C5-160551-INFO
Objets Connectés :
Programmation Microcontrôleurs - TP8

©Bernard Besserer

année universitaire 2021-2022

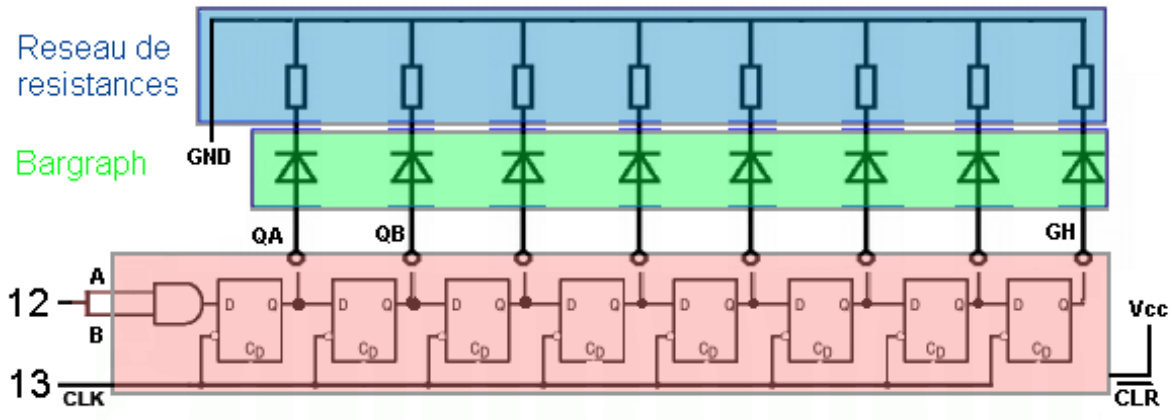
Ce qu'on va apprendre :
- Utiliser un registre à décalage pour réaliser une transmission série rudimentaire pour laquelle on va implémenter un protocole.
- Utiliser un capteur de distance qui se connecte via une liaison série I2C, en exploitant une bibliothèque logicielle prévue pour ce bus I2C.
- Réaliser un ensemble de type "radar de recul" visualisant graphiquement la distance détectée.

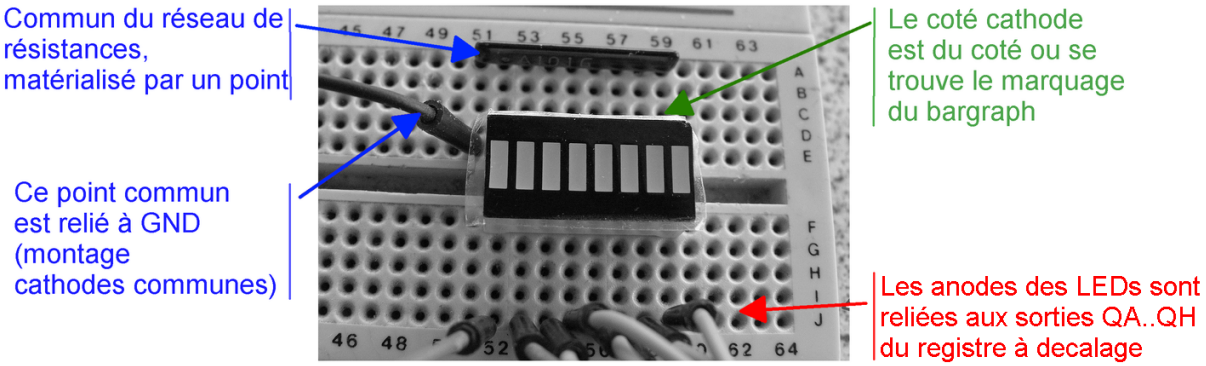
1 Branchement du registre a décalage, du bargraph et vérifications

Effectuez le câblage sur une plaquette d'expérimentation de grande taille.

Comme décrit dans le TD7, utilisez un registre à décalage 74HC164 (demandez à l'intervenant de vous remettre le composant ainsi qu'un afficheur de type bargraph et un réseau de résistances). Les brochages sont en annexe, à la fin du TP.
Réalisez un montage tel que celui proposé en TD (voir ci-après) avec un pilotage via 2 lignes : Horloge (on prendra le GPIO 13) et entrée série (GPIO 12).

On branche les LEDs du bargraph en mode *active high* tel qu'une valeur logique 1 dans une des cellules du registre à décalage allume la LED correspondante, donc relier les anodes des LEDs aux sorties du registre à décalage. NE PAS OUBLIER LE RESEAU DE RESISTANCE (coté cathodes) SINON LES LEDs DU BARGRAPH RISQUENT D'ETRE DETRUITES. Pour l'orientation du bargraph et savoir de quel coté sont les anodes et les cathodes : les cathodes sont du coté ou il y a les inscriptions sur le boîtier du bargraph.





Check list de contrôle :
- Il faut 8 fils, qui relient les sorties Q_A...Q_H du registre à décalage aux anodes des 8 LEDs consécutives du bargraph.
- Le réseau de résistances est-il bien branché ? Le “point commun” du réseau de résistances (la broche avec une marque) est-il bien relié au GND (et uniquement au GND) ?
- Le circuit 74HC164 est-il alimenté (+Vcc sur le 3.3V de l’ESP32, et le GND du circuit au GND du microcontoleur)
- La broche de *reset* du circuit 74HC164 est elle inactive : elle est nommé $\overline{\text{CLR}}$ et doit être branchée au niveau logique 1 (donc Vcc)
- Les entrées A et B du 74HC164 sont-elles reliées ensemble.
- Les entrées (A,B) et CLK du 74HC164 sont-elles reliés aux GPIOs de l’ESP32 ?

Donnez vous 20 minutes pour réaliser le câblage. Si vous étés bloqué avec le câblage, vous pouvez consulter une ressource supplémentaire sur Moodle (avec pénalité).
Pendant qu’un des membre du binôme effectue le câblage, l’autre écrit un petit programme de test :
- Mettre un niveau HIGH sur le GPIO qui correspond à la liaison “Données” (GPIO 12) - Envoyer un top d’horloge (front montant) sur le GPIO qui correspond à la liaison d’horloge ou *Clock* (GPIO 13). Pour créer un front montant, il faut que le GPIO soit à LOW puis passe à HIGH. La première LED du bargraph doit s’allumer. A chaque top, les données dans le registre à décalage se deplacent d’une position. Donc, si vous envoyez 8 tops d’horloge (avec une pause entre chaque top), vous allumez l’une après l’autre les LEDs du bargraph, ce qui permet de verifier l’ordre des branchements.

2 Modes d’affichage avec le bargraph

2.1 Va-et-vient

Écrire les **fonctions** suivantes, comme demandé dans le TD :
- La fonction `BargraphClear()` qui doit éteindre toute les LEDs du bargraph
- Une fonction `SetValue(unsigned char x)` permettant d’allumer le segment x du bargraph (affichage mode trait)

☑ En utilisant vos deux fonctions, écrire le programme qui reproduit l’animation “va et vient”, comme illustré par le GIF animé sur la page Moodle, section TD. Montrer le resultat à l’intervenant.

2.2 Écriture d’une classe

Écrire une classe `bargraph`, comme demandé dans le TD7.
- Pour le constructeur, il faut pouvoir spécifier les GPIOs pour la ligne d’horloge et la ligne de donnée, ainsi que le mode d’affichage (colonne ou trait, voir TD).
  Les valeurs par defaut sont : 13 pour le GPIO reliè à la liaison d’horloge, GPIO 12 pour la liaison “données”, et le mode de fonctionnement par defaut devra être le mode “colonne” (0 = mode colonne, 1 = mode trait)
- Implémentez les methonde suivante : Clear(), une methode SetValue(unsigned int val) et une methode SetMode(unsigned int mode)

La définition de la classe ressemble à :

```
class bargraph
{
    public:
      bargraph(unsigned int DisplayMode = 0, unsigned int pinClk=13, unsigned int pinData=12);  // constructeu
      void SetValue(unsigned int value);
      void Clear();
```

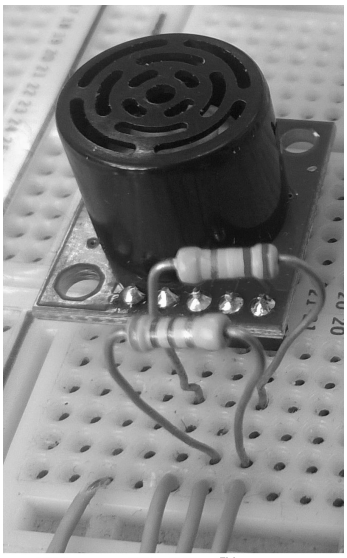
```
      void SetMode(unsigned int mode);

    private:
       // ici methodes privées et variables membres de la classe
};
```

Re-écrivez le code générant l'animation en utilisant votre classe et déposez le code source sur Moodle.

3 Mise en oeuvre d'un capteur utilisant un bus I2C

Ne pas débrancher le bargraph. Branchez le module SFR02. La documentation complète (également déposée sur Moodle) indique un brochage vu de dessous. La photo ci-après est une vue de dessus avec les indications pour les connexions . Pour l'I2C, le module ESP32 Huzzah utilise par défaut les broches 22 (SCL) et 23 (SDA).
Attention, le module SFR02 a été prévu pour une utilisation sous 5V. En conséquence :
- Pour alimenter le module SFR02, prélever le 5V qui arrive du port USB (broche marquée USB sur la platine de l'ESP32).
- Le GND du module SFR02 doit être relié au GND de la platine ESP32.
- Les résistances de tirage (pull-up), nécessaires pour le fonctionnement du bus I2C, seront connectés entre la ligne SDA et le +5V et entre la ligne SCL et le +5V. Utiliser les deux résistances de 18KΩ (brun, gris, orange)) ou de valeur proche.



Pour dialoguer avec le module SFR02, on utilise la bibliothèque Wire (normalement présente et déjà intégrée dans le portage du framework Arduino pour ESP32, elle n'est pas à installer). L'adresse I2C (sur 7 bits) de ce module est 112. Voir code ci-dessous :

```
#include <Wire.h>
...
setup()
{
   ...
   Wire.begin();    // On ne specifie pas les broches, alors broches par defaut SCL=22, SDA=23
}

loop()
{
  ...
  Wire.beginTransmission(112);  // initialisation d'une transmission vers le module à l'adresse #112 (0x70)
  Wire.write(byte(0x00));       // on specifie que l'on envoie une commande
  Wire.write(byte(0x52));       // commande 0x52 : effectuer une mesure du temps de vol (duree jusqu'à reception de l'echo
  Wire.endTransmission();       // fin transmission

  delay(70);                    // Il est suggèré dans la doc d'attendre au moins 65 millisecondes que la mesure se fasse

  Wire.beginTransmission(112);  // initialisation d'une transmission vers le module à l'adresse #112 (0x70)
  Wire.write(byte(0x02));       // on souhaite lire le registre correspondant à echo #1 (0x02)
  Wire.endTransmission();       // fin transmission

  Wire.requestFrom(112, 2);     // demande de lecture : 2 octets de la part du module esclave #112
  if (2 <= Wire.available())    // 2 octets ont été reçus
  {
    reading = Wire.read();     // recuperation de l'octet MSB
    reading = reading << 8;     // on décale
    reading |= Wire.read();     // recupération de l'octet LSB et on combine
    Serial.print(reading);      // affichage valeur
    Serial.println("microsec"); // affichage unite
  }
}
```

1. Dans un nouveau projet, utilisez le code ci-dessus et configurer votre programme pour qu'une mesure soit effectuée toutes les demi-secondes (une LED clignote sur le module SFR02 à chaque mesure) avec affichage du temps de vol sur le Terminal Série.
2. En consultant la documentation, envoyez la séquence de commande adéquate vers le module pour récupérer à chaque mesure **une distance en cm** et non un temps de vol en microsecondes.
3. Ajouter la classe de contrôle du bargraph et écrivez un programme qui affiche la distance sur le bargraph, comme s'il s'agissant d'un radar de recul, par exemple :
   - Bargraph éteint si distance supérieure à un mètre.
   - En dessous de 1 mètre, affichage des LEDs de façon progressive, tel que les 8 LEDs soient allumés pour

une distance de 20cm ou moins (le capteur n'étant plus très précis en dessous de 15cm). Pour le code gérant cet affichage, évitez les cascades de `if then else`. Trouvez la relation nb_led = A $\times$ distance + B.

Montrez le fonctionnementt à l'intervenant.

4 Finale au choix

Choisir l'un des développements parmi les deux proposés.

4.1 Déporter l'affichage

On veut réaliser un radar de recul pour un véhicule. Le microcontrôleur et le capteur à ultra-sons sont à l'arrière du véhicule et il faut transmettre les informations sur le smartphone du conducteur.
Reprendre le code des TPs précédents pour la création d'un HotSpot Wifi et l'instanciation d'un serveur WEB (la première partie du TP7 convient). Lors d'une requete GET à la racine, transmettre un page HTML qui doit afficher la distance sous forme de texte et d'un graphique similaire au bargraph, en vous inspirant de ce projet : `github.com/tomnomnom/vumeter`
Ce dernier code JS - qui n'utilise pas d'autres dépendances - peut facilement être ajouté à votre projet (copier le fichier dans le répertoire "'arduino"' et ajouter la syntaxe RAW STRING pour insérer, via une directive `#include`, l'intégralité du code JS en tant que RAW STRING dans votre fichier .ino

Placer un objet type bargraph sur votre page HTML. En utilisant des requêtes asynchrones toutes les secondes, effectuer une mesure de distance avec le capteur à ultra-sons et affichez le résultat sur le smartphone.
Idéalement, ajoutez des bips audio dans votre HTML (augmentation de la fréquence de répétition des bips quand la distance diminue, et arrêt des bips quand la distance est supérieure à un seuil). Toutes ces opérations se font coté client : les tests par rapport au seuil de distance en JS, et pour les bips, c'est l'objet `<audio>` du HTML et la commande `play()` en JS qui sont mis à contribution. Le signal audio doit être un fichier WAV ou MP3. Si l'ESP32 ne dispose pas d'un système de fichier, le fichier audio peut être encodé (UUENCODE) et mis au format texte dans le code de la page HTML. Voir ici : `jsfiddle.net/7EAgz/`

4.2 Alarme intrusion avec envoi d'email

Effectuer des mesures de distance régulièrement (par exemple toutes les 5 sec) et calculer une moyenne glissante (ou moyenne mobile) sur les 4 dernières mesures.
Si la dernière mesure est différente de la moyenne (à vous de trouver le seuil), une intrusion est détéctée et l'ESP32 doit alerter un utilisateur en lui envoyant un mail ou même un SMS.

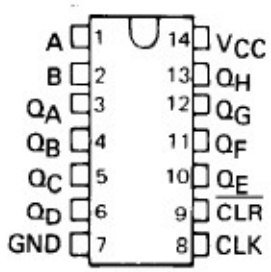
Pour cela, nous allons utiliser un service *cloud* nommé IFTTT (IF This Then That). Il vous faudra - comme trop bien souvent... - créer un compte (mais inscription possible via compte Apple, Google ou Facebook). Le service IFTTT permet de créer des services (nommés applets) comprenant des déclencheurs (*trigger*) et entraînant des actions.
Parmi les *trigger*, on y trouve les `webhooks` : lors de la réception d'une requête GET ou POST sur une adresse particulière et sous réserve d'une identification correcte (clé d'API), vous pouvez déclencher une action, dont l'envoi d'un email voir d'un SMS. Votre ESP32 doit donc se connecter à internet (en mode client, et en passant par votre téléphone mobile qui sera mis en partage de connexion), et lors de la détection d'une intrusion, l'ESP32 devra émettre une requête correctement formatée pour servir de déclencheur à votre applet IFTTT
Vous trouverez les informations utiles ici :
`iotdesignpro.com/projects/how-trigger-led-using-ifttt-and-esp32-email-notification`
Si votre systeme fonctionne et s'il vous reste du temps, rendez votre système économe en énergie en y intégrant les modes de veille :
- Réveil toutes les 5 secondes pour une mesure
- Connexion au réseau Wifi uniquement si c'est nécessaire (intrusion détectée)

A 74HC164

FUNCTIONAL DESCRIPTION

The LS164 is an edge-triggered 8-bit shift register with serial data entry and an output from each of the eight stages. Data is entered serially through one of two inputs (A or B); either of these inputs can be used as an active HIGH Enable for data entry through the other input. An unused input must be tied HIGH, or both inputs connected together.

Each LOW-to-HIGH transition on the Clock (CP) input shifts data one place to the right and enters into Q_0 the logical AND of the two data inputs (A•B) that existed before the rising clock edge. A LOW level on the Master Reset (MR) input overrides all other inputs and clears the register asynchronously, forcing all Q outputs LOW.

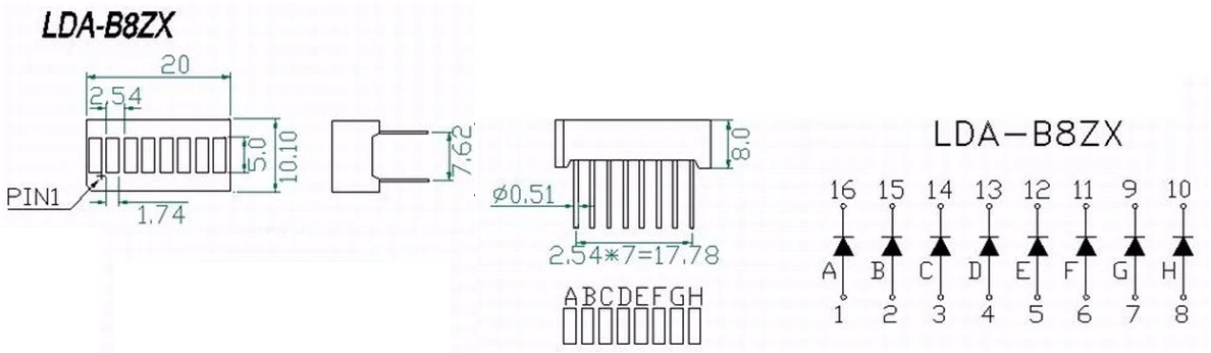


MODE SELECT — TRUTH TABLE

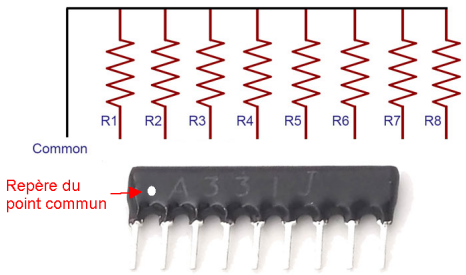
OPERATING MODE	INPUTS			OUTPUTS	
	$\overline{MR}$	A	B	Q_0	Q_1–Q_7
Reset (Clear)	L	X	X	L	L – L
Shift	H	l	l	L	q_0 – q_6
	H	l	h	L	q_0 – q_6
	H	h	l	L	q_0 – q_6
	H	h	h	H	q_0 – q_6

L (l) = LOW Voltage Levels
H (h) = HIGH Voltage Levels
X = Don't Care
q_n = Lower case letters indicate the state of the referenced input or output one set-up time prior to the LOW to HIGH clock transition.

B Bargraph



C Réseau de résistance



D SFR02 (vu coté composants, pas coté transducteur ultrasonique)