

Université Claude Bernard



Lyon 1

Projet ERPDGE

Guitare HERO

28 mai 2013



Table des matières

I. Rappel du cahier des charges	4
1 Le but final	4
2 Synoptique.....	5
II. Partie conception électronique	6
1 Le microcontrôleur : ARDUINO UNO SMD	6
2 La guitare	7
3 L’afficheur LCD.....	8
4 La gestion du son	9
5 La matrice à LEDs.....	10
III. Partie logiciel	13
1 Introduction.....	13
2 Fonction système <i>setup</i> et <i>loop</i>	13
3 Bibliothèque personnelle	13
3.1 Rappels sur les class	13
3.2 Class GH_somo	14
3.2.1 Déclaration	14
3.2.2 Méthodes	14
3.2.3 Attribues.....	14
3.3 Class GH_matrice.....	14
3.3.1 Déclaration	14
3.3.2 Attribues.....	14
3.3.3 Méthodes	15
4 Bibliothèque <i>LyquidCristal</i>	15
5 Clip vidéo et communication série	15
5.1 Introduction.....	15
5.2 Côté <i>Arduino</i>	15
5.3 Côté PC	16
6 Interruption externe.....	16
IV. Quick Start Guide	17
V. Conclusion	18

Guitare Hero

Annexe I : Les boutons poussoirs	19
Annexe II : L’afficheur LCD.....	20
Annexe III : Tableau justifiant le câblage du LCD	21
Annexe IV : Câblage du SOMO 14D	22
Annexe V : Matrice à LEDs 8x4	23
Annexe VI : Décodeur/démultiplexeur 3 vers 8	24
Annexe VII : Décodeur/démultiplexeur 2 vers 4	25
Annexe VIII : Courbes caractéristiques AHC et BC558	26
Annexe IX : Fichier Principal	27
Annexe X : Fichier lib_GH.cpp	30
Annexe XI : Fichier lib_GH.h	34
Annexe XII: Fichier Processing_GH.....	35
Annexe XIII : Tableau récapitulatif interfaçage <i>arduino</i>	36
Bibliographie	37

I. Rappel du cahier des charges

1 Le but final

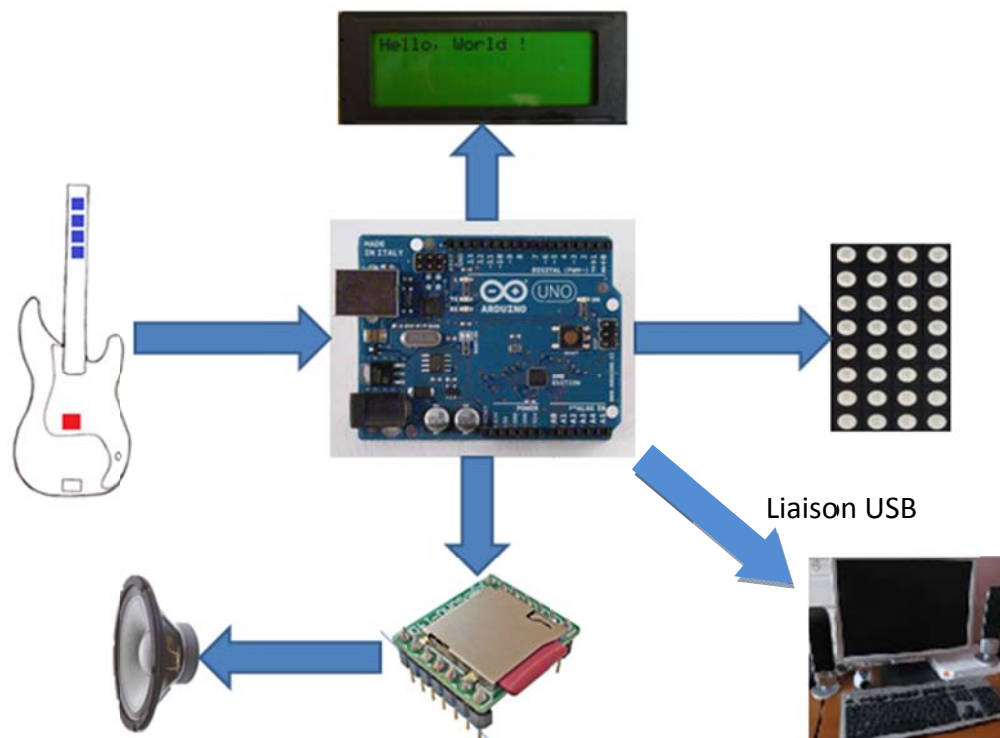
L'objectif était de réaliser le jeu GUITAR HERO. Il s'agissait de faire apparaître les combinaisons sur une matrice à LEDs. Ceci permettant au joueur d'anticiper l'appui sur les bons boutons poussoirs.

L'afficheur permet de guider le joueur dans le déroulement du jeu. Il permet aussi d'indiquer au joueur l'état du score ainsi que d'éventuels bonus.

Un haut-parleur est à disposition pour permettre l'écoute du son joué. L'étude de la gestion du son a été développée et son fonctionnement confirmé, nous lui avons rajouté la lecture du clip de la chanson.

Au final, notre cahier des charges a été modifié. Nous n'aurons donc plus trois niveaux mais une seule chanson. Nous avons réalisé le jeu flash disponible sur internet de GUITAR HERO II. La partition est identique. La lecture du clip de la musique sera lue sur un écran en même temps que le joueur compose.

2 Synoptique



Le cerveau du jeu est le microcontrôleur Arduino Uno. Il commande l'ensemble du système. La seule information qu'il reçoit c'est l'action de la guitare via ses boutons poussoirs. L'Arduino contrôle l'affichage sur l'écran LCD qui sert à guider le joueur dans le bon déroulement du projet. Il affichera en temps réel le score du joueur.

Le microcontrôleur contrôle la gestion de la matrice qui, via le défilement des LEDs, permet au joueur d'anticiper l'appui sur les bons boutons.

Une information est transmise pour la gestion du son géré par le module audio SOMO 14D. Même procédé pour l'affichage du clip via une liaison USB connecté à un ordinateur.

II. Partie conception électronique

1 Le microcontrôleur : ARDUINO UNO SMD



Le microcontrôleur est le cerveau de la gestion du jeu. Il est composé de 14 entrées/sorties digitales (dont 6 peuvent être utilisées en tant que sortie PWM) et de 6 entrées analogiques. Il dispose également d'un cristal à 16 MHz, une connexion USB, une prise jack d'alimentation, un en-tête ICSP ainsi que d'un bouton reset.

Il s'agit de connecter simplement la carte Arduino UNO SMD à un ordinateur à l'aide d'un câble USB ou bien de l'alimenter avec une batterie pour son démarrage.

En résumé :

Fonctionnement Tension 5V

Tension d'entrée (recommandée) 7-12V

Tension d'entrée (limites) 6-20V

14 Broches d'E/S numériques (dont 6 fournissent PWM)

6 Pins d'entrées Analogiques

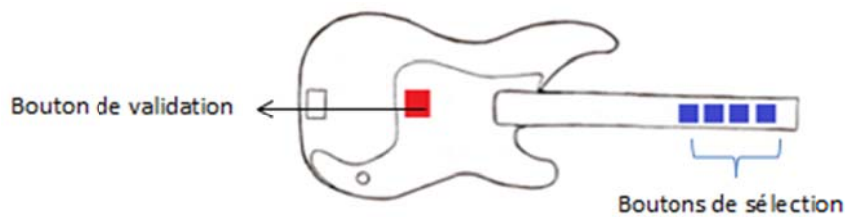
Courant DC 40 mA par Pin I/O

Courant 50 mA pour Pin 3.3V

Vitesse d'horloge 16 MHz

Pour l'utilisation de l'Arduino, on remarque que son site est très complet. Il permet une rapide prise en main de la partie programmation. Il en est de même pour la partie électronique, plusieurs tutoriels sont disponibles.

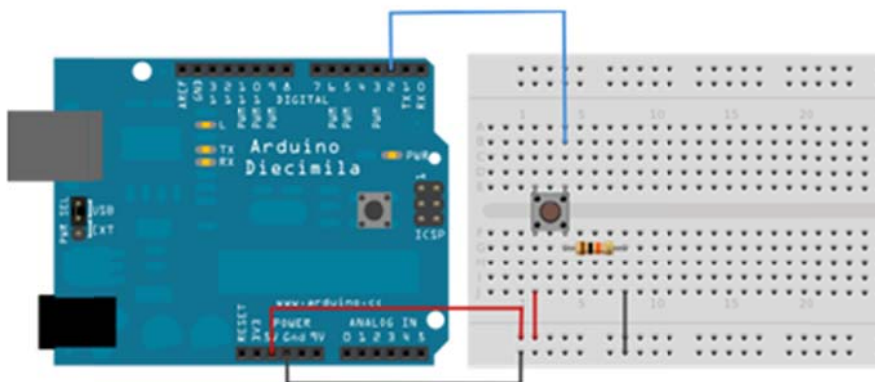
2 La guitare



Voici ci-dessous le montage d'un bouton-poussoir sur une plaque d'essais.

- La plaque d'essais est alimentée par la carte elle-même (5V et masse),
- Le bouton-poussoir est relié à 5V d'un côté et de l'autre à un pin de la carte ainsi qu'à la masse par l'intermédiaire d'une résistance de pull up de 10 K Ω .

5 volts sont envoyés dans le pin lorsque le bouton est pressé. Celui-ci est alors considéré comme une entrée du microcontrôleur.



Ce montage est le même pour les 5 boutons poussoirs qui constituent la guitare.
(Voir le schéma de câblage en ANNEXE I)

3 L'afficheur LCD



Le câblage et la programmation de l'afficheur LCD sont très simples. Il existe un tutoriel qui permet d'être guidé ainsi que de permettre de faire de simples tests d'essais. Le câblage est disponible en ANNEXE II.

En ANNEXE III, le tableau permet de justifier son câblage.

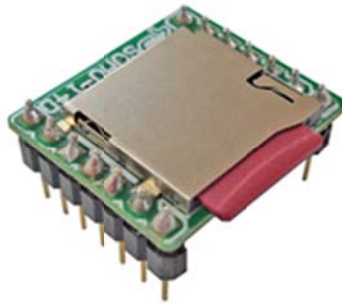
D'après la description du site d'Arduino, il faut créer une variable de type LiquidCrystal en définissant les broches utilisées avec le LCD et son mode de fonctionnement. L'afficheur LCD peut être contrôlé avec 4 ou 8 lignes de données. Si vous souhaitez utiliser l'afficheur en mode 4 bits, il ne faut pas utiliser les numéros des broches d0 à d3 et les laisser non-connectées comme dans notre cas. La broche RW peut-être connectée à la masse (0V) au lieu d'être connectée à une broche de la carte Arduino. De la même façon, si vous ne désirez pas l'utiliser, omettez-là dans les paramètres de la fonction LiquidCrystal.

A savoir que notre afficheur LCD est contrôlé uniquement avec 4 broches de données (d4 à d7) et 2 broches de commandes (RS et E), soit 6 broches seulement, contre 8 broches de données et 3 broches de commandes en connexion complète. On économise ainsi 5 broches de la carte Arduino qui resteront disponibles.

Le réglage du contraste est réglé par un potentiomètre 10 à 20K Ω . En revanche nous avons dû agir sur la tension pour obtenir un bon contraste et lui avons mis une alimentation de 6V.

Notre afficheur LCD est un produit du fabricant POWERTIP PC1602ARU-HWB-G-Q.

4 La gestion du son



La gestion du son est gérée par le module SOMO 14D. En ANNEXE IV, son schéma simplifié symbolise clairement son câblage.

Ce module sert à restituer des fichiers sonores préalablement enregistrés sur une carte microSD en FAT16, celle-ci étant lisible sur un PC acceptant le FAT16 DOS. On est limité par l'utilisation des cartes formatées d'une capacité de 2Go.

Dans un premier temps, il faut convertir vos fichiers WAVE (.wav) ou MP3 (.mp3) en fichier ADPCM (.ad4) reconnus par le module au moyen d'un logiciel disponible en téléchargement. Le logiciel « Audacity » permet de couper la musique en fonction du nombre de minute que l'on veut conserver.

Le logiciel « UsbRecorder » sert à la conversion des fichiers.

Ensuite, il est nécessaire de stocker les fichiers sur une carte microSD et de l'insérer dans le connecteur du "SOMO-14D" prévu à cet effet. Enfin il faudra piloter la restitution des messages audios via votre microcontrôleur en envoyant des ordres très simple via un bus série 2 fils (DATA - CLOCK) au module "SOMO-14D".

Les différentes commandes accessibles permettent de sélectionner un fichier audio, de lancer sa lecture, de la mettre en pause, de la stopper ou encore de modifier le volume sonore. Il est possible de rajouter une résistance de 10 Ohm en série avec le Haut-Parleur pour abaisser le volume sonore.

On n'oubliera pas d'adjoindre, comme indiqué sur le schéma, la capacité de découplage de 220uF minimum à la broche 8.

Il faut prendre en compte que le module n'accepte que des signaux à 3,3V.

Nous utilisons seulement les Pins de DATA et de la CLOCK.

5 La matrice à LEDs



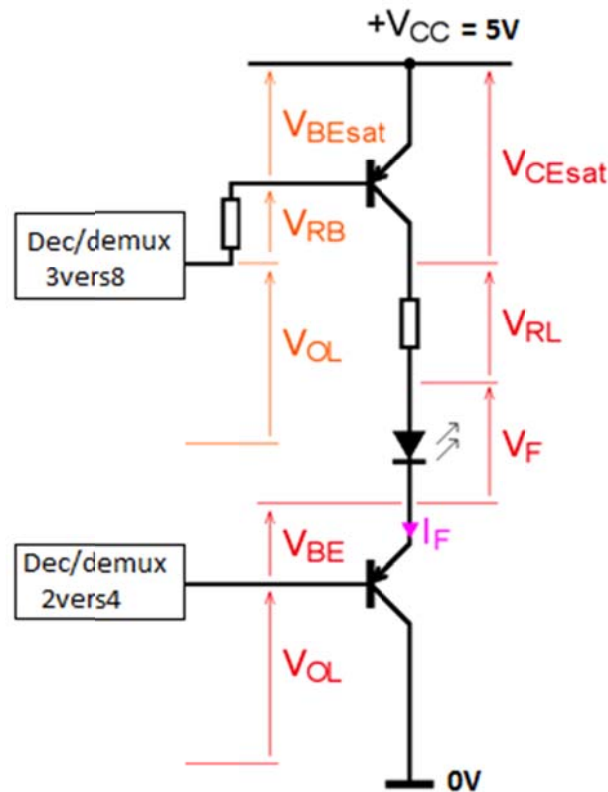
La matrice à LEDs, étant composée de 4 colonnes et 8 lignes de Leds, a nécessité la réduction de nombre de broches utiles pour le bon fonctionnement de la matrice (Annexe V).

C'est pourquoi nous avons opté pour l'utilisation d'un décodeur/démultiplexeur 3 vers 8 pour le pilotage des 8 lignes ce qui nous permet de passer de 8 à 3 broches. Il suffit donc d'envoyer un mot binaire sur les 3 broches du décodeur/démultiplexeur pour lui indiquer la ligne à piloter. Ce principe est le même pour la gestion des colonnes, nous avons utilisé un décodeur/démultiplexeur 2 vers 4 qui nous permet de passer de 4 à 2 broches seulement.

Comme l'indique la datasheet (ANNEXES VI et VII), le décodeur/démultiplexeur met par défaut les broches de sorties à l'état HAUT. Lorsqu'on envoie un mot binaire dans le décodeur/démultiplexeur, celui-ci met la broche voulu à l'état BAS. C'est pourquoi nous avons utilisé des transistors PNP qui deviennent passants lorsque la commande reçoit un zéro sur sa base.

Notre système permet de piloter une LED à la fois. Nous agissons donc sur la partie programmation pour tromper l'œil. En effet, à l'aide de temporisation, l'œil percevra que plusieurs LEDs sont allumées en même temps alors que physiquement elles ne sont allumées qu'une par une.

Voici un schéma simplifié du fonctionnement de notre matrice pour l'allumage d'une LED :



D'après les courbes caractéristiques en ANNEXE VIII :

Concernant le PNP inférieur, la courbe en bas à droite indique que V_{BE} devrait être compris entre $V_{BEon}=0,78V$ et $V_{BEsat}=0,83V$.

La valeur de V_{OL} est assez faible, inférieure à $0,1V$. Sa valeur exacte est négligeable devant la tension d'alimentation du couple résistance+led que l'on cherche.

Par ailleurs, la courbe du haut à droite fait apparaître que pour $I_C=20mA$ et $V_{CE}>0,6V$ (ce qui est bien le cas, puisque $V_{CE} = V_{BE}+V_{OL} > 0,78V$) le courant I_B est indépendant de V_{CE} et égal à $I_B=I_C/\beta$.

$I_B \approx 0,057mA$ correspond ici à un gain statique $\beta \approx 350$, soit plutôt la valeur typique d'un BC557C, mais on peut admettre que même avec le gain minimal $\beta \approx 120$ d'un BC557A on devrait avoir $I_B < 0,17mA$

Le courbe du 74AHC à gauche suggère que pour $I_{OL}=I_B < 0,17mA$, V_{OL} devrait être très faible, inférieure à $0,01V$. Le tableau de la datasheet garantit quant à lui que V_{OL} restera inférieure à $0,1V$.

On sait que $V_{BE}+V_{OL} \approx 1V$ et $V_{CEsat} \approx 0,1V$

D'après la loi des mailles :

$$V_{CC} = V_{CEsat} + V_{RL} + V_F + V_{BE} + V_{OL}$$

qui mène au calcul de la résistance de limitation de la LED.

$$\text{soit } V_{RL} = V_{CC} - V_{CEsat} - V_F - V_{BE} - V_{OL} = 5 - 0,1 - 2 - 1 = 1,9V$$

$$\text{donc } R_{Led} = V_{RL} / I_{Led} = 1,9 / 0,02 = 95\Omega \quad \text{donc } 100\Omega$$

Concernant le PNP supérieur, sur les courbes on trouve $V_{CEsat}=0,1V$ et $V_{BEsat}=0,83V$ à $I_C=20mA$, et pour $I_{OL}=I_{Bsat}=1$ à $2mA$ on a $V_{OL}'<0,05$ à $0,1V$ d'après le 74AHC

D'après la loi des mailles :

$$V_{CC} = V_{BEsat} + V_{RB} + V_{OL}$$

qui mène au calcul de la résistance de base du PNP supérieur.

$$\text{Soit } V_{RB} = V_{CC} - V_{BEsat} - V_{OL}' = 5 - 0,83 - 0,1 = 4.07V$$

$$\text{Donc } R_B = V_{RB} / I_b = 4.07 / 0.001 = 4 \text{ K}\Omega \quad \text{ou } R_b = V_{rb} / I_b = 4.07 / 0.002 = 2 \text{ K}\Omega$$

Donc R_b est compris entre $2K\Omega$ et $4K\Omega$ j'ai donc pris $3.3K\Omega$

III. Partie logiciel

1 Introduction

Le programme gérant les différents éléments a été fait dans un langage proche du C propre aux cartes *arduino*. La bibliothèque personnelle **GH_lib** quant à elle a été faite dans le langage C++. L'architecture général du programme consiste en une première fonction obligatoire *void setup ()* contenant l'initialisation des différentes broches E/S disponibles sur la carte *arduino uno*. Suit d'une seconde fonction obligatoire *void loop ()* qui contient le programme principal. Nous trouvons également la fonction *void test ()* appelée lorsque une interruption externe se produit. Et enfin nous trouvons deux class : **GH_somo** et **GH_matrice** dans la librairie **GH_lib**.

2 Fonction système *setup* et *loop*

Les initialisations des différents composants se font dans la fonction *setup*. Notamment les E/S via la fonction *pinMode*. Le module *somo 14D* via *reset_somo*. L'afficheur *Lcd* via *begin*. La fonction *loop* correspond à la fonction *main* en C à la différence que celle-ci comme son nom l'indique s'exécute en boucle. C'est-à-dire qu'une fois arrivé à la fin le programme recommence du début de la fonction *loop*. Ce que fait cette fonction dans le programme, c'est de lancer les différentes fonctions à sa disposition. Ces fonctions se trouvent réparties dans deux class distincts. La class **GH_somo** et la class **GH_matrice**. L'une étant chargée de tous ce qui est relatif au module SOMO 14D et l'autre de tous ce qui concerne la matrice de LEDs.

3 Bibliothèque personnelle

3.1 Rappels sur les class

Tout d'abords voyons ce qu'est une class : Les class sont à la base de la programmation orientée objet. Une class ressemble beaucoup aux structures. Mais en plus de pouvoir contenir un ensemble de variables ici nommées attribues ; elles peuvent également contenir des fonctions que l'on nomme méthodes. Ces méthodes et attribues sont réparties en deux groupes :

- Un groupe dit *public* ce qui signifie que ces méthodes et attribues sont accessibles à tout moment dans le programme.

- Le second groupe est dit *private* c'est-à-dire que ces méthodes et attribues ci ne sont pas accessibles dans le programme. Seul les méthodes de la class y ont accès.

Une class contient également ce qu'on appelle un constructeur. Sa fonction est de créer un objet de la class en question. Et d'initialiser ces différents attribues.

3.2 Class GH_somo

3.2.1 Déclaration

Intéressons-nous désormais plus en détail à la class **GH_somo**. Sa déclaration est faite dans l'en-tête **GH_lib.h**. Son implémentation se trouve dans le fichier **GH_lib.cpp**. Cette class comporte deux méthodes et trois attribues ainsi qu'un constructeur.

3.2.2 Méthodes

En ce qui concerne les méthodes, le premier se nomme *reset_somo*. Sa fonction est de réinitialiser le *somo 14D* pour qu'il puisse être utilisé par la suite. La seconde méthode sert à envoyer des commandes à celui-ci tel que l'adresse du son pour démarrer la lecture de 0x0000 à 0xFFFF0 ou des arguments systèmes de 0xFFFF1 à 0xFFFF ceux-ci permettent de contrôler le volume ou mettre fin à la lecture en cours.

3.2.3 Attribues

Enfin les trois attribues correspondent aux broches de sorties de l'*arduino* utilisées pour piloter le module *somo 14d*. Il y a la broche *enable*, la broche *clock* ainsi que la broche *data*. La broche *clock* synchronise les transmissions entre la carte *arduino* et le module *somo*. C'est sur la broche *data* que les données sont transmises (argument de la méthode *démarrer_son*). La dernière broche *enable* est utilisée lors du reset du module *somo*.

3.3 Class GH_matrice

3.3.1 Déclaration

Passons maintenant à la class **GH_matrice**. Tout comme la class **GH_somo**, sa déclaration est faite dans le fichier **GH_lib.h** et son implémentation est fait dans le fichier **GH_lib.cpp**. Cette class contient son constructeur, deux méthodes et huit attribues dont un est public les autres étant privées.

3.3.2 Attribues

Commençons par nous intéresser aux attribues. Ils y en a cinq qui correspondent aux broches de sorties de l'*arduino* pilotant la matrice (*_c_1*, *_c_2*, *_l_1*, ...). Il y a deux broches pour gérer les colonnes et trois pour les lignes. Les lignes et les colonnes fonctionnent sur le même principe.

Colonne	<i>_c_1</i>	<i>_c_2</i>
1	0	0
2	1	0
3	0	1
4	1	1

Tableau 1 : Récapitulatifs des sorties activées en fonction de la colonne sélectionné

L'attribue *TabS* contient la séquence à afficher. Sa taille est de 410 octets se qui correspond à 820 lignes différentes pour une durée d'affichage d'environ trois minutes (durée de la chanson **woman**). Il y a également l'attribue *M*, un tableau de huit lignes et quatre colonnes représentant à tout moment l'état de la matrice DELs. Et enfin la variable public *ligne* contenant la valeur que doit reproduire l'utilisateur pour marquer des points.

3.3.3 Méthodes

La première méthode, *actualisation_matrice* a pour objectif de récupérer dans le tableau *TabS* contenant la séquence à faire défiler la nouvelle image de la matrice à afficher et de la placer dans le tableau *M*. Elle actualise aussi la valeur de l'attribue ligne. La seconde méthode affiche l'image pendant une période d'environ 200ms. Elle active successivement chacune des DELs devant être allumée pour une durée de 1ms.

4 Bibliothèque *LyquidCristal*

La bibliothèque *lyquidCristal* met à notre disposition un certain nombre de fonctions. Nous en utilisons quatre. La première *begin* permet d'initialiser l'afficheur lcd. Elle prend pour arguments le nombre de colonnes et le nombre de lignes. La seconde *print* envoie à l'afficheur lcd le texte à afficher. Une autre fonction, *clear* pour effacer tout ce qui est présent sur l'écran lcd. Et enfin la fonction *setCursor* qui place le curseur à l'endroit indiqué par ces paramètres d'entrées.

5 Clip vidéo et communication série

5.1 Introduction

Le lancement du clip vidéo de la chanson **woman** a été un vrai défi. En effet il a fallu mettre en place une communication série entre la carte *arduino* et un ordinateur. Avec du côté du PC un programme s'exécutant sous *processing*. *Processing* est un langage basé sur le java. Il permet une mise en œuvre simplifiée de tout ce qui est graphique ou animation. Donc très pratique pour la lecture de flux vidéo.

5.2 Côté *Arduino*

La carte *arduino uno* contient un port de communication série. Il utilise les broches 0 (RX) et 1 (TX) via le port USB pour communiquer avec le PC. Or ces broches sont également utilisées pour la transmission de données avec l'afficheur lcd. C'est pourquoi après la communication on utilise la fonction *end* qui met un terme à toutes les communications série. Cela rend à nouveau disponible les broches 0 et 1 de la carte *arduino* pour l'afficheur lcd. On utilise également la fonction *begin* pour initialiser le port série. La transmission est faite elle via la fonction *write* qui écrit la valeur de son argument sur le port USB série.

5.3 Côté PC

Un programme sous *processing* tourne sur l'ordinateur. Celui-ci se trouve en écoute du port série. Lorsqu' il reçoit l'ordre de *l'arduino* il lance la lecture du clip vidéo. Pour faire ceci on utilise deux librairies. Une pour la gestion de la communication et une autre pour la gestion de la vidéo. En ce qui concerne la communication on utilise la fonction *read* pour lire sur le port série. Dès lors que le caractère reçu est celui attendu en l'occurrence 1. La fonction *play* de la librairie vidéo s'exécute qui a pour effet de démarrer la lecture de la vidéo.

6 Interruption externe

Seules les broches 2 et 3 de *l'arduino uno* peuvent être utilisées en interruption externe. Nous avons besoin que d'une seule. Elle est connectée au bouton simulant le médiateur côté guitare et à la broche 2 côté *arduino*. Chaque interruption est perçut lors d'un passage de l'état haut à l'état bas. Correspondant à l'argument *FALLING* de la fonction *attachInterrupt*. A cette interruption est associée une fonction : *test* qui est appelé lors de chaque interruption. Elle est en charge de vérifier si ce que le joueur a joué correspond à l'accord attendu. Si c'est le cas alors le score et les bonus sont actualisés. Sinon le score ne s'incrémente pas et les bonus sont remis à zéro.

IV. Quick Start Guide

1. Allumer l'ordinateur avec ses haut-parleurs
2. Il suffit de connecter l'Arduino à l'ordinateur via sa connexion USB



3. Suivez les instructions indiquées sur l'écran LCD



4. Ensuite il s'agira de vous amuser un maximum sur un son de Wolfmother – Woman



V. Conclusion

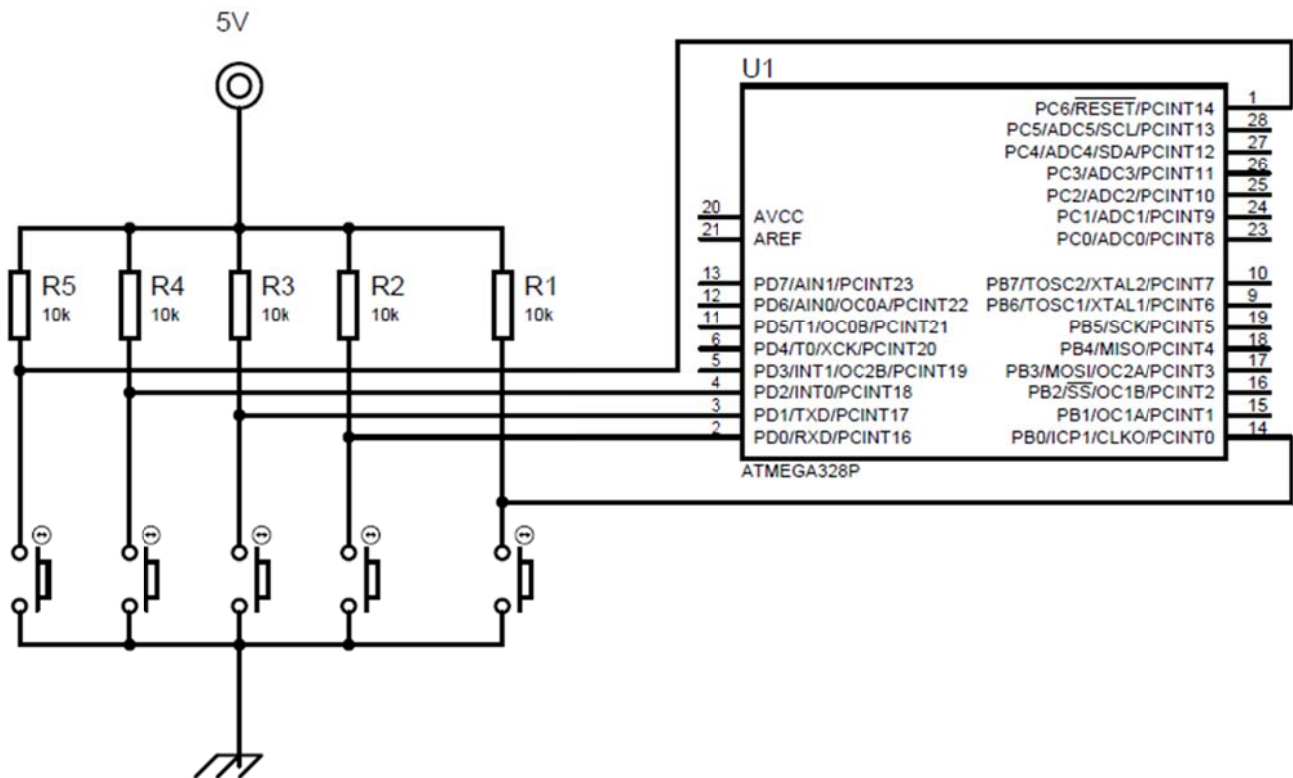
Nous sommes fiers d'avoir pu mener à terme notre projet. La tâche fut assez compliquée. Beaucoup d'éléments se sont greffés à chaque étape de notre progression. Nous étions toujours confiants de notre progression pour au final être freinés par des petites erreurs inattendues tant au niveau de la programmation qu'au niveau l'électronique. Mais nous sommes restés soudés tout en évitant de s'éparpiller.

Nous nous sommes très bien répartis les tâches ce qui nous a permis de rendre le projet dans les temps souhaités.

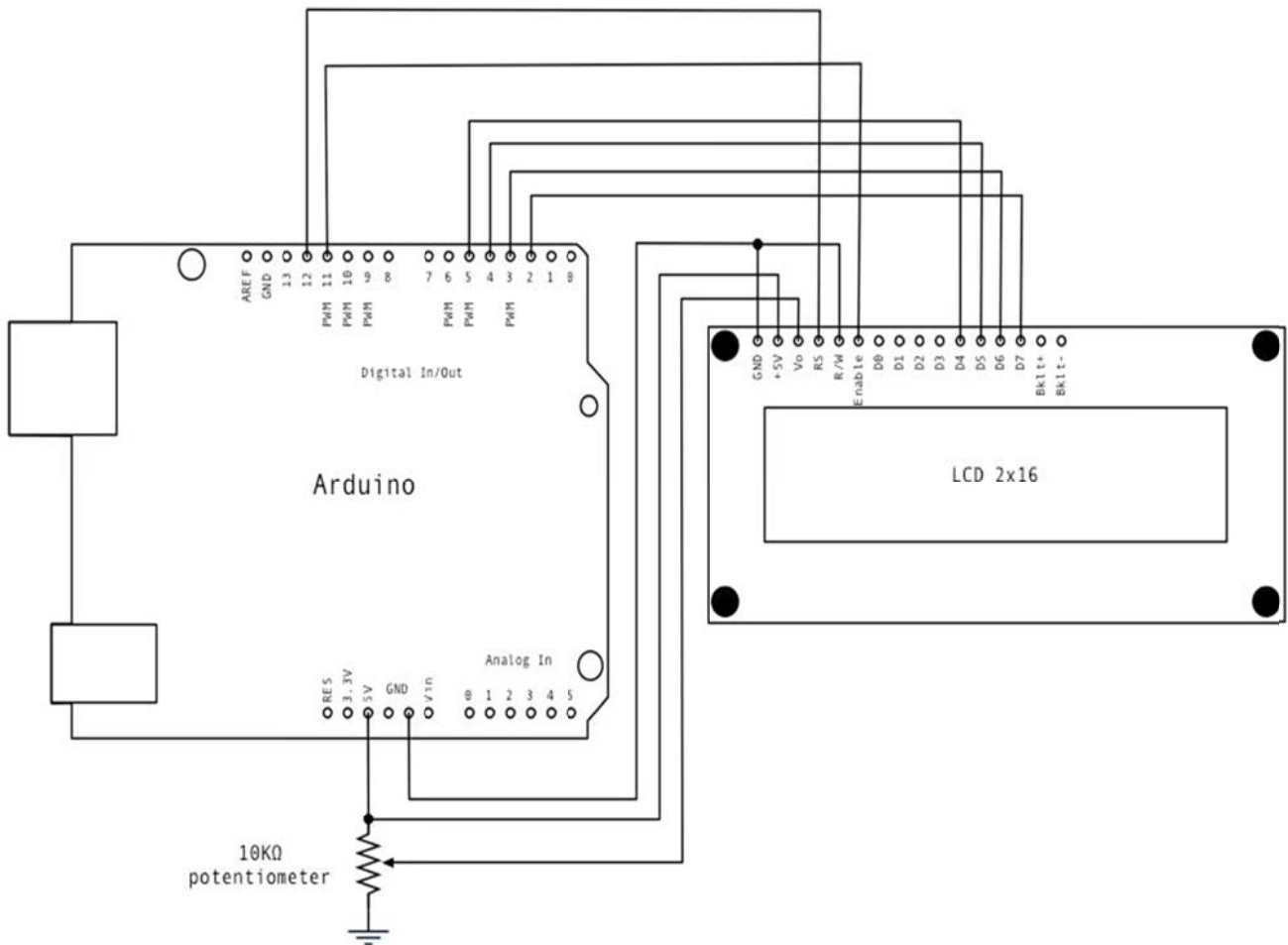
Ce projet nous a permis de nous organiser dans un travail d'équipe très complémentaire. Il nous a aussi permis d'apprendre dans le domaine de la conception électronique et informatique. Nous avons eu la chance de pouvoir découvrir du nouveau matériel, d'être responsable de notre commande de matériel. Nous nous sommes confrontés aux contraintes économiques et temporelles puisqu'il a fallu prendre en compte le délai de réception du matériel.

Nous avons été mis en situation d'entreprise, avec une autonomie complète et un travail d'équipe aussi bien entre binôme qu'entre les autres binômes.

Annexe I : Les boutons poussoirs



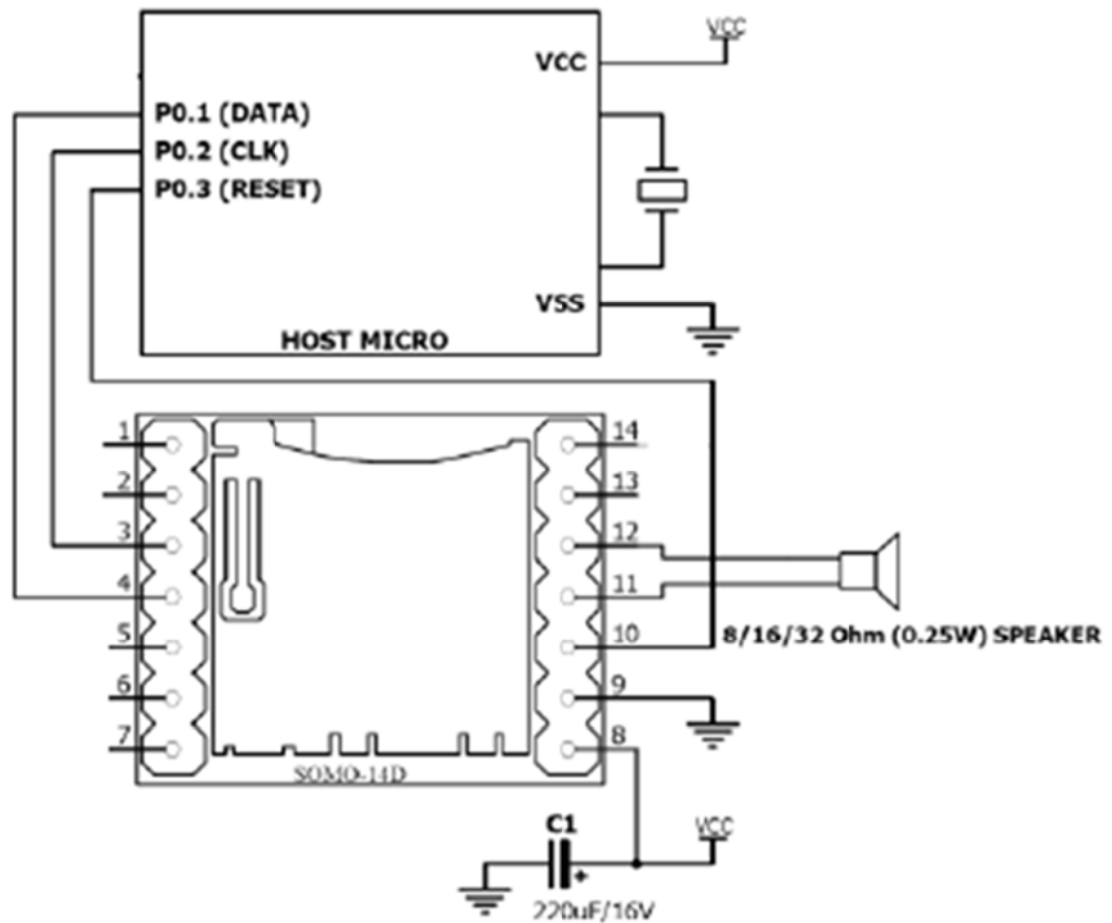
Annexe II : L'afficheur LCD



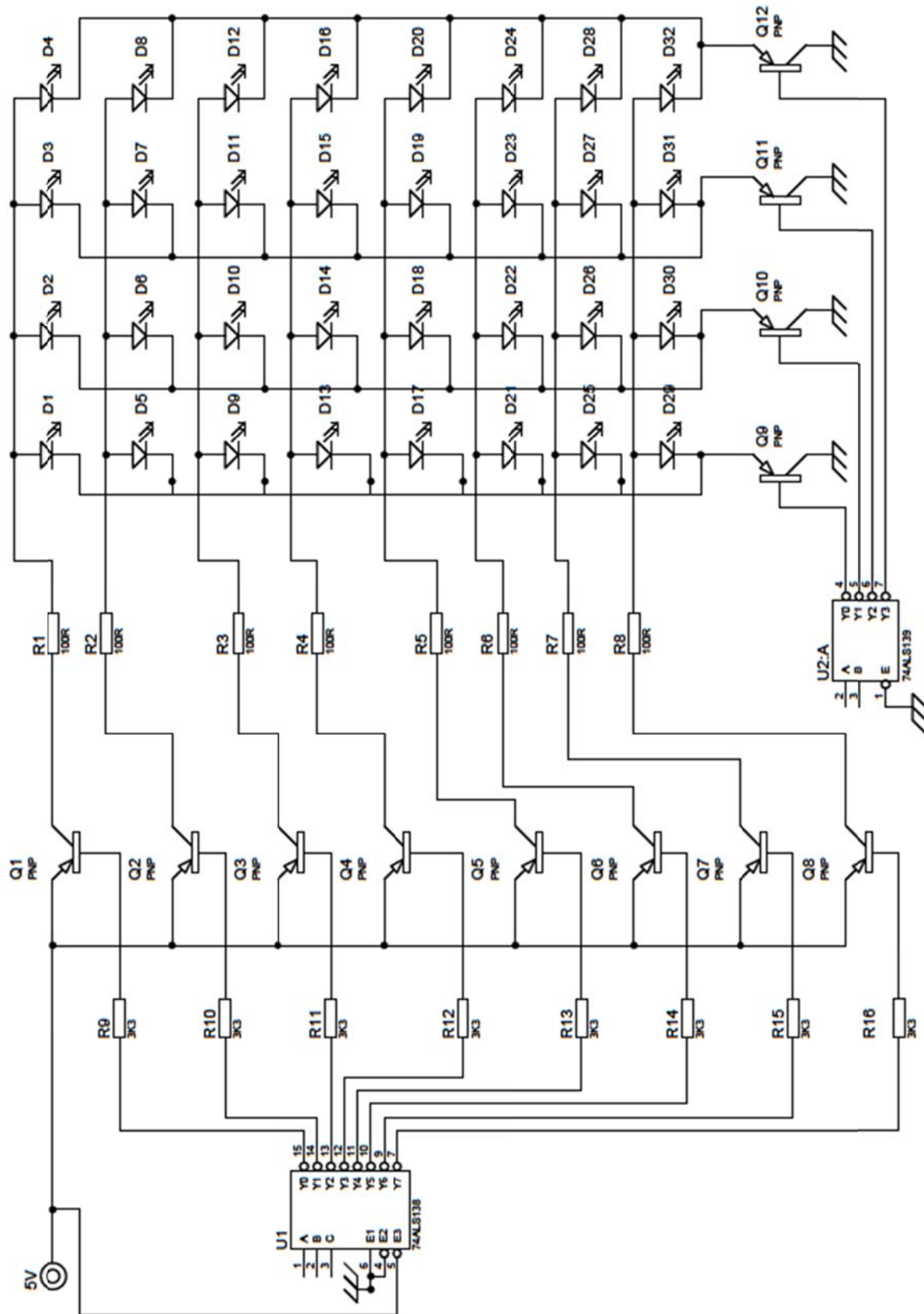
Annexe III : Tableau justifiant le câblage du LCD

Pin No.	Symbol	Function
1	VSS	Signal ground (GND)
2	VDD	Power Supply for logic ($VDD > VSS$)
3	VO	Operating Voltage for LCD (variable) (Réglage du contraste)
4	RS	Register Selection input High = Data register Low = Instruction register (for write) Busy flag address counter (for read)
5	$\overline{R/W}$	$\overline{R/W}$ signal input is used to select the read/write mode High = Read mode, Low = Write mode
6	E	Start enable signal to read or write the data
11~14	DB4~DB7	Four high order bi-directional three-state data bus lines. Used for data transfer between the MPU and the LCD module. DB7 can be used as a busy flag.

Annexe IV : Câblage du SOMO 14D



Annexe V : Matrice à LEDs 8x4

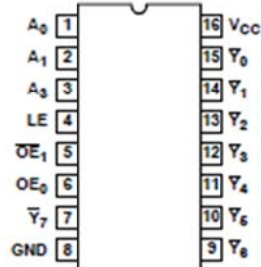


Annexe VI : Décodeur/démultiplexeur 3 vers 8

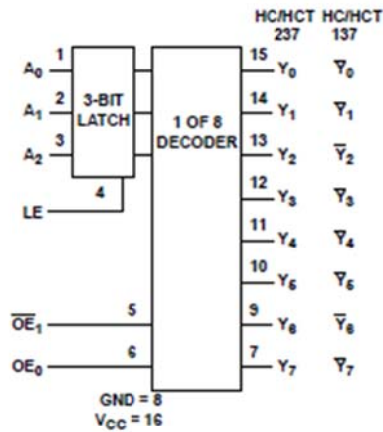
CD74HC137, CD74HCT137, CD54HC237, CD74HC237, CD74HCT237

Pinout

CD54HC237 (CERDIP)
CD74HC137 (PDIP, TSSOP)
CD74HCT137 (PDIP, SOIC)
CD74HC237 (PDIP, SOIC, SOP, TSSOP)
CD74HCT237 (PDIP)
TOP VIEW



Functional Diagram



'HC137, 'HCT137 TRUTH TABLE

INPUTS						OUTPUTS							
LE	OE ₀	OE ₁	A ₂	A ₁	A ₀	Y ₀	Y ₁	Y ₂	Y ₃	Y ₄	Y ₅	Y ₆	Y ₇
X	X	H	X	X	X	H	H	H	H	H	H	H	H
X	L	X	X	X	X	H	H	H	H	H	H	H	H
L	H	L	L	L	L	L	H	H	H	H	H	H	H
L	H	L	L	L	H	H	L	H	H	H	H	H	H
L	H	L	L	H	L	H	H	L	H	H	H	H	H
L	H	L	L	H	H	H	H	L	H	H	H	H	H
L	H	L	H	L	L	H	H	H	L	H	H	H	H
L	H	L	H	L	H	H	H	H	L	H	H	H	H
L	H	L	H	H	L	H	H	H	H	L	H	H	H
L	H	L	H	H	H	H	H	H	H	L	H	H	H
H	H	L	X	X	X	Depends upon the address previously applied while LE was at a logic low.							

H = High Voltage Level, L = Low Voltage Level, X = Don't Care

'HC237, 'HCT237 TRUTH TABLE

INPUTS						OUTPUTS							
LE	OE ₀	OE ₁	A ₂	A ₁	A ₀	Y ₀	Y ₁	Y ₂	Y ₃	Y ₄	Y ₅	Y ₆	Y ₇
X	X	H	X	X	X	L	L	L	L	L	L	L	L
X	L	X	X	X	X	L	L	L	L	L	L	L	L
L	H	L	L	L	L	H	L	L	L	L	L	L	L
L	H	L	L	L	H	L	H	L	L	L	L	L	L
L	H	L	L	H	L	L	L	H	L	L	L	L	L
L	H	L	L	H	H	L	L	L	H	L	L	L	L
L	H	L	H	L	L	L	L	L	L	H	L	L	L
L	H	L	H	L	H	L	L	L	L	L	H	L	L
L	H	L	H	H	L	L	L	L	L	L	L	H	L
L	H	L	H	H	H	L	L	L	L	L	L	L	H
H	H	L	X	X	X	Depends upon the address previously applied while LE was at a logic low.							

H = High Voltage Level, L = Low Voltage Level, X = Don't Care

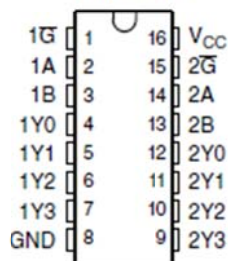
Annexe VII : Décodeur/démultiplexeur 2 vers 4

SN54AHC139, SN74AHC139 DUAL 2-LINE TO 4-LINE DECODERS/DEMULIPLEXERS

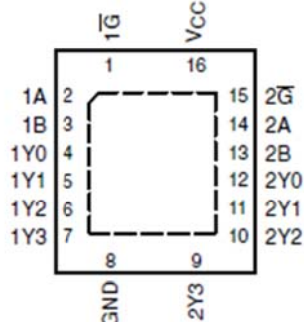
SCLS259K – DECEMBER 1995 – REVISED MARCH 2003

- Operating Range 2-V to 5.5-V V_{CC}
- Designed Specifically for High-Speed Memory Decoders and Data-Transmission Systems
- Incorporate Two Enable Inputs to Simplify Cascading and/or Data Reception
- Latch-Up Performance Exceeds 250 mA Per JESD 17
- ESD Protection Exceeds JESD 22
 - 2000-V Human-Body Model (A114-A)
 - 200-V Machine Model (A115-A)
 - 1000-V Charged-Device Model (C101)

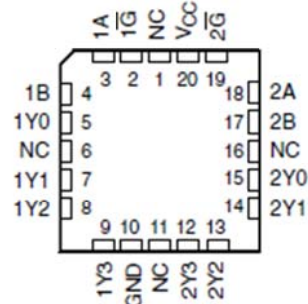
SN54AHC139... J OR W PACKAGE
SN74AHC139... D, DB, DGV, N, NS
OR PW PACKAGE
(TOP VIEW)



SN74AHC139... RGY PACKAGE
(TOP VIEW)



SN54AHC139... FK PACKAGE
(TOP VIEW)



NC – No internal connection

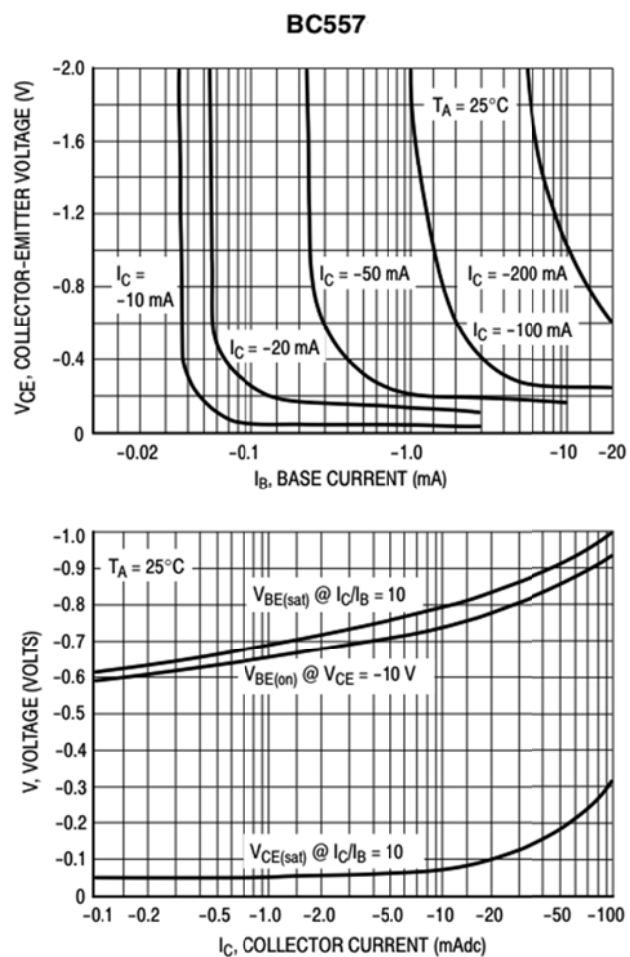
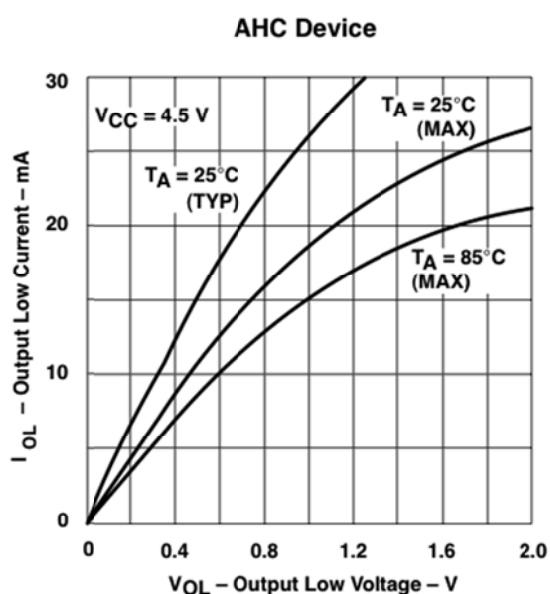
description/ordering information (continued)

The active-low enable (\overline{G}) input can be used as a data line in demultiplexing applications. These decoders/demultiplexers feature fully buffered inputs, each of which represents only one normalized load to its driving circuit.

FUNCTION TABLE
(each decoder/demultiplexer)

\overline{G}	INPUTS		OUTPUTS			
	SELECT		Y0	Y1	Y2	Y3
	B	A				
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

Annexe VIII : Courbes caractéristiques AHC et BC558



Annexe IX : Fichier Principal

```

/*****
Etude et Réalisation d'un Projet dans le Domaine du Génie Electrique
Guitar Hero
*****/

/*****|
|*Fichier principale                                *|
|*void setup ()                                    *|
|*void loop ()                                      *|
*****/

/*librairies*/
#include <LiquidCrystal.h>    //lib afficheur LCD
#include "lib_GH.h"           //lib personnelle (somo - matrice)

byte TabS[420]={0x60,0x60,0x60,0x10,0x60,0x60,0x60,0x10,0x60,0x60,0x60,0x10,
0x55,0x50,0x33,0x30,0x60,0x60,0x60,0x10,0x60,0x60,0x60,0x10,0x60,0x60,0x60,
0x10,0x55,0x50,0x33,0x30,0x60,0x60,0x60,0x10,0x60,0x60,0x60,0x10,0x60,0x60,
0x60,0x10,0x55,0x50,0x33,0x30,0x60,0x60,0x60,0x10,0x60,0x60,0x60,0x10,0x60,
0x60,0x60,0x10,0x55,0x50,0x33,0x30,0x55,0x55,0x55,0x50,0xAA,0xAA,0xAA,0xA0,
0x55,0x55,0x55,0x50,0xAA,0xAA,0xAA,0xA0,0x66,0x66,0x66,0x60,0xCC,0xCC,0xCC,
0xCC,0xCC,0xCC,0xCC,0xC0,0x60,0x60,0x60,0x10,0x60,0x60,0x60,0x10,0x60,0x60,
0x60,0x10,0x55,0x50,0x33,0x30,0x60,0x60,0x60,0x10,0x60,0x60,0x60,0x10,0x60,
0x60,0x60,0x10,0x55,0x50,0x33,0x30,0x60,0x60,0x60,0x10,0x60,0x60,0x60,0x10,
0x60,0x60,0x60,0x10,0x55,0x50,0x33,0x30,0x55,0x55,0x55,0x50,0xAA,0xAA,0xAA,
0xA0,0x55,0x55,0x55,0x50,0xAA,0xAA,0xAA,0xA0,0x66,0x66,0x66,0x60,0xCC,0xCC,
0xCC,0xCC,0xCC,0xCC,0xC0,0x40,0x40,0x20,0x20,0x40,0x40,0x20,0x10,0x40,
0x40,0x20,0x20,0x40,0x40,0x20,0x10,0x40,0x40,0x20,0x20,0x40,0x40,0x20,0x10,
0x40,0x40,0x20,0x20,0x40,0x40,0x20,0x10,0x10,0x22,0x10,0x22,0x10,0x22,0x10,
0x44,0x10,0x10,0x22,0x10,0x22,0x10,0x44,0x10,0x10,0x22,0x20,0x44,0x20,0x44,
0x20,0x44,0x20,0x88,0x40,0x80,0x22,0x22,0x20,0x44,0x44,0x44,0x04,0x01,0x00,
0x44,0x44,0x04,0x80,0x08,0x00,0x42,0x21,0x11,0x11,0x44,0x02,0x01,0x10,0x11,
0x11,0x11,0x10,0x22,0x22,0x22,0x22,0x20,0x44,0x44,0x44,0x44,0x40,0x88,0x88,
0x88,0x88,0x88,0x44,0x22,0x11,0x40,0x00,0x11,0x01,0x04,0x02,0x00,0x44,0x40,
0x44,0x40,0x40,0x80,0x80,0x40,0x40,0x40,0x20,0x11,0x20,0x20,0x10,0x10,0x20,
0x20,0x40,0x40,0x80,0x80,0x40,0x10,0x22,0x20,0x44,0x40,0x88,0x20,0x44,0x40,
0x80,0x80,0x80,0x80,0x80,0x80,0x40,0x40,0x40,0x20,0x20,0x55,0x50,0x33,0x30,
0x60,0x60,0x60,0x10,0x60,0x60,0x60,0x10,0x60,0x60,0x60,0x10,0x55,0x50,0x33,
0x30,0x60,0x60,0x60,0x10,0x60,0x60,0x60,0x10,0x60,0x60,0x60,0x10,0x55,0x50,
0x33,0x30,0x60,0x60,0x60,0x10,0x60,0x60,0x60,0x10,0x60,0x60,0x60,0x10,0x55,
0x50,0x33,0x30,0x55,0x55,0x55,0x50,0xAA,0xAA,0xAA,0xA0,0x55,0x55,0x55,0x50,
0xAA,0xAA,0xAA,0xA0,0x66,0x66,0x66,0x60,0xCC,0xCC,0xCC,0xCC,0xCC,0xCC,0xCC,
0x04,0x04,0x02,0x02,0x04,0x04,0x02,0x01,0x00,0x00,0x00};

const int c_1=6;           //broches matrice
const int c_2=7;
const int l_1=8;
const int l_2=9;
const int l_3=10;
const int somo_clock = 11; //broches somo
const int somo_data = 12;
const int somo_reset = 18;
const int Bouton_1 = 14;   //broches guitare
const int Bouton_2 = 15;
const int Bouton_3 = 16;
const int Bouton_4 = 17;
const int Bouton_val = 0;
const int LCD_RS = 5;      //broches lcd

```

```
const int LCD_E = 13;
const int LCD_DB4 = 4;
const int LCD_DB5 = 3;
const int LCD_DB6 = 1;
const int LCD_DB7 = 0;
volatile int score = 0;
volatile int bonus = 0;

/*Déclaration class*/
GH_matrice matrice(c_1,c_2,l_1,l_2,l_3,TabS);
GH_somo somo(somo_clock,somo_data,somo_reset);
LiquidCrystal lcd(LCD_RS,LCD_E,LCD_DB4,LCD_DB5,LCD_DB6,LCD_DB7);

void setup ()
{
  /*initialisation Broches E/S*/
  pinMode(c_1, OUTPUT);
  pinMode(c_2, OUTPUT);
  pinMode(l_1, OUTPUT);
  pinMode(l_2, OUTPUT);
  pinMode(l_3, OUTPUT);
  pinMode(somo_clock, OUTPUT);
  pinMode(somo_data, OUTPUT);
  pinMode(somo_reset, OUTPUT);
  pinMode(Bouton_1, INPUT);
  pinMode(Bouton_2, INPUT);
  pinMode(Bouton_3, INPUT);
  pinMode(Bouton_4, INPUT);

  lcd.begin(16, 2);          //initialisation afficheur lcd

  somo.Reset_somo();         //initialisation module somo

  attachInterrupt(0, test, FALLING); // attache l'interruption externe n°0
}

void loop ()
{
  int i = 0;
  int niveau = 0;
  static boolean a=1;

  if(a){
    /*menu bienvenu*/
    lcd.clear();
    lcd.print("Guitare Hero");
    lcd.setCursor(0,1);
    lcd.print("Appuyer sur 1");
    do{
      if(digitalRead(Bouton_1)){niveau = 1;}
      if(digitalRead(Bouton_2)){niveau = 2;}
      if(digitalRead(Bouton_3)){niveau = 3;}
    }while (niveau==0);
    lcd.clear();

    Serial.begin(9600);      //initialisation communication série

    Serial.write(1);         //écriture de la valeur 1 sur le port série

    delay(4800);             //attente de 4.8s synchronisation
  }
}
```

```
somo.demarrer_son(0xFFFF);          /*arrêt son*/
somo.demarrer_son(0x0005);          /*lance son*/
delay(10000);                        //attente 10s synchronisation
Serial.end();                        //arrêt de toutes les communications série

lcd.clear();
lcd.print("Women Wolfmother");
for(i=0;i<412;)                      //boucle principale du temps de la séquence jouer
{
    i = matrice.actualisation_matrice(i); //actualise la matrice à afficher
    matrice.affichage_matrice();          //affiche la matrice
    lcd.setCursor(6,1);
    lcd.print(score);                    //affichage score sur le lcd
}
somo.demarrer_son(0xFFFF);/*arrêt son*/

a=0;}
}

/*fonction appelée lors de l'interruption externe*/
void test(void)
{
    volatile byte comp = 0;

    /*lit les touches de la guitare*/
    comp =
digitalRead(Bouton_1)+2*digitalRead(Bouton_2)+4*digitalRead(Bouton_3)+8*dig
italRead(Bouton_4);

    if(comp!=0){                      //vérifie que c'est différent de zéro
    if(comp==matrice.ligne)//vérifie que l'accord est correct
    {score++;++bonus;}                //incrémentatation score bonus
    else{bonus=0;}
    if(bonus==5){score+=5;}
    if(bonus==10){score+=10;bonus=0;}}/*reset bonus*/
    else{bonus=0;}
}
```

Annexe X : Fichier lib_GH.cpp

```

/*****
Etude et Réalisation d'un Projet dans le Domaine du Génie Electrique
Guitar Hero
*****/

/*****
|*Fichier source (C++)                                     *|
|*GH_somo(int somo_clock,int somo_data,int somo_reset)      *|
|*void demarrer_son (int Son)                               *|
|*void Reset_somo()                                         *|
|*GH_matrice(int c_1,int c_2,int l_1,int l_2,int l_3,byte T[40])*|
|*int actualisation_matrice(int z)                         *|
|*int affichage_matrice (void)                             *|
*****/

#include<WProgram.h>
#include<arduino.h>
#include"lib_GH.h"

/*****
implementation des méthodes de la class matrice
*****/

/* constructeur matrice */
GH_matrice::GH_matrice(int c_1,int c_2,int l_1,int l_2,int l_3,byte
TabS[420])
{
    _c_1 = c_1;
    _c_2 = c_2;
    _l_1 = l_1;
    _l_2 = l_2;
    _l_3 = l_3;
    int i,j;
    ligne=0;
    for(i=0;i<420;i++)
    {
        _TabS[i] = TabS[i];
    }
    for(i=0;i<8;i++)
    {
        for(j=0;j<4;j++)
        {
            M[i][j]=0;
        }
    }
}

```

```
/*
décale la matrice d'une ligne et actualise
la première ligne avec une nouvelle
*/
int GH_matrice::actualisation matrice(int z)
{
    static boolean p=0;
    int i,j;
    int new_l=0;

    for(i=7;i>0;i--) //parcourt seulement les 7 dernières lignes
    {
        for(j=0;j<4;j++)
        {
            M[i][j]=M[i-1][j]; //décalage d'une ligne
            M[i-1][j]=0; //mise à zéro de la ligne précédente
        }
    }
    new_l = _TabS[z];
    if(p==0){new_l = new_l>>4;p=1;}else{p=0;z=z+1;}

    ligne = M[7][0]+2*M[7][1]+4*M[7][2]+8*M[7][3];

    for(j=0;j<4;j++) //information contenu dans new_line
    {if(new_l&0x01){M[0][j]=1;}//si dernier bit = 1 alors M = 1
      new_l>>=1;} //décalage vers la droite de 1

    return z;
}

/*
affiche la matrice
*/
void GH_matrice::affichage_matrice (void)const
{
    int a,i,j;

    for(a=0;a<195;) //temps d'affichage de chaque image 195ms
    {
        for(i=0;i<8;i++) //on parcourt la matrice
        {
            for(j=0;j<4;j++)
            {
                if(M[i][j])
                {
                    ++a;
                    /*l'info est contenue dans i et j. On fait les
                    tests relatifs et on active ou pas les sorties*/
                    if(i%2){digitalWrite(_l_1,HIGH);}else{digitalWrite(_l_1,LOW);}

if((i>=6)|| (i==2)|| (i==3)){digitalWrite(_l_2,HIGH);}else{digitalWrite(_l_2,
LOW);}

                    if(i>=4){digitalWrite(_l_3,HIGH);}else{digitalWrite(_l_3,LOW);}
                    if(j%2){digitalWrite(_c_1,HIGH);}else{digitalWrite(_c_1,LOW);}
                    if(j>=2){digitalWrite(_c_2,HIGH);}else{digitalWrite(_c_2,LOW);}

                    delay(1); //temps d'affichage de chaque led 1ms
                }
            }
        }
    }
}
```

```
    }  
  }  
}  
  
/*****  
Implémentation des méthodes de la class somo  
*****/  
  
/* constructeur somo */  
GH_somo:: GH_somo(int somo_clock,int somo_data,int somo_reset)  
{  
    _somo_clock = somo_clock;  
    _somo_data = somo_data;  
    _somo_reset = somo_reset;  
}  
  
/*  
    Cette fonction envoie l'adresse de la chanson à jouer  
    avec le module somo 14D celui-ci lit alors la chanson sélectionnée  
*/  
void GH_somo:: demarrer_son (int Son)const  
{  
  
    int TheSong = Son;  
    int ClockCounter=0;  
    int ClockCycle=15; // un cycle de clock afin d'envoyer uniquement 16 bits  
    (0 a 15)  
  
    digitalWrite(_somo_clock,HIGH); // met la clock à 1 pendant 300msec pour  
    préparer l'envoi de données  
    delay(300);  
    digitalWrite(_somo_clock,LOW); // met la clock à 0 pendant 2msec pour  
    préparer l'envoi de données  
    delay(2);  
  
    while(ClockCounter <= ClockCycle) //16 bits donc compte jusqu'à 16  
    {  
        digitalWrite(_somo_clock,LOW); //met clock à low car écrit sur  
        front descendant  
        if ((TheSong & 0x8000)==0x8000 ) //applique un masque sur le mot de 16  
        bits pour avoir et envoyer uniquement un bit à la fois  
        {  
            digitalWrite(_somo_data,HIGH); //si bit = HIGH data = HIGH  
        }  
        else  
        {  
            digitalWrite(_somo_data,LOW); //sinon data = LOW  
        }  
        TheSong = TheSong << 1; // décale mot de 16 bits pour envoyer  
        un autre bit a la carte de gestion du son  
        delayMicroseconds(200); // attente pendant 200 µs à LOW  
        digitalWrite(_somo_clock,HIGH); // broche clock à l'état HIGH  
        ClockCounter++; // incrémente le compteur de bit  
        delayMicroseconds(200); // attente 200 µs de la clock à HIGH  
    }  
  
    digitalWrite(_somo_data,LOW); // met la broche data à 0 plus de  
    donnée envoyée  
    digitalWrite(_somo_clock,HIGH); // met la broche clock à 1 pour  
    signaler la fin de transmission des données  
}
```



```
//fait un reset avant utilisation du module somo
void GH_somo:: Reset_somo() const
{
    digitalWrite(_somo_reset,LOW); //broche reset à LOW
    delay(50);                      //attente de 50 ms
    digitalWrite(_somo_reset,HIGH); //broche reset à HIGH
}
```

Annexe XI : Fichier lib_GH.h

```

/*****
Etude et Réalisation d'un Projet dans le Domaine du Génie Electrique
Guitar Hero
*****/

/*****
|*Fichier bibliothèque                                     *|
|*class GH_somo                                           *|
|*class GH_matrice                                         *|
*****/

#ifndef lib_GH
#define lib_GH

#include<WProgram.h>
#include<arduino.h>

class GH_matrice
{
public:
    GH_matrice(int c_1,int c_2,int l_1,int l_2,int l_3,byte TabS[420]);
    int actualisation_matrice(int z);
    void affichage_matrice (void)const;
    byte ligne;
private :
    int _c_1;
    int _c_2;
    int _l_1;
    int _l_2;
    int _l_3;
    boolean M[8][4];
    byte _TabS[420];
};

class GH_somo
{
public :
    GH_somo(int somo_clock,int somo_data,int somo_reset);
    void demarrer_son (int Son)const;
    void Reset_somo()const;

private :
    int _somo_clock;
    int _somo_data;
    int _somo_reset;
};

#endif
```

Annexe XII: Fichier Processing_GH

```

/*****
Etude et Réalisation d'un Projet dans le Domaine du Génie Electrique
Guitar Hero
*****/

/*****|
|*Fichier Processing_GH                *|
|*void setup()                        *|
|*void draw()                          *|          *|
|*void movieEvent(Movie m)            *|          *|
|*****/

/*librarie*/
import processing.video.*;
import processing.serial.*;

/*variables*/
Serial myPort2;
Movie myMovie;

void setup() {
    myPort2 = new Serial(this, Serial.list()[0], 9600); //initialisation com
    println(Serial.list());                             //affiche liste de com
    size(600, 450);                                     //création d'une fenêtre
    myMovie = new Movie(this, "totoro.mov");             //initialisation de la vidéo
}
int a=0;
int b=1;
void draw() {
    while (myPort2.available() > 0) {                   //si on reçoit quelque chose
        a = myPort2.read();                             //on met la valeur lu dans a
        if(a==1){if(b==1){
            myMovie.play();b=0;}}}                     //démarrage de la vidéo
        image(myMovie, 0, 0);
        myPort2.clear();                                //arrêt com série
    }
}

void movieEvent(Movie m) {
    m.read();                                           //lit la prochaine image de la vidéo
}

```

Annexe XIII : Tableau récapitulatif interfaçage *arduino*

Broches <i>arduino</i>	Modules
0	LCD DB7
1	LCD DB6
2	Bouton bas Guitare
3	LCD DB5
4	LCD DB4
5	LCD RS
6	Colonne 1 matrice
7	Colonne 2 matrice
8	Ligne 1 matrice
9	Ligne 2 matrice
10	Ligne 3 matrice
11	Somo Clock
12	Somo Data
13	LCD Enable
14	Bouton 1 Guitare
15	Bouton 2 Guitare
16	Bouton 3 Guitare
17	Bouton 4 Guitare
18	Somo Reset
19	//

Bibliographie

<http://f1mvp.perso.sfr.fr/Robotic/somo14d.htm>

<http://www.lextronic.fr/P5732-micro-module-de-restitution-de-fichiers-audio.html>

<https://forum.sparkfun.com/viewtopic.php?f=14&t=21388>

<http://arduino.cc/fr/Main/LiquidCrystal>

http://claudio.dreschel.free.fr/commentfaire/lcd_4.htm

http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.ExempleDigitalReadSerial

<http://regrapide.fr/arduino-tuto>

<http://arduino.cc/fr/Main/DebuterInstallationWindows>