Hsan Ennouri, Shileen Azarin
Mardi, 28 Mai 2024
Numéro de Groupe : 007

# Projet de Microcontrôleurs

Sommaire :

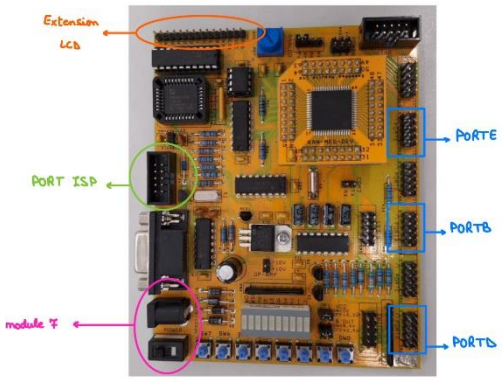

## 1 Introduction :

Ce rapport présente le développement de notre projet réalisé dans le cadre du cours EE-208, Microcontrôleurs et systèmes numériques. Notre projet consiste en un traducteur de code Morse. Ce dernier permet à l'utilisateur de saisir un texte, à l'aide d'un clavier, qui sera affiché sur le LCD puis transformé en code Morse, que nous pouvons ensuite entendre. Un moteur pas à pas tourne tant qu'un des boutons est enfoncé. Ce système facilite la communication à l'aide du code Morse, offrant un outil pratique et éducatif.

## 1.1 Description générale de l'application :

Notre objectif était de concevoir et mettre en œuvre un système simple permettant à l'utilisateur de pouvoir communiquer facilement en code Morse sans devoir connaitre les codes des différents caractères. Nous avons fait en sorte que notre système réponde à certains objectifs spécifiques. Tout d'abord, nous avons intégrés divers périphériques tels que le Keypad, le Buzzer, le LCD et le moteur pas à pas. De plus, nous avons tenus à ce que notre système soit simple d'utilisation et intuitif suivant le modèle des anciens téléphones ou chaque touche permet d'écrire plusieurs caractères (2 caractères par touche dans notre cas) et où il est possible d'effacer des caractères et d'insérer des espaces. Enfin, nous avons accordé une grande importance à la modularité et la structure ainsi qu'à la clarté de notre code afin de faciliter sa compréhension par d'autres utilisateurs qui souhaiteraient y avoir accès.

## 2 Mode d'emploi




### 1. Montage du système

1.Connectez les périphériques comme suit : LCD au port « Extension LCD », moteur pas à pas (module M1) au PORTB, buzzer (module M2) au PORTE et le Keypad sur les pins 0→7 du PORTD comme suit : les entrées 8→5 du keypad sur les pins 0 → 3 respectivement et les entrées 4 → 1 du keypad sur les pins 4 → 7 respectivement.
2.Branchez le programmeur ISP sur un port USB du PC + reliez le programmeur à la carte STK-300 sur le port ISP au moyen du câble gris
3.Connectez l'alimentation du microcontrôleur (et mettre le petit interrupteur sur « ON ») : Module 7
4.Effectuez le téléchargement du code de l'ordinateur vers le MCU. (En utilisant AVRISP-U)

## 2. Utilisation du Système

1. Saisissez un texte sur le clavier.
2. Le KEYPAD est composé de 16 touches. Les touches des trois premières lignes, ainsi que la première touche de la quatrième ligne, sont chacune configurées avec deux lettres de l'alphabet différentes, comme indiqué ci-dessous. Pour taper les premiers caractères des différentes cases du clavier (A par exemple pour la case numéro 0), une brève impulsion sur le bouton correspondant est suffisante alors que pour obtenir le deuxième caractère (B par exemple pour la case numéro 0), une impulsion plus prolongée est nécessaire. Mettre des espaces et effacer des caractères est possible en appuyant sur les boutons correspondants. Tant qu'un des boutons est enfoncés (à l'exception de « ENTER ») le moteur effectue une rotation dans le sens anti-horaire.
3. Saisissez le texte caractère par caractère à l'aide du Keypad comme mentionné ci-dessus
4. Appuyez sur la touche « ENTER » une fois le texte saisi : Cela enclenchera la transformation de la phrase en code Morse avec un silence entre chaque caractère (le curseur du LCD bouge aussi pour indiquer quel caractère est en train d'être « entendu »

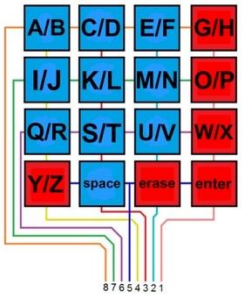



*Figure 1 : Keypad utilisé*

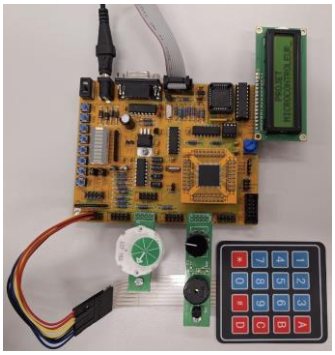


*Figure 2 : Dispositif Branché*

# 3. Fonctionnement du système

## 3.1 Description technique de l'application et du matériel

Dans le cadre de notre projet, quatre périphériques parmi les quatorze disponibles ont été utilisés.

- KEYPAD 4x4

Le premier périphérique, obligatoire, est un keypad 4x4 connecté au microcontrôleur via le PORTD, comme indiqué dans le mode d'emploi. Ce keypad envoie des interruptions lorsque différentes touches sont pressées.
Chaque touche du keypad est configurée pour engendrer une action particulière. Les 13 premières permettent d'écrire et de stocker les 26 différentes lettres de l'alphabet selon si la touche correspondante est enfoncée brièvement (premier caractère de la touche) ou longuement (deuxième caractère de la touche). Les 3 dernières représentent les touches « SPACE », « ERASE » et « ENTER » permettant, respectivement, d'écrire un espace, d'effacer le dernier caractère et de démarrer le mode Morse

- LCD 2x16

Le deuxième périphérique utilisé est l'affichage LCD 2x16 (Hitachi 44780U 2x16 LCD). Les affichages LCD 2x16 caractères sont standards sur toutes sortes de terminaux et permettent de transmettre des informations simples ou des instructions aux utilisateurs sous forme de textes et de messages. Ce périphérique est connecté au microcontrôleur via port « Extension LCD », comme indiqué dans le mode d'emploi.
Sur le LCD, chaque caractère saisi sur le keypad est affiché. Dans le code, les conditions suivantes sont vérifiées à plusieurs endroits : si l'on est à la fin de la première ligne du LCD (dans ce cas, il

faut revenir au début de la deuxième ligne, si l'on est à la fin de la deuxième ligne (dans ce cas, il ne faut rien faire si on cherche à écrire), si l'on est au début de la deuxième ligne et qu'on cherche à effacer un caractère (dans ce cas, il faut revenir au dernier caractère de la première ligne avant d'effacer)
Le LCD prend donc en charge l'affichage des informations à l'utilisateur, assurant une communication claire et efficace.

- Buzzer (M2)

Le troisième périphérique que nous utilisons est le buzzer, qui est utilisé pour produire du son. Le buzzer est connecté au microcontrôleur sur le PORT E comme indiqué dans le mode d'emploi. Dans notre code, la phrase entrée via le keypad est traduite en Morse caractère par caractère à l'aide d'un look up table contenant la séquence des durées des Bips à produire pour chacun des caractères.
Ce dernier, offre donc une rétroaction auditive à l'utilisateur

- Moteur pas à pas (Stepper motor x27)

Et finalement le quatrième périphérique que nous utilisons est le moteur, connecté au microcontrôleur sur le PORT B comme indiqué dans le mode d'emploi. Le moteur se met en marche chaque fois qu'un caractère est pressé sur le keypad et continue de tourner jusqu'à ce que le bouton pressé soit relâché. Cela permet d'indiquer à l'utilisateur si la touche sur laquelle il appuie est détectée par le système évitant ainsi la confusion dans le cas d'un clavier qui dysfonctionne.

→ Ensemble, ces modules forment un système cohérent et fonctionnel, offrant une interaction fluide et intuitive avec l'utilisateur.

## 3.2 Rôle des registres :

- **wr0 :** Détecter la ligne + stocker l'offset pour accéder au caractère correspondant de la table de transcorrespondance (look up table)
- **wr1 :** Détecter la colonne
- **mask :** Stocker un "mask" pour pouvoir identifier la colonne
- **Counter_LCD_Morse :** Identifier la position du curseur du LCD quand on est en mode lecture + compter le nombre de caractère(il s'incrémente à chaque fois) déjà converti en morse quand on est en mode morse (donne l'offset permettant d'accéder différents caractères stockés dans l'espace SRAM alloué)
- **Counter** : Compteur qui identifie l'offset de la SRAM, combien de caractères on a déjà stocké dans le mode écriture + en mode morse

## 3.3 Définition des constantes :

- **KPDD :** DDRD
- **KPDO** : PORTD
- **KPDI** : PIND
- **KPD_DELAY :** 30 ms →délais nécessaires au fonctionnement du code du Keypad
- **DOT :** 100 ms → utilisé pour la représentation de caractères en code morse
- **DASH :** 500 ms →utilisé pour la représentation de caractères en code morse
- **t1 :** 1000us → délais nécessaire au fonctionnement de la Macro MOTOR
- **port_mot :** PORTB

Remarque : comme ça sera expliqué dans la section suivante, quand l'utilisateur cherche à écrire un espace, ou efface un caractère, le caractère qui sera stocké dans l'espace SRAM dédié est « [ » et non «  ». Cela vient du fait qu'en code ASCII, le caractère « [ » suit directement le caractère « Z » et cela permet, en mode morse d'obtenir un offset de 26 et d'aller vers la ligne de morse_tb correspondante et d'évaluer les conditions avec cette valeur d'offset (on teste si égale à 26 pour reconnaitre que c'est un espace)

## 3.4 Fonctionnement du programme (top-down)

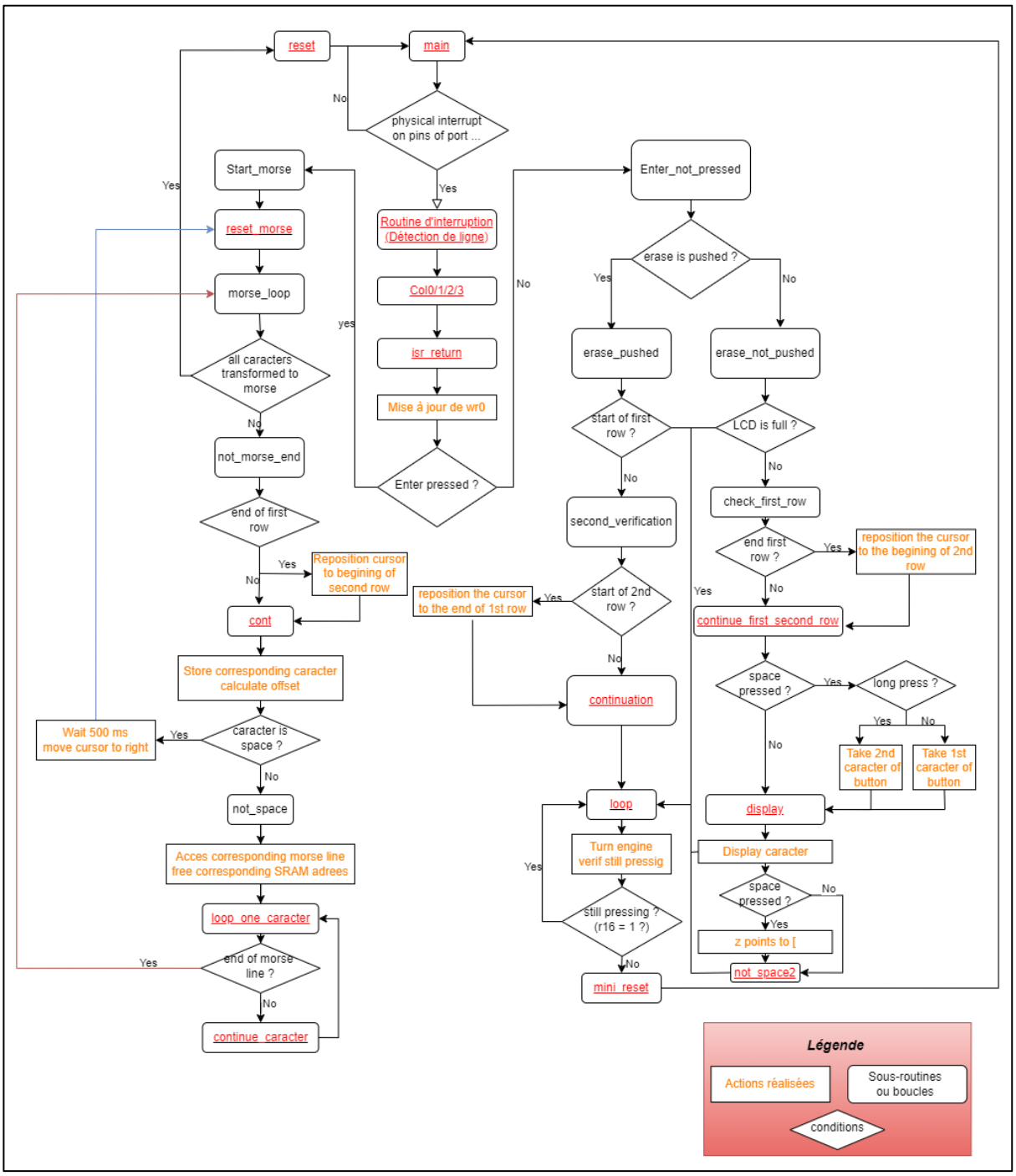



*Figure 3 : Structure du programme et branchements du code*

La figure ci-dessus (figure 3) indique la structure générale de notre programme et la manière dont s'effectuent les différents branchements du code. Vous trouverez ci-dessous une explication approfondie de ce qui se passe dans les boucles et sous-routines qui sont soulignés et mises en rouge dans la figure 3

* *Les boucles/sous-routines présentes dans la figure 3 :*

**-Routines d'interruptions des lignes : isr_ext_int0/ isr_ext_int1/ isr_ext_int2/ isr_ext_int3 :** détecte la ligne sur laquelle se situe l'interruption / met la valeur de wr0 (le « numéro » de la ligne) qui permettra de calculer le numéro de la touche enfoncée / met la valeur du registre mask
**-col0/col1/col2/col3 :** détecte la colonne sur laquelle la touche enfoncée se situe / met la valeur de wr1 (le « numéro » de la colonne) qui permettra de calculer le numéro de la touche enfoncée
**-isr_return :** calcule wr0 = wr0+wr1 : numéro de la touche(voir figure 4) puis wr0 = wr0*2 : offset correspondant au premier caractère de la touche enfoncée
**-Continuation :** déplacer le curseur à gauche / accèder au caractère «  » dans le look up table « lut » avec le curseur z et l'affiche sur le LCD (écrase l'ancienne valeur) / stocker le caractère « [ » dans la SRAM à l'emplacement de l'ancien caractère avec le pointeur z (tout cela se fait en mettant à jour les compteurs counter et counter_LCD_morse) / déplacer le curseur à gauche (car il est auto-incrémentée quand on a affiché l'espace)
**-continue_first_second_row :** si ce n'est pas espace qu'on cherche à afficher : attendre 500 ms / vérifier si on appuie toujours : si c'est le cas r16 = 1 (pour sélectionner le deuxième caractère de la touche) sinon r16 = 0 / wr0=wr0+r16 (permet d'avoir wr0 + 1 si on appuie longuement)
**-display** : z = z+wr0 : z pointe vers la case mémoire du caractère correspondant dans lut / affichage du caractère sur le LCD / incrémentation de counter_LCD_morse / si on appuie sur espace : incrémenter z (pour qu'il pointe sur [ ) / mettre le caractère ou z pointe dans r0
**-not_space2 :** pointer à l'adresse correspondante dans la SRAM / stocker le caractère qui est dans r0 dans la SRAM / incrémenter « counter »
**-loop :** tourner le moteur / attendre 250 ms / vérifier si on appuie toujours : si c'est le cas r16 = 1 sinon r16 = 0
**-cont :** initialiser z sur la case correspondante du caractère à récupérer (de la lut« entries ») / calculer l'offset du caractère par rapport à « A » et le met dans r17 / r17=r17*8 (chaque ligne morse d'un caractère est stockée sur 8 octets) accéder au début de la ligne correspondante dans le look up table morse_tb
**-loop_one_caracter :** récupérer le contenu de la case mémoire pointée / incrémenter z / vérifier si c'est la fin du code morse du caractère donnée (on pointe sur un 0) / si c'est le cas, on bouge le curseur à droite + on attend 300ms + on incrémente counter_LCD_morse
**-continue_caracter :** produire le son correspondant (DOT ou DASH) en mettant dans b0 la durée du son / attendre 200ms
**-Main :** Boucle infinie
**-mini-reset :** met les paramètres initiaux du portD permettant la détection des interruptions sur le PORTD / met les registre wr0, r16 et wr1 à 0
**-reset :** met les paramètres initiaux du portD permettant la détection des interruptions sur le PORTD / active le PORTE (buzzer) et PORTB (moteur) en sortie / initialise le LCD et load le stack pointer met tous les registres utilisés à 0 / active le curseur du LCD / active les interruptions
**-reset_morse :** remet le curseur du LCD à la case 0 / met les registre r16, r17 et counter_LCD_morse à 0

* *Les boucles/sous-routines non présentes dans la figure 3 mais utilisées implicitement :*

**-timer_250MS :** permet de produire un temps d'attente de 250 ms en utilisant le timer0 avec un prescaler de 3 et en utilisant le quartz horloger. Elle configure le timer, met r22 à 0 et active les interruptions puis rentre dans la boucle loop_2
**-loop_2 :** le programme loop tant que r22 = 0 (r22 est mis à 0 à l'overflow du timer dans timer0_overfl)
**-verif_row :** met les entrées des colonnes du keypad à 0 / stockes la sortie du PORTD / compare cette sortie au mask (qui a été initialisé par la valeur correspondante à la ligne détectée) / si cette même ligne est toujours enfoncée alors on est toujours en train d'appuyer sur la touche (dans ce cas on met r16 à 1 sinon on le met à 0)

**-turn_engine :** tourne le moteur en utilisant la macro MOTOR

## 3.5 Allocations mémoire :

Une allocation mémoire consiste à réserver de l'espace dans la mémoire afin d'y stocker des données. Dans notre projet, nous avons utilisé deux types d'allocations mémoire :

- Allocation de mémoire dans la SRAM à l'aide de la directive **.byte** : dans les ligne spécifiées , **'.dseg' → '.org 0x100' → 'entries: .byte 32'**, nous réservons 32 octets de la mémoire SRAM, (débutant à l'adresse mémoire 0x100 et se terminant à l'adresse 0x11F. Cela servira à stocker les caractères à afficher.)
- Allocation de mémoire dans la mémoire Flash à l'aide de la directive **.db** : dans les ligne spécifiées, '**.cseg' → 'lut : .db 'A','B' … ' → 'morse_tb: .db DOT,DASH, …'** nous définissons deux look up tables, respectivement de 28 et 27*8 = 216 octets et servant à stocker, respectivement, les caractères et les séquences sonores (durées des bips) de chaque caractère, dans l'ordre alphabétique

## 3.6 Interruptions:

Dans notre programme, nous avons deux types d'interruptions :

1. Interruptions physiques des pins 0 → 3 du PORTD : ces interruptions surviennent quand nous sommes en mode écriture et donc quand l'utilisateur cherche à écrire sa phrase. Elles surviennent quand l'utilisateur appuie sur une des touches du Keypad. L'interruption se fait sur un des 4 pins mentionnés ci-dessus et force le système à la routine d'interruption correspondante à la ligne qui a été détectée (après être passé par la table des vecteurs d'interruptions)
2. Interruption du timer0 du microcontrôleur : Cette interruption nous sert dans la sous-routine **timer_250MS** qui nous permet de générer un délai de 250 ms. Cela est utilisé dans la boucle **loop** qui reboucle tant que la touche est enfoncée (on vérifie cela toutes les 250 ms) ainsi que dans **continue_first_second_row** quand le système vérifie si on veut afficher le premier ou le second caractère de la case correspondante (si on reste appuyé longuement ou pas). Nous utilisons le timer0 dont l'horloge est le quartz horloger à 32'768 Hz et configuré avec un prescaler de 3 permettant un overflow toutes les 250 ms

**Rq :** Ces interruptions sont désactivées dans le code quand le système rentre en mode morse et réactivées après pour ne pas interrompre la transformation du code en morse.

## 3.7 Accès aux périphériques

### LCD :

L'affichage sur l'écran LCD se fait grâce aux méthodes du fichier lcd.asm fourni pour le cours. Plus précisément, nous avons utilisé les sous-routines suivantes :

| Subroutine | Explanation |
|---|---|
| LCD_putc | Writes value in a0 to current cursor position, and interprets CR |
| LCD_home | Cursor to home position |
| LCD_cursor_[left/right] | Move cursor to the left/right |
| LCD_cursor_on | Turns cursor on |
| LCD_init | Initializes the LCD module |
| LCD_pos | Places the cursor at position given in a0 |

### Keypad:

La carte 4x4 keypad fonctionne comme une matrice d'interconnexion pour détecter quelle touche parmi les seize a été activée. La matrice est accessible via les colonnes X1 à X4 en mode entrée et les lignes Y1 à Y4 en mode sortie. La détection de la touche se fait en trois phases. D'abord, toutes les

colonnes sont mises à GND et toutes les lignes à VDD, permettant de détecter la ligne de la touche activée (par le lancement d'une interruption du pin de la ligne correspondante). Ensuite, chaque colonne est mise successivement à GND pour identifier la colonne activée. Par exemple, si X3 et Y2 sont détectées, cela signifie que la touche à l'intersection de X3 et Y2 est pressée. Un délai de 10-30 ms est nécessaire pour le debouncing. Enfin, la combinaison colonne-ligne détectée est décodée en utilisant une LUT pour identifier le caractère à afficher.

La figure suivante nous montre les numéros des différentes cases du keypad (partie bleu clair) permettant de les identifier (calculés en faisant wr0 = wr0+wr1)
L'offset du caractère à afficher est calculé en multipliant le numéro de la case par 2(faisant wr0=wr0*2) (car chaque case contient deux caractères) puis d'éventuellement lui ajouter 1 dans le cadre d'une pression prolongée de la case (pour sélectionner le second caractère

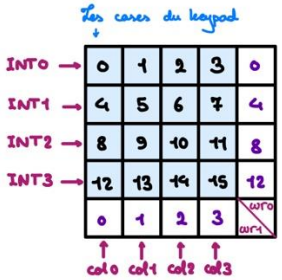



***Figure 4 : Numéros des cases du Keypad (et les wr0/wr1 permettant de l'obtenir)***

## Buzzer :

Pour entendre le son du Buzzer, on utilise la sous-routine **sound** qui se trouve dans le fichier **sound.asm**.

La sous-routine "sound" enregistre la période d'oscillation et la durée du son. Elle utilise les valeurs stockées dans les registres pour contrôler la génération du son. Si la période d'oscillation est égale à zéro, le son est désactivé.
Dans notre code, nous appelons notre code en mettant dans le registre b0 (longueur du son) la valeur de DOT ou de DASH récupéré dans le lut **morse_tb**

```
sound:
; in    a0  period of oscillation (in 10us)
;   b0  duration of sound (in 2.5ms)

    mov b1,b0           ; duration high byte = b
    clr b0              ; duration  low byte = 0
    clr a1              ; period high byte = a
    tst a0
    breq    sound_off   ; if a0=0 then no sound
```

*Figure 5 : sous-routine sound*

## Moteur :

Pour activer le moteur, nous avons utilisé la sous-routine **turn_engine** qui utilise 6 fois, avec des arguments précis, la macro **MOTOR**, qui effectue les actions suivantes :
Cette macro "MOTOR" charge une valeur donnée dans le registre w, puis l'envoie sur le port de contrôle du moteur. Ensuite, il attend un délai spécifié par WAIT_US.

```
.macro  MOTOR
    ldi     w,@0
    out     port_mot,w          ; output motor pin pattern
    WAIT_US t1          ; wait period
.endmacro
```

*Figure 6 : Macro MOTOR*

```
turn_engine :
    MOTOR   0b0101
    MOTOR   0b0001
    MOTOR   0b1011
    MOTOR   0b1010
    MOTOR   0b1110
    MOTOR   0b0100
    ret
```

*Figure 7 : sous-routine turn_engine*

# Références


- Livre du cours – "Microcontrôleurs, Théorie et pratique de l'AVR'', Alexandre Schmid, Raphael Holzer, EPFL Press, 2022
- cours EE-208, "Microcontrôleurs et systèmes numériques", Alexandre Schmid


# Annexe

## Totalité du code source de l'application :