TREX : Microprocesseur et interface
Réalisation d'une calculatrice à commande vocale

Alexandre GRAMFORT
Aurélien LOUIS

10 mars 2004

Table des matières

Introduction

Dans le cadre de ce travail expérimental, nous avons réalisé une calculatrice vocale, c'est-à-dire un système piloté par la voix réalisant les taches d'une calculatrice simple. Nous désirions en effet concilier ce travail avec un de nos centres d'intérêt, à savoir le domaine du traitement du signal.

Au final, notre calculatrice respecte le "cahier des charges" suivant :

- prise en compte des opérations +, -, * sur l'ensemble des entiers relatifs entre -32767 et 32767 (entiers signés codés sur 16 bits) .
- commandes vocales "effacer l'écran" et "affiche le résultat" (=).
- saisie des données de calcul (opérateurs et opérandes) entièrement vocale.
- possibilité de réglage manuel des différents modes (apprentissage, reconnaissance, et effacement mémoire) au moyen de "switch"
- affichage progressif collé à droite sur deux lignes, selon le modèle d'une calculatrice de téléphone portable.

1 Description des principaux éléments du système.

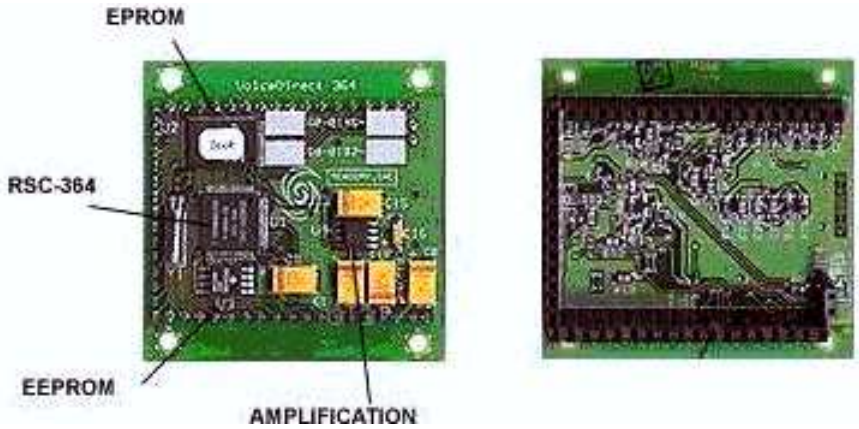
Pour réaliser notre calculatrice vocale, nous avons eu besoin de 3 éléments principaux : un écran LCD pour l'affichage, un module de reconnaissance vocale réalisant l'interface utilisateur-machine, et un micro-contrôleur réalisant les calculs et pilotant le système (cf. fig. 1.3 en page 6 et fig. 2.1 en page 7).

1.1 Le module VoiceDirect 364

Le VoiceDirect 364 est un processeur de reconnaissance vocale fabriqué par la marque Sensory. Il est capable avec un minimum de programmation d'apprendre à reconnaître de la parole mono et multi-locuteur. Pour ce faire il est basé autour d'un micro-contrôleur RSC-364 associé à une pré-amplification des signaux audio, à une EPROM externe 27C512 (dans laquelle est stocké le programme de gestion) et à une EEPROM série 24LC65 (dans laquelle sera stocké le vocabulaire à reconnaître).

Caractéristiques :

- Taux de reconnaissance maximale : 99 %.
- Durée maximale du mot ou de l'expression à reconnaître : 3,5 s (à condition qu'il ne comporte pas de silence supérieur à 0,5 s).
- Tension d'alimentation : + 5V.
- Consommation : 30 à 100 mA environ.
- Dimensions : 50 x 50 x 10 mm



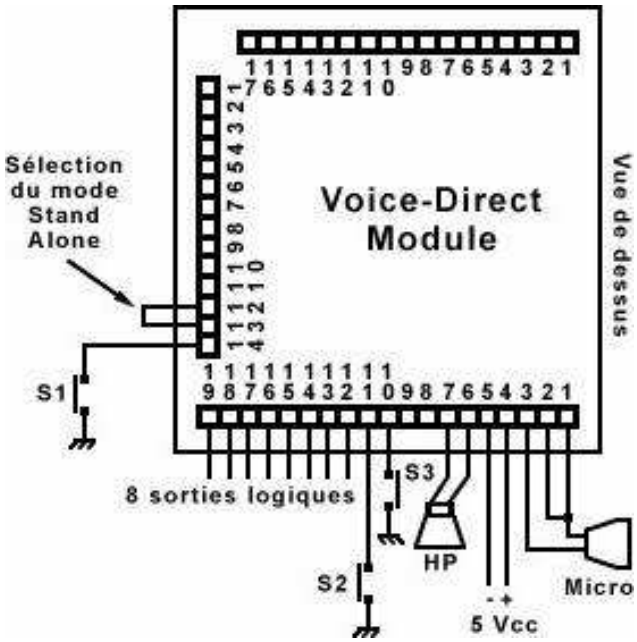


FIG. 1 – Schéma de branchement du VoiceDirect 364 en mode Stand Alone par appel

Mode de fonctionnement : Mode "Stand-Alone" par appel

L'exploitation et la mise en oeuvre du "VoiceDirect module" en mode "Stand Alone par appel" se résume au schéma sur la figure 1.1.

L'entrée **"S1"** sert à réinitialiser le module.

L'entrée "**S2"** sert à "lancer" la phase d'apprentissage.

L'entrée **"S3"** lancer la phase de reconnaissance.

Le haut-parleur sert à restituer des messages d'aide lors des phases d'apprentissage et de reconnaissance.

Le microphone est un "simple" électret.

PHASE D'APPRENTISSAGE :

L'apprentissage du vocabulaire se déclenche en appuyant fugitivement sur le bouton-poussoir "S2". A ce stade, le module (via le haut-parleur) vous demande (en anglais) de prononcer le mot n° 1 : "Say word one". Prononcez alors un premier mot. Le module vous demande de lui répéter à nouveau le mot "Repeat" afin qu'il puisse faire ainsi une sorte de valeur "moyenne" des 2 enregistrements. A ce moment, le processeur vous demande d'enregistrer le 2ème mot "Say word two" et ainsi de suite... (maximum 15 mots). Un mot peut très bien être constitué d'une suite d'expression dont la valeur totale ne doit pas excéder les 3,2 s (ex. ouvre la porte du salon). A noter toutefois que l'expression ne doit pas comporter de silence supérieur à 0,5 s. Si lorsque le module vous demande de répéter le mot, vous lui "donnez" un mot différent du premier, il refusera ce dernier en vous signalant "Training error !" et vous fera reprendre l'enregistrement du mot en question depuis la première phase (vous devrez alors le répéter à nouveau 2 fois). Si en phase d'apprentissage aucun mot n'est prononcé pendant plus de 3 à 4 secondes ou si "S2" est pressé à nouveau, le module retourne en mode "attente de commande". A ce stade, si vous appuyez à nouveau sur "S2", vous reprenez la phase d'enregistrement au N° du mot là où vous l'avez "laissé". Prenez tout votre temps car si vous enregistrez un "mauvais" mot, il vous faudra tout effa-

cer et enregistrer tout votre vocabulaire. Pour ce faire, il suffit d'appuyer simultanément sur "S2" et "S3" pendant 1 seconde pour que le module soit prêt à "repartir" depuis le mot N° 1.

PHASE DE RECONNAISSANCE :

En appuyant fugitivement sur le bouton-poussoir "S3", le module passe en attente de reconnaissance d'un mot. Il vous suffit alors de prononcer un mot préalablement mémorisé pour que la sortie correspondante à ce dernier s'active. Si en revanche, le mot prononcé est trop différent du vocabulaire en mémoire, le module refuse d'activer ses sorties et vous le signale "Word not recognized !". Les sorties correspondantes aux 8 premiers mots mémorisés sont disponibles sur 8 sorties logiques distinctes (apparition d'un +5V pendant une seconde lorsque le mot est reconnu). Le 9ème mot active les sorties 8 et 1, le 10ème les sorties 8 et 2, le 11ème les sorties 8 et 3...

Sorties activées	1	2	3	4	5	6	7	8
Mot N° 1 reconnu	OUI	NON	NON	NON	NON	NON	NON	NON
Mot N° 2 reconnu	NON	OUI	NON	NON	NON	NON	NON	NON
Mot N° 3 reconnu	NON	NON	OUI	NON	NON	NON	NON	NON
Mot N° 4 reconnu	NON	NON	NON	OUI	NON	NON	NON	NON
Mot N° 5 reconnu	NON	NON	NON	NON	OUI	NON	NON	NON
Mot N° 6 reconnu	NON	NON	NON	NON	NON	OUI	NON	NON
Mot N° 7 reconnu	NON	NON	NON	NON	NON	NON	OUI	NON
Mot N° 8 reconnu	NON	NON	NON	NON	NON	NON	NON	OUI
Mot N° 9 reconnu	OUI	NON	NON	NON	NON	NON	NON	OUI
Mot N° 10 reconnu	NON	OUI	NON	NON	NON	NON	NON	OUI
Mot N° 11 reconnu	NON	NON	OUI	NON	NON	NON	NON	OUI
Mot N° 12 reconnu	NON	NON	NON	OUI	NON	NON	NON	OUI
Mot N° 13 reconnu	NON	NON	NON	NON	OUI	NON	NON	OUI
Mot N° 14 reconnu	NON	NON	NON	NON	NON	OUI	NON	OUI
Mot N° 15 reconnu	NON	NON	NON	NON	NON	NON	OUI	OUI

1.2 Le LCD

L'écran LCD (Liquid cristal display) est un Toshiba 2491/2493J0 de 2x16 caractères [1]. Il se connecte facilement à la carte CMD11A8 contenant le micro-contrôleur 68HC11 par l'intermédiaire d'une nappe. Ainsi l'adresse des registres de fonctionnement du LCD sont inclus dans le mapping de la carte qui pilote l'application. Ils sont ainsi d'un accès relativement facile. Il est aussi notable que l'on peut régler par l'intermédiaire d'un potentiomètre présent sur la carte CMD11A8 le contraste de l'écran.

1.3 La carte CMD11A8 et le micro-contrôleur 68HC11

Le micro-contrôleur 68HC11 de Motorola est installé sur une carte CMD11A8 (cf. fig. 1.3 en page 6). Cette carte branchée sur une alimentation 0V-5V se programme par l'intermédiaire d'un PC relié par une liaison série. Il contient en plus du HC11 de l'EEPROM et de la RAM servant à stocker le code à exécuter.

Cette carte se relie facilement à un écran LCD à l'aide d'une nappe et propose en autre une liaison vers une carte extérieur à l'aide d'un port auxiliaire.

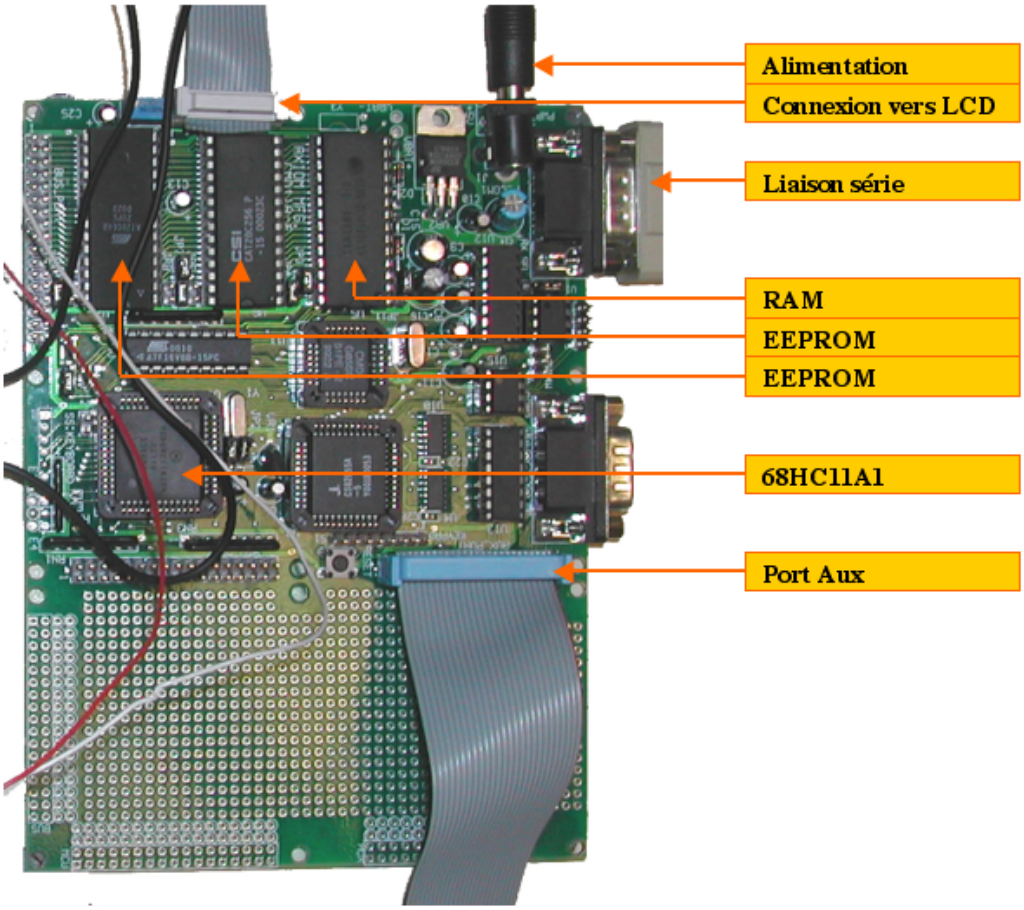


FIG. 2 – Description de la carte CMD11A8

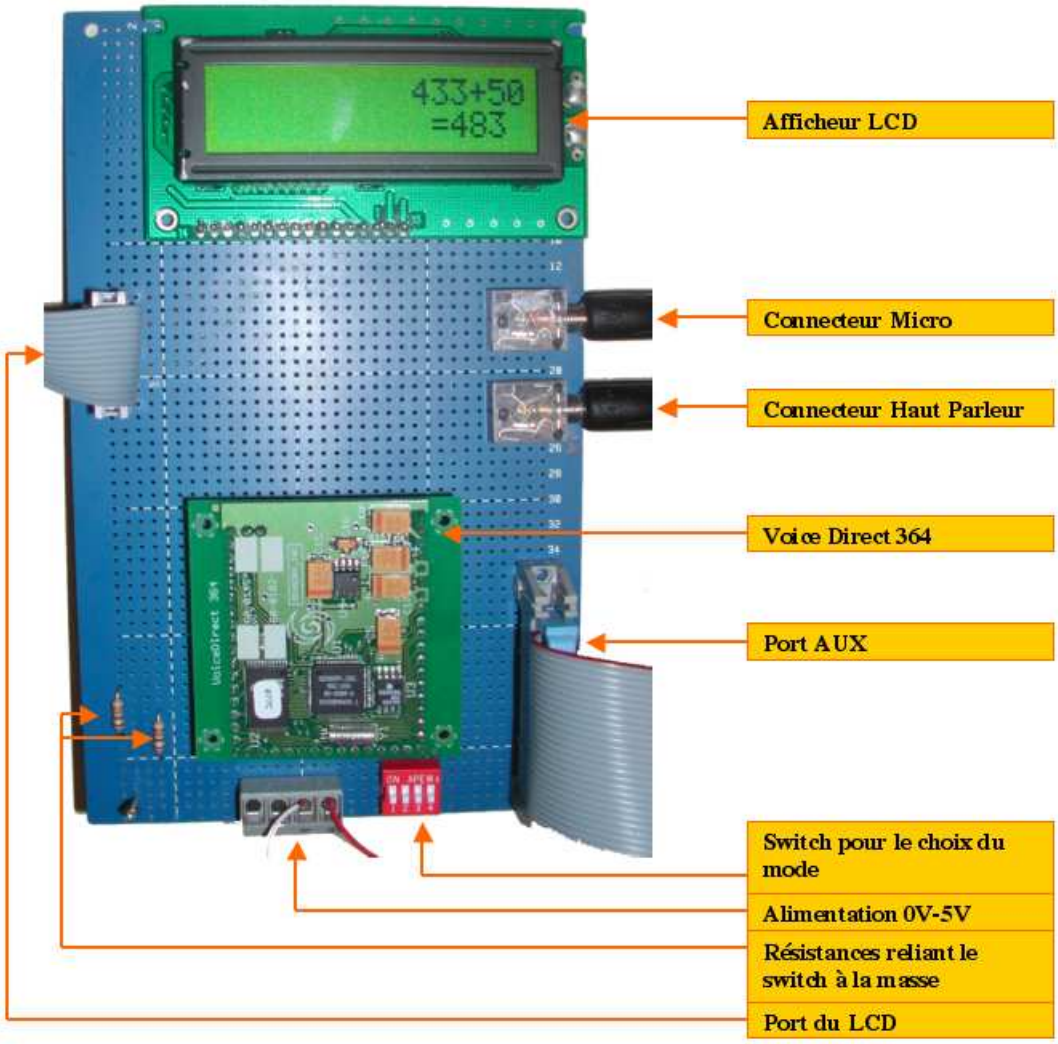


FIG. 3 – Vue face avant

2 La carte réalisée

2.1 Schéma de branchement

Le module Voicedirect est piloté par le micro-contrôleur à travers le port auxiliaire de la carte CMD. Comme le montre le schéma de branchement (cf. fig. 2.1 en page 8), les pattes S1, S2, S3 (qui servent à placer le module en mode reconnaissance ou apprentissage, et à effacer les mots), sont reliés respectivement à PC5, PC6 et PC7. Les 8 pattes de données sont reliées aux pattes A0 jusqu'à A7. Par ailleurs, le module est connecté à une alimentation séparée, ainsi qu'à un microphone et un haut-parleur. Le port aux est en outre relié aux switchs 1 et 4 par les pattes B0 et B1. Les pattes 1 à 14 du LCD sont reliées au port LCD

2.2 Principe de fonctionnement : mode d'emploi de la calculatrice

Il faut tout d'abord alimenter la carte en 5V, brancher le micro et les écouteurs, relier le port aux et le port LCD du 68HC11 à la carte.

A la mise sous tension, les switchs permettent de placer la calculatrice sous un des trois

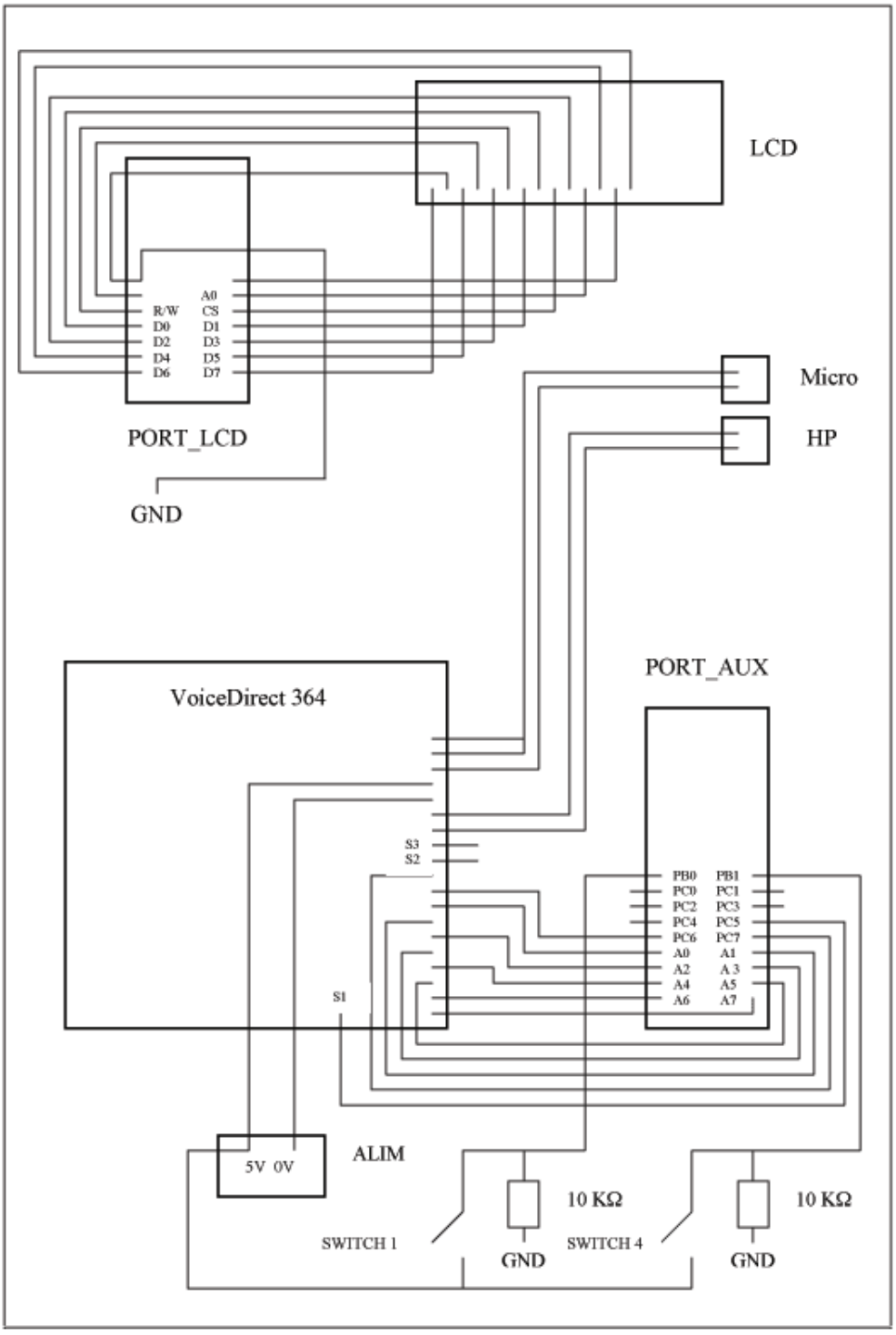


FIG. 4 – Schéma de branchement

Numéro du mot	Signification logicielle
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	0
11	+
12	-
13	*
14	efface
15	=

TAB. 1 – Table des correspondances

mode, reconnaissance, apprentissage et réinitialisation.

Le mode réinitialisation est activé en plaçant les switchs 1 et 4 en position on. Ce mode efface tout d'abord la mémoire mots de la calculatrice (annoncé par un "Memory erased"), puis lance une phase d'apprentissage des commandes.

Le mode apprentissage (switch 1 en position on, switch 4 en off) lance directement la phase d'apprentissage. Les mots 1 à 15 doivent correspondre aux commandes fournies dans le table des correspondances (cf Tab. 1). La calculatrice transmet le message suivant : "Say word ...", il faut alors prononcer distinctement la commande correspondante. A l'annonce "Repeat word...", il faut répéter la commande. Si le mot n'est pas correctement prononcé une deuxième fois, il y a "Training error". Il faut alors relancer la calculatrice en mode apprentissage, qui repartira là où il s'était arrêté. A la fin de l'apprentissage, le message "Training complete" est transmis. On peut alors relancer la calculatrice en mode reconnaissance.

Le mode reconnaissance (switch 1 et 4 off) est le mode principal de la calculatrice, c'est celui qui permet de réaliser les calculs proprement dit. Il faut attendre l'annonce "Say a word" pour prononcer la commande désirée : si elle n'est pas reconnue, la calculatrice dit "Word not recognized". Pour entrer un nombre, il faut lui donner la série de ses chiffres. La commande "efface" efface le calcul, et la commande "égale" permet d'obtenir le résultat du calcul enregistré.

3 La programmation du micro-contrôleur

Il s'agit désormais dans cette partie, d'aborder plus dans le détail, le mode d'initialisation et de pilotage des composants.

3.1 La gestion des interfaces

Les composants sur lesquels le micro-contrôleur agit sont le LCD pour l'affichage et le VoiceDirect pour la reconnaissance des commandes vocales. Le 68HC11 va donc devoir gérer des entrées et des sorties.

Interface LCD :
Comme il en a été fait mention plus haut, l'écran LCD est relié par une nappe à la carte CMD11A8. Les adresses prévues pour le LCD dans le mapping de cette carte sont B5F0 et B5F1 (en hexadécimal) [2].

B5F0 est utilisée lors de l'initialisation du LCD (cf. [1] en page 6 pour un exemple de procédure d'initialisation) ainsi que pour forcer la décrémentation du curseur. Il sert également pour vérifier que l'écran a bien fini de traiter une instruction comme par exemple d'afficher un caractère. Dans notre application nous avons choisi de mettre le "Display Shift" sur 0 (off), le curseur en mode "increment" [4].

Pour l'affichage c'est l'adresse B5F1 qui est utilisée. La table des correspondances entre le code des caractère en ASCII et le code hexadécimal requis par le LCD est fournie dans [4].

Interface Port Aux :
Le 68HC11 grâce au port aux a pour fonction de piloter de manière totalement logicielle le VoiceDirect 364. Il doit donc pouvoir agir avec des broches à la fois en sortie sur S1, S2 et S3 mais aussi en entrée pour lire la sortie du VoiceDirect et la position des switchs.

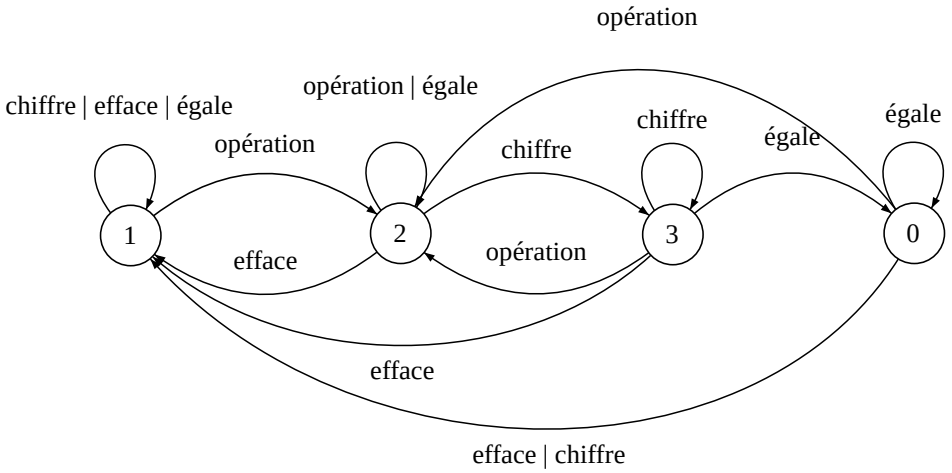
Comme en témoigne le schéma de branchement en fig. 2.1 page 8. Les broches en sortie sont reliées au port C. Les 8 broches fournissant les résultats du VoiceDirect sont connectées au port A et les switchs sont reliés au port B. Ainsi le manuel [2] fournit pour le bit de contrôle en B5F7 la valeur 92 (en hexadécimal) spécifiant l'état d'entrée-sortie des ports A, B et C.

Il est alors possible de vérifier l'état des switchs sur le port B en B5F5 et l'état de la sortie du VoiceDirect sur le port A en B5F4. Pour fixer la valeur de S1, S2 et S3, il suffit de déterminer la valeur du port C en B5F6.

3.2 Principe général de l'algorithme

La calculatrice est en fait un automate à 4 états, qui s'accompagne de 4 registres (variables globales dans le programme C). Ces registres sont "gauche", qui contient l'opérande à gauche, "op", qui contient l'opérateur et "droite" qui contient l'opérande à droite. Son alphabet est l'ensemble des commandes, dans lesquelles on distingue les chiffres, les opérateurs, la commande égale et la commande efface. L'automate est dans l'état 1 lorsque l'utilisateur est en train de rentrer l'opérande à gauche, dans l'état 2 lorsque il rentre l'opérateur, l'état 3 lorsqu'il s'agit de l'opérande droite, et l'état 0 quand il a renvoyé un résultat. Nous détaillons ici en pseudo-code les opérations de l'automate pour chaque transition, ainsi que son graphe des transitions (tableaux 2, 3, 4 et 5).

Nous avons géré l'affichage de la manière suivante, de façon à donner l'impression que les caractères apparaissent de la droite : l'écran ne se déplace pas, et le curseur se déplace vers la droite à chaque entrée. Nous plaçons donc le curseur à la fin de la ligne, puis nous le décalons d'un certain nombre de cases en fonction de ce que nous avons à afficher. Nous affichons ensuite les caractères, qui, éventuellement se superposent aux précédents en les effaçant.



TAB. 2 – Graphe des transitions de l'automate

(3 ; chiffre) → droite = droite*10 + chiffre ;
affiche(op) ;
affiche(droite) ;

(0 ; chiffre) → gauche = chiffre ;
efface écran ;
va en ligne 2 ;
affiche(gauche) ;

(2 ; chiffre) → droite = chiffre ;
affiche(op) ;
affiche(droite) ;

(1 ; chiffre) → gauche = gauche*10 +chiffre ;
efface écran ;
va en ligne 2 ;
affiche(gauche) ;

TAB. 3 – Actions de l'automate pour la saisie d'un chiffre

(3 ; opérateur) → résultat = gauche "op" droite ;
gauche = résultat ;
op = opérateur ;
efface écran ;
va en ligne 1 ;
affiche(gauche) ;
va en ligne 2 ;
affiche(op) ;

(0 ; opérateur) → gauche = résultat ;
op = opérateur ;
efface écran ;
va en ligne 1 ;
affiche(gauche) ;
va en ligne 2 ;
affiche(op) ;

(2 ; opérateur) → op = opérateur ;
efface écran ;
va en ligne 1 ;
affiche(gauche) ;
va en ligne 2 ;
affiche(op) ;

(1 ; opérateur) → op = opérateur ;
efface écran ;
va en ligne 1 ;
affiche(gauche) ;
va en ligne 2 ;
affiche(op) ;

TAB. 4 – Actions de l'automate pour la saisie d'un opérateur

(0|1|2|3 ; chiffre) → efface écran ;
va ne ligne 2 ;
affiche(0) ;
gauche = 0 ;

(3 ; égale) → résultat = gauche "op" droite ;
efface écran ;
va en ligne 1 ;
affiche(gauche) ;
affiche(op) ;
affiche(droite) ;
va en ligne 2 ;
affiche(=) ;
affiche(résultat) ;

(0|1|2 ; égale) → ne fais rien ;

TAB. 5 – Actions de l'automate pour la saisie des commandes efface et égale

Conclusion

Ce TREX fut pour nous très instructif, à deux niveaux. Tout d'abord, il nous a permis d'acquérir des connaissances techniques sur l'utilisation et le fonctionnement des micro-contrôleurs, et de comprendre leur intérêt d'un point de vue industrielle.

Par ailleurs, ce fut l'occasion pour nous de mettre en oeuvre une démarche de projet complète, partant de la définition des objectifs jusqu'à la réalisation complète du système désiré. Outre la satisfaction de mener à terme une telle démarche en combinant "hardware" et "software", et d'obtenir un "produit fini", cela nous a permis de nous rendre compte de ses difficultés inhérentes. Par exemple, nous avons du faire quelquefois des marches arrière dans notre projet, parce que nous avions mal compris le fonctionnement ou les capacités techniques de tel ou tel composant ; ou bien il nous est arrivé parfois de passer beaucoup de temps à débusquer la source d'une erreur, du fait de la grande complexité du système global (l'erreur pouvant être "hardware" ou bien "software").

Références

[1] *Trex-Modex Annexe G Centres de Travaux Expérimentaux*

[2] *Majeure ECS - Éléments matériels et logiciels sur la carte cible CMD11A8*

[3] *Majeure ECS - Éléments matériels et logiciels sur le 68HC11*

[4] *Documentation sur le LCD - "lcd_cmd.PDF"*

[5] *"VoiceDirect 364 Manual.pdf"* disponible sur *http ://www.sensoryinc.com/html/products/voicedirect364.html*

Annexes

Code C

```
/*
#code    0xE000
#data    0x2000
#stack   0x7FFF
*/
#code    0x3000
#data    0x2000
#include <startup.c>
/* adresse de base des registres internes dans le HC11 */
#define  REG_BASE 0x1000
/* la liste des offsets internes du HC11 */
#include <68hc11.h>
#define DELAY1MS 73
/*variables globales*/
unsigned char etat;
/*
etat vaut 0x00 si on sort d'une opération
et qu'on a un déjà un res à utiliser
etat vaut 0x01 si on est à gauche
etat vaut 0x02 si on traite l'op
etat vaut 0x03 si on est à droite
*/
unsigned char c; /* char courant */
unsigned char op;
```

```
int gauche;
int droite;
int resultat;
main ()
{
        pokeb(0xB5F7, 0x92); /* initialisation des ports aux A, B, C. */

        initvoice();
        initLCD();

        delayms(1000);

        if ((peekb(0xB5F5) & 0x03) == 0x03) {
                resetmots();
                apprentissage();
                }
        else if ((peekb(0xB5F5) & 0x01) == 0x01) {
                apprentissage();
                }
        else {
                init();
                reconnaissance();
                }
        #asm
        SWI
        #endasm
}
longueur(n)
int n;
{
        int p,a,res;
        p=1;
        res = 0;
        if (n<0) {
                a = 0-n;
                res++;
        } else a = n;
        while (a / p > 10) {
                p = p*10;
                res++;
        }
        return res;
}
resetmots()
{
        /* affichage de effacement */
        allerun();
        decremente(12);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x45);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x46);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x46);
```

```
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x41);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x43);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x45);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x4D);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x45);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x4E);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x54);

        allerdeux();
        decremente(11);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x4D);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x45);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x4D);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x4F);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x49);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x52);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x45);

        pokeb(0xB5F6, 0x20);
        delayms(3000);
        pokeb(0xB5F6, 0xE0);
}
void init() {
        /*init des var globales et de l'affichage*/
        etat = 0x01;
        gauche = 0;
        droite = 0;
        clearLCD();
        allerdeux();
        affiche(0x82);
}
void initLCD(){
        int i;
        delayms(15);
        pokeb(0xB5F0, 0x30);
        delayms(5);
        pokeb(0xB5F0, 0x30);
        delayms(1);
        pokeb(0xB5F0, 0x30);
        delayms(1);
```

```
        pokeb(0xB5F0, 0x3C);
        delayms(1);
        pokeb(0xB5F0, 0x08);
        delayms(1);
        pokeb(0xB5F0, 0x01);
        delayms(1);
        pokeb(0xB5F0, 0x06);
        delayms(1);
        pokeb(0xB5F0, 0x0C);
        delayms(1);
}
void clearLCD(){
        delayms(1);
        pokeb(0xB5F0, 0x01);
        delayms(2);
        pokeb(0xB5F0, 0x06);
        delayms(1);
        pokeb(0xB5F0, 0x0C);
        delayms(1);
}
void decremente(n)
int n;
{
    int i;
    for (i=1; i<n; i++) {
        while((peekb(0xB5F0) & 0x80) == 0x80){};
        pokeb(0xB5F0, 0x10);
    }
}
void reconnaissance() {
/* si la sortie du voice change */
        int i;
        int nb;

        /* affichage de reconnaissance */
        clearLCD();
        allerun();
        decremente(14);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x52);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x45);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x43);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x4F);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x4E);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x4E);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x41);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x49);
```

```
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x53);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x53);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x41);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x4E);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x43);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x45);
        delayms(1000);
        init();

        nb = 0;
        while ( nb < 5 ) {
                pokeb(0xB5F6, 0x60);
                delayms(1000);
                pokeb(0xB5F6, 0xE0);
                delayms(1000);
                i = 0;
                while( (peekb(0xB5F4) == 0x00) && (i++ < DELAY1MS*10000) ){};
                delayms(100);
                if (i > DELAY1MS*9999) nb++;
                else {
                        nb = 0;
                        c = peekb(0xB5F4);
                        traite_data();
                }
        }
        while(peekb(0xB5F4)!= 0x00){};
}
void initvoice(){
        pokeb(0xB5F6, 0xE0);
        delayms(1000);
}
void apprentissage(){
        /* affichage de apprentissage */
        clearLCD();
        allerun();
        decremente(13);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x41);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x50);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x50);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x52);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x45);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x4E);
```

```
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x54);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x49);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x53);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x53);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x41);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x47);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F1, 0x45);
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F6, 0xA0);
        delayms(1000);
        pokeb(0xB5F6, 0xE0);
}
void traite_data() {
        switch(c) {
        case 0x82 : { traite_entier(0); break; };
        case 0x01 : { traite_entier(1); break; };
        case 0x02 : { traite_entier(2); break; };
        case 0x04 : { traite_entier(3); break; };
        case 0x08 : { traite_entier(4); break; };
        case 0x10 : { traite_entier(5); break; };
        case 0x20 : { traite_entier(6); break; };
        case 0x40 : { traite_entier(7); break; };
        case 0x80 : { traite_entier(8); break; };
        case 0x81 : { traite_entier(9); break; };
        case 0x84 :;
        case 0x88 :;
        case 0x90 : { traite_op(); break; }
        case 0xA0 : { init(); break; }
        case 0xC0 : { traiteegal(); break; }
        default :;
        }
}
void traite_entier(n)
int n;
{
        int d;
        switch(etat) {
        case 0x00 : {
                etat = 0x01;
                gauche = 0;
        clearLCD();
        allerdeux();
        };
        case 0x01 : {
                gauche = (gauche*10) + n;
                d = longueur(gauche);
                clearLCD();
```

```
        allerdeux();
                decremente(d);
                affiche_entier(gauche);
                break;
        }
        case 0x02 : {
                etat = 0x03;
                droite = n;
                d = longueur(droite);
                decremente(d+2);
                affiche(op);
                affiche_entier(droite);
                break;
        };
        case 0x03 : {
                droite = (droite*10) + n;
                d = longueur(droite);
                decremente(d+2);
                affiche(op);
                affiche_entier(droite);
                break;
        }
        default :;
        }
}
void traite_op() {
        int d;
        switch(etat) {
        case 0x03 : {
                calcule();
        };
        case 0x00 : {
                gauche = resultat;
        }
        case 0x01 : {
                op = c;
                etat = 0x02;
                clearLCD();
                allerun();
                d = longueur(gauche);
                decremente(d);
                affiche_entier(gauche);
                allerdeux();
                affiche(op);
                break;
        };
        case 0x02 : {
                op = c;
                allerdeux();
                affiche(op);
                break;
        };
        default :;
        }
```

```
}
void traiteegal() {
        int d;
        switch(etat) {
        case 0x00 : break;
        case 0x01 : break;
        case 0x02 : break;
        case 0x03 : {
                calcule();
                clearLCD();
                allerun();
                d = longueur(gauche) + longueur(droite) + 2;
                decremente(d);
                affiche_entier(gauche);
                affiche(op);
                affiche_entier(droite);
                allerdeux();
                d = longueur(resultat)  + 1;
                decremente(d);
                affiche(0xC0);
                affiche_entier(resultat);
                etat = 0x00;
                break;
                };
        default :;
        }
}
void calcule() {
/* met dans résultat le res de l'opération */
switch (op) {
        case 0x84 : { resultat = gauche + droite; break; }
        case 0x88 : { resultat = gauche - droite; break; }
        case 0x90 : { resultat = gauche * droite; break; }
        default :;
        }
}
void allerun(){
/* place le curseur en fin de ligne 1 */
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F0,0x8F);
}
void allerdeux(){
/* place le curseur en fin de ligne 2 */
        while((peekb(0xB5F0) & 0x80) == 0x80) {};
        pokeb(0xB5F0,0xCF);
}
void affiche(d)
unsigned char d;
{
        while((peekb(0xB5F0) & 0x80) == 0x80) {};

        switch(d){
                case 0x82 : pokeb(0xB5F1, 0x30); break; /* 0 */
                case 0x01 : pokeb(0xB5F1, 0x31); break; /* 1 */
```

```
                case 0x02 : pokeb(0xB5F1, 0x32); break; /* 2 */
                case 0x04 : pokeb(0xB5F1, 0x33); break; /* 3 */
                case 0x08 : pokeb(0xB5F1, 0x34); break; /* 4 */
                case 0x10 : pokeb(0xB5F1, 0x35); break; /* 5 */
                case 0x20 : pokeb(0xB5F1, 0x36); break; /* 6 */
                case 0x40 : pokeb(0xB5F1, 0x37); break; /* 7 */
                case 0x80 : pokeb(0xB5F1, 0x38); break; /* 8 */
                case 0x81 : pokeb(0xB5F1, 0x39); break; /* 9 */
                case 0x84 : pokeb(0xB5F1, 0x2B); break; /* + */
                case 0x88 : pokeb(0xB5F1, 0x2D); break; /* - */
                case 0x90 : pokeb(0xB5F1, 0x2A); break; /* * */
                case 0xA0 : break;                      /* efface */
                case 0xC0 : pokeb(0xB5F1, 0x3D); break; /* = */
                default  : break;
        }
        while(peekb(0xB5F4)!= 0x00){};
}
affiche_entier(n)
/* fait une serie de affiche pour afficher n */
int n;
{
        int p,a;
        if (n==0) {
                affiche(0x82);
                return;
                }
        if (n<0) {
                affiche(0x88); /* - */
                a = 0-n;
        } else a = n;
        p = 1;
        while (a / p > 10) {
                p = 10*p;
        }
        while (p>0) {
                switch(a/p) {
                        case 0 : { affiche(0x82); break; };
                        case 1 : { affiche(0x01); break; };
                        case 2 : { affiche(0x02); break; };
                        case 3 : { affiche(0x04); break; };
                        case 4 : { affiche(0x08); break; };
                        case 5 : { affiche(0x10); break; };
                        case 6 : { affiche(0x20); break; };
                        case 7 : { affiche(0x40); break; };
                        case 8 : { affiche(0x80); break; };
                        case 9 : { affiche(0x81); break; };
                        default : pokeb(0xB5F1, 0x3F); break;
                        }
                a = a%p;
                p = p/10;
        }
}
void delayms(ms)
unsigned int ms;
```

```
{
        unsigned int i,j;
        if(ms>0)
                {
                for(j=0;j<=ms;j++)
                        for(i=0;i<=DELAY1MS;i++);
                }
        return;
}
```