GIF-3002 Périphériques, Général

Ce document présente l'accès aux périphériques par les programmes, comment sont gérés les périphériques et quels sont les méthodes générales pour accéder aux périphériques.

1 Introduction

Entrées/Sorties (E/S, Input/Output ou I/O en anglais) désigne l'ensemble des transferts de données qui permettent au microprocesseur et à la mémoire de communiquer avec le restant du monde. Une entrée est une donnée allant du monde extérieur vers le microprocesseur. Une sortie est une donnée allant du microprocesseur vers le monde extérieur.

Un périphérique est un appareil qui interagit avec microprocesseur et la mémoire. Certains périphériques sont branchés à l'intérieur de l'ordinateur (disques durs, carte réseau,…) alors que d'autres sont branchés sur des interfaces externes de l'ordinateur (clavier, écrans, souris, imprimantes, etc.).

Le traitement des E/S est complexe pour plusieurs raisons:

- Les périphériques ont des modes de fonctionnement variés.
- Les périphériques ont souvent leurs propres formats de données.
- Les périphériques ne vont pas à la même vitesse que le microprocesseur. Beaucoup sont très lents par rapport à ce dernier alors que certains sont plus rapide…

Pour chaque périphérique, il existe une unité spéciale appelée module d'E/S (I/O module) qui sert d'interface entre le périphérique et le microprocesseur.

2 Accès aux périphériques et accès à la mémoire

Du point de vue du microprocesseur, l'accès aux périphériques se fait de la même façon que l'accès aux données de la mémoire : des instructions demandent au microprocesseur de lire ou d'écrire un périphérique.

Dans certains systèmes (ARM par exemple), l'accès aux périphériques se fait avec les mêmes instructions que les accès à la mémoire : les périphériques ont seulement des adresses différentes. On dit alors que les périphériques sont "mappés" en mémoire, c'est-à-dire que des plages d'adresses leurs sont attribuées (MMIO = Memory Mapped I/O).

Dans les autres systèmes (x8086 par exemple), il existe des instructions réservées pour les accès aux périphériques. Dans ces systèmes, les périphériques ont souvent un espace d'adresse différent de celui de la mémoire et les adresses de chaque périphériques sont appelées des ports (PMIO = Port Mapped I/O).

Pour accéder aux périphériques, le cœur ARM utilisera les instructions LOAD et STORE, au même titre que pour les accès à la mémoire. Pour accéder aux

périphériques, le x86 utilisera des instructions IN/OUT alors que l'instruction MOV est utilisée pour accéder à la mémoire.

3 Gestion des périphériques : modules d'E/S

Les modules d'E/S sont des interfaces entre le microprocesseur et un périphérique spécifique. Ces modules sont habituellement appelés « contrôleurs ». Par exemple, le module d'E/S servant d'interface entre le microprocesseur et un disque dur sera appelé contrôleur de disque.

Les modules d'E/S ont plusieurs fonctions. En voici les principales:

- *Lire ou écrire des données du périphérique.*
- *Lire ou écrire des données du microprocesseur/Mémoire. Cela implique du décodage d'adresses, de données et de lignes de contrôle. Certains modules d'E/S doivent générer des interruptions ou accéder directement à la mémoire.*
- *Contrôler le périphérique et lui faire exécuter des séquences de tâches.*
- *Tester le périphérique et détecter des erreurs.*
- *Mettre certaines données du périphérique ou du microprocesseur en mémoire tampon, dans le module d'E/S lui-même, afin d'ajuster les vitesses de communication.*

Il existe plusieurs techniques pour communiquer à partir du microprocesseur/Mémoire vers un périphérique à travers un module d'E/S. Les trois principales techniques sont les E/S programmées, les E/S avec interruptions et le DMA.

Le diagramme ci-dessous montre les principales composantes d'un module d'E/S. Ainsi qu'illustré, le module comprends de la mémoire tampon pour des données (sous forme de registres dans l'illustration), une logique de contrôle pour décoder l'adresse et les lignes de contrôle (ou pour faire du DMA), et une ou plusieurs interfaces avec un ou plusieurs périphériques. Ce diagramme est simple. Certains modules d'E/S sont très rudimentaires alors que d'autres sont très complexes (exemples de caractéristiques qui rendent le module complexe: DMA, support pour interruptions, programmes exécutés à l'intérieur du module, etc.).

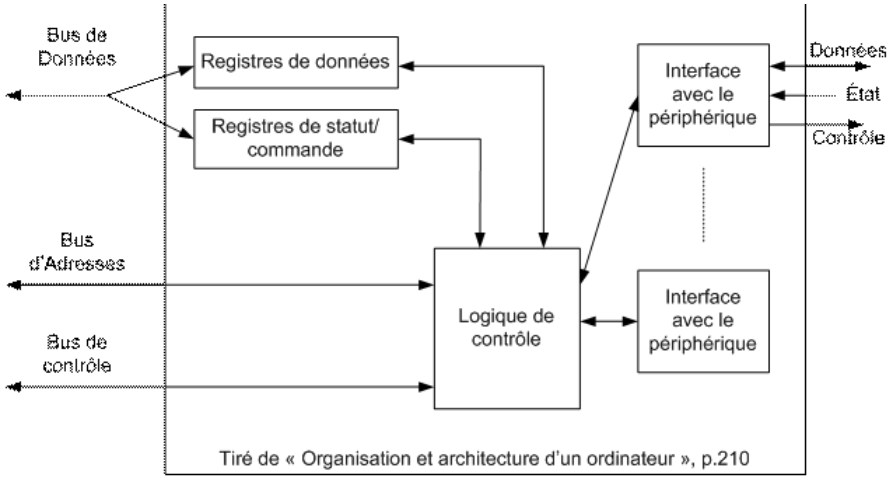


Figure 1 - Module d'E/S

Dans les microcontrôleurs, on accède généralement à un périphérique en écrivant ou en lisant les registres du module d'E/S. Une adresse, ou un port, est attribuée à chaque registre du module d'E/S.

4 Méthodes d'accès aux périphériques

4.1 E/S Programmées

La technique d'E/S programmées consiste à concevoir un programme, exécuté par le microprocesseur, qui communiquera avec le module d'E/S afin d'obtenir des données du périphérique.

Le microprocesseur peut tester, lire, écrire ou contrôler le périphérique à travers le module d'E/S.

Les E/S programmées sont très simples. Le microprocesseur a un contrôle direct sur le périphérique. Pour accéder à un périphérique, il exécute un programme qui aura les grandes lignes suivantes[1]:

- *Vérifier si le périphérique est prêt*
- *Envoyer une requête au périphérique*
- *Attendre que la requête soit finie en interrogeant les registres de statut du périphérique (polling)*
- *Envoyer une autre requête et attendre encore. Recommencer ces deux opérations tant que le programme n'est pas terminé…*

[1] Dans le texte qui suit, périphérique désigne en fait le module d'E/S du périphérique. Par exemple, "Vérifier si le périphérique est prêt" sera plutôt "Le programme contiendra une instruction disant de lire le registre de statut du module d'E/S.

Les E/S programmées ont un désavantage majeur évident: c'est très lent. Le microprocesseur attend après un périphérique pour passer à la prochaine séquence d'instructions! Les E/S programmées ont d'autres désavantages. Par exemple, il peut y avoir des tâches complexes au niveau matériel qui exigent un contrôle très rapide et très poussé du périphérique. Dans de tel cas, il peut devenir impossible de faire un programme qui pourra contrôler le périphérique

4.2 E/S par Interruptions

Une façon d'éliminer les délais d'attente du microprocesseur est d'utiliser des interruptions. La méthode d'E/S avec interruption consiste à relier le module d'E/S au contrôleur d'interruptions afin de lui permettre de signaler un évènement particulier (la fin d'une tâche par exemple!) au microprocesseur.

Nous avons déjà vu les interruptions. Néanmoins, voici un rappel de ce qui se produit lorsqu'une interruption survient:

- *Le périphérique émet un signal d'interruption*
- *Le microprocesseur termine l'instruction en cours*[2].
- *Le microprocesseur détecte l'interruption et détermine la source de l'interruption.*
- *Le microprocesseur vérifie s'il doit traiter l'interruption en fonction de sa priorité et de ses registres internes.*
- *Le microprocesseur sauvegarde l'emplacement du programme en cours sur la pile, puis il appelle une routine de traitement de l'interruption.*
- *La routine de traitement de l'interruption est exécutée*
- *Le microprocesseur dépile l'emplacement du programme exécuté avant l'interruption lorsque l'interruption finit.*

La séquence d'évènement précédente ne s'applique pas à tous les systèmes microprocesseurs et à toutes les interruptions. Par exemple, un registre de lien est parfois utilisé plutôt que la pile pour sauvegarder l'adresse de retour. Il arrive aussi bien souvent que ce soit le contrôleur d'interruptions plutôt que le microprocesseur qui détermine la priorité des interruptions. Bref, la séquence d'évènements lors d'une interruption varie quelque peu d'un système à l'autre. Cependant, les grandes lignes demeurent.

Lorsque plusieurs périphériques peuvent faire des interruptions, il faut identifier qui fait l'interruption et déterminer quelle interruption est la plus prioritaire. Souvent, des registres du contrôleur d'interruptions permettent d'assigner des priorités ou de masquer des interruptions (voir la section sur le NVIC dans Présentation de Microcontrôleur-Matériel.doc pour un exemple).

[2] Lorsque le microprocesseur a un pipeline d'instructions, terminer l'instruction en cours est plus complexe que cela ne semble! Pour avoir une interruption précise, il faut terminer l'instruction en cours… et toutes les instructions précédentes encore dans le pipeline. Il faut aussi éviter d'écrire un résultat pour une instruction qui suit l'instruction en cours et déjà débutée dans le pipeline…

4.3 DMA

Voir la section sur le DMA dans SMI_C4_Mémoires-Matériel.doc.

GIF-3002 Périphériques : General Purpose Input/Output

1 Introduction

Les entrées-sorties d'usage général sont parmi les premiers périphériques à avoir été intégrés dans les microcontrôleurs.

Une entrée-sortie générale est soit une entrée ou une sortie, lisant ou mettant une tension digitale sur une broche du microprocesseur. La direction (entrée ou sortie) de l'entrée-sortie est donnée par un registre du module d'E/S qui contrôle les GPIOs.

Lorsqu'un GPIO est une entrée, il est possible de lire la valeur digitale présente sur la broche. Bien entendu, il s'agira d'une tension analogique, mais un "1" ou un "0" sera lu en fonction de la tension analogique.

Les paramètres importants d'une entrée sont les seuils de tensions qui feront changer la valeur lue (voir SMI_C1_Introduction-Composantes analogiques.ppt), la tension maximale supportée par l'entrée et la vitesse d'échantillonnage.

Pour le TTL (Transistor-Transistor Logic) par exemple, un « 0 » (LOW) sera entre 0 et 0.8V alors qu'un « 1 » HIGH sera entre 2.2V et 5V. La valeur digitale lue pour une tension entre 0.8V et 2.2V est indéterminée. Ces valeurs sont indicatives et peuvent changer d'un circuit intégré à l'autre.

Lorsqu'un GPIO est une sortie, la sortie "imposera" une tension sur la broche reliée au GPIO en fonction d'un registre de contrôle. En écrivant un registre du module d'E/S du microcontrôleur, la sortie permet de signaler un "0" ou un "1".

Les paramètres importants d'une sortie sont la nature de la sortie et les courants maximums que peut tirer ou pousser la sortie (source – sink) et ses protections contre les court-circuits.

Les natures les plus communes des sorties sont totem pole (même chose que push-pull) et open collector (même chose que open drain). Une sortie totem pole reliera la broche du GPIO au 0V-GND lorsqu'un "0" est demandé. Elle reliera la broche du GPIO à VCC (5V, 3.3V, autre) lorsqu'un "1" est demandé. Une sortie open collector reliera aussi la broche du GPIO au 0V lorsqu'un "0" est demandé, mais elle laissera la sortie flottante lorsqu'un "1" est demandé.

Une sortie "pousse" du courant (source) lorsque le courant sort de la sortie et se dirige vers l'extérieur. C'est souvent le cas lorsque la sortie est à "1". Une sortie tire du courant (sink) lorsque le courant entre dans la sortie, provenant de l'extérieur. C'est souvent le cas lorsque la sortie est à "0".

2 Contrôle des GPIOs

Dans la plupart des microcontrôleurs, il existe plusieurs registres pour contrôler les GPIOs. De manière générale, on retrouve :

- Un registre de DIRECTION sert à déterminer si la broche est une entrée ou une sortie.
    - Pour éviter d'éventuelles collisions, les broches sont toujours des entrées après un reset.
- Un registre de VALEUR sert à lire une entrée, ou imposer une tension sur une sortie.
- Un registre de FONCTION sert à déterminer quel sera le rôle de la broche.
    - Habituellement, le rôle par défaut des broches est GPIO.

On retrouve aussi souvent d'autres registres, optionnels, qui servent à configurer les GPIOS :

- Un registre de PULL-UP ou PULL-DOWN sert à mettre une PULL-UP ou une PULL-DOWN sur la broche.
- Un registre de COURANT déterminant quel courant peut circuler à travers la broche.

2.1 Un exemple de module d'E/S pour GPIO

La figure suivante illustre ce que pourrait être un module d'E/S pour une broche d'entrée/sortie tout usage : le module a deux registres (valeur et direction) qui permettent d'imposer une valeur sur la broche. Il est aussi possible, de lire la tension sur la broche en adressant le registre de valeur et en activant le signal de lecture du bus de contrôle.

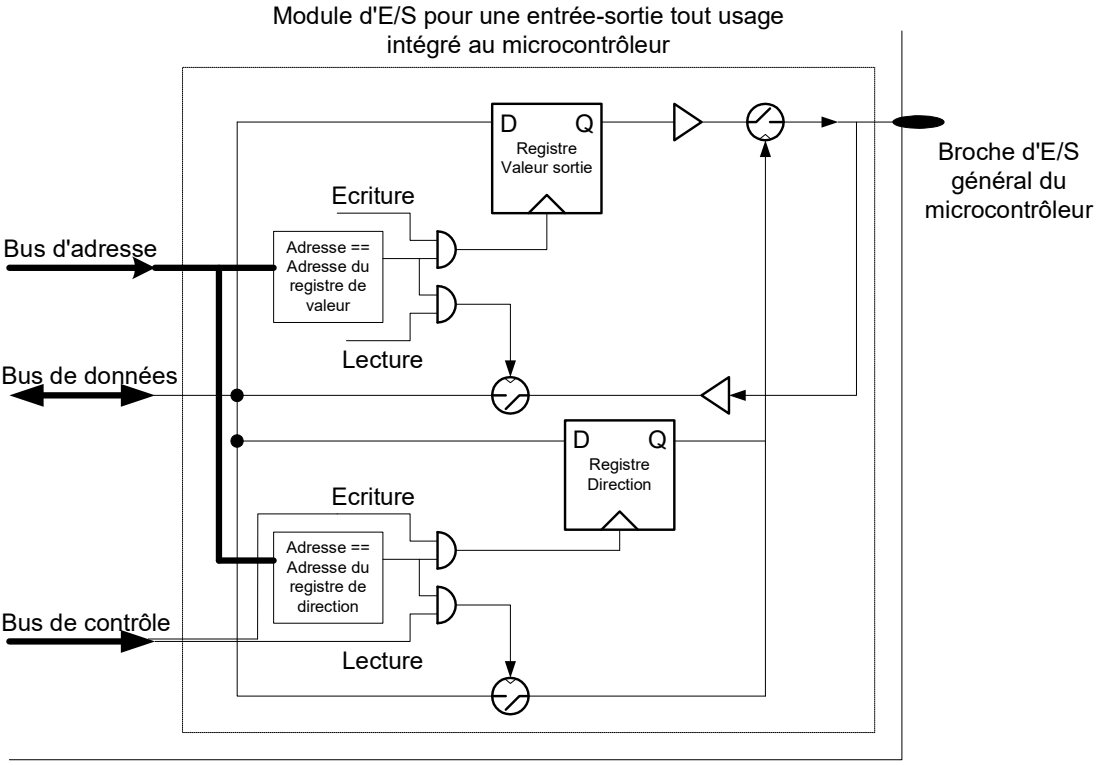


Figure 2 - Module d'E/S pour GPIO très simple

2.2 Ports de GPIO

Les GPIOs sont toujours regroupés par « ports » de 8-16-32 bits. Ainsi, le registre de direction peut être un registre 32 bits dont chacun des bits établit la direction d'une broche…

> *Il a été vu précédemment qu'un port désigne l'adresse d'un périphérique. Même si le mot « port » a souvent une signification qui dépend du contexte, de manière générale, en électronique, un port désigne un point de connexion entre un contrôleur et un périphérique.*

Un port, dans un contexte de GPIO, désigne un ensemble d'entrées/sorties digitales qui peuvent faire des opérations en groupe : lecture d'entrées en parallèle ou contrôle simultané de sorties dans une seule instruction.

Les signaux d'un port sortent du die et sont reliés à une broche du boîtier, à l'intérieur du boitier. En fonction du nombre de broches du microcontrôleur ou du format du boîtier, le

numéro de la broche reliée à un port peut changer à travers les microcontrôleurs d'une même famille.

2.3 GPIO et manipulations de bit

Il est fréquent de manipuler des bits lorsqu'on travaille avec les GPIOs. L'approche traditionnelle pour mettre à "1" un bit à l'intérieur d'un mot est d'utiliser un OU (RegistreGPIO = RegistreGPIO | 2^bit). L'approche traditionnelle pour mettre à "0" un bit à l'intérieur d'un mot est d'utiliser un ET ainsi qu'une négation (RegistreGPIO = RegistreGPIO & ~(2^bit).

> *Si on veut mettre le bit 3 d'un nombre = 1 par exemple (bit7 …**bit3** bit2 bit1 bit0), on fera nombre |= 8. Si on veut mettre le bit 3 d'un nombre à 0, on fera nombre &= ~8;*

L'approche traditionnelle a le désavantage d'être longue : il faut trois instructions pour mettre un GPIO à 1 (lire le RegistreGPIO, faire l'opération logique, écrire le RegistreGPIO). Pour accélérer l'écriture de bits dans les GPIOs, plusieurs méthodes ont été adoptées :

La plus commune des méthodes adoptées pour accélérer ou éviter la manipulation de bits dans les registres de GPIO est de mettre deux registres par registre de GPIO : un registre pour écrire et un autre pour lire. Par exemple, vous retrouverez parfois les registres GPIO_DIRECTION_SET et GPIO_DIRECTION_CLEAR pour établir la direction des GPIOs.

2.4 Comment exploser sa broche de GPIO

Les GPIOs sont configurés en input par défaut, pour tous les microcontrôleurs. Cela évite la cause la plus cause de dommage la plus fréquente : deux sorties qui imposent des tensions différentes sur la même broche. Si deux sorties sont reliées entre-elles et si une sortie impose 0V alors que l'autre sortie impose 3.3V, les deux sorties créent un court-circuit…

Une autre façon courante d'endommager une entrée/sortie de microcontrôleur est de faire passer trop de courant. Par exemple, on branche une sortie du microcontrôleur pour allumer directement une LED 20mA alors que la sortie peut fournir 2mA maximum. C'est l'idéal pour griller des composantes dans le contrôleur de GPIO!

Les décharges électrostatiques sont aussi la cause de plusieurs dommages même si de bonnes habitudes permettent de limiter leurs apparitions.

Enfin, appliquer trop de voltage sur une entrée peut également l'endommager. Souvent, les tensions maximum et minimum supportées par une entrée sont souvent exprimée par rapport à Vss et Vcc. Par exemple, si la tension maximum est VCC+0.3V et que VCC est 3.6V, appliquer 5V sur la broche risque d'endommager le matériel…

Dans le même ordre d'idée, mettre de la tension sur une entrée de microcontrôleur qui n'est plus alimenté peut faire endommager le microcontrôleur.

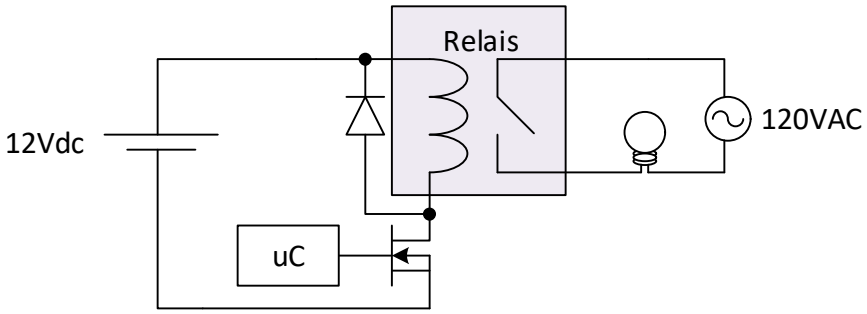
3 Interrupteurs

3.1 BJT et FET

Les transistors sont des interrupteurs par excellence en électronique, pour commuter des signaux basse-tension (sous 40Vdc). Les transistors sont des semi-conducteurs qui ne coupent pas mécaniquement le contact, mais empêche le courant de circuler.
Les BJT sont des interrupteurs contrôlés par un courant.
Les FET sont des interrupteurs contrôlés par un voltage.
Les transistors de type N (BJT-NPN ou NMOS) servent habituellement à relier un signal au 0V (GND).
Les transistors de type P (BJT-PNP ou PMOS) servent habituellement à relier un signal au 3.3V (VCC).

3.2 Relais

Les relais permettent de couper un fil ou de fermer un circuit à l'aide d'un signal électrique. Un relais est constitué d'une bobine et d'un contact. Lorsqu'un courant circule dans la bobine, cela crée un champ magnétique qui ouvre ou ferme le contact.



Les relais sont très utiles pour contrôler du haut-voltage tout en isolant le microprocesseur.

3.3 Autres interrupteurs

Il existe une multitude d'interrupteurs disponibles sur le marché. Par exemple, vous retrouverez les thyristors et les triacs qui sont des interrupteurs pour la haute-puissance. Il s'agit de semi-conducteurs comme les transistors et les diodes qui conduisent en fonction du courant circulant dans la gâchette. Vous retrouverez aussi des opto-coupleurs, des opto-triacs, des diacs, des circuits intégrés qui servent d'interrupteur, des portes AND qui servent de GATE, et plus.

Les critères pour choisir un interrupteur sont les suivants :

- Comment se fait le contrôle de l'interrupteur? Quels signaux et quelle puissance est requise pour ouvrir ou fermer le contact?
- Quel est l'état par défaut de l'interrupteur et quelle est le nombre d'entrées/sortie de l'interrupteur : ouvert, fermé, avec deux contacts de sortie dont un ouvert et l'autre fermé…

- Lorsque l'interrupteur est ouvert, quelle tension pouvez-vous appliquer aux bornes du contact sans créer d'arc électrique.
- Lorsque l'interrupteur est fermé, quel est le courant maximum pouvant circuler dans l'interrupteur?
- Quelle puissance sera perdue ou dissipée dans le contact?
- Quelle est l'isolation entre le contact de l'interrupteur et le circuit de contrôle de l'interrupteur?

4 LEDs et Boutons

Les périphériques les plus communs connectés aux GPIOs sont incontestablement les LEDs et les boutons. Les LED (Light Emitting Diode) sont des diodes qui éclairent lorsqu'il passe un courant et les boutons sont des contacts mécaniques.

Les figures qui suivent montrent quelques exemples de connexion des LEDs et des boutons à une entrée/sortie générale. Ces figures sont commentées ci-dessous.

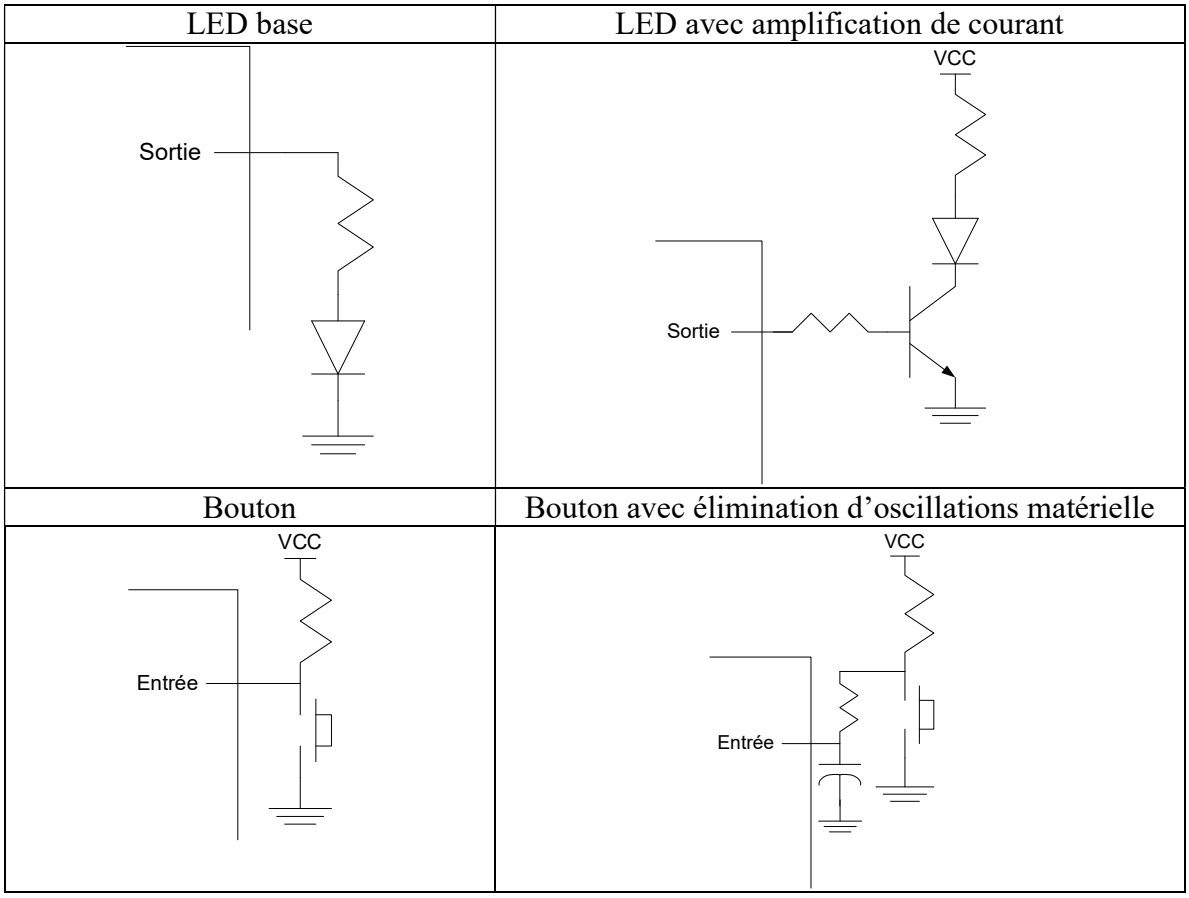


Figure 3 – Exemples de branchements de LED et bouton à un microprocesseur

Quelques points à retenir lorsqu'on interface avec des LEDs :

- Si vous connectez une LED directement à du 5V ou du 3.3V, elle grillera ou l'alimentation grillera ou les deux. Brancher une LED à l'envers ne cause habituellement pas de dommage, mais elle n'allume pas.
- Avec le circuit adéquat, il est possible d'allumer ou d'éteindre une LED lorsque la sortie est un "1". Il est aussi possible d'allumer ou d'éteindre une LED lorsque la sortie est un "0". Le choix du circuit est habituellement déterminer par le courant que peut tirer ou pousser la sortie ainsi que par l'état par défaut de la lumière (allumée initialement, éteinte initialement).
- La perte de tension aux bornes d'une LED (tension Vf) est souvent plus que 0.7V, qui est une perte de tension commune pour une diode normale. Des valeurs typiques de perte de tension pour les LEDs sont de 1V à 2.5V.
- Pour allumer une LED, il faut souvent un minimum de courant, mais, dans bien des cas, il est possible d'avoir un éclairage réduit avec un courant inférieur au courant nominal de la LED. Des courants typiques pour allumer des LEDs sont 10mA, 15mA ou 20mA. Il existe des LEDs avec de plus petits courants (2mA par exemple), et le courant requis pour faire de la lumière dépend souvent de la couleur de la LED.
- Lorsque les sorties d'un microprocesseur ne peuvent pas fournir assez de courant pour allumer une LED, il est habituel d'ajouter un transistor pour que le courant traversant la LED soit fourni par l'alimentation plutôt que par la broche du transistor.
- Il n'est pas nécessaire d'allumer une LED 100% du temps pour qu'elle paraisse allumée. Dans plusieurs systèmes et pour plusieurs raisons (économie de puissance, réduction du nombre de broches pour contrôler un tableau de LED, etc.), des trains de pulse seront utilisés pour "allumer" une LED.

La tension Vf d'une LED dépend de la longueur d'onde émise. Plus la longueur d'onde est courte (fréquence élevée), plus il faut d'énergie pour produire la lumière et plus il faut de puissance pour qu'elle éclaire. En d'autres mots, le Vf d'une LED bleue est plus élevé que le Vf d'une LED verte qui est plus élevé que celui d'une LED rouge. On peut prédire cette information avec le spectre électromagnétique de la lumière visible!

Quelques points à retenir lorsqu'on interface avec des boutons et des interrupteurs:

- Les boutons/interrupteurs ne sont pas tous de simples contacts. Certains interrupteurs ont plusieurs entrées (poles) et/ou plusieurs sorties (through). Certains boutons relient plusieurs broches ensembles.
- Pour détecter si un bouton est enfoncé, on peut lire un "0" ou un "1", en fonction du circuit associé au bouton.
- Lorsque l'usager appuiera sur le bouton ou l'interrupteur, il peut arriver une décharge électrostatique. Décharger quelques kilovolts dans une entrée de microcontrôleur peut endommager cette entrée, voire même endommager le microcontrôleur au complet. Habituellement, des résistances et des diodes sont ajoutées aux circuits des boutons afin de limiter le courant et la tension pouvant apparaître sur la broche du microcontrôleur

- Un contact mécanique ne se fait pas instantanément : il est courant de voir des oscillations apparaître lorsque le contact se fait ou se défait. En quelques millisecondes, un bouton peut conduire, ne plus conduire, conduire, ne plus conduire… Le contact rebondit (“bounce”). Le debounce peut se faire matériellement, avec un circuit RC, ou, plus souvent, avec un traitement logiciel (voir annexe A, debounce de bouton).
- D’un point logiciel, un bouton semble être une entrée strictement booléenne. Cependant, il est classique de gérer les boutons avec des machines d’état (ON –> ON allant vers OFF –> OFF –> OFF allant vers ON –> ON).

5 Interface avec un clavier

Deux documents seront présentés brièvement en classe afin d’illustrer une interface avec un clavier : les notes de cours de 2009 et un document bien vulgarisé sur les principes généraux du clavier.

5.1 Notes de cours de 2009 sur le clavier

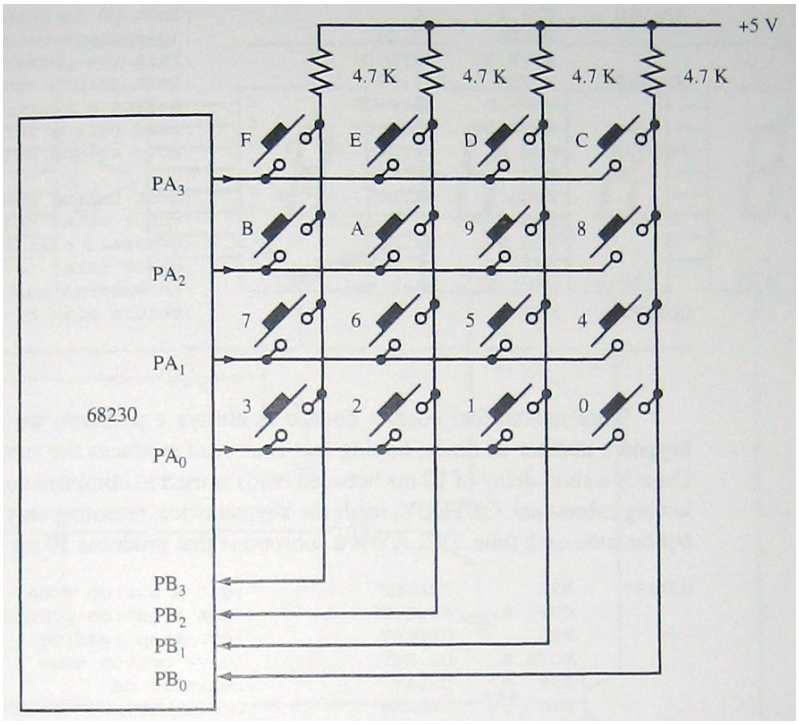


Figure 4 – Exemple de branchement de clavier

“Comment multiplexer un clavier avec un port parallèle:

PA configuré en sortie. On envoie successivement les motifs suivants sur PA0 à PA3: 0111; 1011; 1101; 1110 pour effectuer un « linescan »

Pour chaque envoi sur PA, on lit le résultat sur PB. Si PB != 1111 alors une ou plusieurs touches ont été pesées.
À ce moment, la routine envoie un
ScanCode constitué de:

PA0-3 qui indique la ligne testée et PB0-3 qui indique quelles touches sur cette ligne ont été pesées.
Attention aux appuis simultanés: PAi doit pouvoir supporter le courant de 4 pull-up en parallèle, or, les résistances sont calculées ici pour un appui à la fois seulement.

Les claviers modernes procèdent ainsi, mais cachent cette partie pour en faire abstraction aux codeurs. »

5.2 Principes d’interface avec un clavier

Le document suivant sera présenté en classe
http://www.dribin.org/dave/keyboard/keyboard.pdf

6 Interface avec un LCD à caractère

6.1 Principes généraux

Les LCDs sont habituellement constitués d'une matrice de points servant à afficher un symbole :

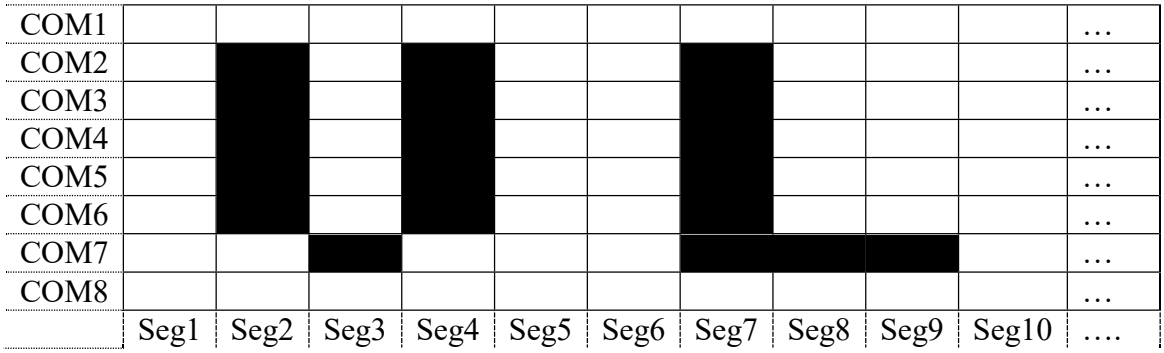


Figure 5 – Matrice de points d'un LCD

Pour afficher un symbole, on balaie généralement les lignes communes (COMx) une après l'autre, en changeant la valeur de chaque segment en fonction du symbole que l'on veut afficher. Le LCD est rafraîchit à une fréquence prédéterminée qui dépend de la vitesse de l'horloge et du nombre de lignes communes.

Pour afficher une séquence de caractères, il faut gérer plusieurs lignes de contrôle avec un timing précis. Pour cette raison, les LCDs sont souvent contrôlés par un microcontrôleur dédié à cette tâche : il s'agit d'un microcontrôleur simple avec beaucoup de broches et, souvent, contenant l'information nécessaire, en mémoire non-volatile, pour afficher des caractères précis.

La plupart des LCDs (environ 50%) de 80 caractères ou moins sont contrôlés par le HD44780 de Hitachi.

6.2 Interface avec un LCD ayant un contrôleur HD44780

> *La meilleure préparation avant d'interfacer avec un LCD avec HD44780 est sûrement de lire les sections pertinentes de la datasheet du HD44780… Les détails qui manquent ci-dessous se retrouveront dans ce document!*

Le HD44780 gère l'affichage d'un LCD ayant jusqu'à 4 lignes et 80 caractères. Il possède des mémoires internes pour retenir les symboles à afficher (DDRAM) et savoir quels segments (en fonction de la ligne commune) activer pour afficher ces symboles (CGROM = Caracter Generator ROM). Il est possible d'ajouter quelques symboles à l'afficheur (CGRAM). Par ailleurs, le LCD gère un curseur et peut faire clignoter les symboles.

L'interface entre un système microprocesseur et un HD44780 est assez simple. Elle se compose des broches suivantes :

Broche	Description
1- VSS	Référence, Ground
2- VCC	Alimentation
3- VLC	Intensité de l'éclairage
4- RS	Register Select : 0 pour une instruction, 1 pour une donnée
5- R/W	Lecture (R) or écriture (W) : 0 pour lire, 1 pour écrire
6- E	Enable : Activation du LCD lorsque 1
7- D0	Ligne de donnée
8- D1	Ligne de donnée
…	Ligne de donnée
14-D7	Ligne de donnée

Les instructions supportées par le HD44780 sont décrites dans la datasheet du HD44780, la table 6 fournissant un excellent résumé.

Le LCD est un périphérique lent. La mise à jour de l'affichage n'est pas instantanée et exécuter une instruction peut prendre quelques millisecondes. C'est pourquoi chaque instruction pour le LCD devrait être précédée ou suivie d'un temps d'attente convenable ou c'est pourquoi l'instruction Read Busy Flag devrait être employée avant chaque instruction tel que présenté à la Figure 15 de la datasheet.

Pour gérer le LCD, la séquence d'instructions suivante est une séquence minimale d'instructions envoyées au HD44780 pour le faire fonctionner correctement:

1) Initialiser le LCD. Faire les instructions suivantes dans n'importe quel ordre :
    a. Display ON/OFF Control: Déterminer les options de curseur, de blink et activer/d/sactiver l'affichage.
    b. Function Set : Décrit le LCD utilisé au HD44780 et le mode de communication de l'information entre le HD44780 et l'extérieur (4 bits de données ou 8 bits de données dans l'interface, nombre de lignes d'affiche, et nombre de points (5*8 ou 5*10) par symbole.
    c. Clear Display : Remet l'affichage à zéro et le curseur au début…
2) Écrire vos données à afficher ou gérer le curseur…
    a. Write Data to DDRAM

Pour des exemples de code, voir les deux sites web suivants :

Simple et minimaliste : http://www.beyondlogic.org/parlcd/parlcd.htm
Beau et complet : http://ee.cleversoul.com/lcd-project.html

7 Interface avec un LCD TFT

7.1 Description

Un LCD TFT (Thin Film Transistor) est un LCD avec des transistors déposés en couche mince sur un substrat de verre et utilisés pour contrôler des LEDs ou des crystaux liquides. Les LCDs TFT forment de petits écrans avec une résolution moyenne qui peuvent servir pour des téléphones cellulaires ou des tablettes électroniques.

7.2 Interface avec le LCD TFT

Comme pour le LCD à caractère, il y a souvent un contrôleur de LCD entre votre microprocesseur et le LCD, intégré au LCD lui-même. Parfois, lorsque le LCD est gros (disons 5 pouces de diagonale ou 480*272 pixels) et que votre microprocesseur est petit, il est même possible de retrouver un contrôleur graphique en plus :

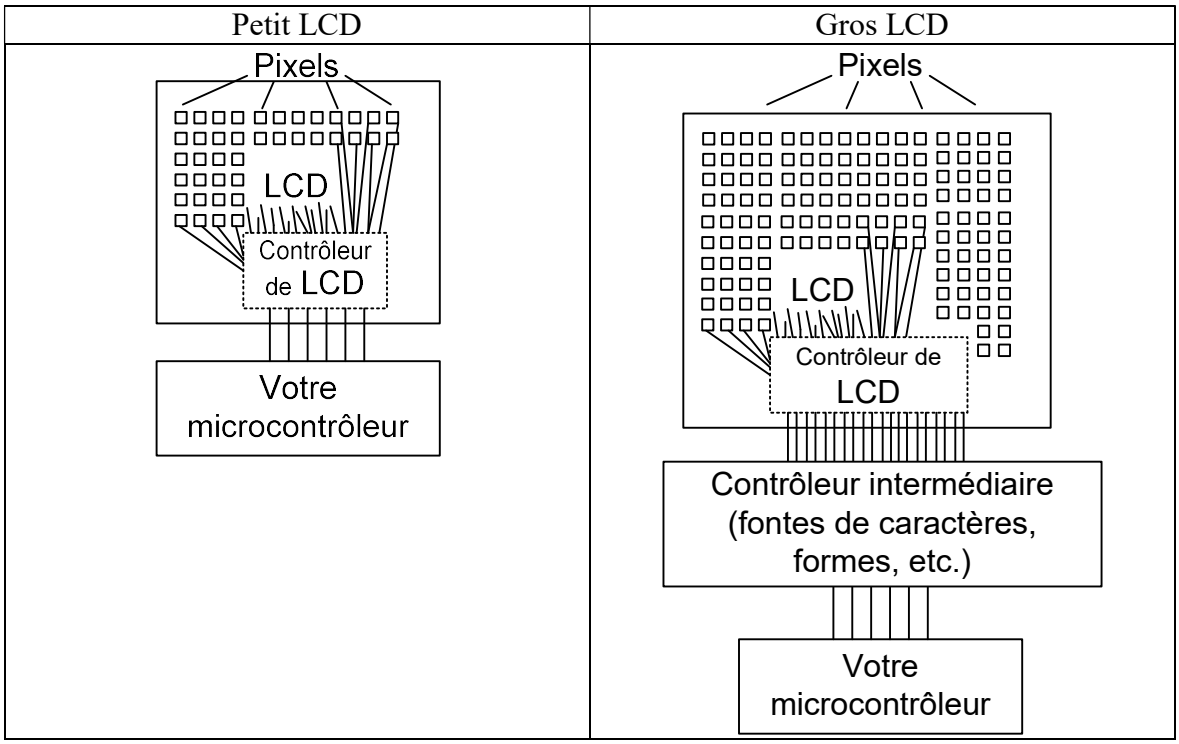


Figure 6 – Contrôleurs de LCD TFT

> *Le choix d'utiliser ou de ne pas utiliser de contrôleur intermédiaire dépend de plusieurs facteurs. Si on peut l'éviter, on l'évite pour réduire les coûts et l'espace requis. Cependant, il faut beaucoup de puissance de calcul et de bande passante pour gérer des milliers de pixel…*

Il y a plusieurs interfaces possibles entre votre microcontrôleur et le contrôleur de LCD. La plupart des contrôleurs de LCD (Driver IC) ou même des contrôleurs intermédiaires de LCD implémentent des interfaces parallèles, des interfaces séries, des interfaces

mémoire (chaque pixel est traité comme une adresse de mémoire), des interfaces analogiques (comme les vieilles télévisions), etc.

Dans le cadre de ce document, seules les interfaces parallèles de type intel-8080 ou motorola-6800 sont décrites.

7.2.1 Interface parallèle intel-8080 et motorola-6800 pour LCD

L'interface parallèle intel-8080 pour LCD (notée i8080 ci-dessous pour alléger le texte) est une interface standard avec les broches suivantes :

- **!CE** : Chip Enable --- Activer le LCD quand 0
- **!WR** : Write Strobe --- Activer l'écriture quand 0
- **!RD** : Read Strobe --- Activer la lecture quand 0
- **D/ !C** : Data/Command --- Déterminer si on écrit une commande quand 0 ou des données quand 1
- **!RES** : Reset --- Permet de faire un reset du LCD
- **D[8, 16 ou X]** : Data --- Bus parallèle de données, habituellement 8 bits ou 16 bits. Sert pour la commande, les données et les adresses

Les diagrammes temporels suivants illustrent le rôle de chaque broche:

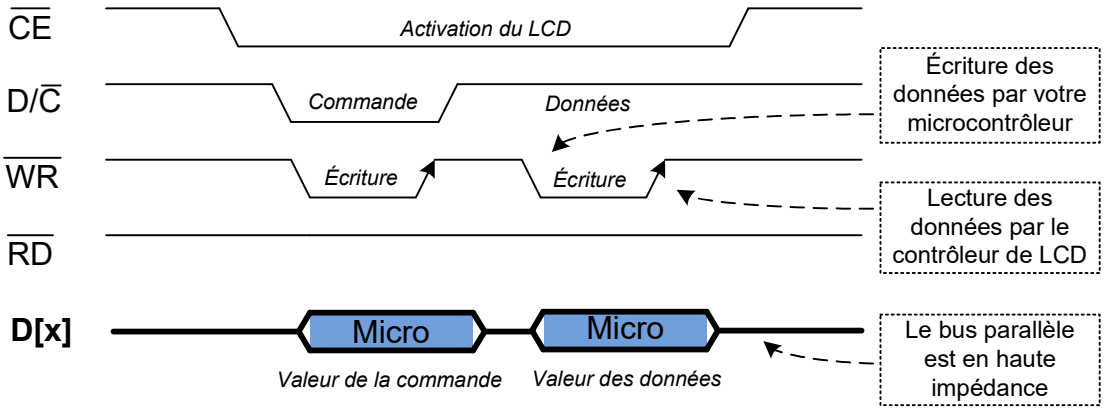


Figure 7 – Écriture de LCD TFT

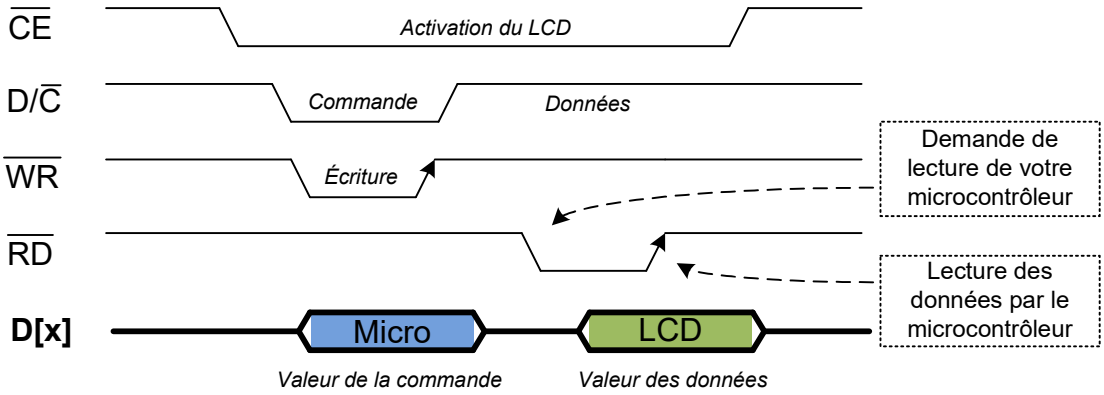


Figure 8 – Lecture de LCD TFT

Les interfaces parallèles intel-8080 et motorola-6800 pour LCD sont presque identiques. Seuls les rôles de quelques broches changent. Par exemple plutôt que d'avoir WR et RD strobe comme le 8080, l'interface 6800 implémente une ligne Strobe et une ligne R/W…

> *La plupart des microcontrôleurs modernes intègrent une interface parallèle permettant de générer les signaux d'accès au LCD automatiquement. Ces signaux sont similaires à des signaux d'accès mémoire et vous n'avez habituellement pas à contrôler chaque broche avec des instructions, même s'il est possible de le faire…*

8 Interface avec un écran tactile

8.1 Principe de fonctionnement de l'écran tactile

Dans l'industrie, on retrouve plusieurs types de touch screen. Ce document décrit les touchscreens résistifs et les touch screen capacitifs qui sont les plus communs.

Voir https://en.wikipedia.org/wiki/Touchscreen pour plus de détails sur les divers touchscreen existants.

8.1.1 Touch Screen Résistif

Le touch screen résistif a les caractéristiques suivantes :

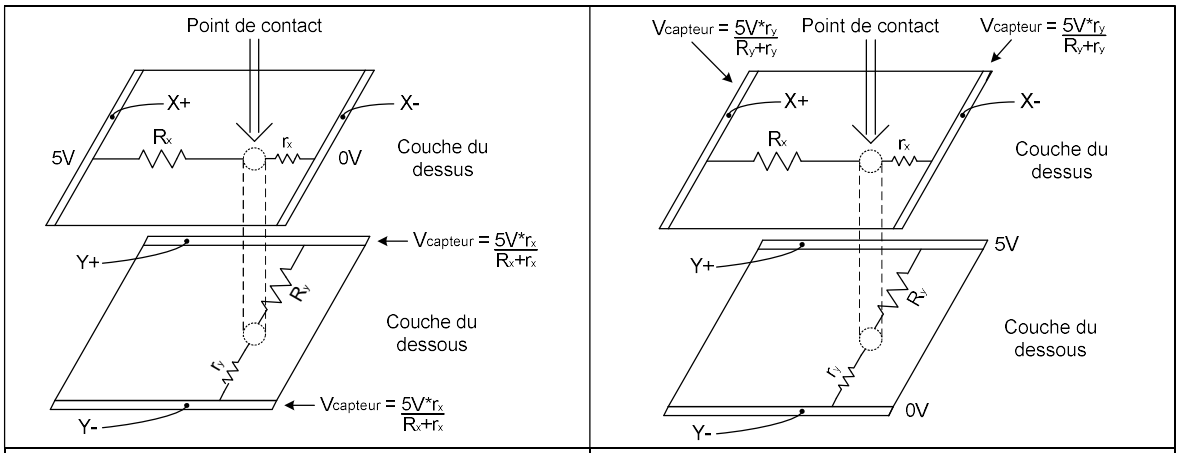
- Détecte la pression
- S'insère autour et au-dessus du LCD
- Coût faible relativement aux autres touch pad
- Moins transparent que les autres touch pad (rendement lumineux de 75%)
- Permet de détecter une seule touche à la fois
- Peut s'user : les deux couches de TIO (voir plus loin) qui se touchent font de petits arcs électriques. Ces « étincelles » peuvent abîmer l'écran avec le temps.
- Peut être endommagé par des objets tranchants

Les écrans tactiles résistifs sont utilisés dans la plupart des applications industrielles et pour certains appareils électroniques, là où la robustesse compense la perte de luminosité.

Un touch screen résistif est fait avec deux couches d'un conducteur résistif transparent (oxyde d'indium avec étain ou ITO ou Indium Tin Oxyde), séparée par une mince couche d'air. Les deux couches se font face et se touchent lorsqu'on appuie sur l'écran tactile.

> *L'indium est l'élément 49 du tableau périodique. Il s'agit d'un métal, qui, comme plusieurs métaux utilisés en électronique, est de plus en plus rare. En 2001, avant d'être utilisé dans les LCD, l'indium se vendait 70$/kg. En 2010, il se vendait environ 500$/kg… Le Canada dispose d'environ 25% des réserves mondiales qui pourraient être épuisées dans une vingtaine d'année !*

La lecture de la position appuyée se fait en deux temps. Dans un premier temps, on applique un voltage sur la couche du haut et la couche conductrice du bas sert de capteur. Dans un second temps, on applique un voltage sur la couche du bas et la couche du haut sert de capteur. Les résistances entre les bords de la couche conductrice et le point appuyé forment un diviseur de tension qui permet de déterminer la position du point appuyé, tel qu'illustré ci-dessous :



Schémas électrique du capteur tactile lors d'un contact, lecture de l'axe X	Schémas électrique du capteur tactile lors d'un contact, lecture de l'axe Y

Figure 9 – Construction d'un écran tactile résistif

Pour lire la position de contact, un microprocesseur utilise généralement des broches qui peuvent être à la fois des GPIOs (pour appliquer le 3.3V/5V) et des ADCs (voir *SMI_C6_Periph_Timer_PWM_ADC.doc*)…

8.1.2 Touch Screen Capacitif

Le touch screen capacitif a les caractéristiques suivantes :

- Détecte une charge capacitive (comme un doigt!) qui apparaît à un point du LCD.
- S'insère autour et au-dessus du LCD
- Coût plus élevé que le touchscreen résistif, mais plus faible que d'autres touch screen
- Très bonne transparence (rendement lumineux de 90%)
- Permet de détecter une seule touche à la fois
- Ne détecte pas les objets non-conducteurs comme les crayons ou les gants de caoutchouc
- Permet de détecter une seule touche à la fois
- Les technologies actuelles ne permettent que de petits/moyens écrans

Les écrans tactiles capacitifs sont utilisés dans plusieurs appareils électroniques comme les tablettes et les téléphones cellulaires.

Un écran tactile capacitif est habituellement construit avec une couche mince d'un conducteur électrique comme l'ITO (voir section précédente). Une charge est appliquée sur le conducteur et cette charge change lorsqu'on appuie sur l'écran. Cette diminution de charge est détectée et donne la position.

Il y a plusieurs façons de détecter le changement de charges sur la couche de conducteur ITO. La figure suivante illustre l'une de ces façons :

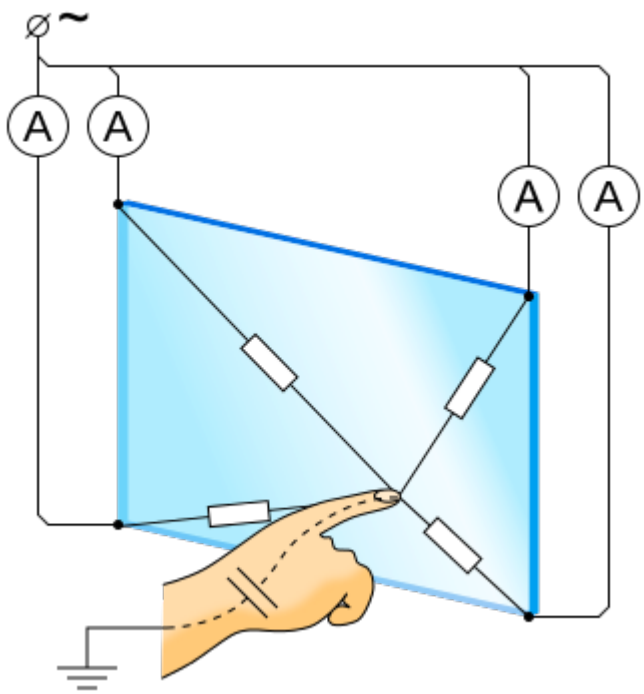


Figure 10 – Construction d'un écran tactile capacitif --- Image provenant de Mercury13 sur Wikipedia : https://commons.wikimedia.org/wiki/File:TouchScreen_capacitive.svg

L'écran tactile illustré à la Figure 10 fonctionne ainsi : une tension AC est appliquée sur l'écran à partir des quatre coins. Lorsque l'utilisateur appuie sur l'écran, du courant passe entre l'écran et la terre : cela décharge l'écran à l'endroit appuyé. En fonction du courant relatif qui passe dans chaque coin, mesuré avec des ampèremètres précis, on peut déterminer la position appuyée…

> *La capacité d'un corps humain est de quelques dizaines de pF à quelques centaines de pF. Pour l'ESDA (Electrostatic Discharge Association), le modèle électrique d'un corps humain est de 100pF avec une résistance série de 1.5kOhms.*

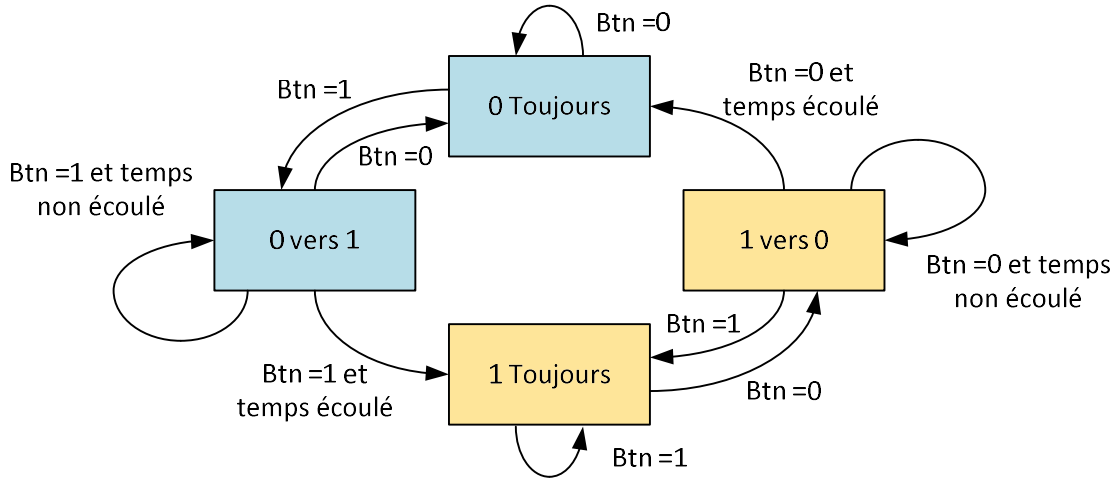
8.2 Interface

Il existe plusieurs circuits intégrés qui permettent d'interfacer avec des écrans tactiles. Par exemple, l'ADS7843 gère les signaux qui permettent de contrôler un écran tactile résistif (voir http://www.ti.com/lit/ds/sbas090b/sbas090b.pdf). Il existe aussi souvent des exemples de code qui permet de le faire dans votre microcontrôleur, tant que celui dispose de plusieurs ADCs assez précis…

9 Annexe A, Debounce de bouton logiciel

Habituellement, les rebonds d'un bouton sont gérés avec une machine d'état d'un point de vue logiciel : lorsque la valeur lue sur la broche du bouton passe de « 0 » à « 1 » ou de « 1 » à « 0 », l'état de la machine d'état change.

Le debounce de bouton se fait habituellement avec un minimum de quatre états : « 0 toujours », « 0 allant vers 1 », « 1 toujours » et « 1 allant vers 0 ». Quand le bouton est dans un état stable (toujours enfoncé ou toujours relâché), l'état ne change pas : il reste « X Toujours ». Par contre, lors d'une transition de X vers Y, l'état devient « X allant vers Y ». Si on lit Y pendant un temps assez long (environ 100ms!), l'état deviendra Y toujours. Sinon, le moindre retour à l'état X nous ramènera à X toujours : on interprétera la brève lecture « Y » comme du bruit à ignorer.



Voici un exemple de code :

```
#define BTN_DEBOUNCE_TIME_MS 50          //Habituellement, les oscillations d'un bouton durent 5-10ms.
//Si on met le temps de debounce trop long, l'usager doit appuyer longtemps sur le bouton pour que le système réagisse

#define READ_MY_BTN()     0  //On doit mettre le registre de valeur du port du bouton ici!

typedef enum
{
  StateButton0,
  StateButton1to0,
  StateButton1,
  StateButton0to1,
}ButtonStateEnum;

typedef enum
{
        BtnNoEvent,
        ButtonPressedEvent,
        ButtonReleasedEvent,
}BtnEventEnum;
```

```
unsigned int msctr;
unsigned int ButtonStateChangeTimestamp;

BtnEventEnum DebounceBouton(void);
void InitBouton(void);

void InitBouton(void)
{
 return;
}

void main(void)
{
	BtnEventEnum IsButtonEvent;
	InitBouton(); //Configure le bouton en entrée digitale avec pull-up (souvent l'état par défaut)

	while(1)
	{
		IsButtonEvent = DebounceBouton();
		if(ButtonPressedEvent)
		{
			//Tache ici
		}
		if(ButtonReleasedEvent)
		{
			//Autre tache ici
		}
	}
}

//On assume que MsTimerISR est appelée quand un timer expire périodiquement à toutes les ms
void MsTimerISR(void)
{
	msctr++;
}

//La fonction DebounceBouton retourne des évènements lorsque le bouton est appuyé ou relâché
//Button pressed is 0 (quand on appuie sur le bouton, la fonction READ_BTN() retourne 0)
//Button release is 1 (quand relâche le bouton, la fonction READ_BTN() retourne 1)
//Lorsqu'on appuie our relâche le bouton, il faut qu'il soit BTN_DEBOUNCE_TIME_MS millisecondes dans un état
pour que cet état soit accepté.

ButtonStateEnum    ButtonState;

BtnEventEnum DebounceBouton(void)
{
 switch(ButtonState)
 {
  case StateButton1:
  {
   if(READ_MY_BTN() == 0)
   {
    ButtonStateChangeTimestamp = msctr;
    ButtonState = StateButton1to0;
   }
   return BtnNoEvent;
  }
  case StateButton1to0:
  {
   if(READ_MY_BTN() == 1)
   {
    ButtonState = StateButton1;
```

```
      return BtnNoEvent;
     }
     else
     {
      if(msctr - ButtonStateChangeTimestamp >= BTN_DEBOUNCE_TIME_MS)
      {
       ButtonState = StateButton0;
       return ButtonPressedEvent;
      }
      else
      {
       return BtnNoEvent;
      }
     }
    }
    case StateButton0:
    {
     if(READ_MY_BTN() == 1)
     {
      ButtonStateChangeTimestamp = msctr;
      ButtonState = StateButton0to1;
     }
     return BtnNoEvent;
    }
    case StateButton0to1:
    {
     if(READ_MY_BTN() == 0)
     {
      ButtonState = StateButton0;
      return BtnNoEvent;
     }
     else
     {
      if(msctr - ButtonStateChangeTimestamp >= BTN_DEBOUNCE_TIME_MS)
      {
       ButtonState = StateButton1;
       return ButtonReleasedEvent;
      }
      else
      {
       return BtnNoEvent;
      }
     }
    }
   }

   return BtnNoEvent;
}
```