Université de Clermont Auvergne
École Universitaire de Physique et d'Ingénierie
Master 2 Mécatronique

Rapport de projet microcontrôleurs : Afficheur LCD

Albaladejo Joris

10 Décembre 2022

Professeur référent : François Marmoiton

Liste des abréviations

ASCII American Standard Code for Information Interchange

LCD Liquid Crystal Display

Table des matières

Table des figures

Liste des tableaux

1 Introduction

Au cours de ce projet, nous avions à programmer l'affichage d'une phrase sur un écran Liquid Crystal Display (LCD) de taille 16×2 (voir figure 1.1) à l'aide d'un microcontrôleur *ATMEGA 328P* de chez *Atmel* (voir photo 1.2).

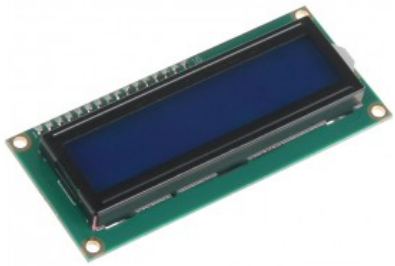

FIGURE 1.1 – Afficheur LCD 16×2

FIGURE 1.2 – Microcontrôleur *ATMEGA 328P*

Ce projet s'inscrit dans la continuité de plusieurs séances de TP ayant permis de prendre en main la programmation d'un microcontrôleur via ses registres.

Ce rapport sera organisé de la manière suivante : dans un 1^{er} temps nous présenterons le cahier des charges du projet, par la suite nous réaliserons une description explicative de celui-ci puis nous aborderons le travail réalisé. Enfin, nous conclurons sur l'aboutissement du projet.

2 Cahier des charges

2.1 Contexte et définition du problème

Ce projet est lié au module de microntrôleurs réalisé dans le cadre du Master 2 Automatique/Robotique parcours Mécatronique. Il est réalisé dans le but de mettre en pratique sur un cas concret les différentes notions étudiées en cours ainsi que tout au long des différents TP.

2.2 Objectif

Pour mener à bien ce projet, nous avions à programmer un afficheur LCD 16×2 via un un microcontrôleur *ATMEGA 328P*. Pour faire cela nous ne devons pas utiliser les fonctions des différentes librairies Arduino, afin d'optimiser l'espace mémoire disponible sur le microcontrôleur.

2.3 Durée & date butoire

Nous avons eu 3 séances de cours (12h30) entre le 17 Octobre 2022 et le 25 Octobre 2022 ainsi que du temps en autonomie jusqu'au 10 Décembre 2022 pour atteindre l'objectif.

2.4 Contraintes

Nous avions à respecter un certain branchement entre l'afficheur LCD et la carte Arduino. Pour optimiser l'espace mémoire, nous devions programmer directement les différents registres du microcontrôleur, plutôt que les fonctions préfaites disponibles dans les librairies Arduino permettant de gérer l'affichage sur un écran LCD.

Etant donné que nous n'avions pas un afficheur LCD par élève, pour pouvoir travailler en autonomie en dehors des séances de cours et pour ne pas perdre de temps à devoir effectuer de nouveau le branchement à chaque séance, nous utilisions un simulateur en ligne disponible sur le site Wokwi.

2.5 Ressources à disposition

Pour programmer nous pouvions utiliser l'IDE Arduino installé sur nos ordinateurs ou bien celui directement disponible sur Wokwi. Nous avions à notre disposions la documentation de l'*ATMEGA 328P* et celle de l'afficheur LCD pour chercher ce dont nous avions besoin.

Nous pouvions également nous aider des codes réalisés lors des TP précédents, de nos notes de cours, de ce que nous trouvions sur internet, et du code source de la librairie Arduino *LiquidCrystal*, qui nous permettait de voir comment sont construites les fonctions de cette dernière, et donc de comprendre comment programmer sans avoir à les utiliser.

2.6 Tableau récapitulatif

Afin de faciliter la référence aux ressources et aux contraintes du projet dans la suite du rapport, un tableau récapitulatif est disponible ci-dessous.

Contexte	Projet lié au module de microcontrôleurs visant à mettre en pratique les différentes notions étudiées
Objectif	Réaliser la programmation d'un afficheur LCD 16×2 via un microcontrôleur *ATMEGA 328P*
Durée	12h30 en salle de cours + temps personnel
Date butoire	10 Décembre 2022
Contraintes	Branchement de l'afficheur LCD sur la carte Arduino Utilisation des registres du microcontrôleur plutôt que les librairies Arduino Utilisation du simulateur du site Wokwi
Ressources à disposition	IDE Arduino Documentation de l'*ATMEGA 328P* Documentation de l'afficheur LCD Programmes réalisés lors des TP précédents Notes de cours Internet

TABLE 2.1 – Cahier des charges du projet

3 Description explicative

3.1 Câblage de l'afficheur LCD sur la carte Arduino

Comme expliqué dans le cahier des charges, nous avons utilisé le logiciel Wokwi afin de simuler notre montage (voir figure 3.1.1).

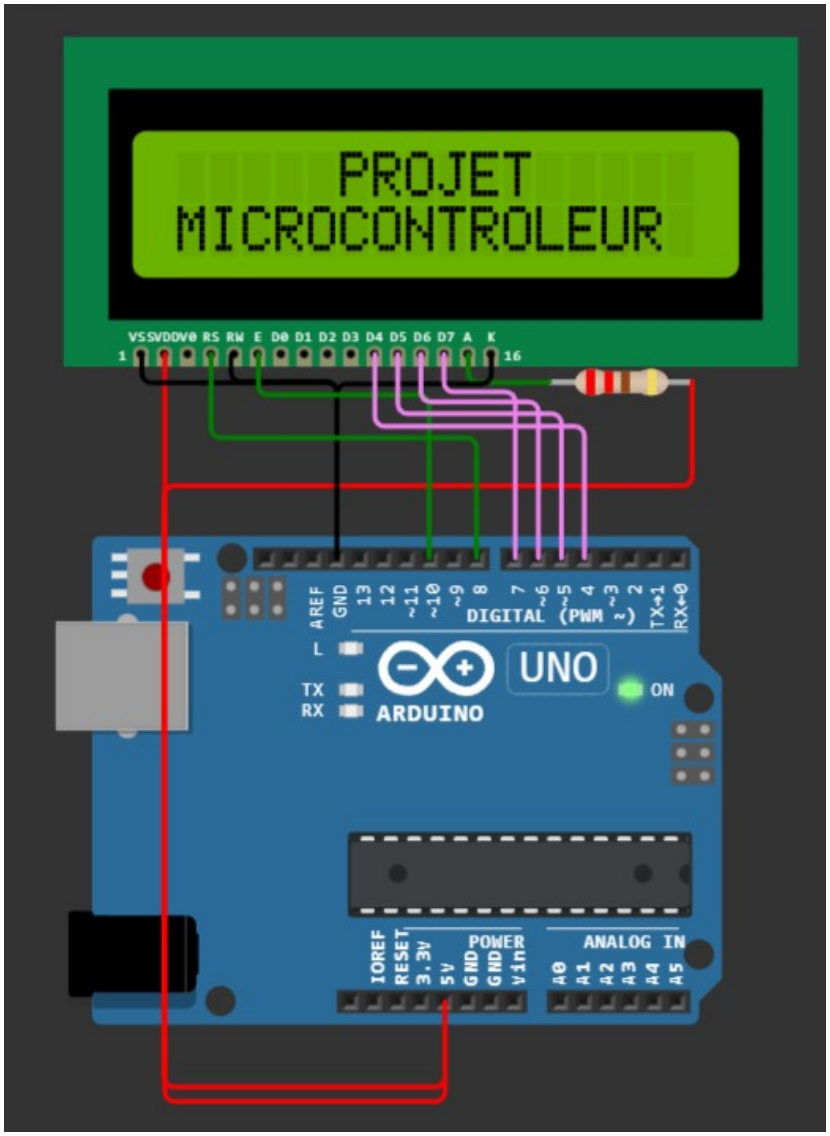

FIGURE 3.1.1 – Simulation du montage

Comme nous pouvons le voir ici, le montage se compose simplement de 3 éléments :
— 1 carte Arduino Uno
— 1 afficheur LCD 16×2
— 1 résistance de 220Ω

On peut également y voir que l'on utiliser les ports D4 à D7, B0 et B2 de la carte Arduino, une masse et une entrée d'alimentation 5V.

Ce montage impacte la partie programmation du projet puisqu'il définit les ports du microcontrôleur avec lesquels nous allons intéragir via notre programme.

Etant donné que l'afficheur utilise un codage American Standard Code for Information Interchange (ASCII) sur 8 bits, nous devons séparer l'envoi d'information en 2 messages de 4 bits.

3.2 Configuration des ports du microcontrôleur

Avant de pouvoir programmer l'envoi de bits, il faut commencer par configurer les broches que nous allons utiliser (voir figure 3.2.1).

```
DDRB |= 0x01;// PB0 en sortie RS: signal pour afficher les DATA
DDRB |= 0x04;// PB2 en sortie E: autorisation d'afficher des DATA

DDRD |= 0x80; // PD 7 en sortie
DDRD |= 0x40; // PD 6 en sortie
DDRD |= 0x20; // PD 5 en sortie
DDRD |= 0x10; // PD 4 en sortie
```

FIGURE 3.2.1 – Définitions des différents ports en sorties

Ici on utilise donc les registre DDRB et DDRD pour définir respectivement les ports B0, B2 et D4 à D7 comme sorties, en initialisant le bit à 1 via un masque.

3.3 Affichage via l'écran LCD

Comme expliqué ci-dessus, pour envoyer les 8 bits correspondants à un caractère ASCII, il faut découper le message en 2 informations de 4 bits chacunes. Afin de déterminer comment procéder, il est donc nécessaire de se référer à la documentation de l'afficheur (voir figure 3.3.1).

● Write mode

Characteristic	Symbol	Min.	Typ.	Max.	Unit	Test pin
E cycle time	t_C	500	---	---	ns	E
E rise time	t_r	---	---	25	ns	E
E fall time	t_f	---	---	25	ns	E
E pulse width (High, Low)	t_W	220	---	---	ns	E
R/W and RS set-up time	t_{SU1}	40	---	---	ns	R/W, RS
R/W and RS hold time	t_{h1}	10	---	---	ns	R/W, RS
Data set-up time	t_{SU2}	60	---	---	ns	$DB_0 \sim DB_7$
Data hold time	t_{h2}	10	---	---	ns	$DB_0 \sim DB_7$

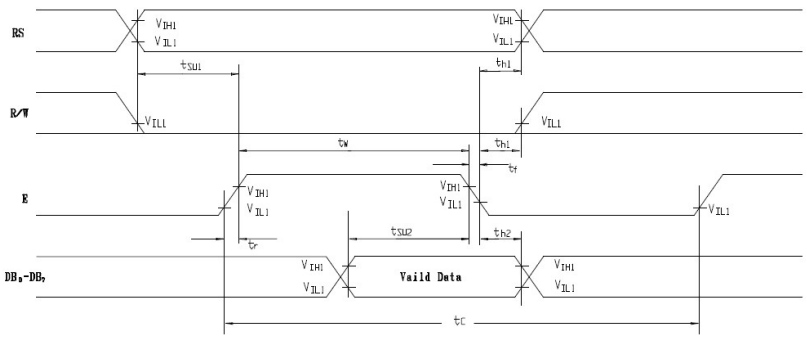

FIGURE 3.3.1 – Spécifications techniques du LCD

Cette documentation nous permet de comprendre sur quels registres il est nécessaire d'agir et quels bits il faut activer pour réussir à faire un affichage.

Ici on peut donc voir qu'il est nécessaire de programmer 7 ports de l'afficheur (comme présenté en figure 3.1.1) :
— RS la lecture et l'écriture de données
— R/W pour la lecture d'adresses ou de données
— E pour autoriser l'affichage de données
— D4 à D7 pour les caractères que l'on souhaite envoyer

Dans le cas présenté ici, le port E de l'afficheur est connecté au port B2 du microcontrôleur. Il faut donc écrire une fonction pour définir que nous voulons autoriser l'affichage des données en alternance (voir figure 3.3.2).

```
void pulse(){
  PORTB &=~ 0x04; // mise a 0 de E: PB2
  PORTB |= 0x04; // mise a 1 de E: PB2
  PORTB &=~ 0x04; // mise a 0 de E: PB2
  time_ms();
}
```

FIGURE 3.3.2 – Fonction de configuration du signal Enable

À partir de là il est possible d'écrire sur 8 bits en écrivant les 4 premiers bits, puis en réalisant un décalage à gauche avant d'écrire les 4 suivants (voir figure 3.3.3).

```
//ECRITURE DU 1ER MOT
void write_word_1(){
  unsigned  char mot[16] = "PROJET";

  for (int i = 0; i<strlen(mot);i++){
  uint8_t value = mot[i];
  PORTB |= 0x01; // mise a 1 de RS: PB0
  PORTD = value; // upper 4 bit ascii

  pulse();
  PORTD = PORTD << 4;// lower 4bit ascii

  time_ms();
  pulse();
  }
}

// ECRITURE DU 2EME MOT
void write_word_2(){
  unsigned  char mot[16] = "MICROCONTROLEUR";

  for (int i = 0; i<strlen(mot);i++){
  uint8_t value = mot[i];
  PORTB |= 0x01; // mise a 1 de RS: PB0
  PORTD = value; // upper 4 bit ascii

  pulse();

  PORTD = PORTD << 4;// lower 4bit ascii

  time_ms();
  pulse();
  }
}
```

FIGURE 3.3.3 – Fonctions d'écriture du 1er et du 2nd mot

Les 2 fonctions utilisées sont codées de manière similaire, la seule différence est que chacune permet d'écrire un mot différent.

On remarque ici que le bit RS est ici définit à 1, car d'après la documentation cela permet d'écrire les caractères que l'on souhaite envoyer (voir figure 3.3.4).

Command	RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0	Execution Time (f_{osc} = 250kHz)	Remark
WRITE DATA	H	L	Write Data								46μs	Write data into DD or CG RAM
READ DATA	H	H	Read Data								46μs	Read data from DD or CG RAM

FIGURE 3.3.4 – Valeur des bits pour l'écriture de données

dans cette fonction on peut noter l'utilisation d'une fonction " time_ms() " (voir figure 3.3.5) qui permet de réaliser une temporisation entre chaque étape ainsi que la fonction " pulse() " qui a été présentée en figure 3.3.2.

```
//FONCTION DE TEMPORISATION
void time_ms(){
  for ( unsigned long int i = 0; i < 10000; i++ ){
    asm("nop"); //nop: instruction assembleur qui ne fait rien
  }
}
```

FIGURE 3.3.5 – Fonction de temporisation

En dernier lieu, il faut réaliser l'initialisation de l'afficheur sans utiliser les librairies Arduino. Cette partie a été réalisée en utilisant les chronogrammes présentés en figure 3.3.1 mais n'a malheureusement pas aboutie. La fonction qui va être présentée ci-dessous n'est pas donc pas entièrement fonctionnelle et ne permet pas de réaliser l'initialisation de l'afficheur sans utiliser les librairies Arduino.

```
//FONCTION D'INITIALISATION
void lcd_begin(){
  for ( unsigned long int i = 0; i < 3000000; i++ ){
    asm("nop"); //nop: instruction assembleur qui ne fait rien
  }
  PORTB &=~ 0x01;
  PORTD &=~ 0xF0;
  PORTD |= 0x30;
  pulse();

  time_ms();
  PORTB &=~ 0x01;
  PORTD &=~ 0xF0;
  PORTD |= 0x30;
  pulse();
```

```
time_ms();
PORTB &=~ 0x01;
PORTD &=~ 0xF0;
PORTD |= 0x30;
pulse();

time_ms();
PORTB &=~ 0x01;
PORTD &=~ 0xF0;
PORTD |= 0x20;
pulse();

time_ms();
PORTB &=~ 0x01;
PORTD &=~ 0xF0;
PORTD |= 0x20;
pulse();

PORTB &=~ 0x01;
PORTD &=~ 0xF0;
PORTD |= 0xC0;
pulse();

time_ms();
PORTB &=~ 0x01;
PORTD &=~ 0xF0;
PORTD |= 0x00;
pulse();
PORTB &=~ 0x01;
PORTD &=~ 0xF0;
PORTD |= 0x80;

time_ms();
pulse();
PORTB &=~ 0x01;
PORTD &=~ 0xF0;
PORTD |= 0x00;
pulse();
```

```
  time_ms();
  pulse();
  PORTB &=~ 0x01;
  PORTD &=~ 0xF0;
  PORTD |= 0x00;
  pulse();
  PORTB &=~ 0x01;
  PORTD &=~ 0xF0;
  PORTD |= 0x02;
  pulse();
}
```

FIGURE 3.3.6 – Tentative de fonction d'initialisation du LCD

4 Travail réalisé

Dans cette section nous allons voir la version du code la plus avancée qui est fonctionnelle, même si toutes les librairies n'ont pas pu être remplacées.

```
// LCD1602 to Arduino Uno connection example

#include <LiquidCrystal.h>

LiquidCrystal lcd(8, 10, 4, 5, 6, 7); // bib init (initialisation)

void pulse(){
  PORTB &=~ 0x04; // mise a 0 de E: PB2
  PORTB |= 0x04; // mise a 1 de E: PB2
  PORTB &=~ 0x04; // mise a 0 de E: PB2
  time_ms();
}

void setup() {

  DDRB |= 0x01;// PB0 en sortie RS: signal pour afficher les DATA
  DDRB |= 0x04;// PB2 en sortie E: autorisation d'afficher des DATA

  DDRD |= 0x80; // PD 7 en sortie
  DDRD |= 0x40; // PD 6 en sortie
  DDRD |= 0x20; // PD 5 en sortie
  DDRD |= 0x10; // PD 4 en sortie
  initialisation();

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

 lcd.begin(16, 2); // bib init (initialisation)
}
```

Une fonction d'initialisation est ici utilisée afin de définir la valeur initiale des bits associés aux ports utilisés, après qu'on les ait définit comme des sorties.

```
void loop() {
  decal_cursor_col();
  write_word_1();
  decal_cursor_row();
  write_word_2();
}
////////////////////////////////////////////////////////////////////////////

void initialisation(){
  PORTB |= 0x01; // mise a 1 de RS: PB0
  PORTB |= 0x04; // mise a 1 de E: PB2
  PORTD = 0xF0;
}

//FONCTION DE TEMPORISATION
void time_ms(){
  for ( unsigned long int i = 0; i < 10000; i++ ){
    asm("nop"); //nop: instruction assembleur qui ne fait rien
  }
}
```

La fonction " loop() " contient ici l'écriture de nos deux mots ainsi que des fonctions permettant de réaliser le décalage de nos caractères sur la 1ère ligne, puis écrire le 2ème mot sur la 2ème ligne de l'afficheur, comme nous l'avons vu sur la figure 3.1.1.

```
//DECALAGE CURSEUR COLONNES
void decal_cursor_col(){
  PORTB &=~ 0x01;
  uint8_t value = 0x85;
  PORTD = value; // upper 4 bit ascii

  pulse();

  PORTD = PORTD << 4;// lower 4bit ascii

  time_ms();
  pulse();

}

//DECALAGE CURSEUR LIGNES
void decal_cursor_row(){
  PORTB &=~ 0x01; //Mise à 0 de RS
  uint8_t value = 0xC0;
  PORTD = value; // upper 4 bit ascii

  pulse();

  PORTD = PORTD << 4;// lower 4bit ascii

  time_ms();
  pulse();

}
```

```
//ECRITURE DU 1ER MOT
void write_word_1(){
  unsigned  char mot[16] = "PROJET";

  for (int i = 0; i<strlen(mot);i++){
  uint8_t value = mot[i];
  PORTB |= 0x01; // mise a 1 de RS: PB0
  PORTD = value; // upper 4 bit ascii

  pulse();
  PORTD = PORTD << 4;// lower 4bit ascii

  time_ms();
  pulse();
  }
}

// ECRITURE DU 2EME MOT
void write_word_2(){
  unsigned  char mot[16] = "MICROCONTROLEUR";

  for (int i = 0; i<strlen(mot);i++){
  uint8_t value = mot[i];
  PORTB |= 0x01; // mise a 1 de RS: PB0
  PORTD = value; // upper 4 bit ascii

  pulse();

  PORTD = PORTD << 4;// lower 4bit ascii

  time_ms();
  pulse();
  }
}
```

5 Conclusion

Finalement l'objectif n'es pas entièrement atteint, nous n'avons pas réussi à implémenter notre message sur l afficheur LCD entièrement grâce à nos fonctions. Malgré tout, même si l'initialisation se fait encore à l'aide de fonctions Arduino, l'écriture de données sur l'afficheur est effective. Pour réaliser cela, il a fallu utiliser les compétences acquises tout au long de l'UE ainsi que les documentations que nous possédions.

Ce projet a donc permis de mettre en pratique sur un cas concret les méthodes vues en cours, et poser des bases pour ouvrir un champ des possibles quand à notre utilisation future de microcontrôleurs.