01/12/2017

Compte rendu

TP SEM1

Guilpain Léo & Legris Thomas

Table des matières

Introduction

L'objectif de ces travaux pratiques était de manipuler les microcontrôleurs Atmega16, de mettre en pratique le cours théorique et donc d'apprendre à programmer en langage C. Nous avons pu par exemple allumer des diodes avec des boutons, afficher des textes sur un écran ou bien réagir à des signaux analogiques du style mouvement sur un Joystick.

Chapitre 2 : Communication série RS-232

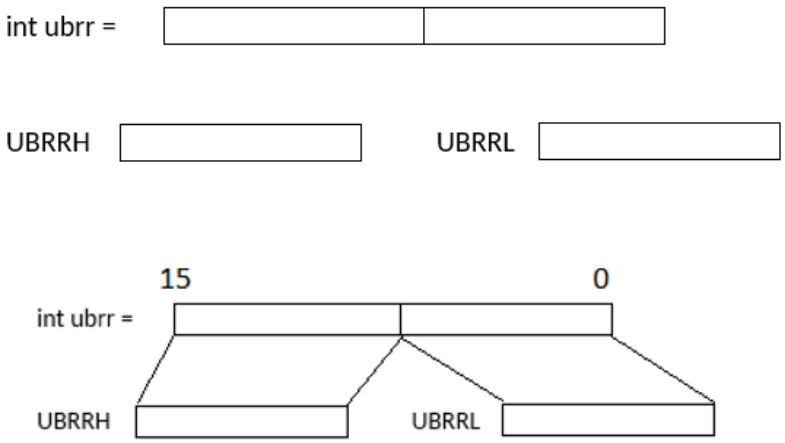
1. Configuration de la communication série

Les caractéristiques de la communication sont gérées dans les paramètres de la communication et non dans le code.

```
void usart_init(unsigned int debit){
    UCSRB |= (1<<RXEN) | (1<<TXEN); //def USCSRB (doc)
    UCSRC |= (1<<URSEL)| (1<<UCSZ1)| (1<<UCSZ0);//def USCSRC (doc)
    unsigned int ubrr = (F_CPU/debit/16)-1;

    UBRRL =(unsigned char)ubrr; //on met l'octet de poid faible dans UBRRL, on prend les 8 premiers (on part de la droite)
    UBRRH =(unsigned char)(ubrr>>8);
}
```

Dans cette fonction, on effectue différentes actions :

- Dans le registre UCSRB :
  - RXEN à 1 : Active la réception
  - TXEN à 1 : Active la transmission
- Dans le registre UCSRC :
  - URSEL à 1 : Sélection de UCSRC
  - UCSZ1 à 1 : 8 bits de données
  - UCSZ0 à 1 : 8 bits de données



2. Envoi d'un caractère

```
//fonction 1 char
void  usart_putc(char c){
    while ((UCSRA&(1<<UDRE))==0){
    }
    UDR = c;
}
```

On vérifie dans un premier temps que UDR est prêt à recevoir de nouvelles données. Si UDR est vide cela signifie qu'il est prêt, on peut ainsi stocker la valeur du caractère dans UDR.

3. Envoi d'une chaîne de caractères

```
//fonction chaine de caractère

void  usart_puts(char*s){
    int i=0;
    while(s[i]!='\0'){
        usart_putc(s[i]);
        i++;
    }
}
```

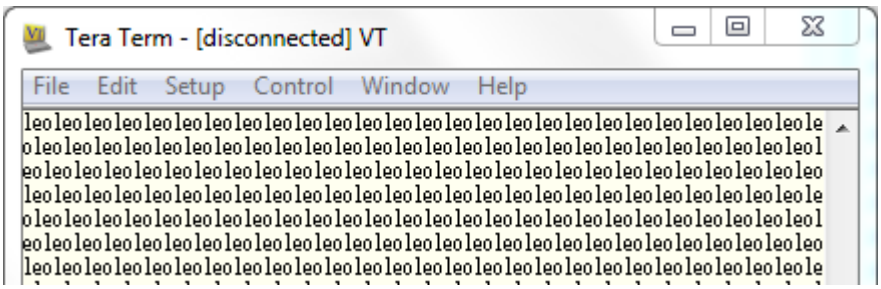
On parcourt la chaine de caractère s (par exemple « toto ») jusqu'à la fin qui est determinée par « \0 ». A chaque itération, on appelle la fonction créée précedement. On stocke ainsi le caractère lu dans UDR à chaque itération.

4. Réception d'une chaine caractère

```
char usart_getc(void){ //on recoit sur le micro le code correspondant à ce que l'on a tapé
    while((UCSRA&(1<<RXC))==0){//si c'est égale à 0 on fait rien
    }
    return UDR; // si ==1 on recoit la donnée
}
```

Tant que toutes les données n'ont pas été reçues on ne fait rien, ensuite on renvoie les données qui ont été stockées dans UDR.

Dans l'exemple, on a envoyé la chaine de caractère « leo ».

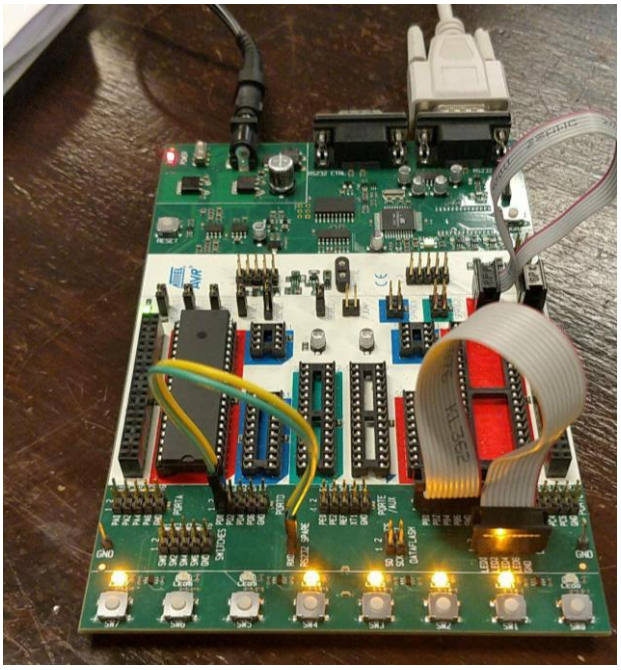


5. Communication bidirectionnelle

Voici le code complet avec la réception de la chaine de caractère :

```
void usart_init(unsigned int debit);
void  usart_putc(char c);
void  usart_puts(char*s);
char usart_getc(void);

int main(void){// on initialise le serial
    usart_init(57600);//freq

    usart_puts("tapez");
    while(1){
        char c = usart_getc();//on appelle la fonction d'envoi de chaîne
        DDRB = 0xFF; // init
        PORTB = c;  //on stock "c" qui va allumer les diodes selon la valeur de c (
    }
    return 1;
}

void usart_init(unsigned int debit){
    UCSRB |= (1<<RXEN) | (1<<TXEN); //def USCSRB (doc)
    UCSRC |= (1<<URSEL)| (1<<UCSZ1)| (1<<UCSZ0);//def USCSRC (doc)
    unsigned int ubrr = (F_CPU/debit/16)-1;

    UBRRL =(unsigned char)ubrr; //on met l'octet de poid faible dans UBRRL, on prend les 8 premiers (on part de la droite)
    UBRRH =(unsigned char)(ubrr>>8);
}

//fonction 1 char
void  usart_putc(char c){
    while ((UCSRA&(1<<UDRE))==0){} // tant que il n'y a pas de reception, on ne fait rien
    UDR = c;    //sinon on stock c
}


//fonction chaine de caractère

void  usart_puts(char*s){
    int i=0;
    while(s[i]!='\0'){ //tant que la chaine de caractères n'est pas finie
        usart_putc(s[i]);
        i++;
    }
}


char usart_getc(void){ //on recoit sur le micro le code correspondant à ce que l'on a tapé
    while((UCSRA&(1<<RXC))==0){//si c'est égale à 0 on fait rien
    }
    return UDR; // si ==1 on recoit la donnée
}
```

Voici le résultat sur les LEDS lorsque l'on appuie sur la lettre « a » du clavier. C'est le code ASCII qui est renvoyé sur les LEDS



Chapitre n°3 : Télémètre à ultrasons

```
void pulse(void){
    DDRD |= (1<<PD2) ;
    PORTD |= (1<<PD2);

}
```

Ici, on a créé une fonction *pulse(void)* permettant de mettre le PORTD n°2 en sortie et à 1. On génère ici un front montant.

```
#include<avr/io.h>;
#define F_CPU 3686400
#include <util/delay.h>
void pulse(void);

int main(void){
    while(1){
    _delay_ms(50);//on attend 50ms avant de générer un nouveau pic
    pulse();
    _delay_us(12); //délai de 12ms avant de redescendre
    PORTD &=~(1<<PD2);  //remise à zéro
    }
    return 1;
}
```

Ensuite, dans la fonction main, on va créer des impulsions (pics avec front montant et front descendant).

On peut voir qu'on génère un front montant, puis on patiente 12 micro- secondes avant de générer le front descendant (remise à zéro du PORTD n°2).

Ensuite, on gère les interruptions :

```
int main(void){
    TIMSK |=(1<<TICIE1); //1 par source d'interuption
    TCCR1B |=(1<<CS10)|(1<<ICES1);
    sei(); //interupt enable global, ici il est activé
```

Comme écrit ci-dessus, on active dans un premier temps les interruptions (sei).

- Registre TIMSK :
    - TICIE1 à 1 : Activation de l'entrée capture du Compteur/Timer

```
ISR(TIMER1_CAPT_vect){

    if ((TCCR1B & (1<<ICES1)) == 0){

        t2=ICR1;
        t=t2-t1;
        dist = t/(3.6*58); //l'horloge est à 3.6 GHz donc il faut divisé par 3.6 pour avoir le temps
        PORTB = t/(3.6*58);
        char s[20];
        sprintf(s, "Dist=%d \n\r", dist);
        usart_puts(s);
        TCCR1B |= (1<<ICES1);
        }
    else{
        t1=ICR1;
        TCCR1B &=~ (1<<ICES1);
    }
}
```

Lorsqu'il y a une interruption, si on a un front descendant (ICES1 à 0), on stocke la valeur de ICR1 dans la variable t2 puis on fait passer la valeur de ICES1 de 0 à 1. Si on a un front montant (ICES1 à 1), on stocke la valeur de ICR1 dans la variable t1 et on fait passer ICES1 de 1 à 0.

Ensuite, on calcule le temps final t = t2 – t1.

Le SFR04 fournit une echo pulsation proportionnelle à la distance. On a mesuré la largeur du pic en microseconde, c'est pourquoi, pour avoir la distance en centimètre, il faut diviser par 58. Ensuite on divise par la fréquence de l'horloge.

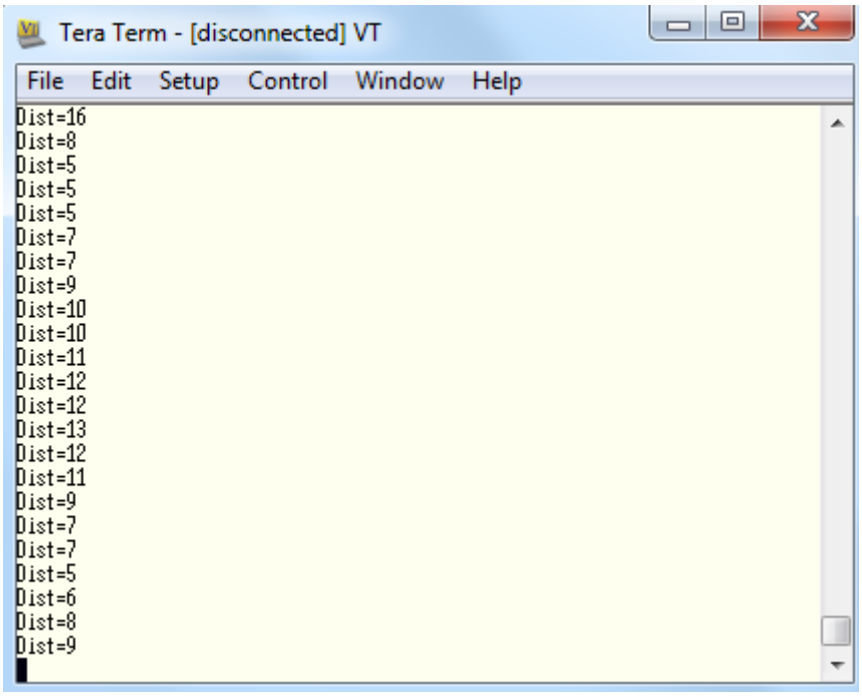
On stocke cette distance sur le PORTB.

On réutilise les fonctions du TP précédent pour faire une communication série et transmettre une chaine de caractères.

Voici le code complet (raccord de deux images) :

```
#include<avr/io.h>
#define F_CPU 3686400
#include <util/delay.h>
#include<avr/interrupt.h>
#include <stdio.h>
void pulse(void);
volatile unsigned int t1;
volatile unsigned int t2;
volatile unsigned int t;
int dist;
ISR(TIMER1_CAPT_vect);
void usart_putc(char c);
void usart_puts(char*s);
void usart_init(unsigned int debit);

ISR(TIMER1_CAPT_vect){


    if ((TCCR1B & (1<<ICES1)) == 0){


        t2=ICR1;
        t=t2-t1;
        dist = t/(3.6*58); //l'horloge est à 3.6 GHz donc il faut divisé par 3.6 pour avoir le temps
        PORTB = t/(3.6*58);
        char s[20];
        sprintf(s, "Dist=%d \n\r", dist);
        usart_puts(s);
        TCCR1B |= (1<<ICES1);
        }
    else{
        t1=ICR1;
        TCCR1B &=~ (1<<ICES1);
    }
}

void pulse(void){
    DDRD |= (1<<PD2) ; //on utilise PD2 = 0 | 4
    PORTD |= (1<<PD2); //on a juste PD2 en sortie

}

int main(void){
    TIMSK |=(1<<TICIE1); //1 par source d'interuption
    TCCR1B |=(1<<CS10)|(1<<ICES1);
    sei(); //interupt enable global, ici il est activé

    while(1){
    _delay_ms(50);//on attend 50ms avant de générer un nouveau pic
    pulse();
    _delay_us(12); //délai de 12ms avant de redescendre
    PORTD &=~(1<<PD2);  //remise à zéro
    }
    return 1;


}
```

Lorsque la connection série est activée, on obtient ceci (distance en centimètre) :



Chapitre n° 4 : Joystick

```
#include <avr/io.h>
#define F_CPU 3686400


void usart_puts (char* s);
void usart_putc (char c);
void USART_INIT(unsigned int debit);

int main(void){

    USART_INIT(57600);
    DDRB = 0xFF;
    ADMUX |= (1<<ADLAR);
    ADCSRA |= (1<<ADEN)|(1<<ADATE);
    while(1){
        //1er axe
        ADMUX &=~(1<<MUX0);
        ADCSRA |= (1<<ADSC);
        while ((ADCSRA & (1<<ADIF))==0){
            }
        PORTB=ADCH;
        char s[20];
        sprintf (s,"Horizontal=%d\n\r",PINB);
        usart_puts(s);

        //2nd axe
        ADMUX |= (1<<MUX0);
        ADCSRA |= (1<<ADSC);
        while ((ADCSRA & (1<<ADIF))==0){
            }
        PORTB=ADCH;
        char t[20];
        sprintf (t,"Vertical=%d\n\r",PINB);
        usart_puts(t);


    }
    return 1;

}
```

On réutilise les fonctions des TPs précédents pour la communication série.

- Registre ADMUX :
  - ADLAR à 1 : Ajuster le résultat à gauche. On prend que les bits de poids fort.
- Registre ADCSRA :
  - ADEN à 1 : Active l'ADC
  - ADATE à 1 : Active l'auto ADC Trigger

On ne peut pas gérer les deux axes simultanément, il faut gérer un axe après l'autre.

<u>Sur le 1er axe :</u>

- Registre ADMUX :
  - MUX0 à 0 : Correspond à ADC0
- Registre ADCSRA :
  - ASDC à 1 : Démarre la conversion

Tant que le bit d'interruption n'est pas activé, on ne fait rien (ADIF à 0). En revanche, dès qu'il est activé, on stocke les valeurs de ADCH sur PORTB.

Ensuite, on envoie les valeurs de PINB dans la connexion série et on récupère les valeurs.

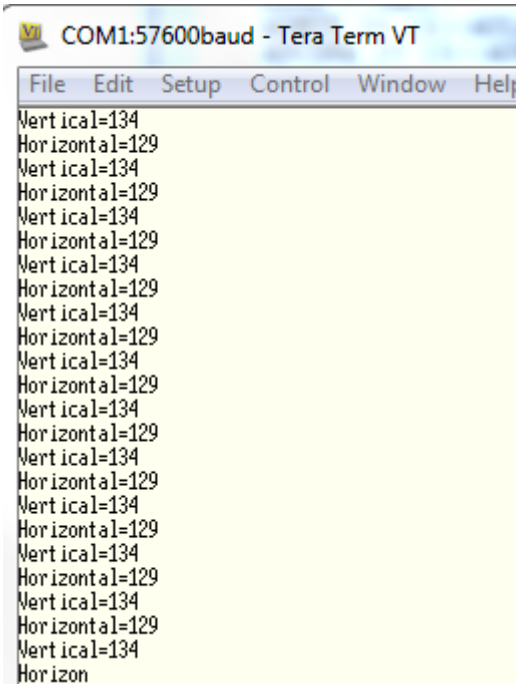
Sur le 2ème axe :

- Registre ADMUX :
  - MUX0 à 0 : Correspond à ADC1
- Registre ADCSRA :
  - ASDC à 1 : Démarre la conversion

Tant que le bit d'interruption n'est pas activé, on ne fait rien (ADIF à 0). En revanche, dès qu'il est activé, on stocke les valeurs de ADCH sur PORTB.

Ensuite, on envoie les valeurs de PINB dans la connexion série et on récupère les valeurs.

Après la connexion en série, on obtient ceci :



Chapitre n°5 : Afficheur LCD

1. Bus de données

```
void port_data(unsigned char octet){
    //PORTA
    if ((octet&(1<<0))!=0){
        PORTA |=(1<<0);
    }
    else {
        PORTA &=~(1<<0);
    }

    if ((octet&(1<<1))!=0){
        PORTA |=(1<<3);
    }
    else {
        PORTA &=~(1<<3);
    }

    if ((octet&(1<<2))!=0){
        PORTA |=(1<<4);
    }
    else {
        PORTA &=~(1<<4);
    }

    if ((octet&(1<<3))!=0){
        PORTA |=(1<<7);
    }
    else {
        PORTA &=~(1<<7);
    }



    //PORTD

    if ((octet&(1<<4))!=0){
        PORTD |=(1<<0);
    }
    else {
        PORTD &=~(1<<0);
    }

    if ((octet&(1<<5))!=0){
        PORTD |=(1<<3);
    }
    else {
        PORTD &=~(1<<3);
    }

    if ((octet&(1<<6))!=0){
        PORTD |=(1<<4);
    }
    else {
        PORTD &=~(1<<4);
    }

    if ((octet&(1<<7))!=0){
        PORTD |=(1<<7);
    }
    else {
        PORTD &=~(1<<7);
    }
```

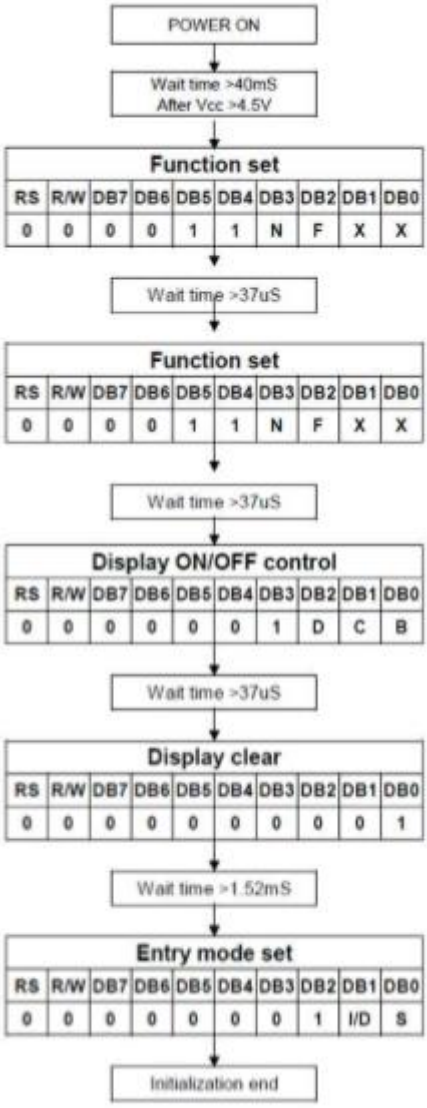
Cette fonction permet d'envoyer l'octet « octet » au bus de données.

2. Commande du LCD

```
void lcd_command(unsigned char cmd) {

    PORTB|=(1<<PB2); // activation
    port_data(cmd);
    _delay_ms(5);
    PORTB&=~(1<<PB2);

}
```

On active le bit E en mettant PB2 à 1, puis on envoie la commande « cmd » sur le bus de données puis après avoir attendu 5 millisecondes, on désactive le bit E.

3. Initialisation du LCD




```
void lcd_init(){

    _delay_ms(40);

    //function set
    lcd_command(0x38);
    _delay_us(37);

    //display on/off
    lcd_command(0x0F);
    _delay_us(37);

    //display clear
    lcd_command(0x01);
    _delay_ms(5);

    //entry mode set
    lcd_command(0x06);

}
```

Pour initialiser le LCD, il faut suivre la procédure de gauche.

Les différentes commandes sont :

- Function Set (0x38)
- Display ON/OFF (0x0F)
- Display clear (0x01)
- Entry mode set (0x06)

4. Envoi d'un caractère au LCD

```
void lcd_output(unsigned char data){
    PORTB|=(1<<PB2); // activation
    PORTB|=(1<<PB0); //RS : "1" envoie de données
    port_data(data);
    _delay_ms(5);
    PORTB&=~(1<<PB2);
}
```

Le but est d'envoyer la donnée au LCD (ici « data »).

Pour cela, on fait passer le bit E (PB2) à 1 et on fait passer le bit RS (PB0) à 1, cela correspond à l'envoie de données. Ensuite on envoie la donnée « data » sur le bus, puis après 5 millisecondes, on fait passer le bit E à 0.

5. Envoie d'une chaine de caractères au LCD

```
void chaine (char* s) {

    unsigned char i =0 ;
    while (s[i] != '\0')
    {
        char x = s[i];
        lcd_output(x);
    i++;
    }
}
```

On envoie un caractère après l'autre sur le bus de données. Voici ce que l'on obtient après avoir envoyé « toto ».

Voici le code complet :

```
#include <avr/io.h>
#include <util/delay.h>
void port_data(unsigned char octet);
void lcd_command(unsigned char cmd);
void lcd_output(unsigned char data);
void chaine (char* s);
void lcd_init();


int main(void){
    DDRA = 0xFF;
    DDRB = 0xFF;
    DDRD = 0xFF;
    lcd_init();
    //lcd_output('s');
    chaine("toto");
}


void port_data(unsigned char octet){
    //PORTA
    if ((octet&(1<<0))!=0){
        PORTA |=(1<<0);
    }
    else {
        PORTA &=~(1<<0);
    }

    if ((octet&(1<<1))!=0){
        PORTA |=(1<<3);
    }
    else {
        PORTA &=~(1<<3);
    }

    if ((octet&(1<<2))!=0){
        PORTA |=(1<<4);
    }
    else {
        PORTA &=~(1<<4);
    }

    if ((octet&(1<<3))!=0){
        PORTA |=(1<<7);
    }
    else {
        PORTA &=~(1<<7);
    }


    //PORTD

    if ((octet&(1<<4))!=0){
        PORTD |=(1<<0);
    }
    else {
        PORTD &=~(1<<0);
    }

    if ((octet&(1<<5))!=0){
        PORTD |=(1<<3);
    }
    else {
        PORTD &=~(1<<3);
    }

    if ((octet&(1<<6))!=0){
        PORTD |=(1<<4);
    }
    else {
        PORTD &=~(1<<4);
    }

    if ((octet&(1<<7))!=0){
        PORTD |=(1<<7);
    }
    else {
        PORTD &=~(1<<7);
    }


}
```

```
void lcd_command(unsigned char cmd) {

    PORTB|=(1<<PB2); // activation
    port_data(cmd);
    _delay_ms(5);
    PORTB&=~(1<<PB2);

}


void lcd_init(){

    _delay_ms(40);

    //function set
    lcd_command(0x38);
    _delay_us(37);

    //display on/off
    lcd_command(0x0F);
    _delay_us(37);

    //display clear
    lcd_command(0x01);
    _delay_ms(5);

    //entry mode set
    lcd_command(0x06);

}
void lcd_output(unsigned char data){
    PORTB|=(1<<PB2); // activation
    PORTB|=(1<<PB0); //RS : "1" envoie de données
    port_data(data);
    _delay_ms(5);
    PORTB&=~(1<<PB2);
}
void chaine (char* s) {


    unsigned char i =0 ;
    while (s[i] != '\0')
    {
        char x = s[i];
        lcd_output(x);
    i++;
    }
}
```

Conclusion

Pour conclure, les différents chapitres nous ont permis de gérer la communication série et de gérer des programmes très simple. On a ensuite évolué vers des programmes plus complexes afin de gérer des interruptions, les réactions par rapport aux signaux analogiques tels que le changement de direction d'un joystick, qui correspondait à une certaine combinaison de diodes allumées.