.

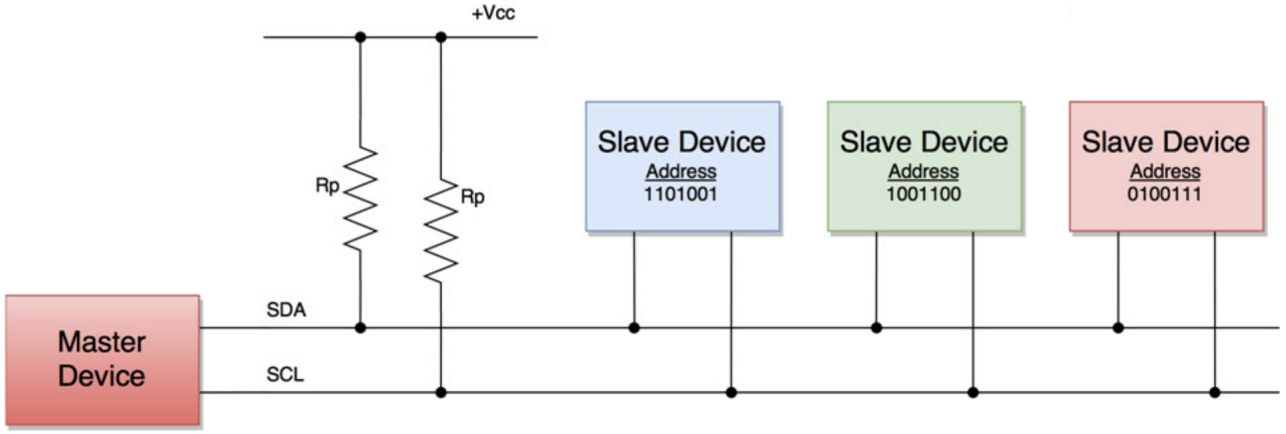
Table des Matières

1. Communication I2C entre 2 Arduino

I2C est l'abréviation de Inter-Integrated Circuit, un protocole de communication série synchrone développé par Phillips pour la communication entre un microcontrôleur rapide et des périphériques relativement lents (comme la mémoire ou les capteurs) en utilisant seulement deux fils. Par conséquent, il est parfois également connu sous le nom de TWI (Two Wire Interface).

En utilisant I2C, on peut transmettre des données à des débits de 100 kbit/s (horloge 100 kHz – Standard Mode), 400 kbit/s (horloge 400 kHz – Fast Mode), 1 Mbit/s (horloge 1 MHz – Fast Mode Plus) et 3,4 Mbit/s (horloge 3,4 MHz – High Speed Mode).

Le bus I2C se compose de deux fils appelés données série (SDA) et horloge série (SCL). Les données sont transmises via la ligne SDA tandis que la ligne SCL est utilisée pour synchroniser les appareils avec le signal d'horloge.



Il existe deux types d'appareils qui se connectent au bus I2C : maître et esclave. Les maîtres de bus sont responsables de l'envoi et de la réception de données vers et depuis les dispositifs esclaves. Le signal d'horloge est également fourni par le maître.

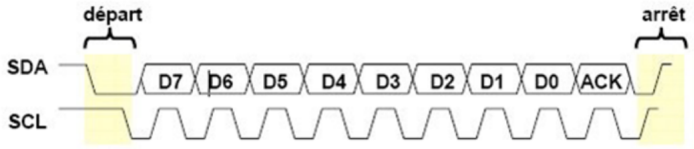
Le réseau I2C prend en charge plusieurs maîtres et plusieurs esclaves (généralement un seul maître et plusieurs esclaves). Chaque appareil esclave connecté au bus I2C possède une adresse unique de 7 bits.
En utilisant cette adresse, le maître sélectionne un esclave particulier pour la transmission de données (envoi ou réception) et l'esclave sélectionné répond en fonction de la demande.

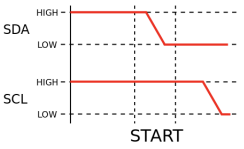
- Le maître envoie sur le bus l'adresse du composant avec qui il souhaite communiquer.
- L'esclave qui reconnaît son adresse répond à son tour par un signal de confirmation, puis le maître continue la procédure de communication ...(écriture/lecture).

Les transactions seront confirmées par un acquittement (ACK : acknowledge). ACK est actif à 0.

Prise de contrôle du bus I2C :

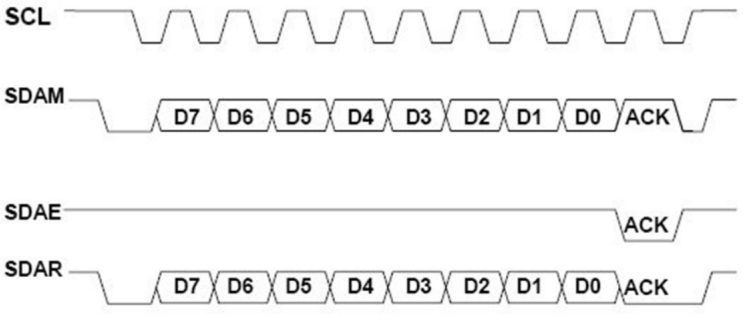
- Le bus doit être au repos avant la prise de contrôle : SDA=1 et SCL=1.
- Pour transmettre des données, il faut surveiller :
  - La condition de départ : SDA passe à 0 et SCL reste à 1.
  - La condition d'arrêt : SDA passe à 1 et SCL reste à 1.





Transmission d'un octet :

- Le maître transmet le bit de poids fort D7 sur SDA.
- Il valide la donnée en appliquant un niveau 1 sur SCL.
- Lorsque SCL retombe à 0, il poursuit avec D6, et ainsi de suite jusqu'à ce que l'octet soit transmis.
- Il envoie le bit ACK à 1 en scrutant l'état de SDA.
- L'esclave doit imposer un 0 pour indiquer que la transmission s'est effectuée correctement.
- Le maître voit le 0 et peut passer à la suite.



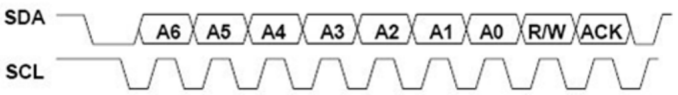
SCL : horloge imposée par le maître.
SDAM : niveaux de SDA imposés par le maître.
SDAE : niveaux de SDA imposés par l'esclave.
SDAR : niveaux de SDA réels résultants.

Fonctionnement de l'acquittement :

- Le maître libère la ligne SDA.
- L'esclave force la ligne SDA à 0.
- Le maître envoie une impulsion sur l'horloge SCL.
- Lorsque l'impulsion retombe à 0, l'esclave libère SDA.

Transmission d'une adresse :

Chaque composant possède une adresse codée sur 7 bits et le maître indique s'il souhaite lire ou écrire dans ce composant grâce au bit R/W (0 en écriture et 1 en lecture).



Écriture d'une donnée :

— Le maître transmet l'adresse du composant avec R/W à 0.

— Il envoie le bit ACK à 1 en scrutant l'état de SDA.

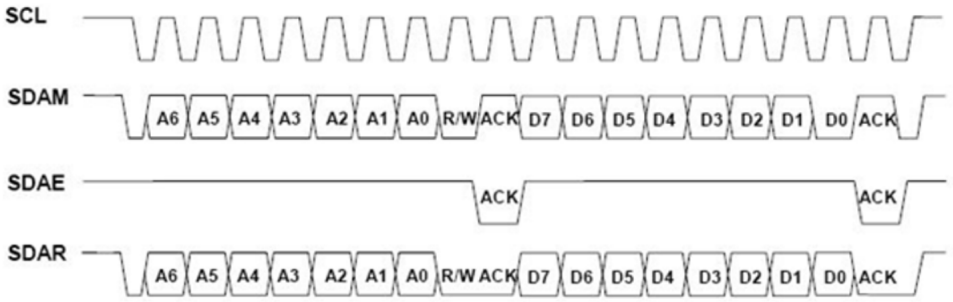
— L'esclave doit imposer un 0 pour indiquer que la transmission s'est effectuée correctement.

— Le maître transmet la donnée en commençant par D7.

— Il envoie le bit ACK à 1 en scrutant l'état de SDA.

— L'esclave doit imposer un 0 pour indiquer que la transmission s'est effectuée correctement.

— Le maître voit le 0 et peut passer à la suite.



Lecture d'une donnée :

— Le maître transmet l'adresse du composant avec R/W à 1.

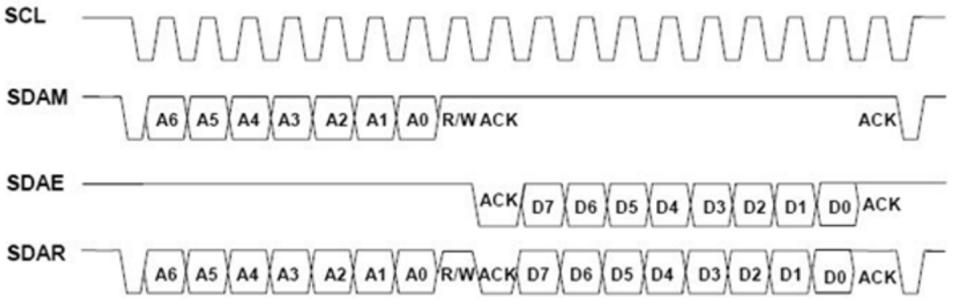
— Il envoie le bit ACK à 1 en scrutant l'état de SDA.

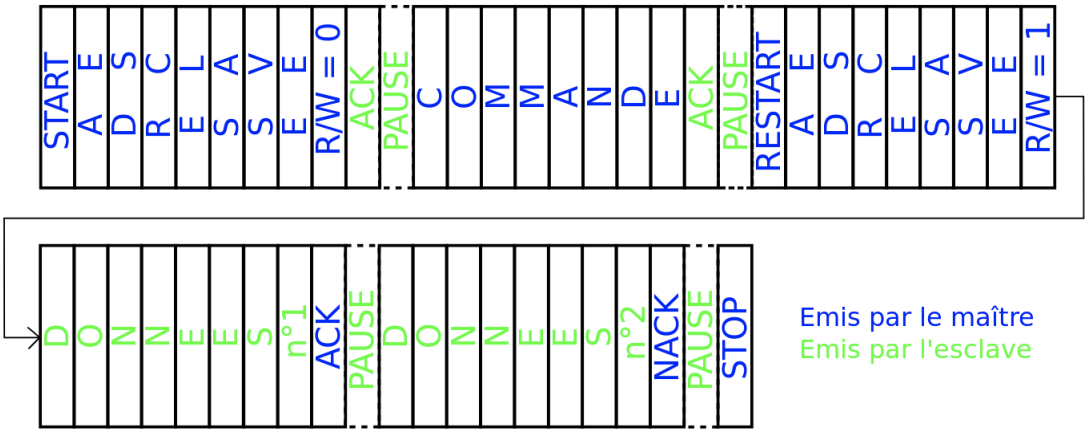
— L'esclave doit imposer un 0 pour indiquer que la transmission s'est effectuée correctement.

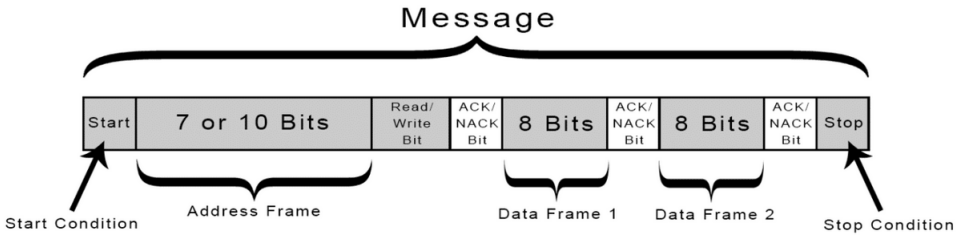
— L'esclave transmet la donnée en commençant par D7.

— Il envoie le bit ACK à 1 en scrutant l'état de SDA.

— Le maître doit imposer un 0 pour indiquer que la transmission s'est effectuée correctement.







1.1 I2C dans Arduino :

Arduino prend en charge la communication I2C : les broches d'entrée analogique A4 et A5 ont une fonction alternative d'I2C.
La broche A4 agit comme SDA tandis que la broche A5 agit comme SCL. Dans de l'Arduino UNO R3 d'origine, il y a deux autres broches près de Digital IO Pin 13 (près de la prise USB), dédiées à SDA et SCL.

Utiliser l'interface Arduino I2C :

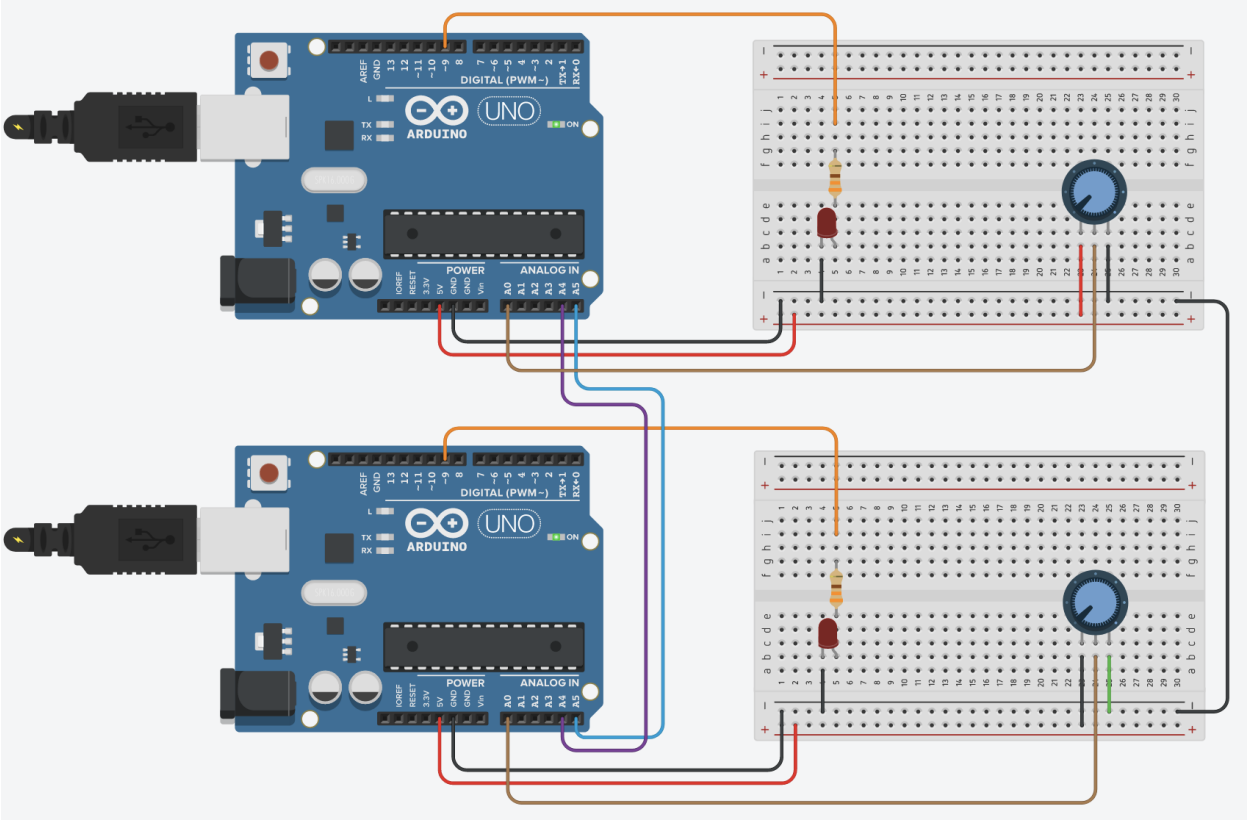
On connecte deux cartes Arduino UNO pour communiquer via le bus I2C. Pour rendre les choses intéressantes et voir réellement la communication, on ajoute quelques LEDs et potentiomètres (pour chaque carte Arduino).

Les potentiomètres sont connectés aux broches d'entrée analogique respectives (A0) et les LEDs sont connectées à une broche E/S numérique avec PWM (broche 9). Une carte Arduino est configurée en tant que maître de bus I2C et l'autre UNO est configurée en tant que périphérique esclave.

Lorsque on ajuste le potentiomètre connecté au Master Arduino, il capture la lecture analogique du potentiomètre, la convertit en une valeur numérique (dans la plage de 0 - 1023), la mappe à une valeur PWM appropriée (dans la plage de 0 – 255) et transmet cette valeur à l'Arduino esclave via le bus I2C.

L'Arduino esclave, lors de la réception de la valeur PWM, ajuste la luminosité de sa LED. De plus, l'Arduino esclave envoie sa propre valeur de potentiomètre convertie en PWM à l'Arduino maître (sur demande du maître).

L'Arduino maître lit ensuite la valeur PWM de l'Arduino esclave et ajuste la luminosité de sa LED en fonction de cette valeur. Cette communication se poursuit et se répète de manière transparente sur le bus I2C.


DIGITAL (PWM~)
UNO
ARDUINO
POWER
ANALOG IN

Code Arduino Master :

Arduino Master BTS SN : KB

```
1
2  /*Arduino Master I2C*/
3
4  #include <Wire.h>
5  #define ledPin 9
6
7  byte rcvData;
8  int potValue;
9
10 void setup()
11 {
12   Wire.begin();
13   rcvData = 255;
14   pinMode(ledPin, OUTPUT);
15 }
16
17 void loop()
18 {
19   potValue = analogRead(A0);
20   potValue = map(potValue, 0, 1023, 0, 255);
21
22   Wire.beginTransmission(0x14);
23   Wire.write(potValue);
24   Wire.endTransmission();
25
26   Wire.requestFrom(0x14, 1);
27   if(Wire.available())
28   {
29     rcvData = Wire.read();
30   }
31
32   analogWrite(ledPin, rcvData);
33 }
```

Code Arduino Slave :

Dans le code esclave Arduino, on définit l'adresse esclave comme 0x14. Il peut s'agir de n'importe quelle valeur inférieure à 128. Un point important à noter à propos de la bibliothèque ArduinoI2C est qu'elle utilise une adresse I2C 7 bits sans le bit de lecture/écriture.

Donc, si on a une adresse 8 bits (qui inclut le bit R/W), décaler l'adresse de 1 à droite, puis on l'utilise dans la bibliothèque. La bibliothèque ajustera automatiquement l'adresse en fonction de l'opération de lecture ou d'écriture.
On doit s'assurer également que l'adresse de l'esclave est unique et que deux esclaves ne doivent pas avoir la même adresse.

Dans le code, on a déclaré deux fonctions "DataReceive" et "DataRequest" à appeler lorsque des données sont reçues par l'esclave ou que des données sont demandées à l'esclave. Les données reçues par l'esclave dans la fonction DataReceive contiennent la valeur PWM envoyée par le maître.
Les données à transmettre via la fonction DataRequest sont la valeur PWM de l'esclave au maître.

Arduino Slave BTS SN : KB

```
1
2  /*Arduino Slave I2C*/
3
4  #include <Wire.h>
5  #define ledPin 9
6
7  byte rcvData;
8  int potValue;
9
10 void setup()
11 {
12   Wire.begin(0x14);
13   /*Event Handlers*/
14   Wire.onReceive(DataReceive);
15   Wire.onRequest(DataRequest);
16
17   rcvData = 255;
18   pinMode(ledPin, OUTPUT);
19 }
20
21 void loop()
22 {
23   potValue = analogRead(A0);
24   potValue = map(potValue, 0, 1023, 0, 255);
25   analogWrite(ledPin, rcvData);
26 }
27
28 void DataReceive(int numBytes)
29 {
30   while(Wire.available())
31   {
32     rcvData = Wire.read();
33   }
34 }
35
36 void DataRequest()
37 {
38   Wire.write(potValue);
39 }
```

Questions :

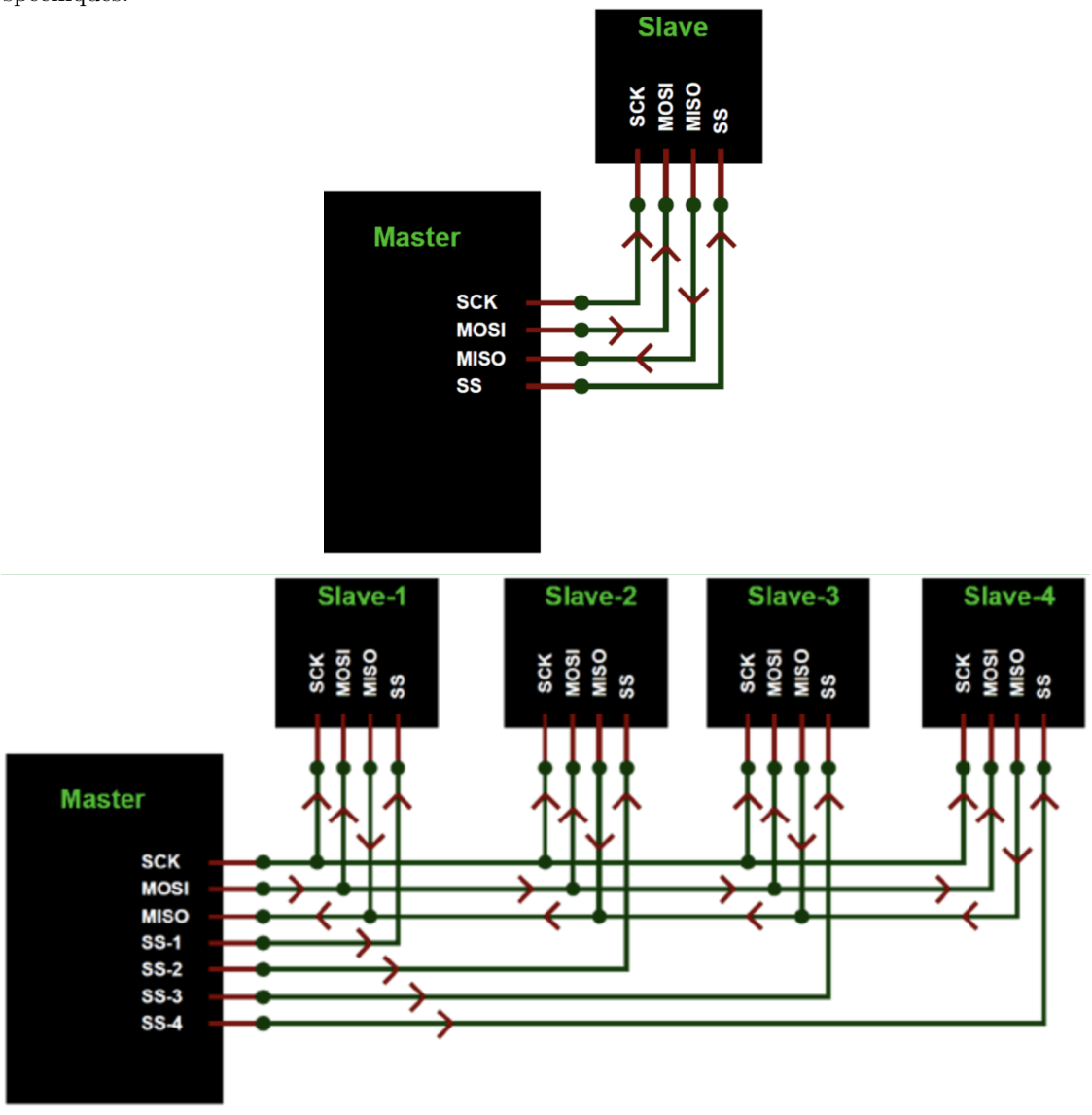
Certains oscilloscopes peuvent réaliser l'analyse de trame (analyseur logique). Vérifier sur internet si l'oscilloscope utilisé possède cette fonctionnalité (en mentionnant la marque et la référence).

1. Visualiser les différents signaux I2C à l'oscilloscope.
2. Afficher les trames.
3. Commenter les résultats.

2. Communication SPI entre 2 Arduino

SPI comprend 4 lignes : MISO, MOSI, SS, et CLK.

— MISO (Master in Slave Out) - La ligne Esclave pour l'envoi de données au maître.
— MOSI (Master Out Slave In) - La ligne Master pour l'envoi de données aux périphériques.
— SCK (Serial Clock) - Les impulsions d'horloge qui synchronisent la transmission des données générées par le maître.
— SS (Slave Select) – Le maître peut utiliser cette broche pour activer et désactiver des périphériques spécifiques.



Les données échangées sont des octets. La transmission s'effectue sur 2 fils monodirectionnels (nommés MOSI, MISO). Une horloge indépendante fixée par le maître synchronise les échanges (en général sur front). La fréquence de l'horloge de transmission est comprise entre 1 Mhz et 20 Mhz (selon les performances des circuits reliés au bus).
Il est possible de choisir le type d'horloge grâce à une combinaison de 2 bits :

— CPOL (Clock POLarity) qui détermine le niveau logique de la ligne SCK au repos.
— CPHA (Clock PHAse) qui détermine le front sur lequel la donnée est modifiée et le front sur lequel la donnée va être lue.

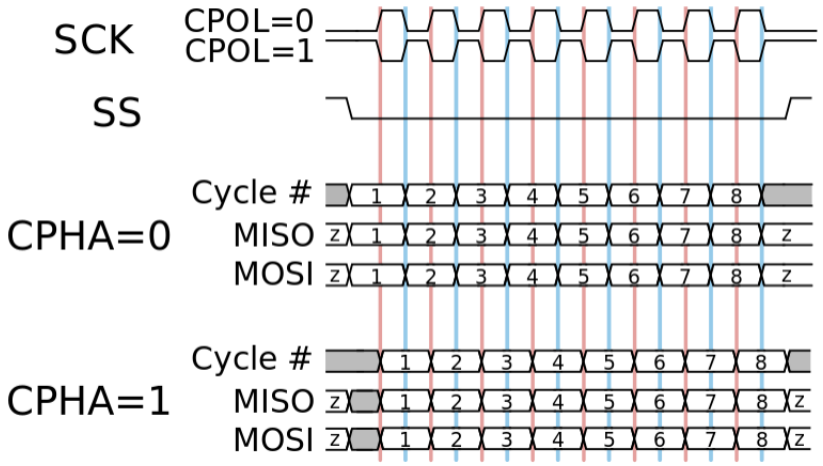
Nous avons ainsi 4 modes de fonctionnement :

— mode 0 : CPOL=0 et CPHA=0.

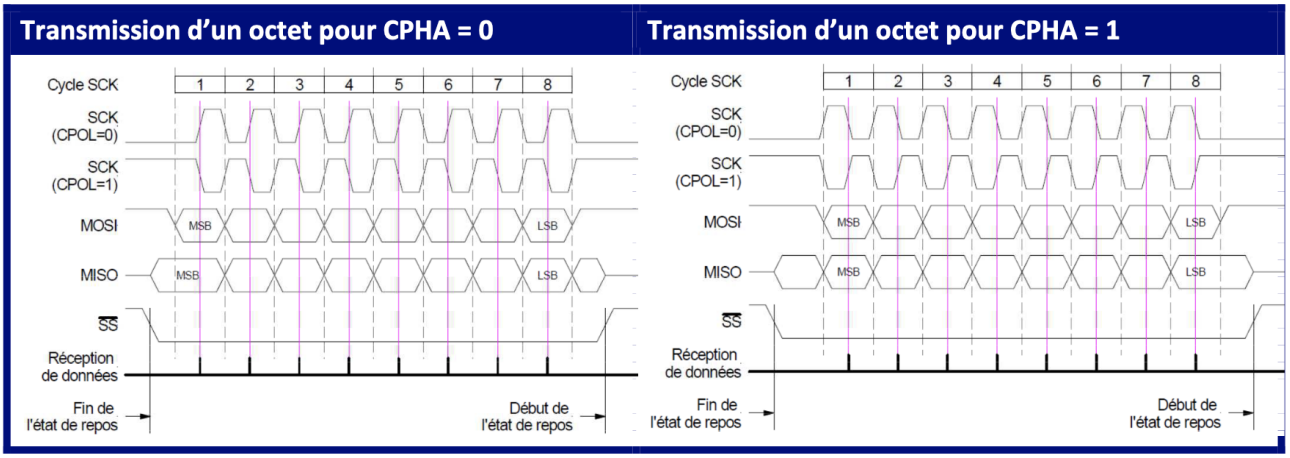
— mode 1 : CPOL=0 et CPHA=1.

— mode 2 : CPOL=1 et CPHA=0.

— mode 3 : CPOL=1 et CPHA=1.

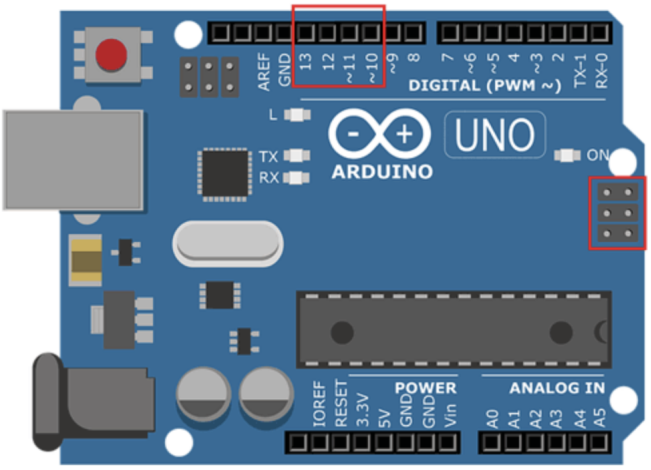


Il n'y a pas d'adressage des esclaves (comme sur un bus I2C par exemple). L'esclave devient actif au moyen d'une ligne de sélection de boîtier dédiée (généralement active à l'état bas). La ligne est constituée de 3 fils auxquels il faut ajouter les fils de sélection d'esclave.
Les communications sont initiées par le maître, qui commence par sélectionner l'esclave. Au cours de chaque cycle d'horloge, le maître envoie un bit à l'esclave, qui le lit sur la ligne MOSI.
En parallèle, l'esclave envoie un bit au maître, qui le lit sur la ligne MISO. Ces lectures et écritures simultanées entre le maître et l'esclave forment une communication Full-Duplex.



2.1 Programmer Arduino pour la communication SPI :

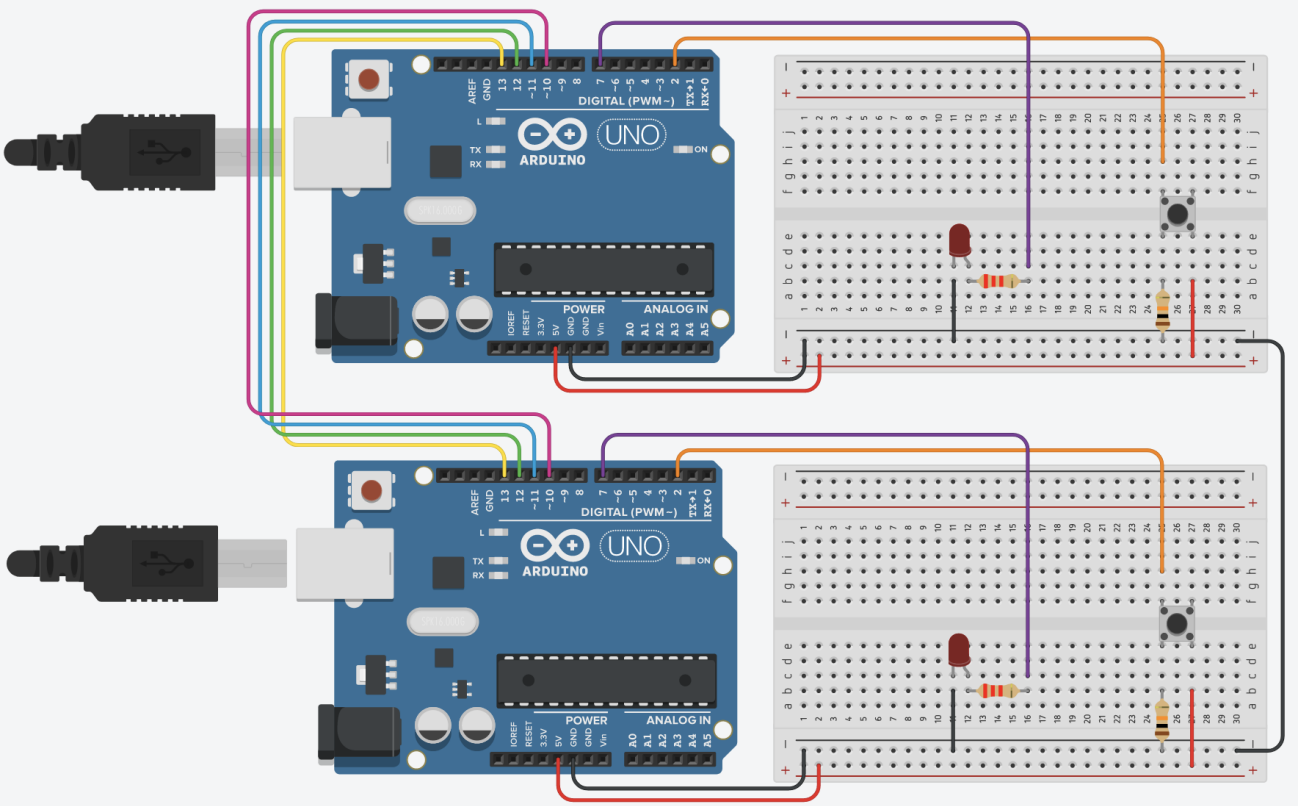
Broches numériques utilisées pour SPI		
Signal SPI	Carte Arduino Uno	Carte Arduino Mega
SS (choix de l'esclave)	10	S3
MOSI (entrées de données)	11 ou ICSP-4	S1
MISO (sorties de données)	12 ou ICSP-1	S0
SCLK (horloge)	13 ou ICSP-3	S2



On a besoin de deux programmes, l'un pour l'Arduino maître et l'autre pour l'Arduino esclave. Les programmes complets pour les deux côtés sont donnés à la fin.
Le montage et le programme réalisent la communication entre 2 Arduino ;

— Lorsque le bouton-poussoir du côté maître est enfoncé, la LED du côté esclave s'allume.

— Et lorsque le bouton-poussoir du côté esclave est enfoncé, la LED du côté maître s'allume.



Explication de la programmation principale Arduino SPI

Tout d'abord, nous devons inclure la bibliothèque SPI pour utiliser les fonctions de communication SPI. #include<SPI.h>

Dans void setup()

On commence la communication série au débit en bauds 115200. Serial.begin(115200);

Fixer la LED à la broche 7 et le bouton-poussoir à la broche 2 et régler ces broches OUTPUT et INPUT respectivement. pinMode(ipbutton,INPUT);

pinMode(LED,OUTPUT);

Ensuite, on commence la communication SPI SPI.begin();

On définit le diviseur d'horloge pour la communication SPI. Ici, on a défini le diviseur 8.

SPI.setClockDivider(SPI_CLOCK_DIV8);

Ensuite, on définit la broche SS HIGH car on n'a commencé aucun transfert vers l'Arduino esclave. digitalWrite(SS,HIGH);

Dans void loop() :

On lit l'état de la broche du bouton-poussoir connectée à la broche 2 (Arduino maître) pour envoyer ces valeurs à l'Arduino esclave. buttonvalue = digitalRead(ipbutton);

Définir la logique pour le réglage de la valeur x (à envoyer à l'esclave) en fonction de l'entrée de la broche 2

```
if(buttonvalue == HIGH)
 {
   x = 1;
 }
 else
 {
   x = 0;
 }
```

Avant d'envoyer la valeur, on doit abaisser la valeur de sélection de l'esclave pour commencer le transfert vers l'esclave depuis le maître. digitalWrite(SS, LOW);

Voici l'étape importante, dans la déclaration suivante, on envoie la valeur du bouton poussoir stockée dans la variable Mastersend à l'Arduino esclave et reçoit également la valeur de l'esclave qui sera stockée dans la variable Mastereceive. Mastereceive=SPI.transfer(Mastersend);
Après cela, en fonction de la valeur Mastereceive, nous allons allumer ou éteindre la LED Master Arduino.

```
if(Mastereceive == 1)
{
  digitalWrite(LED,HIGH);          //Sets pin 7 HIGH
  Serial.println("Master LED ON");
}
else
{
 digitalWrite(LED,LOW);           //Sets pin 7 LOW
 Serial.println("Master LED OFF");
}
```

Remarque : on utilise .println() pour afficher le résultat dans Serial Motor of Arduino IDE.

Explication de la programmation de l'esclave Arduino SPI

Tout d'abord, on doit inclure la bibliothèque SPI pour utiliser les fonctions de communication SPI.include¡SPI.h¿

Dans void setup()
On commence la communication série au débit en bauds 115200. Serial.begin(115200);
Fixer la LED à la broche 7 et le bouton-poussoir à la broche 2 et régler ces broches OUTPUT et INPUT respectivement.
pinMode(ipbutton,INPUT);
pinMode(LED,OUTPUT);
L'étape importante ici est les déclarations suivantes pinMode(MISO,OUTPUT);
L'instruction ci-dessus définit MISO comme OUTPUT (Avoir envoyer des données au Master IN). Les données sont donc envoyées via MISO de l'esclave Arduino.

Activer maintenant SPI en mode esclave en utilisant le registre de contrôle SPI SPCR l= _BV(SPE);
Activer ensuite l'interruption pour la communication SPI. Si des données sont reçues du maître, la routine d'interruption est appelée et la valeur reçue est extraite du SPDR (registre de données SPI) SPI.attachInterrupt();

La valeur du maître est extraite de SPDR et stockée dans la variable Slaves Received. Cela a lieu dans la fonction de routine d'interruption suivante.

```
ISR (SPI_STC_vect)
{
  Slavereceived = SPDR;
  received = true;
}
```

Ensuite, dans la void loop(), on définit la LED Arduino de l'esclave pour qu'elle s'allume ou s'éteigne en fonction de la valeur reçue de l'esclave.

```
  if (Slavereceived==1)
   {
   digitalWrite(LEDpin,HIGH); //Sets pin 7 as HIGH LED ON
                                    Serial.println("Slave LED ON");
     }
else
      {
  digitalWrite(LEDpin,LOW);      //Sets pin 7 as LOW LED OFF
  Serial.println("Slave LED OFF");
      }
```

Ensuite, on lit l'état du bouton poussoir esclave Arduino et stocke la valeur dans l'envoi esclave pour envoyer la valeur au maître Arduino en donnant la valeur au registre SPDR.

```
 buttonvalue = digitalRead(buttonpin);
  if (buttonvalue == HIGH)
      {
        x=1;
      }
else
      {
       x=0;
      }
            Slavesend=x;
            SPDR = Slavesend;
```

Remarque : on utilise serial.println() pour afficher le résultat dans Serial Motor of Arduino IDE.

Questions :

Certains oscilloscopes peuvent réaliser l'analyse de trame (analyseur logique). Vérifier sur internet si l'oscilloscope utilisé possède cette fonctionnalité (en mentionnant la marque et la référence).

1. Visualiser les différents signaux SPI à l'oscilloscope.
2. Afficher les trames.
3. Commenter les résultats.

Code Arduino Master :

Arduino Master BTS SN : KB

```
1
2  //SPI MASTER (ARDUINO)
3  //COMMUNICATION SPI ENTRE 2 ARDUINO
4
5  #include<SPI.h>                              //Librairie SPI
6  #define LED 7
7  #define ipbutton 2
8  int buttonvalue;
9  int x;
10 void setup (void)
11
12 {
13   Serial.begin(115200);                      //Baud Rate 115200
14
15   pinMode(ipbutton,INPUT);                   // 2 input
16   pinMode(LED,OUTPUT);                       // 7 Output
17
18   SPI.begin();                               //Début de la communication SPI
19   SPI.setClockDivider(SPI_CLOCK_DIV8);       //Réglage de l'horloge de communication
20                                              //SPI à 8 (16/8=2Mhz)
21   digitalWrite(SS,HIGH);                     // Réglage SlaveSelect à l'état HIGH
22                                  //(Sinon le maître ne se connecte pas avec l'esclave )
23 }
24
25 void loop(void)
26 {
27   byte Mastersend,Mastereceive;
28
29   buttonvalue = digitalRead(ipbutton);    //Lire l'état de la pin 2
30
31   if(buttonvalue == HIGH)                 //Logique pour le réglage de la valeur x
32                                           //(à envoyer à l'esclave) en fonction de
33                                           //l'entrée de la pin 2
34   {
35     x = 1;
36   }
37   else
38   {
39     x = 0;
40   }
41
42   digitalWrite(SS, LOW);                   //Démarrer la communication avec
43                                            //l'esclave connecté au maître
44
45   Mastersend = x;
46   Mastereceive=SPI.transfer(Mastersend); //Envoyer la valeur mastersend à l'esclave reçoit
47                                          //également la valeur de l'esclave
48
49   if(Mastereceive == 1)                   //Logique de réglage de la sortie LED en
50                                           //fonction de la valeur reçue de l'esclave
51   {
52     digitalWrite(LED,HIGH);               // pin 7 HIGH
53     Serial.println("Master LED ON");
54   }
55   else
56   {
57    digitalWrite(LED,LOW);                // pin 7 LOW
58    Serial.println("Master LED OFF");
59   }
60   delay(1000);
61 }
```

Code Arduino Slave :

Arduino Slave BTS SN : KB

```
1
2  //SPI SLAVE (ARDUINO)
3  //COMMUNICATION SPI ENTRE 2 ARDUINO
4
5  #include<SPI.h>
6  #define LEDpin 7
7  #define buttonpin 2
8  volatile boolean received;
9  volatile byte Slavereceived,Slavesend;
10 int buttonvalue;
11 int x;
12 void setup()
13
14 {
15   Serial.begin(115200);
16
17   pinMode(buttonpin,INPUT);              // pin 2 : INPUT
18   pinMode(LEDpin,OUTPUT);                // pin 7 : OUTPUT
19   pinMode(MISO,OUTPUT);                  //Définit MISO comme OUTPUT (il faut envoyer
20                                          //des données à Master IN
21
22   SPCR |= _BV(SPE);                      //Activer SPI en mode esclave
23   received = false;
24
25   SPI.attachInterrupt();                 //L'interruption ON est définie pour
26                                          //la communication SPI
27
28 }
29
30 ISR (SPI_STC_vect)                       //Fonction de routine d'interruption
31 {
32   Slavereceived = SPDR;                  // Valeur reçue du maître si stockée
33                                          //dans la variable  slavereceived
34   received = true;
35 }
36
37 void loop()
38 { if(received)                           //Logique pour définir LED ON ou OFF en fonction
39                                          //de la valeur reçue du maître
40    {
41       if (Slavereceived==1)
42       {
43         digitalWrite(LEDpin,HIGH);        // pin 7 : HIGH LED ON
44         Serial.println("Slave LED ON");
45       }else
46       {
47         digitalWrite(LEDpin,LOW);         //pin 7 : LOW LED OFF
48         Serial.println("Slave LED OFF");
49       }
50
51       buttonvalue = digitalRead(buttonpin);  // Lit l'état de la pin 2
52
53       if (buttonvalue == HIGH)               //Logique pour définir la valeur de x à
54                                              //envoyer au maître
55       {
56         x=1;
57
58       }else
59       {
60         x=0;
61       }
62
63   Slavesend=x;
64   SPDR = Slavesend;                          //Envoie la valeur x au maître via SPDR
65   delay(1000);
66 }
67 }
```