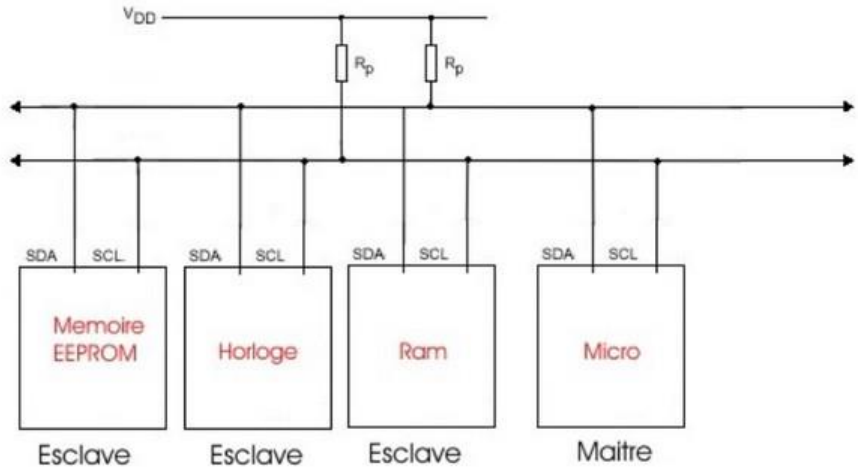
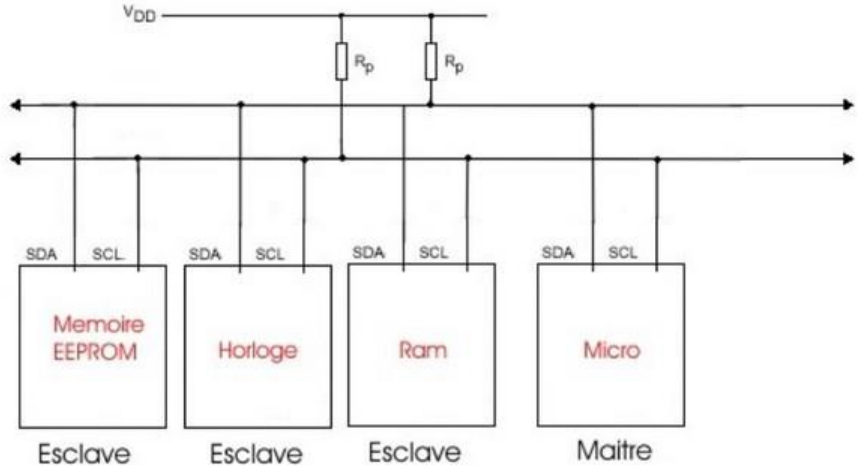
Fonctionnement et utilisation de la liaison I2C



Le bus I2C (Inter-Integrated Circuit) est **un bus série synchrone, bi-directionnel, half duplex**. Conçu par Philips pour les applications de domotique et d'électronique domestique, il permet de relier facilement un microprocesseur et différents circuits périphériques. Ce bus porte parfois le nom de TWI (Two Wire Interface) chez certains constructeurs.



Plusieurs modules peuvent être connectés au bus :

- Un module qui génère un message est un émetteur.
- Un module qui reçoit un message est un récepteur.

Le module qui commande le transfert du message est le maître.
Les modules qui sont commandés par le maître sont les esclaves.

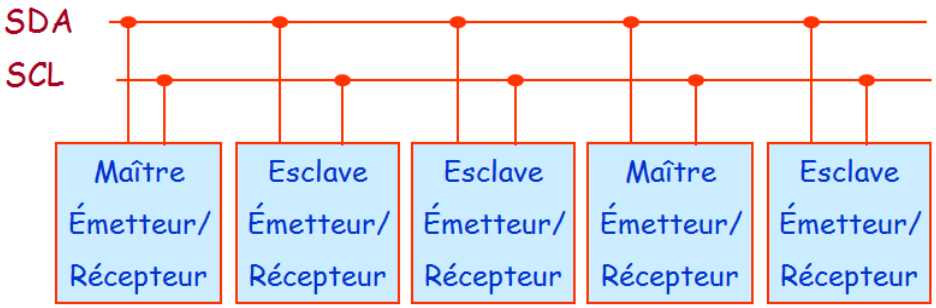
Les échanges ont lieu :

- toujours entre un seul maître et un (ou tous les) esclave(s),
- toujours à l'initiative du maître et donc jamais de maître à maître ou d'esclave à esclave.

Un composant peut passer du statut de maître à celui d'esclave et réciproquement.

Lorsqu"il y a plusieurs modules maîtres, la configuration est dite multimaître. Dans ce cas, une procédure d'arbitrage doit être réalisée lorsque plusieurs maîtres essaient simultanément de commander un transfert afin de n'en autoriser qu'un seul à prendre le contrôle du bus.

Le bus possède 2 lignes : la ligne des données série (**SDA: Serial Data Line**) et la ligne d'horloge série (**SCL: Serial Clock Line**). L'horloge est générée par le maître.



Les mots qui circulent sur le bus ont un format de 8 bits.
Chaque transfert :

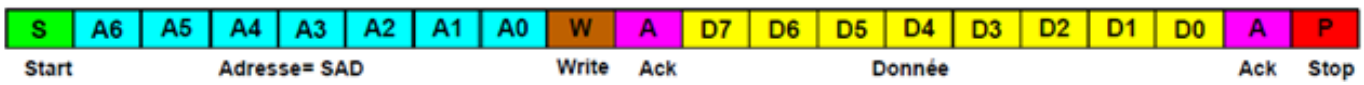
- commence par une condition de **START**
- se termine par une condition de **STOP**.

Ces conditions sont générées par le maître. Après la condition de START, le bus est dit **occupé**. Un certain temps après la condition de STOP, le bus est **libre**.

Après la condition de START :

- le premier mot transmis par le maître est constitué de **l'adresse de l'esclave** (sur 7 bits) avec qui il souhaite communiquer
- puis **d'un bit R/W**
  - à **1** pour la lecture **(read)**
  - à **0** pour l'écriture **(write)**
- L'esclave émet ensuite un **bit d'acquittement Ack** pour signaler la bonne réception de l'octet.
- L'émetteur (maitre ou esclave) émet ensuite les **mots de données**. Après chaque mot transféré, le récepteur émet un bit d'acquittement.
- Quand le maître est le récepteur, il positionne également le bit d'acquittement à **NACK** pour interrompre le dialogue, avant d'envoyer la condition de STOP.

Emetteur donnée : maitre									
Maitre	Start	Adresse	Write		Data		Data		Stop
Esclave				Ack		Ack		Ack	
Emetteur données : esclave									
Maitre	Start	Adresse	Read			Ack		NAck	Stop
Esclave				Ack	Data		Data		

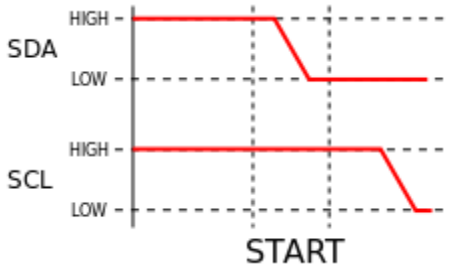


Adresses : Les adresses sont codées sur 7 bits. Elles sont constituées d'une partie fixe sur 4 bits et d'une partie libre sur 3 bits. Les constructeurs de circuits intégrés I2C ne sont pas libres du choix de l'adresse fixe. Celle-ci est imposée par l'I2C Commitee.
Grâce aux 3 derniers bits libres, on peut utiliser 8 circuits identiques sur un même bus.

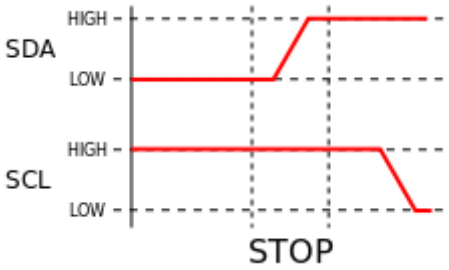
Adresse		sens
$A_6A_5A_4A_3$	$A_2A_1A_0$	R/W
fixe	libre	1 : lecture 0 : écriture

Codage des bits
Les lignes SDA et SCL sont à l'état haut au repos. Le codage utilisé est de type NRZ (non retour à 0).

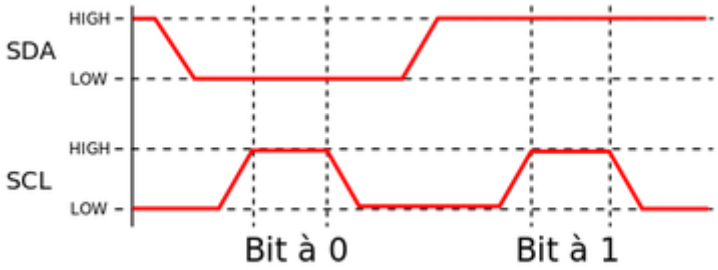
Condition de START : La ligne SDA passe de l'état haut à l'état bas alors que la ligne SCL reste à l'état haut.



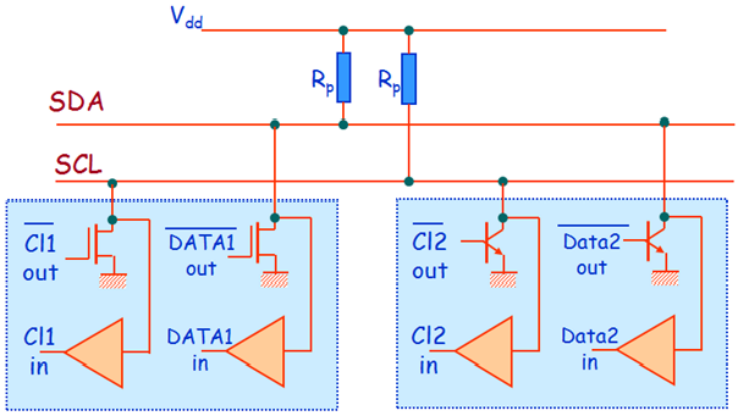
Condition de STOP : La ligne SDA passe de l'état bas à l'état haut alors que la ligne SCL reste à l'état haut.



Emission d'un bit de donnée : Un bit est transféré lors d'une impulsion d'horloge. Pour que le bit soit valide, il faut que la ligne SDA se maintienne à l'état haut ou à l'état bas pendant toute la durée de l'état haut de SCL. Les changements d'état de SDA ne peuvent donc avoir lieu que pendant l'état bas de SCL.



Câblage des entrées-sorties des circuits I2C : Les circuits I2C sont à sortie collecteur ouvert ou drain ouvert. Les 2 lignes sont « tirées » au niveau de tension VDD à travers des **résistances de pull-up** (R_P).



L'entrée des circuits est réalisée sur la même broche que la sortie, ce qui permet au circuit de s'auto-espionner. Le câblage réalise la fonction « ET câblé », ce qui veut dire qu'en cas d'émission simultanée de deux équipements, la valeur 0 « écrase » la valeur 1. On dit que :

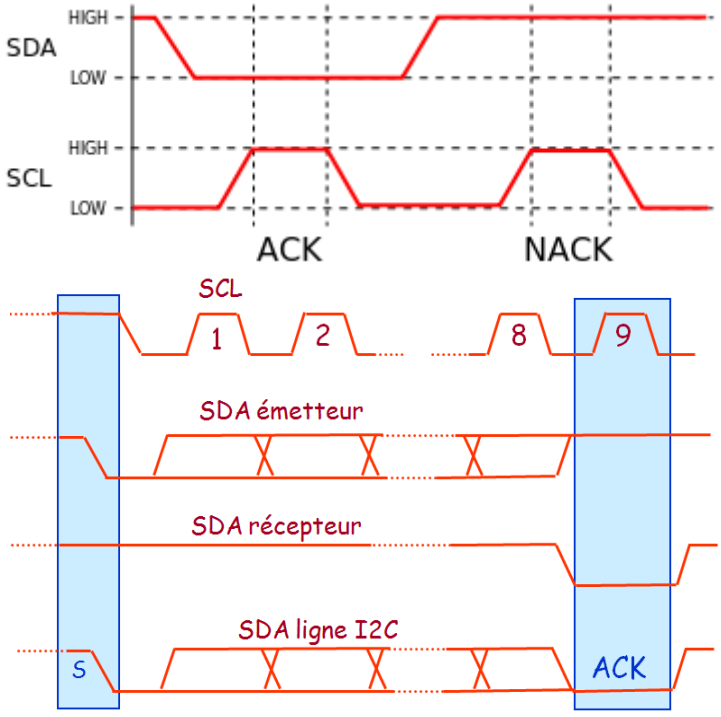
- l'état logique **« 0 »** ou « LOW » est l'état **« dominant »,**
- l'état logique **« 1 »** ou « HIGH » est l'état **« récessif ».**

Lorsqu'il n'y a pas d'émission, la sortie est au niveau haut.

Bit d'acquittement

Sur le neuvième coup d'horloge, l'émetteur libère la ligne de données SDA, c'est-à-dire la place à l'état récessif haut. Le récepteur peut alors émettre le bit d'acquittement :

- « **ACK** », en forçant la ligne SDA au niveau « LOW », pour signaler la bonne réception de l'octet. Il est donc dans ce cas équivalent à un bit à **0**,
- « **NACK** », en laissant la ligne SDA au niveau « HIGH », pour signaler un défaut dans la réception de l'octet, le niveau étant équivalent à un bit à **1**.



Vitesses de transmission : Il existe cinq vitesses de transmission :

- « Standard mode (Sm) » ≤ 100 kbit/s,
- « Fast mode (Fm) » ≤ 400 kbit/s,
- « Fast plus mode (Fm+) » ≤ 1 Mbit/s,
- « High-speed mode (Hs-mode) » ≤ 3,4 Mbit/s,
- « Ultra-fast mode (UFm) » ≤ 5 Mbit/s, unidirectionnel uniquement.

Le bus étant synchrone, le maître impose l'horloge via la ligne SCL, il existe des temps minimums à respecter pour les paliers des niveaux « LOW » et « HIGH » sur cette ligne :

Mode	t_{LOW} min	t_{HIGH} min
Standard	4,7µs	4µs
Fast	1,3µs	0,6µs
Fast plus	0,5µs	0,26µs

Pour les deux vitesses supérieures, les temps dépendent de la capacité du bus (C_B).

Niveaux de tension : Les niveaux utilisés entre les lignes de bus sont proportionnels à VDD :

Etat	**Niveau**
Dominant ou « 0 » ou « LOW »	de -0,5V à 0,3xVDD
Récessif ou « 1 » ou « HIGH »	de 0,7xVDD à VDD

Résistance de Pull-up :
Les temps et les niveaux de tension dépendent de la capacité du bus (C_B) et de la valeur des résistances de pull-up (R_P). Il est difficile de modifier la valeur de la capacité du bus qui est la somme des capacités de sortie des différents circuits + longueur du câble*capacité linéique du câble (ordre de grandeur typique : 100pF/m) mais on peut choisir la valeur des résistances pull-up.
$R_P$min : La valeur minimale des résistances de pull-up est limitée par le courant des sorties SDA et SCL (I_{OL}) lorsqu'elles sont à l'état LOW (V_{OL}) :

Mode	**V_{OL}max**	**I_{OL}**	**$R_P$min pour V_{DD}=5V**
Standard	0,4V	3mA	1534Ω
Fast	0,6V	6mA	733Ω
Fast plus	0,4V	20mA	230Ω

R_PMax : La valeur maximale de R_P est limitée par les temps tr de montée et de descente des signaux SDA et SCL.

Mode	**C_B**	**tr**	**$R_P$max**
Standard	400pF	1µs	2950Ω
Fast	400pF	300ns	885Ω
Fast plus	550pF	120ns	257Ω

Vous pouvez trouver des informations plus complètes sur le Bus I2C à la page:
https://fr.wikipedia.org/wiki/I2C
La documentation de la bibliothèque Arduino Wire (qui permet la communication I2C) est située à la page : https://www.arduino.cc/en/reference/wire

Questions préliminaires:

Combien de lignes sont nécessaires à la communication I2C?

Comment se nomment-t-elles et quelles sont leur rôle?

Qui génère le signal d'horloge?

Peut-on brancher plusieurs esclaves? Combien? Sous quelle condition?

Quelles sont les pattes de l'arduino Uno utilisées pour cette communication?

Sur combien de bits code-t-on une adresse d'esclave?

Un composant I2C a une partie d'adresse fixe et une partie d'adresse variable, combien de bits composent chaque partie?

Si on veut brancher plusieurs esclaves occupe-t-on plus de pattes de l'arduino?

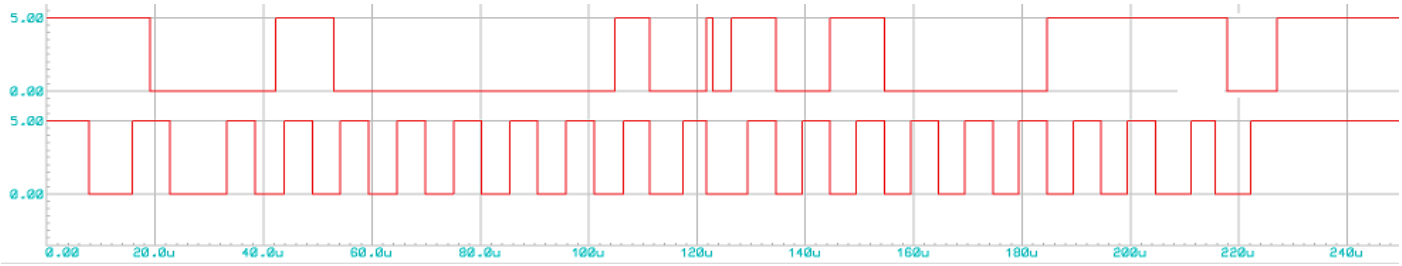
Pour s'entrainer à lire une trame :
Exercice d'application :

BUS I2C Transmission de données Adressage simple 7bits lecture

Master
Slave

S	A6	A5	A4	A3	A2	A1	A0	RW	A	D7	D6	D5	D4	D3	D2	D1	D0	A	P

On relève la trame I^2C suivante : SDA en concordance de temps avec SCL



Décodage de la trame :

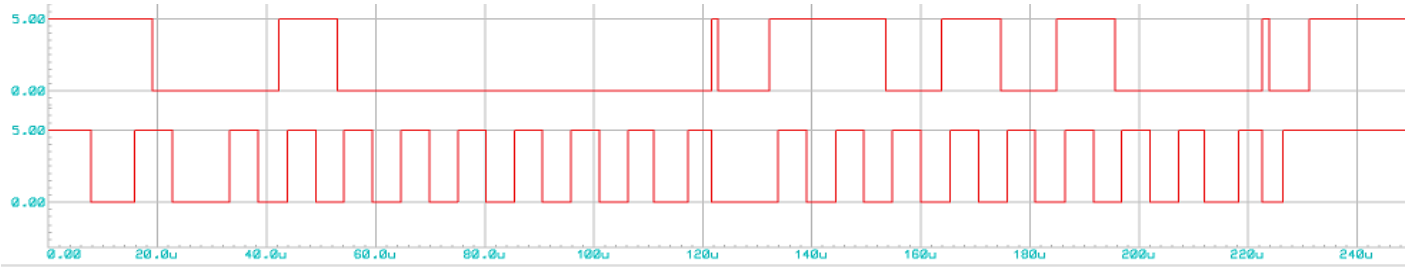
- Entourez sur la trame le bit de START
- Relevez l'adresse de l'esclave. La mettre en hexadécimal
- Entourez sur la trame le bit de R/W
- Quel est son état logique et que cela signifie-t-il ?
- Entourez sur la trame le bit d'acquittement (ACK)
- Quel est son état logique et que cela signifie-t-il ?
- Relevez les 8 bits de données de l'esclave lues. Les mettre en hexadécimal
- Entourez sur la trame le bit de non-acquittement (NACK)
- Quel est son état logique et que cela signifie-t-il ?
- Entourez sur la trame le bit de STOP

BUS I2C Transmission de données Adressage simple 7bits écriture

Master
Slave

S	A6	A5	A4	A3	A2	A1	A0	RW	A	D7	D6	D5	D4	D3	D2	D1	D0	A	P

On relève la trame I^2C suivante : SDA en concordance de temps avec SCL



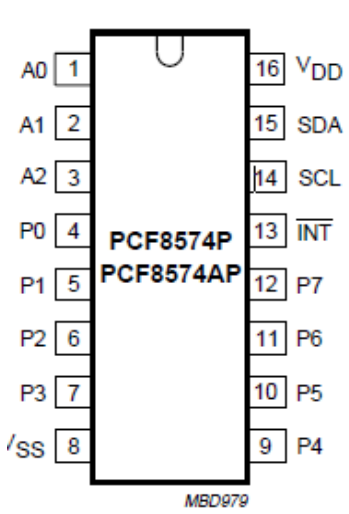
Décodage de la trame :

- Entourez sur la trame le bit de START
- Relevez l'adresse de l'esclave. La mettre en hexadécimal
- Entourez sur la trame le bit de R/W
- Quel est son état logique et que cela signifie-t-il ?
- Entourez sur la trame le bit d'acquittement (ACK)
- Quel est son état logique et que cela signifie-t-il ?
- Relevez les 8 bits de données ecrits. Les mettre en hexadécimal
- Entourez sur la trame le bit d'acquittement (ACK)
- Quel est son état logique et que cela signifie-t-il ?
- Entourez sur la trame le bit de STOP

La carte d'extension I2C PCF

La carte d'extension I2C réalisée à l'IUT est basée autour de 2 PCF8574.
Le circuit 8574 fait la conversion entre un port parallèle E/S et le bus série I2C. Il possède :

- Une partie fixe de son adresse (A6, A5, A4 et A3) égale à **0100**
- Une interface I2C (SDA et SCL)
- 3 bits d'adresse configurables (A0, A1 et A2)
- Un port d'entrées-sorties 8 bits bidirectionnel (P0 à P7).



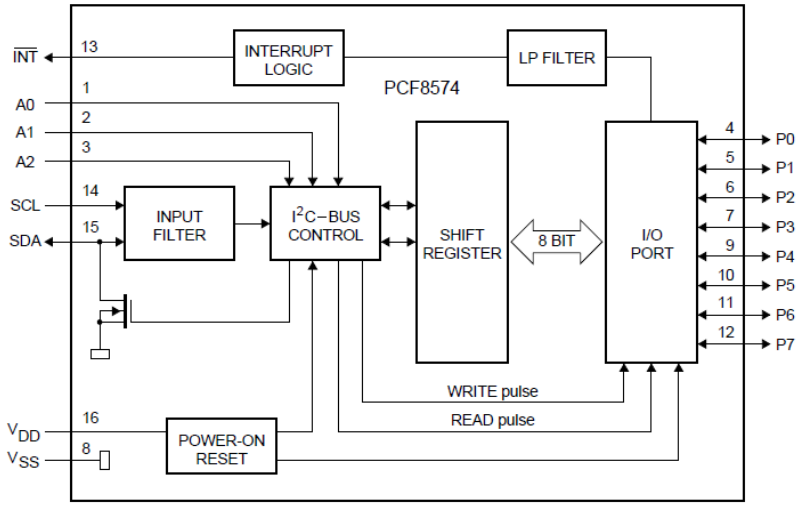
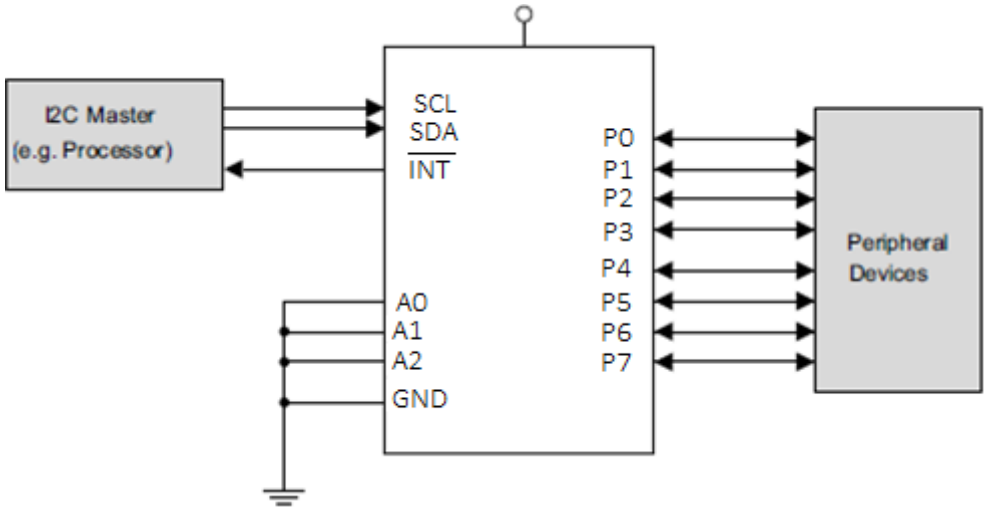
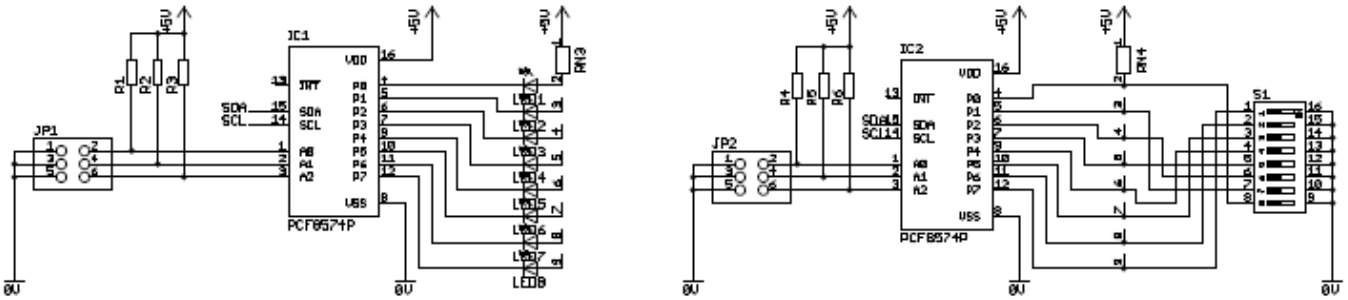


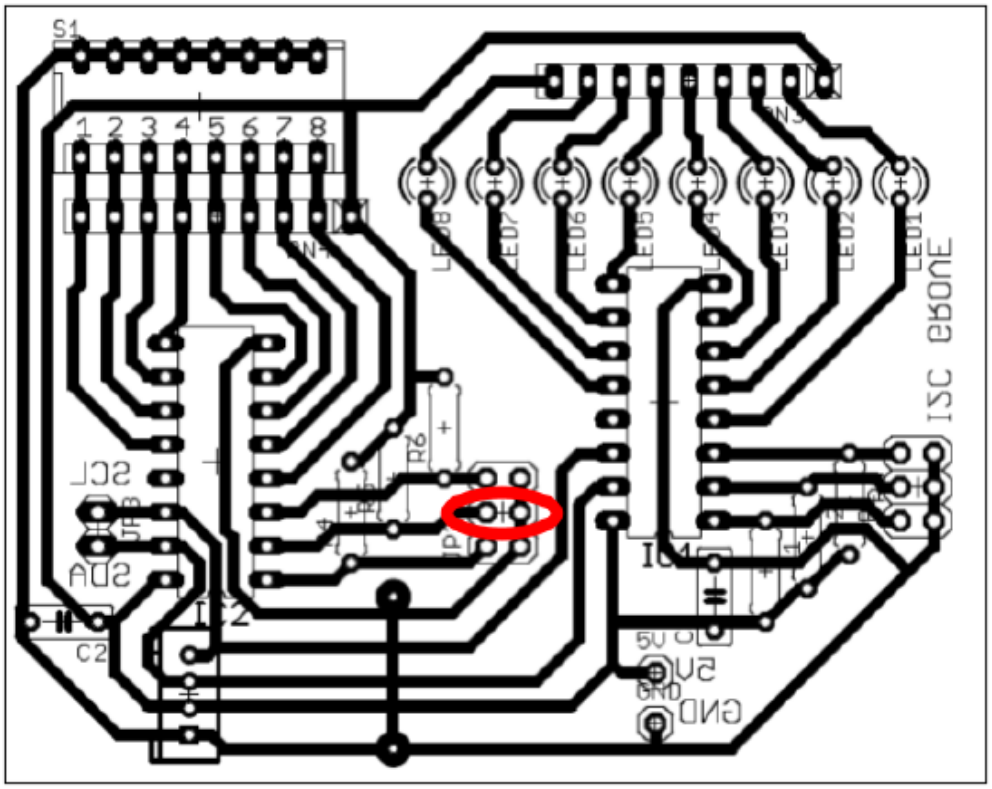
Schéma type d'utilisation :



Voici le schéma de la carte d'extension (vous le trouvez aussi sous AMETICE), sur ce schéma vous pouvez voir qu'un PCF8574 communique avec des leds et l'autre avec des switches:



Voici le routage de la carte d'extension :



ATTENTION !! pour que les 2 PCF 8574 fonctionnent, il faut un cavalier sur les broches du milieu du circuit relié aux switches et aucun autre cavalier sur l'autre.

Etudiez le schéma de la carte d'extension. Repérez les broches du 8574 qui permettent de fixer la partie variable de l'adresse ? Repérez sur le schéma et sur la carte, les cavaliers qui la font varier.
On n'utilise aucun cavalier pour le circuit relié aux leds. Quelle est alors son adresse ?
On place un cavalier sur les broches du milieu pour le circuit relié aux switches. Quelle est alors son adresse ?
On a donc 2 composants identiques sur le bus I2C, mais leurs adresses sont différentes, il n'y aura donc pas de conflit entre eux.

Programme 1 : Côté LED

Nous allons dans un premier temps utiliser le 8574 en mode sortie pour piloter des LED.

La trame SDA sera donc la suivante :

S	A6	A5	A4	A3	A2	A1	A0	W	A	D7	D6	D5	D4	D3	D2	D1	D0	A	P
Start	Adresse= SAD							Write	Ack	Donnée								Ack	Stop

Pour l'émission de la trame, on a recours aux fonctions de la bibliothèque wire.

Recopiez ce programme en remplaçant XXX par la valeur adaptée et YY par 0b10100101 (valeur binaire) ou 0xA5 (valeur en hexadécimale) ou encore 165 (valeur en décimale). L'adresse XXX peut aussi être définie dans ces 3 formats.

```
#include <Wire.h> //bibliothèque I2C
void setup()
{
	Wire.begin(); // connexion au bus I2C, normalement 1 seule fois par programme
	Wire.beginTransmission( XXX ); // début de transmission à l'adresse XXX que vous avez calculée pour les leds
	Wire.write( YY ); // envoie l'octet de données YY à afficher sur les leds
	Wire.endTransmission(); // fin de transmission
}
```

```
void loop()
{
//boucle vide
}
```

Si l'adresse est correcte certaines leds s'allument, d'autres pas, selon la donnée envoyée.
Changez la valeur YY, quel niveau binaire allume une led ?
Changez l'adresse XXX, que se passe-t-il ?

Pour observer les signaux I2C, modifiez le programme : laisser l'instruction Wire.begin dans le setup et déplacez les 3 autres instructions dans la loop. Ajoutez un délai de 200 ms.
Avec une adresse fausse, observez simultanément les 2 signaux SDA et SCL. Pour vous aider à mieux les voir, synchronisez l'oscilloscope sur le front descendant de SDA en mode single.
Relevez les signaux proprement sur du papier à carreaux en indiquant la base de temps.
Comment voit-on de coups d'horloge sur SCL ? Quelle est la valeur de SDA correspondante ? Quelle est la valeur de SDA au moment du 9ème coups d'horloge ?
Interprétez ces signaux en vous aidant de la description de la trame au début de ce texte. Y a-t-il eu acquittement ?

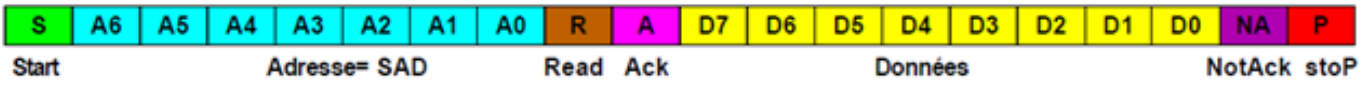
Modifiez l'adresse XXX pour que celle-ci corresponde au circuit relié aux leds. Relevez à nouveau les signaux. Combien y a-t-il maintenant de coups d'horloge ? Interprétez les signaux à nouveau.

Programme 2 : côté switches

Dans ce programme, l'arduino va demander et récupérer l'état des switches de la carte d'extension. La trame est presque la même. Seules différences :

- Le bit R/W est à 1 pour indiquer une lecture
- Les données sont envoyées par l'esclave

La trame SDA aura l'allure suivante :



Quelle est l'adresse du circuit relié aux switches si le cavalier est au milieu comme précédemment ?
Recopiez ce programme en remplaçant XXX par l'adresse de votre circuit:

```
#include <Wire.h> //bibliothèque I2C
void setup()
{
        Wire.begin(); // connexion au bus I2C, normalement 1 seule fois par programme
        Serial.begin(9600); // démarre liaison série
}

void loop()
{
        Wire.requestFrom( XXX , 1); // demande à lire 1 octet de l'esclave d'adresse XXX (que vous avez calculée)
        byte c = Wire.read(); // lire cet octet
        Serial.println(c,BIN); // émet l'octet vers le moniteur
        delay(500) ;
}
```

On observe normalement sur le moniteur série la valeur des switches de la carte.

Programme 3 : leds sur switches

Combinez les 2 programmes précédents pour afficher l'état des switches sur les leds.

Programme 4 : le chenillard 1

Vous avez désormais assez de connaissances pour écrire un programme de chenillard : toutes les leds de la carte I2C sont éteintes, sauf une qui se déplace de gauche à droite, avec 200 ms entre chaque changement de led.

Programme 5 : le chenillard 2

Nous allons utiliser les switches de la carte d'extension pour changer le fonctionnement du chenillard :

- Sw0 change la vitesse de défilement (200/500 ms)
- Sw1 change le sens de défilement (gauche/droite)
- Sw7 autorise / interdit le défilement

Modifiez le programme précédent pour prendre en compte la valeur des switches

Programme 6 : l'accordeur de guitare

Retrouvez le programme accordeur de guitare (TP4 INFO2) et ajouter l'affichage sur les leds de la carte I2C :

- Les leds 7 à 2 représentent les cordes de la guitare, une seule est allumée, pour indiquer la corde détectée.
- Les leds 1 et 0 représentent respectivement l'indication trop tendue / trop molle pour la corde détectée.

Exemple : si l'accordeur trouve que la corde 6 est jouée et qu'elle est trop tendue, on voit sur les leds :

Corde 6 trop tendue

Programme 7 : utilisation d'un composant I2C

Nous allons utiliser désormais un composant autre que le PCF8574, il s'agit du BMP280. Vous trouverez une présentation de ce composant sur le lien suivant :
https://wiki.seeedstudio.com/Grove-Barometer_Sensor-BMP280/

D'après cette documentation :
Quelles grandeurs physiques sont mesurées par ce composant ?
Par quels biais peut-on communiquer avec ce composant ?
Comment change-t-on de moyen de communication ?
Quelle est l'adresse I2C de ce composant ?
Peut-on en changer ?
Si oui comment ?
Quelles sont les adresses I2C possibles pour ce module ?

Vous trouverez sur ce lien la bibliothèque associée à ce composant :
https://github.com/Seeed-Studio/Grove_BMP280
Pour télécharger la bibliothèque :
cliquez sur code / dowload zip, mettre le fichier zippé dans votre répertoire documents/aduino/librairies
Puis indiquez au logiciel arduino où est cette bibliothèque en cliquant sur croquis/inclure une bibliothèque/ajouter la bibliothèque .ZIP, indiquez alors le répertoire zippé et cliquez sur ouvrir
Vous pouvez désormais utiliser cette bibliothèque et notamment le programme d'exemple.
Si besoin inclure la bibliothèque string.h
Vérifiez le bon fonctionnement de ce composant.

Programme 8 : utilisation d'une matrice de leds

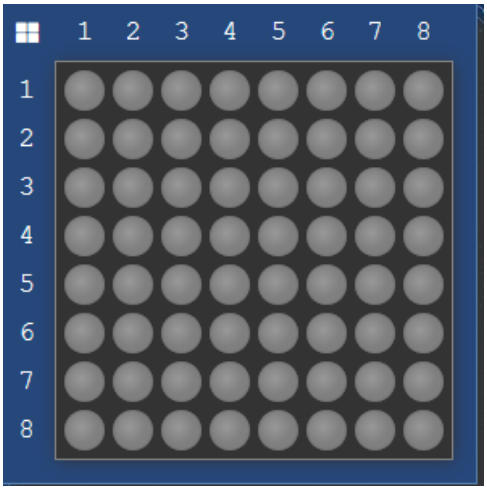
La matrice utilisée est une matrice 8 lignes, 8 colonnes.
Vous trouverez sa description sur le lien :
https://wiki.seeedstudio.com/Grove-Red_LED_Matrix_w_Driver/

et sa bibliothèque sur le lien : https://github.com/Seeed-Studio/Grove_LED_Matrix_Driver_HT16K33

La démarche d'installation de la bibliothèque est identique à la démarche utilisée pour le BMP280.

D'après la documentation technique, quelle est l'adresse I2C par défaut de ce module ?
Quelles sont les autres adresses possibles ?
Peut-on utiliser ce module et le BMP280 dans une même application ?
Sous quelle condition peut-on les utiliser ensemble ?

La matrice se présente sous cette forme :



On s'adresse à la matrice en donnant une valeur hexadécimale de 16 chiffres. Chaque chiffre correspond à 4 leds d'une ligne.

L8 L7 L6 L5 L4 L3 L2 L1

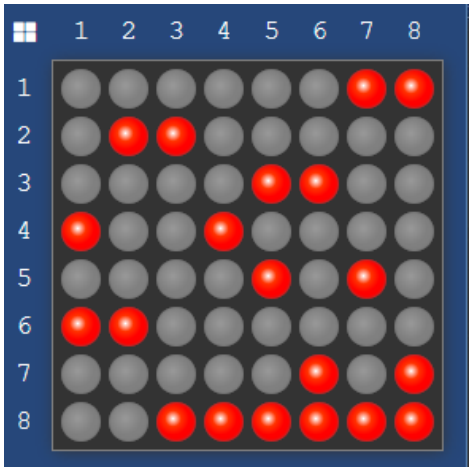
Valeur = 0x00 00 00 00 00 00 00 00

1 : led (1,1)
2 : led (1,2)
4 : led (1,3)
8 : led (1,4)

1 : led (1,5)
2 : led (1,6)
4 : led (1,7)
8 : led (1,8)

Et ce raisonnement est identique pour toutes les lignes.

Exemple pour allumer la matrice de cette manière :



Il faut envoyer à la matrice la valeur suivante : 0xfca00350093006c0

Une fois que vous avez téléchargé la bibliothèque, lancez l'exemple display_number.ino
Observez le résultat.
Lancez l'exemple display_icon.ino et observez le résultat.

Copiez l'exemple :

```
#include "Grove_LED_Matrix_Driver_HT16K33.h"
#include <Wire.h>

const uint64_t Emoji[] =
{
  0x3c4299a581a5423c,
  0x3c4281bd81a5423c,
  0x3c42a59981a5423c,
};

Matrix_8x8 matrix;

void setup()
{
    Wire.begin();
    matrix.init();
    matrix.setBrightness(0);
    matrix.setBlinkRate(BLINK_OFF);
}

void loop()
{
  for (int i = 0;i < 3;i++)
    {
        matrix.writeOnePicture(Emoji[i]);
        matrix.display();
        delay(500);
    }
}
```

Qu'observez-vous ?
Grâce au site : https://xantorohara.github.io/led-matrix-editor/
Amusez-vous à changer le tableau Emoji de ce programme (attention à la taille de votre tableau).

Programme 9 : le chenillard 3

En utilisant une des matrices qui suivent, écrivez le programme permettant de réaliser un chenillard.
Améliorez votre programme pour que celui-ci obéisse au cahier des charges du programme 5 (changement de vitesse, de sens, et marche/pause en fonction des switches)

- Voici le tableau de valeurs pour réaliser un chenillard en diagonale :

```
const uint64_t IMAGES[] = {
  0x0100000000000000,
  0x0201000000000000,
  0x0402010000000000,
  0x0804020100000000,
  0x1008040201000000,
  0x2010080402010000,
  0x4020100804020100,
  0x8040201008040201,
  0x0080402010080402,
  0x0000804020100804,
  0x0000008040201008,
  0x0000000080402010,
  0x0000000000804020,
  0x0000000000008040,
  0x0000000000000080
};
const int IMAGES_LEN = sizeof(IMAGES)/8;  //permet de connaitre la longueur de la matrice
```

- Voici le tableau de valeurs pour réaliser un chenillard carré :

```
const uint64_t IMAGES[] = {
  0x0000001818000000,
  0x00003c24243c0000,
  0x007e424242427e00,
  0xff818181818181ff,
  0x007e424242427e00,
  0x00003c24243c0000
};
const int IMAGES_LEN = sizeof(IMAGES)/8;
```

- et enfin le tableau de valeurs pour chenillard qui éteint les leds au fur et à mesure en partant du milieu.

```
const uint64_t IMAGES[] = {
  0xffffffff7fffffff,
  0xffffffe7ffffffff,
  0xffffffe7efffffff,
  0xffffffe7e7ffffff,
  0xffffffe7e3ffffff,
```

0xffffffe3e3ffffff,
0xfffffbe3e3ffffff,
0xfffff3e3e3ffffff,
0xffffe3e3e3ffffff,
0xffffc3e3e3ffffff,
0xffffc3c3e3ffffff,
0xffffc3c3c3ffffff,
0xffffc3c3c3dfffff,
0xffffc3c3c3cfffff,
0xffffc3c3c3c7ffff,
0xffffc3c3c3c3ffff,
0xffffc3c3c3c1ffff,
0xffffc3c3c1c1ffff,
0xffffc3c1c1c1ffff,
0xffffc1c1c1c1ffff,
0xfffdc1c1c1c1ffff,
0xfff9c1c1c1c1ffff,
0xfff1c1c1c1c1ffff,
0xffe1c1c1c1c1ffff,
0xffc1c1c1c1c1ffff,
0xff81c1c1c1c1ffff,
0xff8181c1c1c1ffff,
0xff818181c1c1ffff,
0xff81818181c1ffff,
0xff8181818181ffff,
0xff8181818181bfff,
0xff81818181819fff,
0xff81818181818fff,
0xff818181818187ff,
0xff818181818183ff,
0xff818181818181ff,
0xff818181818180ff,
0xff818181818080ff,
0xff818181808080ff,
0xff818180808080ff,
0xff818080808080ff,
0xff808080808080ff,
0xfe808080808080ff,
0xfc808080808080ff,
0xf8808080808080ff,
0xf0808080808080ff,
0xe0808080808080ff,
0xc0808080808080ff,
0x80808080808080ff,
0x00808080808080ff,
0x00008080808080ff,
0x00000080808080ff,
0x00000000808080ff,

```
  0x00000000008080ff,
  0x00000000000080ff,
  0x00000000000000ff,
  0x000000000000007f,
  0x000000000000003f,
  0x000000000000001f,
  0x000000000000000f,
  0x0000000000000007,
  0x0000000000000003,
  0x0000000000000001,
  0x0000000000000000
};
const int IMAGES_LEN = sizeof(IMAGES)/8;
```

Programme 10 : l'horloge temps réel de la carte Seeed Starter Shield

Le circuit VS1307N de la carte d'extension est une horloge temps réel : il compte jours, heures, minutes et secondes. Il est branché sur le port I2C.
Sa documentation est sur le site : https://wiki.seeedstudio.com/Starter_Shield_EN/
Et sa bibliothèque sur le site : https://github.com/Seeed-Studio/Starter_Shield
Pour l'utiliser, copiez le répertoire TickTockShieldV2 dans votre répertoire librairies et ajoutez-y le fichier streaming.h que vous trouverez dans le répertoire librairies de la bibliothèque chargée.

```
#include <Wire.h>
#include <TTSTime.h>

TTSTime time;

int hour;
int min;
int sec;

void setup()
{
   Serial.begin(9600);
   Wire.begin();
   Serial.println("hello world");
   //time.setTime(12, 59, 55);
}

void loop()
{
   Serial.print(time.getHour()) ;
   Serial.print(" : ") ;
   Serial.print(time.getMin()) ;
   Serial.print( " : ") ;
   Serial.println(time.getSec()) ;
   delay(1000);
}
```

Essayez le programme précédent avec la ligne time.setTime(12,59,55) ; en commentaire. Puis réessayer en décommentant cette ligne. Quelle modification voyez-vous ?
Changez la valeur entre parenthèses. Réessayez.
Avec cette bibliothèque, on ne peut pas régler la date. Essayez désormais ce programme :

```
#include <Wire.h>
 byte i,j,k;
void setup()
{
	Wire.begin();
	Wire.beginTransmission(0x68); //0x68
	Wire.write(0);
	Wire.write(0);
	Wire.endTransmission();
	Serial.begin(9600);
	Wire.beginTransmission(0x68);
	Wire.write(0);
	Wire.write(0x23);
	Wire.write(0x59);
	Wire.write(0x23);
	//Wire.write(0x02);
	//Wire.write(0x12);
	//Wire.write(0x05);
	Wire.endTransmission();
}

void loop()
{
	Wire.beginTransmission(0x68); //0x68
	Wire.write(0);
	Wire.endTransmission();
	Wire.requestFrom(0x68,3);
	i = Wire.read();
	j = Wire.read();
	k = Wire.read();
	//lecture de l,m,n
	Wire.endTransmission();
	Serial.print(k, HEX);
	Serial.print(" h ");
	Serial.print(j, HEX);
	Serial.print(" mn ");
	Serial.print(i, HEX);
	Serial.println(" s");
	//affichage de l,m,n
	delay(1000);
}
```

Ce programme fonctionne t'il comme le précédent ?
Décommentez les lignes en commentaire et rajouter les variables l, m ,n au niveau de la lecture (changez le 3 en 6 dans le Wire.requestFrom), remplissez-les et affichez-les dans la loop. D'après la documentation technique du VS1307 que vous trouverez sur AMETICE, à quoi correspondent ces valeurs ?

Reprenez le même programme en enlevant le contenu du setup, fonctionne-t-il encore ?
Conclusion : on utilise une horloge temps réel dans les applications pour horodater les prises de données, il faut alors dans un premier programme configurer la date et l'heure, puis (tant qu'il y a la pile), l'horloge évolue et nous pouvons l'interroger pour connaitre l'heure dans un second programme.

Programme 11 : pour aller plus loin

Reprenez le programme 5 ou le programme 9 du chenillard et rendez-le plus réactif, pour cela :

- Remplacez les switches de la carte d'extension par les boutons K1, K2 et K3 de la carte Starter Shield
- Initialisez le chenillard en mode pause, première led (ou premier motif) allumée, delay à 200 ms.

- Configurez les interruptions sur changement de valeur des boutons K1, K2 et K3 et dans leur programme d'interruption faites-en sorte que K1 modifie le temps de délai, K2 le mode de défilement et K3 met en marche ou pause le système.

Bilan : vous savez désormais ce qu'est une communication I2C, connecter un module I2C et communiquer avec, faire attention à l'adresse des modules I2C, allumer une matrice de leds et gérer une horloge temps réel.