

BE Robot - Projet BOB

réalisé par Taha Bouzid



Contents

1	Introduction	3
2	Partie Hardware	3
2.1	Étude de l'unité centrale	3
2.2	réalisation du decodeur sous quartus	7
3	Tests elementaires	10
3.1	Afficheur 7 segments	10
3.2	Les Leds	11
3.3	Les switchs	12
3.4	Joystick	13
3.5	moteurs	16
3.6	capteurs distance	17
3.7	capteurs de lignes	17
4	partie Software	19
5	Conclusion	26

1 Introduction

L'objectif de ce projet consiste à concevoir un robot (BOB) doté d'une autonomie de pilotage. Le projet est divisé en deux parties principales :

- La première se concentre sur l'aspect matériel du robot, mettant en œuvre le processeur et son environnement (capteurs, actionneurs, etc.).
- La seconde tourne autour de l'aspect logiciel, impliquant le développement de programmes en langage C pour plusieurs modes de déplacement du robot, chacun présentant des niveaux de difficulté croissants.

2 Partie Hardware

Notre robot, BOB, est constitué d'une unité centrale et de différents capteurs et moteurs.

2.1 Étude de l'unité centrale

Le robot est équipé d'une carte DE1-SoC comprenant :

- FPGA Cyclone V
- Afficheurs 7 segments
- Switches
- LEDs

Nous allons configurer un processeur de type NIOS (32 bits) sur la FPGA, permettant la communication avec les différents périphériques assurant le bon fonctionnement du robot. Ces périphériques incluent les 6 afficheurs 7 segments, les 10 switches, les 10 LEDs, les capteurs de ligne et de distance ultrason, ainsi que le joystick.

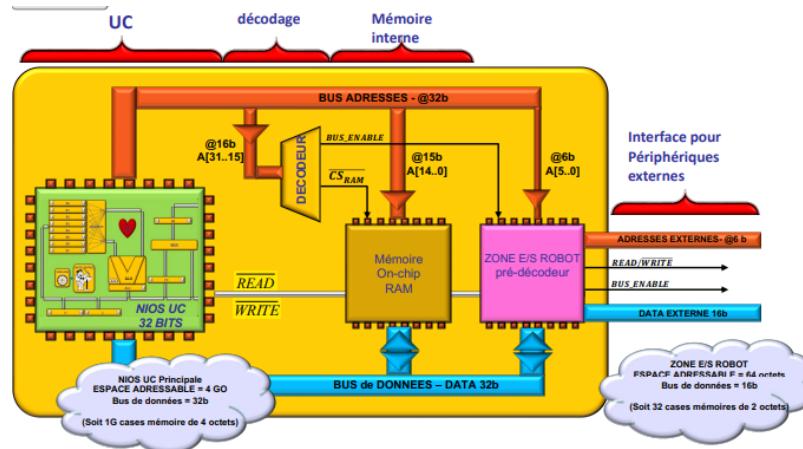
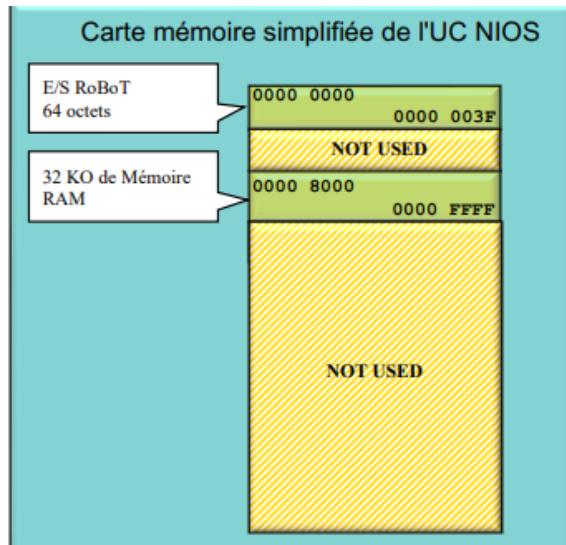


Figure 1: Architecture générique d'une UC et ses périphériques

La communication avec les différents périphériques E/S et la mémoire s'effectue à travers des bus d'adresses et des décodeurs. Occupons-nous dans un premier temps de réaliser ces décodeurs. Pour cela, il faudra d'abord comprendre l'architecture de la mémoire.



On peut alors déduire les débuts et fins des zones mémoire en binaire pour ensuite nous permettre de réaliser un décodeur.

Address	A31	...	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2
0x000000	0	...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x000003F	0	...	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
0x0008000	0	...	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0000FFF	0	...	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

on en déduit alors les équations de décodage Pour l'expression $CS_{ram} = \overline{A31} \times \overline{A30} \times \dots \times \overline{A16} \times \overline{A15}$, on peut simplifier en utilisant la loi de De Morgan pour $\overline{CS_{ram}}$:

$$\overline{CS_{ram}} = A31 + A30 + \dots + A16 + \overline{A15}$$

Ensuite, pour l'expression $BUS_{Enable} = \overline{A31} \times \dots \times \overline{A6}$, on peut déduire CS_{robot} complément en utilisant la complémentation de chaque terme:

$$\overline{BUS_{Enable}} = A31 + \dots + A6$$

ensuite pour mieux gerer les péripherique entrées sortie nous allons réaliser un sous codages pour les 64 octets dédié au robot en la decoupant en 8 sous partie de 8 octets.

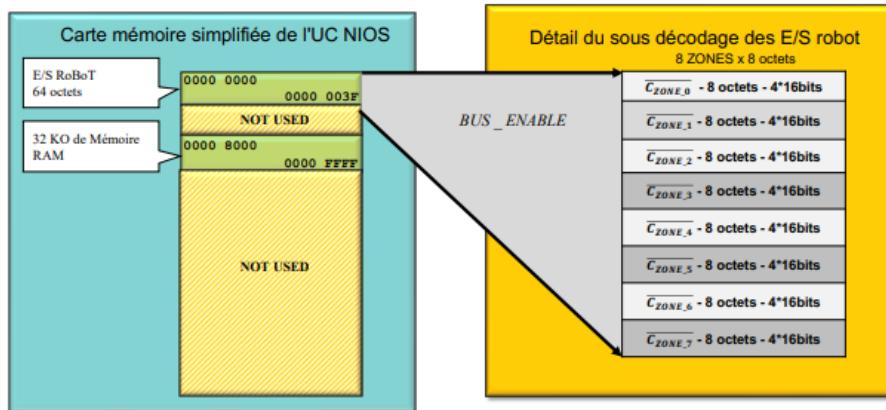


Figure 2: organisation de la memoire

chacune des 8 sous zones de la partie mémoire consacrée au robot est dédiée à l'un des périphériques entrée sortie :

C_{S0} - 4*16bits	Pérophérique d'affichage pour pilotage des afficheurs 7 segments
C_{S1} - 4*16bits	Pérophérique d'affichage pour pilotage des leds
C_{S2} - 4*16bits	Pérophérique d'affichage pour pilotage des moteurs
C_{S3} - 4*16bits	Pérophérique de gestion des capteurs et du joystick
C_{S4} - 4*16bits	Pérophérique de contrôle des switches
C_{S5} - 4*16bits	Pérophérique de mesure distance obstacle centre
C_{S6} - 4*16bits	Pérophérique de mesure distance obstacle gauche
C_{S7} - 4*16bits	Pérophérique de mesure distance obstacle droit

Figure 3: découpage des zones mémoire

Le découpage donné par les figures 2 et 3 conduit aux adresses de début et de fin de chaque zone données par:

Pour $\overline{C_{zone0}}$: Début :0x00000000, Fin :0x00000007 Binaire :0000 0000 à 0000 0111
 Pour $\overline{C_{zone1}}$: Début :0x00000008, Fin :0x0000000F Binaire :0000 1000 à 0000 1111
 Pour $\overline{C_{zone2}}$: Début :0x00000010, Fin :0x00000017 Binaire :0001 0000 à 0001 0111
 Pour $\overline{C_{zone3}}$: Début :0x00000018, Fin :0x0000001F Binaire :0001 1000 à 0001 1111
 Pour $\overline{C_{zone4}}$: Début :0x00000020, Fin :0x00000027 Binaire :0010 0000 à 0010 0111
 Pour $\overline{C_{zone5}}$: Début :0x00000028, Fin :0x0000002F Binaire :0010 1000 à 0010 1111
 Pour $\overline{C_{zone6}}$: Début :0x00000030, Fin :0x00000037 Binaire :0011 0000 à 0011 0111
 Pour $\overline{C_{zone7}}$: Début :0x00000038, Fin :0x0000003F Binaire :0011 1000 à 0011 1111

Alors, en écrivant ces adresses sur un tableau, on pourra facilement regarder que les bits A5, A4 et A3 sont les seuls qui caractérisent les différentes zones. En utilisant ces bits d'indexation, on peut en déduire les équations suivantes pour les signaux de sélection:

$$\begin{aligned} CS_{zone0} &= \overline{A3} \cdot \overline{A4} \cdot \overline{A5} \\ CS_{zone1} &= A3 \cdot \overline{A4} \cdot \overline{A5} \\ CS_{zone2} &= \overline{A3} \cdot A4 \cdot \overline{A5} \\ CS_{zone3} &= A3 \cdot A4 \cdot A5 \\ CS_{zone4} &= \overline{A3} \cdot \overline{A4} \cdot A5 \\ CS_{zone5} &= \overline{A4} \cdot A3 \cdot A5 \\ CS_{zone6} &= A5 \cdot A4 \cdot \overline{A3} \\ CS_{zone7} &= A3 \cdot A4 \cdot A5 \end{aligned}$$

D'où, par passage au complémentaire:

$$\begin{aligned} \overline{CS_{zone0}} &= A3 + A4 + A5 \\ \overline{CS_{zone1}} &= \overline{A3} + A4 + A5 \\ \overline{CS_{zone2}} &= A3 + \overline{A4} + A5 \\ \overline{CS_{zone3}} &= \overline{A3} + \overline{A4} + A5 \\ \overline{CS_{zone4}} &= A3 + A4 + \overline{A5} \\ \overline{CS_{zone5}} &= A4 + \overline{A3} + \overline{A5} \\ \overline{CS_{zone6}} &= \overline{A5} + A4 + A3 \\ \overline{CS_{zone7}} &= \overline{A3} + \overline{A4} + \overline{A5} \end{aligned}$$

Alors, le signal de sélection de ces zones à partir du bus original de 32 bits n'est autre que BUS-enable multiplié par chacun de ces signaux. Autrement dit, pour i allant de 0 à 7, on a :

$$Pour i allant de 0 à 7, \quad C_{si} = BUS-enable \cdot CS_{zonei}$$

ou encore :

$$\overline{C_{si}} = \overline{BUS-enable} + \overline{CS_{zonei}}$$

2.2 réalisation du decodeur sous quartus

réalisons alors le decodeur sous quartus:

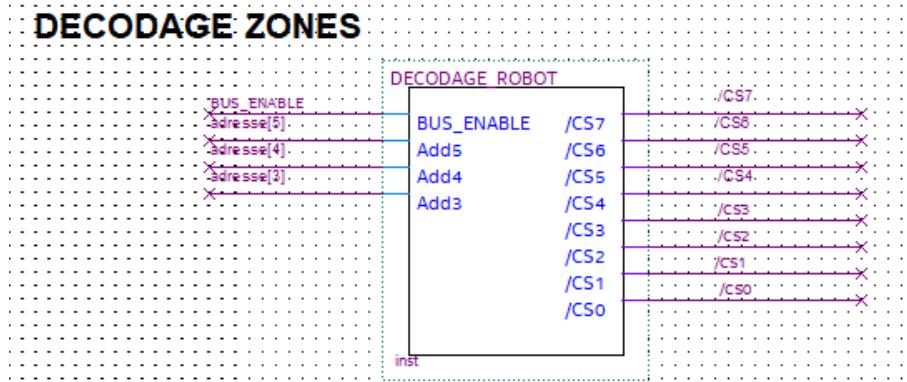


Figure 4: block decodeur dans quartus

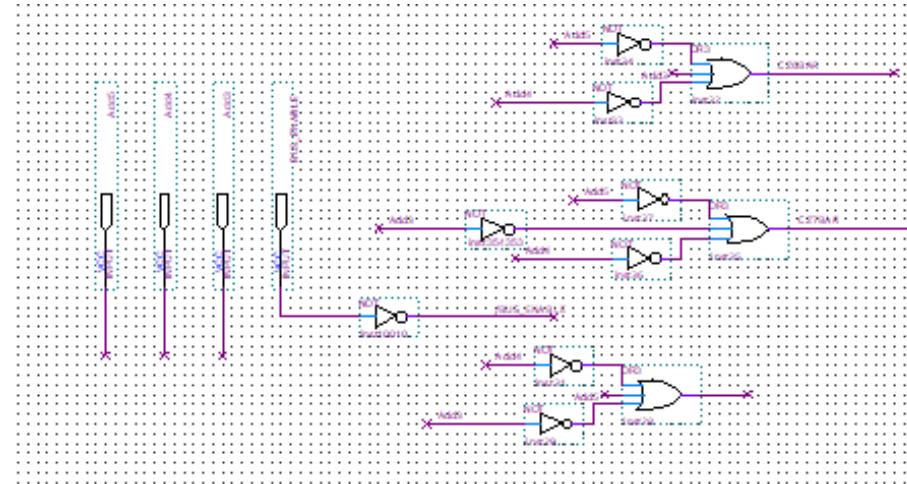


Figure 5: réalisation du decodeur

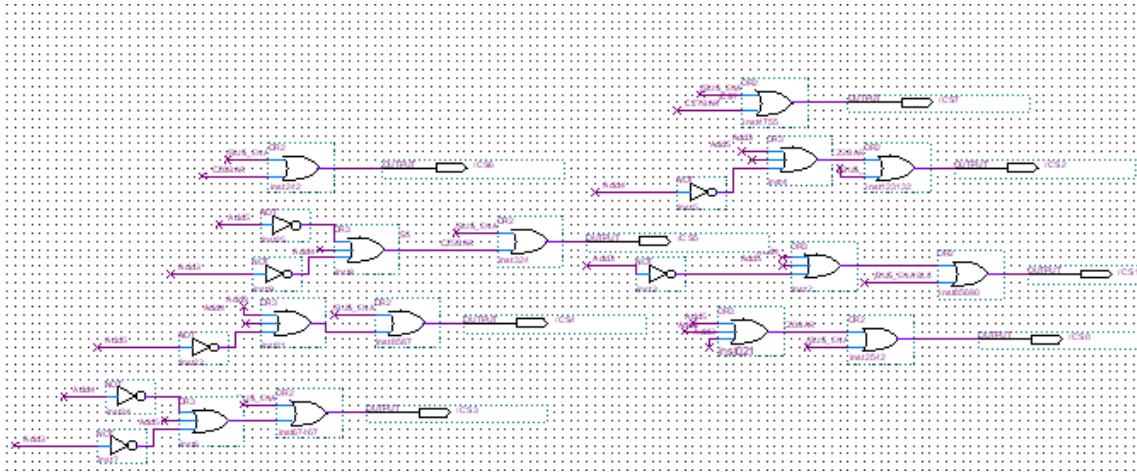


Figure 6: réalisation du decodeur (suite)

pour vérifier que ces signaux sont bien opérationnels nous allons utiliser L'environnement Logiciel ECLIPSE. Après avoir relié les différents signaux de décodage aux pins disponibles sur le robot et après initialisation du projet sur eclipse il nous faut écrire un programme simple d'accès aux différentes zones mémoire puis de vérifier à l'aide de l'oscilloscope que les signaux de décodage sont correctement réalisés.

```
// ZONE 0 - /CS0
#define CS_0 ((volatile short*) (TO_EXTERNAL_BUS_BRIDGE_0_BASE))
// ZONE 1 - /CS1
#define CS_0 ((volatile short*) (TO_EXTERNAL_BUS_BRIDGE_0_BASE+8))
// ZONE 2 - /CS2
#define CS_0 ((volatile short*) (TO_EXTERNAL_BUS_BRIDGE_0_BASE+16))
// ZONE 3 - /CS3
#define CS_0 ((volatile short*) (TO_EXTERNAL_BUS_BRIDGE_0_BASE+24))
// ZONE 4 - /CS4
#define CS_0 ((volatile short*) (TO_EXTERNAL_BUS_BRIDGE_0_BASE+32))
// ZONE 5 - /CS5
#define CS_0 ((volatile short*) (TO_EXTERNAL_BUS_BRIDGE_0_BASE+40))
// ZONE 6 - /CS6
#define CS_0 ((volatile short*) (TO_EXTERNAL_BUS_BRIDGE_0_BASE+48))
// ZONE 7 - /CS7
#define CS_0 ((volatile short*) (TO_EXTERNAL_BUS_BRIDGE_0_BASE+56))
```

Figure 7: définition des adresses des 8 zones

```

short var;

while (1) {
    *CS_0 = 85; // Ecriture 2 premiers octets ZONE0
    *CS_1 = 85; // Ecriture 2 premiers octets ZONE1
    *CS_2 = 85; // Ecriture 2 premiers octets ZONE2
    *CS_3 = 85; // Ecriture 2 premiers octets ZONE3
    *CS_4 = 85; // Ecriture 2 premiers octets ZONE4
    *CS_5 = 85; // Ecriture 2 premiers octets ZONE5
    *CS_6 = 85; // Ecriture 2 premiers octets ZONE6
    var=*CS_7 ; // Lecture 2 premiers octets ZONE7
}

```

Figure 8: boucle infinie d'écriture sur les différentes zones

ensuite on Visualise les 8 signaux de sélection ainsi que les signaux R/W et BUS-ENABLE en prenant comme référence toujours le premier signal CS-0 en branchant l'oscilloscope pour visualiser la tension entre chaque pin et le pin de cs-0 on obtient alors l'allure suivante:

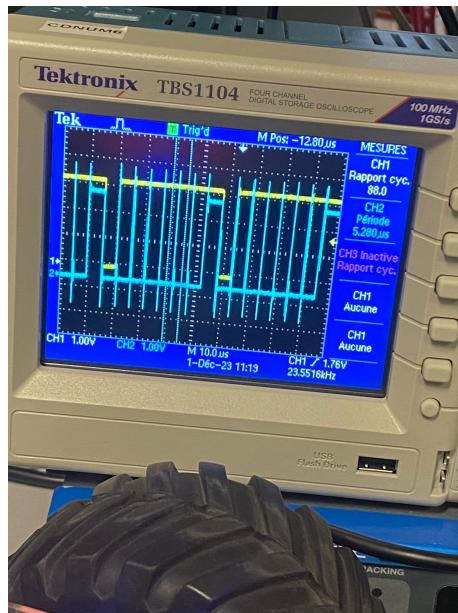


Figure 9: visualisation du signal $CS_4 - CS_0$

On conclut alors que le temps de cycle d'une opération de lecture ou écriture sur la zone robot est de $5 \mu\text{s}$.

3 Tests elementaires

dans cette partie on va tester le fonctionnement elementaire de chacun des périphériques

3.1 Afficheur 7 segments

Comme mentionné précédemment, les afficheurs 7 segments utilisent la zone mémoire CS0 et leur adressage interne est donné par la Figure 10.

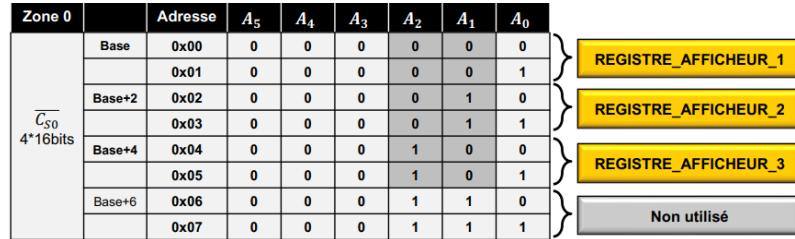


Figure 10: Situation de A7S par rapport au mapping

Les adresses sont définies dans le fichier ‘robot-def.h’.

```
//CS0
// Adressage des afficheurs
#define AFFICHEUR_1 ((volatile short*) (TO_EXTERNAL_BUS_BRIDGE_0_BASE))
#define AFFICHEUR_2 ((volatile short*) (TO_EXTERNAL_BUS_BRIDGE_0_BASE + 2))
#define AFFICHEUR_3 ((volatile short*) (TO_EXTERNAL_BUS_BRIDGE_0_BASE + 4))
```

Figure 11: définition des adresses des afficheurs

On réalise ensuite les trois sous-programmes nommés ‘PILOTAGE-AFFICHEUR-1’, ‘2’ et ‘3’, qui prennent par argument un nombre et l'affichent dans l'A7S correspondant s'il est inférieur à 100 ou affiche ”EE” sinon, permettant le pilotage des périphériques.

```
void PILOTAGE_AFFICHEUR_1 (short valeur){
|   *AFFICHEUR_1=valeur;
};
void PILOTAGE_AFFICHEUR_2 (short valeur){
|   *AFFICHEUR_2=valeur;
};

void PILOTAGE_AFFICHEUR_3 (short valeur){
|   *AFFICHEUR_3=valeur;
};
```

Figure 12: fonctions de pilotage des afficheurs

3.2 Les Leds

de la même manière la situation par rapport au mapping des LEDs est donnée par la figure 11 on

Zone 1	Adresse	A_5	A_4	A_3	A_2	A_1	A_0	
C_{S1} 4*16bits	Base+8 0x08	0	0	1	0	0	0	REGISTRE_LEDs_ROUGES
	Base+9 0x09	0	0	1	0	0	1	Non utilisé
	Base+10 0x0A	0	0	1	0	1	0	Non utilisé
	Base+11 0x0B	0	0	1	0	1	1	Non utilisé
	Base+12 0x0C	0	0	1	1	0	0	Non utilisé
	Base+13 0x0D	0	0	1	1	0	1	Non utilisé
	Base+14 0x0E	0	0	1	1	1	0	Non utilisé
	Base+15 0x0F	0	0	1	1	1	1	Non utilisé

Figure 13: situation des LEDS par rapport au mapping

definit alors les adresses dans le fichier robot-def.h est on teste le fonctionnement des leds

```
//CS1
// Adressage des LEDS
#define LED_ROUGES ((volatile short*) (TO_EXTERNAL_BUS_BRIDGE_0_BASE + 8))
```

Figure 14: définition des adresses des led

puis on réalise un sous-programme qui prend comme argument un nombre binaire sur 10 bits et allume les led ou les éteint suivant la valeurs de chaque bits : si le bit du poids faible est 0 alors la led0 est éteinte

```
void PILOTAGE_LEDs(short valeur){
    *LED_ROUGES=valeur;
};
```

Figure 15: fonction pilotage LEDS

3.3 Les switchs

la situation par rapport au mapping des switchs est donnée par la figure 16

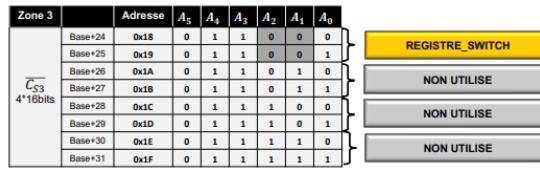


Figure 16: situation des switchs par rapport au mapping

on definit alors les adresses dans le fichier robot-def.h

puis on realise un sous programme LEcTure-switch qui permet de lire l'etat de chaque qwitches et les renvoie sous la forme d'un mot binaire fig 17

```
short LECTURE_SWs(){
    short switches;
    switches=*SWITCHES;
    return switches;
};
```

Figure 17: fonctions lecture etat des switches

puis on teste les switchs on associant les switchs aux leds testés precedement au sein du programme proncipale on realise la boucle de fonctionnement suivantes (event loop)

```
while(1){
    PILOTAGE_LEDs(LECTURE_SWs());
};
```

Figure 18: test de fonctionnement des switchs

3.4 Joystick

Le périphérique de gestion du joystick et les capteurs de lignes partagent le même signal de sélection /Cs2

Le registre REGISTRE-JOYSTICK (d'adresse allant de 0x12 à 0x13) est accessible en lecture seulement et contient 16 bits dont uniquement les 5 de poids faible sont utilisés:

Zone 2 C_{S2} 4*16bits	Adresse	A_5	A_4	A_3	A_2	A_1	A_0	
Base+16	0x10	0	1	0	0	0	0	
Base+17	0x11	0	1	0	0	0	1	
Base+18	0x12	0	1	0	0	1	0	
Base+19	0x13	0	1	0	0	1	1	
Base+20	0x14	0	1	0	1	0	0	
Base+21	0x15	0	1	0	1	0	1	
Base+22	0x16	0	1	0	1	1	0	
Base+23	0x17	0	1	0	1	1	1	

REGISTRE_CAPTEURS_ROBOT
REGISTRE_JOYSTICK
NON UTILISE
NON UTILISE

Figure 19: situation du joystick par rapport au mapping

afin de tester son bon fonctionnement , on utilisera l'afficheur sept segments du robot, en affichant différents nombres selon l'orientation du joystick comme est montré dans le schéma suivant:

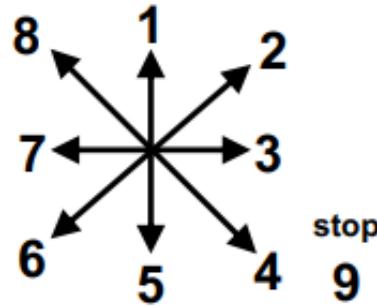


Figure 20: afichage désiré selon direction

on commence par réaliser des sous programme élémentaire qui peuvent indépendamment lire les 5 bits du poids faible pour savoir les directions choisies par le joystick grâce au masquage .

```

short TEST_JOY_GAUCHE(short valeur){
    short mask=4;
    short v;
    v=(mask&valeur)>>2;
    return v;
};

short TEST_JOY_DROITE(short valeur){
    short mask=2;
    short v;
    v=(mask&valeur)>>1;
    return v;
};

short TEST_JOY_BAS(short valeur){
    short mask=8;
    short v;
    v=(mask&valeur)>>3;
    return v;
};

short TEST_JOY_HAUT(short valeur){
    short mask=16;
    short v;
    v=(mask&valeur)>>4;
    return v;
};

short TEST_JOY_BOUTON(short valeur){
    short mask=1;
    short v;
    v=(mask&valeur);
    return v;
};

```

Figure 21: fonctions lecture direction

ces fonctions elementaire seront utilisé pour realisé l'affichage précisé par la fig 20 cette fois ci au sein du programme principale on écrit la boucle suivante:

```

while(1){
vtestjoy=*CAPTEURS_JOYSTICK
if (TEST_JOY_HAUT(vtestjoy)&&TEST_JOY_DROITE(vtestjoy)) {
PILOTAGE_AFFICHEUR_1(2);

}
else if (TEST_JOY_HAUT(vtestjoy)) {
PILOTAGE_AFFICHEUR_1(1);

}
else if (TEST_JOY_DROITE(vtestjoy)) {
PILOTAGE_AFFICHEUR_1(3);

}

else if (TEST_JOY_GAUCHE(vtestjoy)) {
PILOTAGE_AFFICHEUR_1(7);

}

else if (TEST_JOY_HAUT(vtestjoy)&&TEST_JOY_GAUCHE(vtestjoy)) {
PILOTAGE_AFFICHEUR_1(8);

}
else if( TEST_JOY_BOUTON(vtestjoy)) {
PILOTAGE_AFFICHEUR_1(9);
}
else if (TEST_JOY_GAUCHE(vtestjoy)&&TEST_JOY_BAS(vtestjoy)) {
PILOTAGE_AFFICHEUR_1(6);

}
else if (TEST_JOY_BAS(vtestjoy)) {
PILOTAGE_AFFICHEUR_1(5);

}
else if (TEST_JOY_DROITE(vtestjoy)&&TEST_JOY_BAS(vtestjoy)) {
PILOTAGE_AFFICHEUR_1(4);
}
}

```

3.5 moteurs

Le contrôle des roues du robot s'effectue par paire (gauche/droite), alimentées par un moteur à courant continu. Ainsi, en ajustant la tension appliquée au moteur, nous pouvons régler la direction de déplacement du robot.

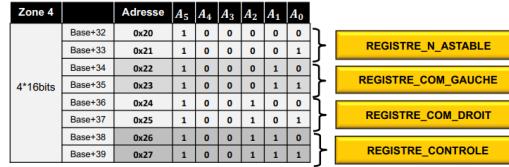


Figure 22: situation des moteurs par rapport au mapping

Après avoir défini les adresses dans `robo_def.h`, on teste le fonctionnement des moteurs grâce au programme suivant:

```

short vtestjoy;
*COMMAND_MOTEUR_MAITRE = 1000;
*COMMAND_MOTEUR_DROIT = STOP;
*COMMAND_MOTEUR_GAUCHE = STOP;
*START_MOTEURS = 7;

while (1) {
    vtestjoy = *CAPTEURS_JOYSTICK;

    if (TEST_JOY_DROITE(vtestjoy)) {
        *COMMAND_MOTEUR_GAUCHE = STOP + MARCHE_AV;
    }
    else if (TEST_JOY_GAUCHE(vtestjoy)) {
        *COMMAND_MOTEUR_DROIT = STOP + MARCHE_AV;
    }
    else if (TEST_JOY_HAUT(vtestjoy)) {
        *COMMAND_MOTEUR_DROIT = STOP + MARCHE_AV;
        *COMMAND_MOTEUR_GAUCHE = STOP + MARCHE_AV;
    }
    else if (TEST_JOY_BAS(vtestjoy)) {
        *COMMAND_MOTEUR_DROIT = STOP + MARCHE_AR;
        *COMMAND_MOTEUR_GAUCHE = STOP + MARCHE_AR;
    }
    else {
        *COMMAND_MOTEUR_GAUCHE = STOP;
        *COMMAND_MOTEUR_DROIT = STOP;
    }
}

```

3.6 capteurs distance

e périphérique de mesure de distance évalue la distance d'un objet à proximité du robot en utilisant des ondes ultrasonores. Le robot est équipé de trois détecteurs de présence identiques, positionnés à droite, à gauche et au centre d'un obstacle.

Ce dispositif comprend quatre registres :

Registres accessibles en écriture :

REGISTRE-IMPULSION : 5000 REGISTRE-PERIODE : 1 REGISTRE-LIMITE : Distance maximale de détection en cm Registre accessible en lecture :

REGISTRE-DISTANCE : Contient la distance mesurée entre le capteur et l'obstacle en cm.

Alors apres avoir encore definie les adresses grace a la situation par rapport ai mapping et a l'adressage interne on realise le programme suivant pour verifier leurs fonctionnement Afin de tester ce périphérique, la distance maximale de détection a été fixée à 50 cm. Les distances mesurées par les capteurs gauche, central et droit sont respectivement affichées sur l'afficheur 1, 2 et 3.

```
*COMMANDE_TRIGGER_ASTABLE_CENTRE = 5000;
*COMMANDE_TRIGGER_RAPPORT_CENTRE = 1;
*ECHO_LIMITE_CENTRE = 50;

*COMMANDE_TRIGGER_ASTABLE_DROITE = 5000;
*COMMANDE_TRIGGER_RAPPORT_DROITE = 1;
*ECHO_LIMITE_DROITE = 50;

*COMMANDE_TRIGGER_ASTABLE_GAUCHE = 5000;
*COMMANDE_TRIGGER_RAPPORT_GAUCHE = 1;
*ECHO_LIMITE_GAUCHE = 50;

unsigned short vdistcentre;
unsigned short vdistdroite;
unsigned short vdistgauche;

while (1) {
    vdistgauche = *ECHO_GAUCHE;
    vdistcentre = *ECHO_CENTRE;
    vdistdroite = *ECHO_DROITE;

    PILOTAGE_AFFICHEUR_1(vdistcentre);
    PILOTAGE_AFFICHEUR_2(vdistdroite);
    PILOTAGE_AFFICHEUR_3(vdistgauche);
}
```

3.7 capteurs de lignes

Ce sont trois émetteurs/récepteurs permettant de détecter une différence Blanc/Noir, ils sont positionnés au dessous des capteurs de distance ils seront utilisés pour permettre au robot de suivre une ligne noir sur le sol. Les informations issues de ce périphériques sont stockées dans un registre accessible en lecture uniquement donc apres avoir definie l'adresse des capteurs robot on desire les

tester en réalisant un programme qui permet d'afficher différents valeurs sur les afficheurs pour différents états des capteurs.

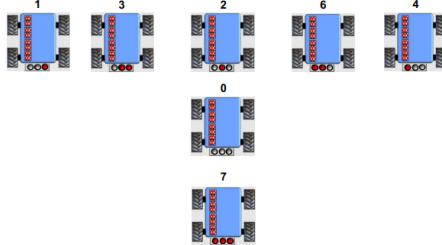


Figure 23: affichage désiré selon état des capteurs

cependant nous avons trouvé cette méthode de test un peu très compliquée pour un simple test des capteurs nous avons donc pensé à utiliser les LEDS pour ce test puisque les deux partagent une interprétation en binaire. d'une façon similaire à ce qu'on a fait pour le joystick on réalisera d'abord 3 fonctions élémentaires permettant de savoir si un capteur est actif ou pas . puis nous allons lire sur les leds l'état des capteurs le LED du poid le plus faible correspond au capteur gauche

```

short testCapteurGauche(short valeur) {
    short mask = 1;
    short v = (mask & valeur) >> 0;
    if (v == 0) {
        v = 1;
    } else {
        v = 0;
    }
    return v;
}

short testCapteurCentre(short valeur) {
    short mask = 2;
    short v = (mask & valeur) >> 1;
    if (v == 0) {
        v = 1;
    } else {
        v = 0;
    }
    return v;
}

short testCapteurDroite(short valeur) {
    short mask = 4;
    short v = (mask & valeur) >> 2;
    if (v == 0) {
        v = 1;
    }
}

```

```

    } else {
        v = 0;
    }
    return v;
}

```

puis dans le programme principal :

```

int main() {
    short va;

    while (1) {
        capteurL = *CAPTEURS_ROBOT;
        short b1 = testCapteurGauche(va);
        short b2 = testCapteurCentre(va);
        short b3 = testCapteurDroite(va);

        short b4 = b1 + (b2 << 1) + (b3 << 2);
        PILOTAGE_LEDs(b4);
    }

    return 0;
}

```

4 partie Software

pour cette partie on desire avoir plusieurs mode de fonctionnement:

Mode de Déplacement 0 - JOY V0

L'utilisateur pilote le robot en utilisant le joystick avec les instructions précédemment définies. Maintenir la pression sur le joystick est nécessaire pour que le robot suive la direction souhaitée. Relâcher le joystick arrête le robot. En cas de détection d'un obstacle à moins de 30 cm, seules les commandes de marche arrière et d'arrêt sont possibles. L'affichage dépend des choix de l'utilisateur via les commutateurs. Il peut afficher les distances entre le robot et les obstacles mesurées par les capteurs gauche, centre et droite respectivement. Sinon, le mode (00) est affiché sur l'afficheur 1.

Mode de Déplacement 1 - JOY V1

Si le mode 1 est sélectionné, le robot mémorise la consigne du joystick. L'utilisateur n'a pas besoin de maintenir constamment le joystick en position. Il doit appuyer sur le bouton d'arrêt pour stopper le robot. Les commandes du joystick restent les mêmes que pour le mode 0. En cas de détection d'un obstacle à moins de 30 cm, seules les commandes de marche arrière et d'arrêt sont possibles. Les afficheurs 7 segments peuvent afficher les distances ou le mode (01).

Mode de Déplacement 2 - Suivi de Ligne

Le robot suit automatiquement une ligne tracée au sol à l'aide de capteurs infrarouges. S'il quitte la ligne ou rencontre un obstacle, le robot s'arrête. En mode "JOY", un compteur de barres peut être affiché sur l'afficheur 2 à chaque détection de barre.

Mode de Déplacement 3 - Mode ESCLAVE

Dans ce mode, le robot suit un autre objet en essayant de maintenir une distance fixe (40 cm). Grâce aux capteurs, le robot détecte s'il y a un autre robot devant ou à côté. S'il se trouve entre 40 et 50 cm, il avance vers lui. Si aucun des capteurs ne détecte rien dans un rayon de 50 cm, "EE" est affiché sur les afficheurs 7 segments.

Programme Robot

```
// debut des definitions
// Vitesses
#define D_MARCHE_AV (MARCHE_AV / 2)
#define D_MARCHE_AR (MARCHE_AR / 2)

// Moteur
*COMMANDER_MOTEUR_MAITRE = 1000;
*COMMANDER_MOTEUR_DROIT = STOP;
*COMMANDER_MOTEUR_GAUCHE = STOP;
*START_MOTEURS = 7;

// Capteurs distance
*COMMANDER_TRIGGER_ASTABLE_CENTRE = 5000;
*COMMANDER_TRIGGER_RAPPORT_CENTRE = 1;
*COMMANDER_TRIGGER_ASTABLE_DROITE = 5000;
*COMMANDER_TRIGGER_RAPPORT_DROITE = 1;
*COMMANDER_TRIGGER_ASTABLE_GAUCHE = 5000;
*COMMANDER_TRIGGER_RAPPORT_GAUCHE = 1;

//fin des definitions

// Fonctions pour tester si le joystick pointe vers une direction (retourne 0 ou 1)

short TEST_JOY_GAUCHE(short valeur) {
    short v = valeur & 4;
    return v << 2;
}

short TEST_JOY_DROITE(short valeur) {
    short v = valeur & 2;
    return v << 1;
```

```

}

short TEST_JOY_HAUT(short valeur) {
    short v = valeur & 16;
    return v << 4;
}

short TEST_JOY_BAS(short valeur) {
    short v = valeur & 8;
    return v << 3;
}

short TEST_BOUTON(short valeur) {
    return valeur & 1;
}

// Fonctions pour tester les capteurs
short TEST_CAPTEUR_DROIT(short valeur) {
    short v = valeur & 4;
    return v << 2;
}

short TEST_CAPTEUR_CENTRE(short valeur) {
    short v = valeur & 2;
    return v << 1;
}

short TEST_CAPTEUR_GAUCHE(short valeur) {
    return valeur & 1;
}

// Sous-programme commande du moteur (on va l'utiliser dans chacun des modes )
void PILOT_MOTEUR(short valeur) {
    if (TEST_BOUTON(valeur)) {
        *COMMANDE_MOTEUR_DROIT = STOP;
        *COMMANDE_MOTEUR_GAUCHE = STOP;
    }
    else if (TEST_JOY_DROITE(valeur) && TEST_JOY_HAUT(valeur)) {
        *COMMANDE_MOTEUR_DROIT = STOP + D_MARCHE_AV;
        *COMMANDE_MOTEUR_GAUCHE = STOP + MARCHE_AV;
    }
    else if (TEST_JOY_DROITE(valeur) && TEST_JOY_BAS(valeur)) {
        *COMMANDE_MOTEUR_DROIT = STOP + D_MARCHE_AR;
        *COMMANDE_MOTEUR_GAUCHE = STOP + MARCHE_AR;
    }
    else if (TEST_JOY_GAUCHE(valeur) && TEST_JOY_BAS(valeur)) {

```

```

        *COMMAND_MOTEUR_DROIT = STOP + MARCHE_AR;
        *COMMAND_MOTEUR_GAUCHE = STOP + D_MARCHE_AR;
    }
    else if (TEST_JOY_GAUCHE(valeur) && TEST_JOY_HAUT(valeur)) {
        *COMMAND_MOTEUR_DROIT = STOP + D_MARCHE_AV;
        *COMMAND_MOTEUR_GAUCHE = STOP + MARCHE_AV;
    }
    else if (TEST_JOY_GAUCHE(valeur)) {
        *COMMAND_MOTEUR_DROIT = STOP + D_MARCHE_AV;
        *COMMAND_MOTEUR_GAUCHE = STOP;
    }
    else if (TEST_JOY_DROITE(valeur)) {
        *COMMAND_MOTEUR_DROIT = STOP;
        *COMMAND_MOTEUR_GAUCHE = STOP + D_MARCHE_AV;
    }
    else if (TEST_JOY_HAUT(valeur)) {
        *COMMAND_MOTEUR_DROIT = STOP + MARCHE_AV;
        *COMMAND_MOTEUR_GAUCHE = STOP + MARCHE_AV;
    }
    else if (TEST_JOY_BAS(valeur)) {
        *COMMAND_MOTEUR_DROIT = STOP + MARCHE_AR;
        *COMMAND_MOTEUR_GAUCHE = STOP + MARCHE_AR;
    }
    else {
        *COMMAND_MOTEUR_DROIT = STOP;
        *COMMAND_MOTEUR_GAUCHE = STOP;
    }
}

// Définition de Variables
short MODE;
short vtestjoy;
short capteurL;
short capteurD;

// Programme principal
while (1) {
    MODE = (*SWITCHES << 7) & 3;
    *ECHO_LIMITE_CENTRE = 30;
    *ECHO_LIMITE_CENTRE = 30;
    *ECHO_LIMITE_CENTRE = 30;

    // MODE 0
    while (MODE == 0 && (*SWITCHES << 9) != 0) {
        vtestjoy = *CAPTEURS_JOYSTICK;
        capteurD = *ECHO_CENTRE + *ECHO_DROITE + *ECHO_GAUCHE;

```

```

    if (capteurD == 0) {
        PILOT_MOTEUR(vtestjoy);
    }
    else if (TEST_JOY_BAS(vtestjoy)) {
        *COMMAND_MOTEUR_DROIT = STOP + D_MARCHE_AR;
        *COMMAND_MOTEUR_GAUCHE = STOP + D_MARCHE_AR;
    }
    else {
        *COMMAND_MOTEUR_DROIT = STOP;
        *COMMAND_MOTEUR_GAUCHE = STOP;
    }

    *LED_ROUGES = (*CAPTEURS_ROBOT & 7) + (*SWITCHES & 896);

    if (*SWITCHES & 1) {
        PILOTAGE_AFFICHEUR_1(*ECHO_DROITE);
        PILOTAGE_AFFICHEUR_2(*ECHO_CENTRE);
        PILOTAGE_AFFICHEUR_3(*ECHO_GAUCHE);
    }

    if (!(*SWITCHES & 1)) {
        PILOTAGE_AFFICHEUR_1(0);
    }
}

// MODE 1
while (MODE == 1 && (*SWITCHES << 9) != 0) {
    if (vtestjoy != 0) {
        capteurD = *ECHO_CENTRE + *ECHO_DROITE + *ECHO_GAUCHE;

        if (capteurD == 0) {
            PILOT_MOTEUR(vtestjoy);
        }
        else if (TEST_JOY_BAS(vtestjoy)) {
            *COMMAND_MOTEUR_DROIT = STOP + D_MARCHE_AR;
            *COMMAND_MOTEUR_GAUCHE = STOP + D_MARCHE_AR;
        }
        else {
            *COMMAND_MOTEUR_DROIT = STOP;
            *COMMAND_MOTEUR_GAUCHE = STOP;
        }
    }
    else {
        *COMMAND_MOTEUR_DROIT = STOP;
        *COMMAND_MOTEUR_GAUCHE = STOP;
    }
}

```

```

}

*LED_ROUGES = (*CAPTEURS_ROBOT & 7) + (*SWITCHES & 896);

if (*SWITCHES & 1) {
    PILOTAGE_AFFICHEUR_1(*ECHO_DROITE);
    PILOTAGE_AFFICHEUR_2(*ECHO_CENTRE);
    PILOTAGE_AFFICHEUR_3(*ECHO_GAUCHE);
}
else {
    PILOTAGE_AFFICHEUR_1(1);
}
}

short a = 0;

// MODE 2
while (MODE == 2 && (*SWITCHES << 9) != 0) {
    capteurL = *CAPTEURS_ROBOT;
    capteurD = *ECHO_CENTRE + *ECHO_DROITE + *ECHO_GAUCHE;

    if (TEST_CAPTEUR_CENTRE(capteurL) == 1 && TEST_CAPTEUR_DROIT(capteurL) == 0 && TEST_CAPTEUR_GAUCHE(capteurL) == 0) {
        *COMMANDE_MOTEUR_DROIT = STOP + MARCHE_AV;
        *COMMANDE_MOTEUR_GAUCHE = STOP + MARCHE_AV;
    }
    else if (TEST_CAPTEUR_DROIT(capteurL) == 1 && TEST_CAPTEUR_GAUCHE(capteurL) == 0) {
        *COMMANDE_MOTEUR_DROIT = STOP + D_MARCHE_AV;
        *COMMANDE_MOTEUR_GAUCHE = STOP + MARCHE_AV;
    }
    else if (TEST_CAPTEUR_GAUCHE(capteurL) == 1 && TEST_CAPTEUR_DROIT(capteurL) == 0) {
        *COMMANDE_MOTEUR_DROIT = STOP + D_MARCHE_AV;
        *COMMANDE_MOTEUR_GAUCHE = STOP + MARCHE_AV;
    }
    else if (TEST_CAPTEUR_CENTRE(capteurL) == 1 && TEST_CAPTEUR_DROIT(capteurL) == 1 && TEST_CAPTEUR_GAUCHE(capteurL) == 0) {
        *COMMANDE_MOTEUR_DROIT = STOP + MARCHE_AV;
        *COMMANDE_MOTEUR_GAUCHE = STOP + MARCHE_AV;
        a = a + 1;
        PILOTAGE_AFFICHEUR_2(a);
    }
    else if (capteurD != 0) {
        if (TEST_JOY_BAS(vtestjoy)) {
            *COMMANDE_MOTEUR_DROIT = STOP + D_MARCHE_AR;
            *COMMANDE_MOTEUR_GAUCHE = STOP + D_MARCHE_AR;
        }
        else {
            *COMMANDE_MOTEUR_DROIT = STOP;

```

```

        *COMMAND_MOTEUR_GAUCHE = STOP;
    }
}

*LED_ROUGES = (*CAPTEURS_ROBOT & 7) + (*SWITCHES & 896);

if (*SWITCHES & 1) {
    PILOTAGE_AFFICHEUR_1(*ECHO_DROITE);
    PILOTAGE_AFFICHEUR_2(*ECHO_CENTRE);
    PILOTAGE_AFFICHEUR_3(*ECHO_GAUCHE);
}
}

// MODE 3
*ECHO_LIMITE_CENTRE = 50;
*ECHO_LIMITE_CENTRE = 50;
*ECHO_LIMITE_CENTRE = 50;

while (MODE == 3 && (*SWITCHES << 9) != 0) {
    short vdistcentre = *ECHO_CENTRE;
    short vdistdroite = *ECHO_DROITE;
    short vdistgauche = *ECHO_GAUCHE;

    if (vdistcentre > 40) {
        *COMMAND_MOTEUR_DROIT = STOP + MARCHE_AV;
        *COMMAND_MOTEUR_GAUCHE = STOP + MARCHE_AV;
    }
    else if (vdistcentre != 0 && vdistcentre <= 40) {
        *COMMAND_MOTEUR_DROIT = STOP + D_MARCHE_AR;
        *COMMAND_MOTEUR_GAUCHE = STOP + D_MARCHE_AR;
    }

    if (vdistdroite > 40 || (vdistgauche != 0 && vdistgauche <= 40)) {
        *COMMAND_MOTEUR_DROIT = STOP;
        *COMMAND_MOTEUR_GAUCHE = STOP + D_MARCHE_AV;
    }

    if (vdistgauche > 40 || (vdistdroite != 0 && vdistdroite <= 40)) {
        *COMMAND_MOTEUR_DROIT = STOP + D_MARCHE_AV;
        *COMMAND_MOTEUR_GAUCHE = STOP;
    }

    if (vdistcentre == 0 && vdistdroite == 0 && vdistgauche == 0) {
        *COMMAND_MOTEUR_DROIT = STOP;
        *COMMAND_MOTEUR_GAUCHE = STOP;
        PILOTAGE_AFFICHEUR_1(100);
    }
}
}

```

```

        PILOTAGE_AFFICHEUR_2(100);
        PILOTAGE_AFFICHEUR_3(100);
    }
    else {
        PILOTAGE_AFFICHEUR_1(vdistdroite);
        PILOTAGE_AFFICHEUR_2(vdistcentre);
        PILOTAGE_AFFICHEUR_3(vdistgauche);
    }

    *COMMANDER_MOTEUR_DROIT = STOP;
    *COMMANDER_MOTEUR_GAUCHE = STOP;
}

```

5 Conclusion

En conclusion, ce bureau d'études pour le développement d'un système robotique a été couronné de succès. La conception robuste du contrôle du robot, nous a offert l'occasion de mettre en pratique les connaissances théoriques acquises lors des cours d'architecture et de programmation en assembleur et nous ouvre les yeux à la complexité du hardware des outils de tous les jours et c'est un très bon projet pour s'initialiser à la robotique . qui mélange l'aspect hardware de d'architecture et de programmation en assembleur et software de la programmation en C . et finalement Nous exprimons notre gratitude envers tous nos encadrants qui ont apporté une assistance précieuse, nous guidant et résolvant de nombreuses difficultés tout au long des deux sessions consacrées à ce projet aussi ludique qu'instructif.