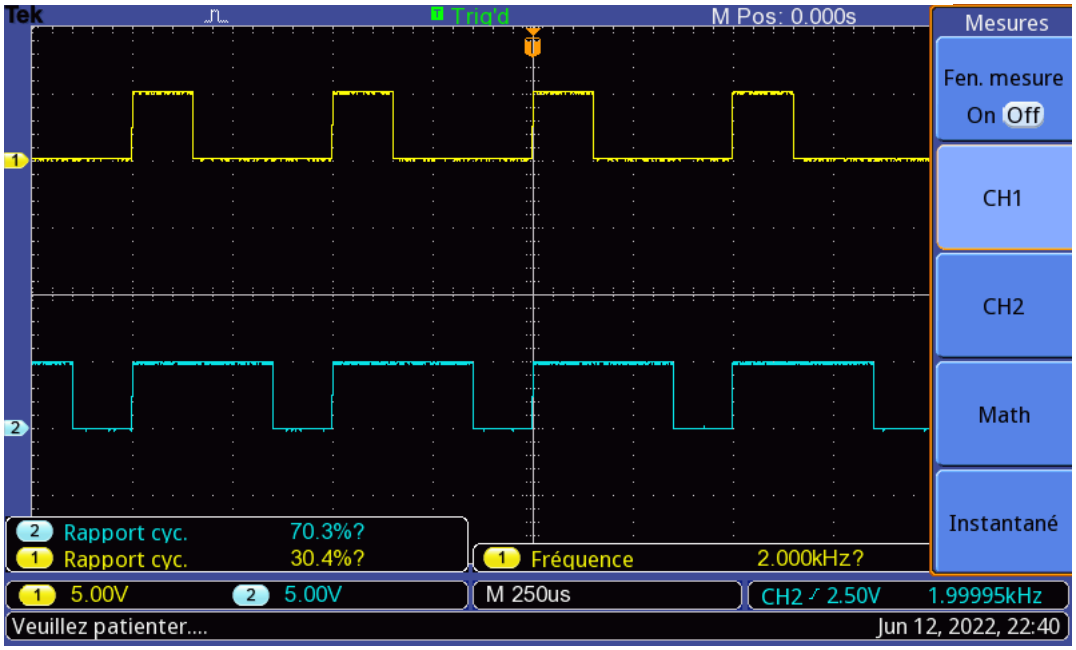
```
// IDE -> Type de carte: Arduino Léonardo

void setup() {
  Init_PWM_Timer4(2000);
}

void loop() {

}
```



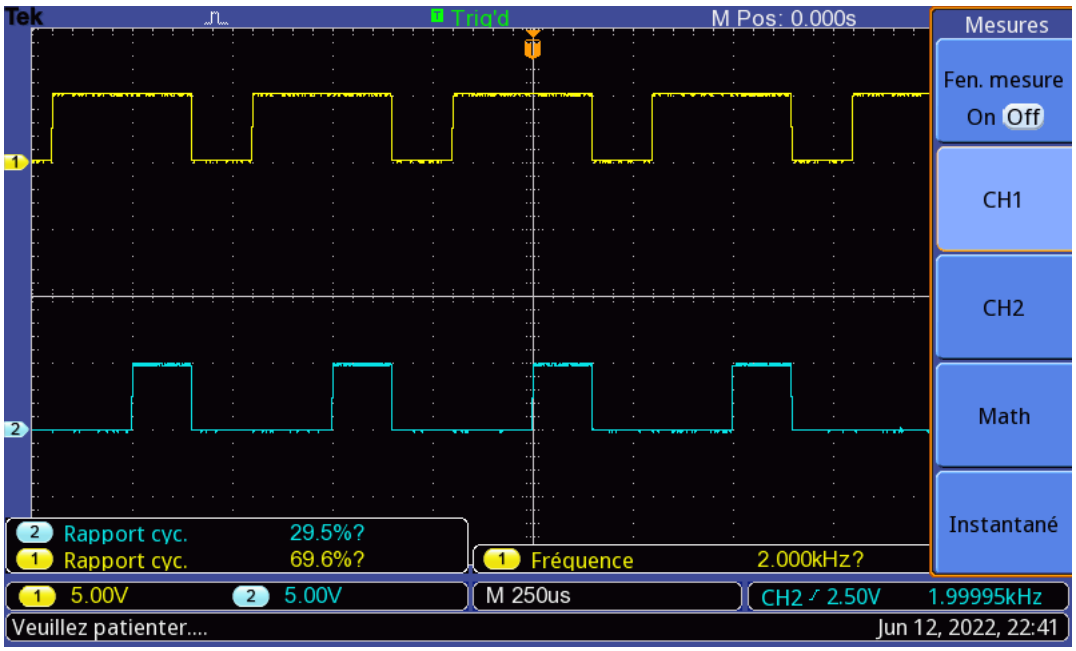
1.2) Programme d'initialisation (Le Timer 4)

- Modification du registre TCCR4B -

```
//Test à 2KHz
OCR4A = 37;             // 30%
OCR4D = 87;             // 70%
TCCR4B = 0b10000111;    // (Inversion) Prescaler /64
```




```
//Test à 2KHz
OCR4A = 37;             // 30%
OCR4D = 87;             // 70%
TCCR4B = 0b00000111;    // (Pas d'inversion) Prescaler /64
```



- **Bit 7 - PWM4X: PWM Inversion Mode**

When this bit is set (one), the PWM Inversion Mode is selected and the Dead Time Generator outputs, OC4x and $\overline{OC4x}$ are inverted.

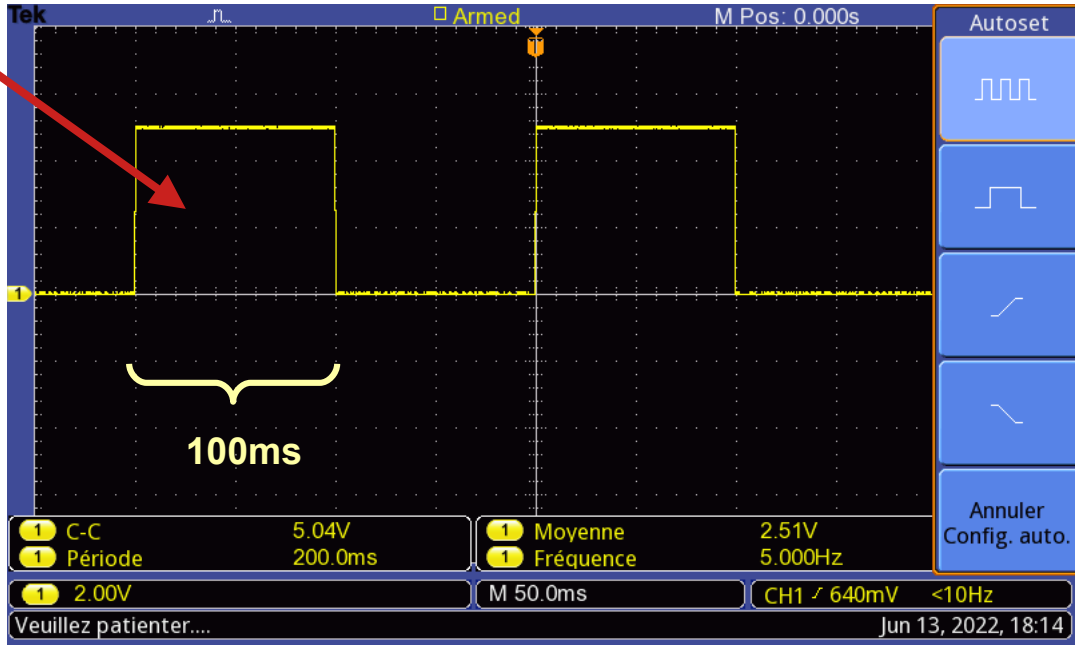
1.3) Programme d'initialisation (Le Timer 1 et interruption)

Romeo_V2_prog | PWM_Timer4 | Timer1_CAN

```
// Timer 1 avec interruption
// Prise de mesure ADC toutes les T1_ms
// ADC Auto Trigger -> Timer/Counter1 Compare Match B
// Point test sur la broche D2

void Init_Timer1(unsigned int T1_ms){

  DDRD |= 0b00000010;      // PD1 (D2 - Point test) -> Output

  TCCR1A = 0x00;
  TCCR1B = 0x03;             // Mode normal, prescaler /64
  OCR1B = 250*T1_ms - 1;
  TIMSK1 = 0b00000100;       // Bit 2 - OCIE1B: Timer/Countern
                             // Output Compare B Match Interrupt Enable
  TCNT1 = 0x00;
}

// Gestion de l'interruption Timer 1
ISR(TIMER1_COMPB_vect){

  TCNT1 = 0x00;
  PORTD ^= 0b00000010;       // XOR -> Toggle
}
```

Romeo_V2_prog | PWM_Timer4 | Timer1_CAN

```
// IDE -> Type de carte: Arduino Léonardo

void setup() {
  cli();
  Init_PWM_Timer4(2000);
  Init_Timer1(100);
  sei();
}

void loop() {

}
```



1.4) Le Timer 1, le CAN et interruptions

Romeo_V2_prog § | PWM_Timer4 | Timer1_CAN §

```
// ADC0 (PF0) -> broche A5 - voir schéma
// ADC Auto Trigger -> Timer/Counter1 Compare Match B

unsigned int ADC_Val;
float Volt_CAN, Volt_Arret;
bool Flag_ADC, Flag_Stop;

void Init_CAN(char Factor, float Volt_Stop){

  ADMUX = 0b01000000;              // Vref = +5V, ADLAR = 0, MUX = A0
  ADCSRA = 0b10101000 | Factor;    // Bit 7 - ADEN: ADC Enable
                                   // Bit 5 - ADATE: ADC Auto Trigger Enable
                                   // Bit 3 - ADIE: ADC Interrupt Enable

  ADCSRB = 0x05;                   // Timer/Counter1 Compare Match B

  Volt_Arret = Volt_Stop;          // Tension "arret des moteurs"
  Flag_ADC = false;
  Flag_Stop = false;
}

// Gestion de l'interruption CAN
ISR(ADC_vect){

  //ADC_Val = ADCL | (ADCH << 8);
  *(char*)(&ADC_Val) = ADCL;
  *((char*)(&ADC_Val) + 1) = ADCH;

  Volt_CAN = (5.0 * ADC_Val) / 1023;  // ADC Conversion Result (Page 309)

  if(Volt_CAN > Volt_Arret){
    // Arret des moteurs
    OCR4A = PWM_Val >> 1;
    OCR4D = OCR4A;
    Flag_Stop = true;
  }
  else{
    Flag_Stop = false;
    // Pour le test
    OCR4A = PWM_Val * 0.3;       // 30%
    OCR4D = PWM_Val * 0.7;       // 70%
  }

  Flag_ADC = true;
}
```

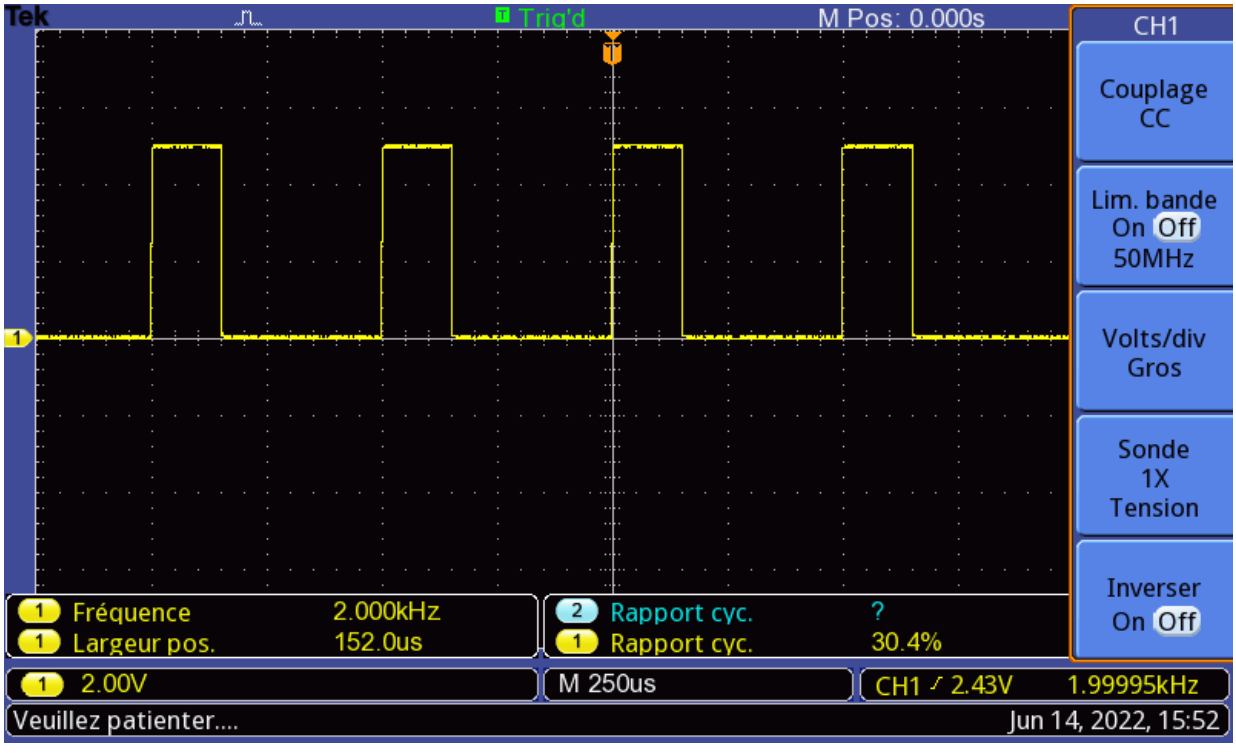
2 méthodes

Traitement de l'arrêt
des moteurs dans l'interruption

1.4) Suite du programme

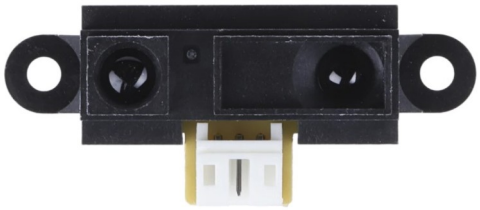
Romeo_V2_prog § | PWM_Timer4 | Timer1_CAN

```
// IDE -> Type de carte: Arduino Léonardo

unsigned char PWM_Val;

void setup() {
  cli();
  PWM_Val = Init_PWM_Timer4(2000);
  Init_Timer1(100);
  Init_CAN(0x05, 1.25);        // Divisor Factor 32
                               // A0 = 1.25v -> Arret moteurs
  sei();

  // Pour le test
  OCR4A = PWM_Val * 0.3;       // 30%
  OCR4D = PWM_Val * 0.7;       // 70%
}

void loop() {

}
```



1.4.1) Essai

SHARP GP2Y0A21YK0F



Voir document constructeur