Systèmes embarqués automobiles

Master D'Université Spécialisé en
Ingénierie des Systèmes Embarqués

Chapitre 3: Le Bus I2C

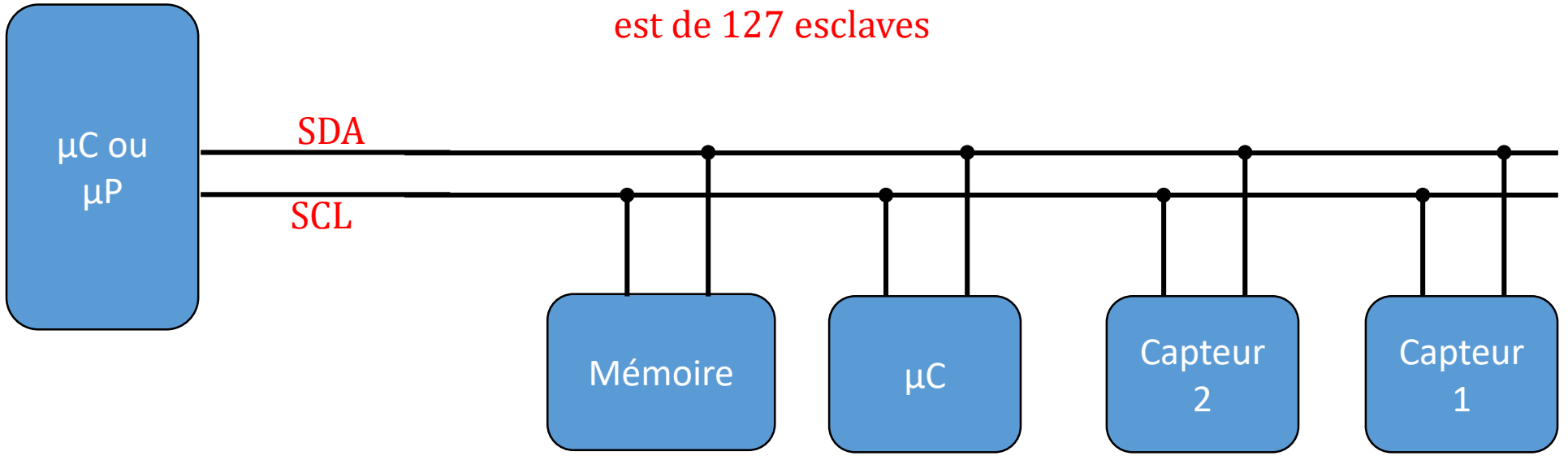
Année universitaire 2025/2026

Dr. Ing. LASSIOUI Abdellah

Chapitre 3: Le Bus I2C

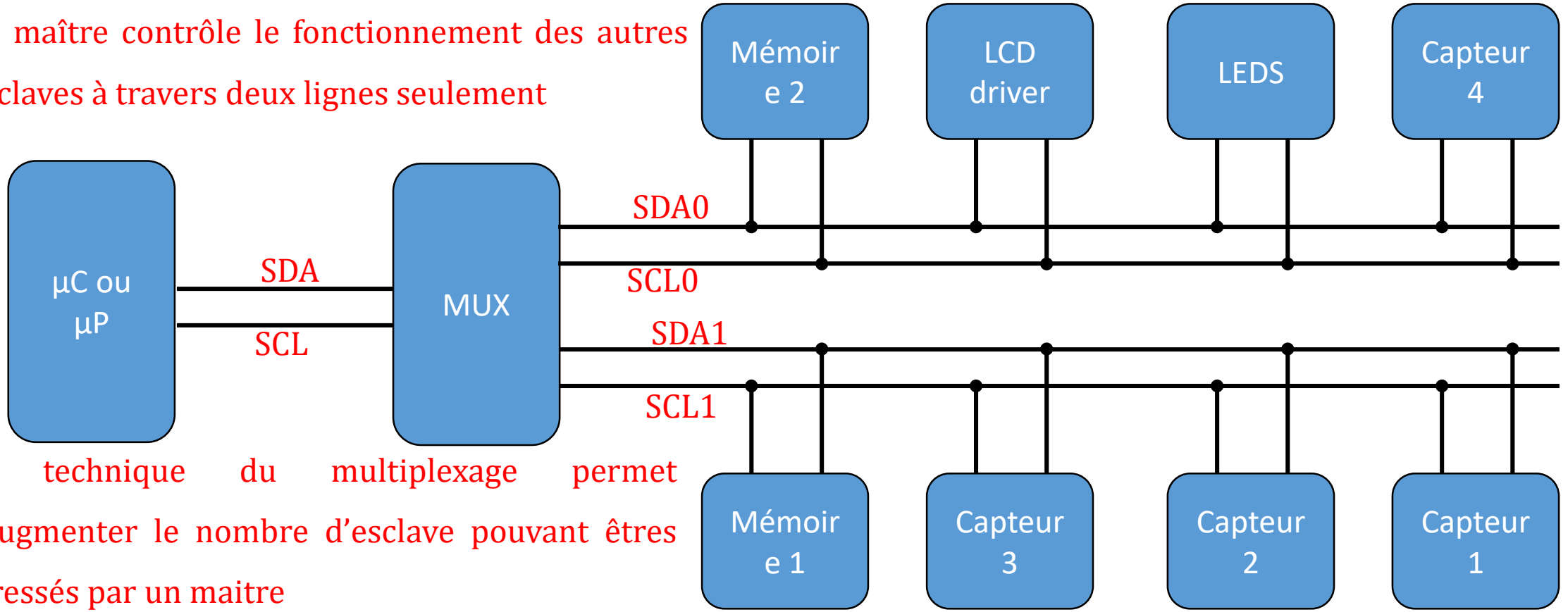
Introduction

- Le bus de données I2C (Inter Integrated Circuit) est un bus très largement utilisé pour la communication entre un master (ou plusieurs masters) et un ou plusieurs esclaves.
- La figure suivante illustre un exemple de connexion entre un nœud de type Master (maître) et plusieurs esclaves
- Le maître contrôle le fonctionnement des autres esclaves à travers deux lignes seulement
- Le nombre maximal d'esclave pouvant être atteint est de 127 esclaves



Introduction

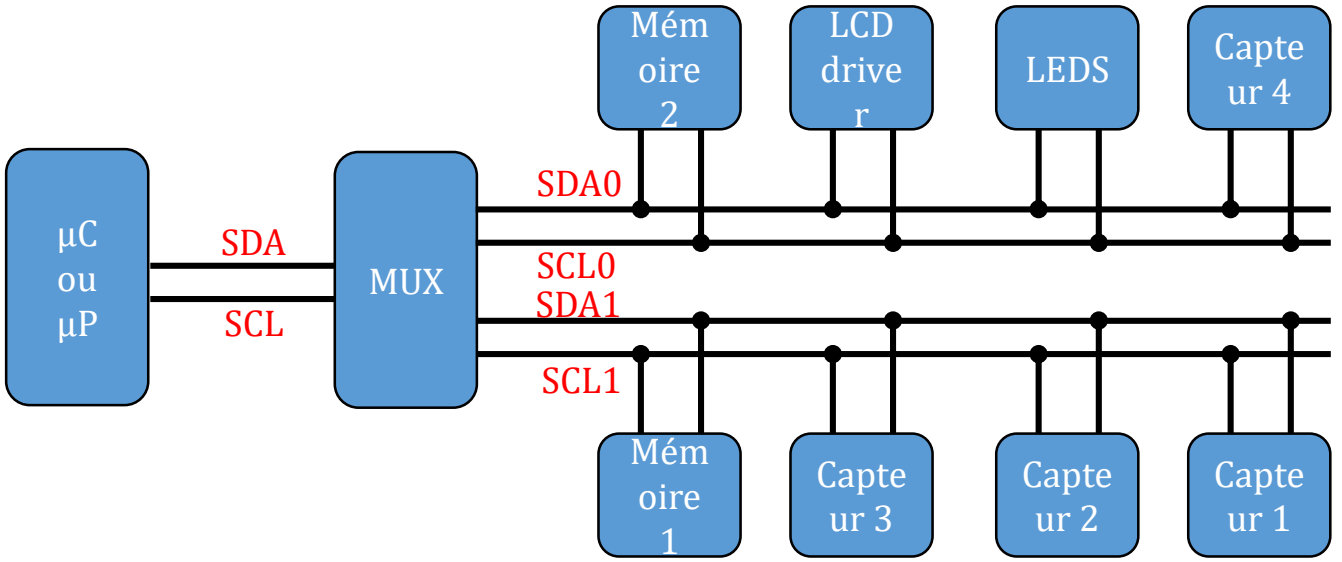
- Le bus de données I2C (Inter Integrated Circuit) est un bus très largement utilisé pour la communication entre un master (ou plusieurs masters) et un ou plusieurs esclaves.
- La figure suivante illustre un exemple de connexion entre un nœud de type Master (maître) et plusieurs esclaves
- Le maître contrôle le fonctionnement des autres esclaves à travers deux lignes seulement



- La technique du multiplexage permet d'augmenter le nombre d'esclave pouvant êtres adressés par un maitre

Les lignes SDA et SCL

- Le bus utilise deux lignes pour la communication entre le maître et les esclaves: SDA et SCL
- Le maître peut accéder à n'importe quel esclave avec une seule adresse envoyée à travers SDA



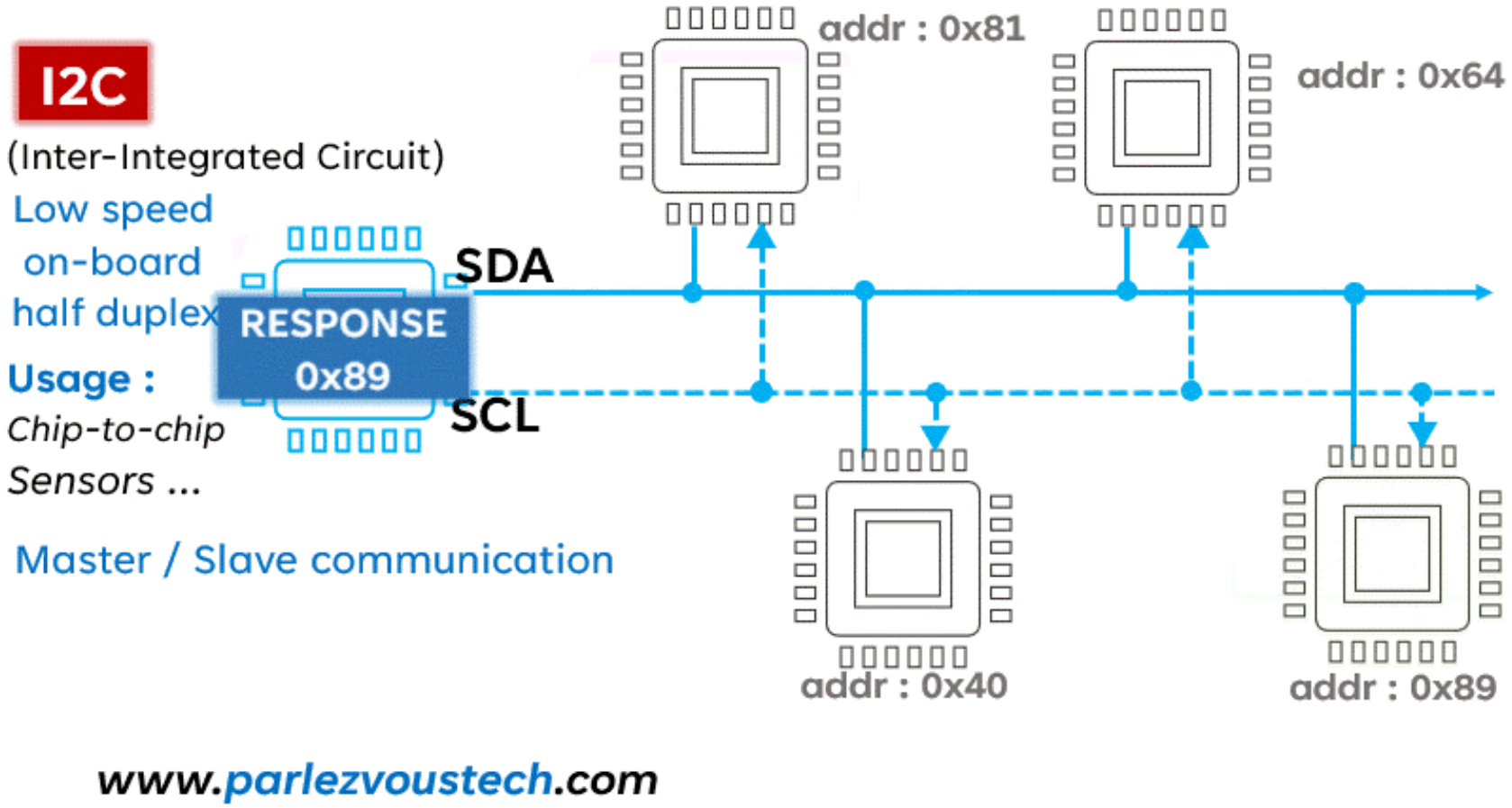
SDA: Serial Data Line : **SDA (Serial Data Line)** : C'est la ligne de données qui transporte les bits d'information entre les maîtres et les esclaves sur le bus I2C.

Les données sont transmises en série, un bit à la fois. SDA est une ligne bidirectionnelle, ce qui signifie qu'elle peut être utilisée par le maître pour envoyer des commandes aux esclaves, et par les esclaves pour renvoyer des données au maître.

SCL : Serial Clock Line **SCL (Serial Clock Line)** : C'est la ligne d'horloge utilisée pour synchroniser le transfert des données sur SDA. Le dispositif maître génère l'horloge, et chaque impulsion sur SCL permet de transférer un bit de données sur SDA. Ainsi, SCL garantit que les dispositifs connectés restent synchronisés pendant l'échange de données.

Les lignes SDA et SCL

- Le bus utilise deux lignes pour la communication entre le maître et les esclaves: SDA et SCL
- Le maître peut accéder à n'importe quel esclave avec une seule adresse envoyée à travers SDA



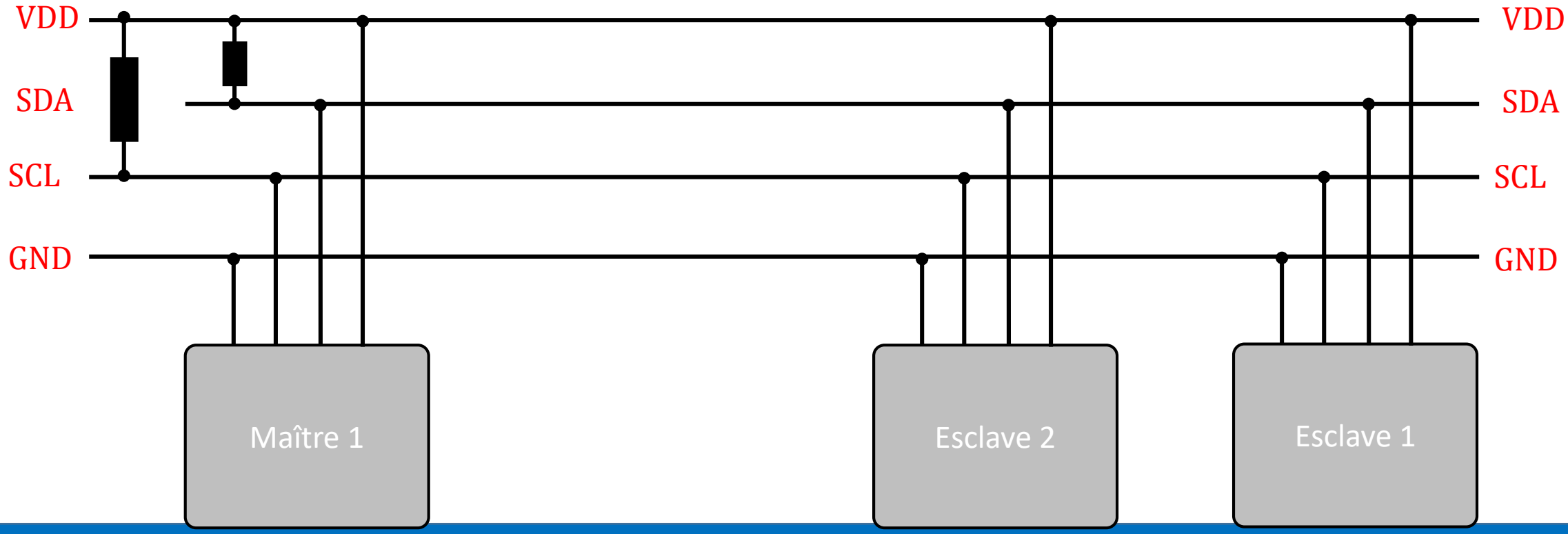
Les vitesses de communications

- La vitesse de communication est choisie en fonction des données à transporter, des performances des nœuds maîtres et des nœuds esclaves
- La charge capacitive du bus joue également un rôle essentiel dans le choix de la vitesse de transferts de données

Mode I2C	Vitesse maximale	Description
Standard Mode	100 kbit/s	Mode de base pour les applications ne nécessitant pas de grande vitesse. (capteurs de température, convertisseurs ADC/DAC…)
Fast Mode	400 kbit/s	Utilisé pour des applications nécessitant une vitesse plus élevée que le mode standard (afficheurs LCD…)
Fast Mode Plus (Fm+)	1 Mbit/s	Introduit pour répondre aux besoins de vitesse plus élevés dans certaines applications.
High Speed Mode (Hs)	3.4 Mbit/s	Mode à haute vitesse, avec des exigences de conception plus strictes pour les circuits (communication avec les DSP, FPGA…).
Ultra-Fast Mode (UFm)	5 Mbit/s	**Mode unidirectionnel (Write Only)** conçu pour des applications nécessitant des transferts très rapides (LIDAR…).

La couche physique du bus I2C

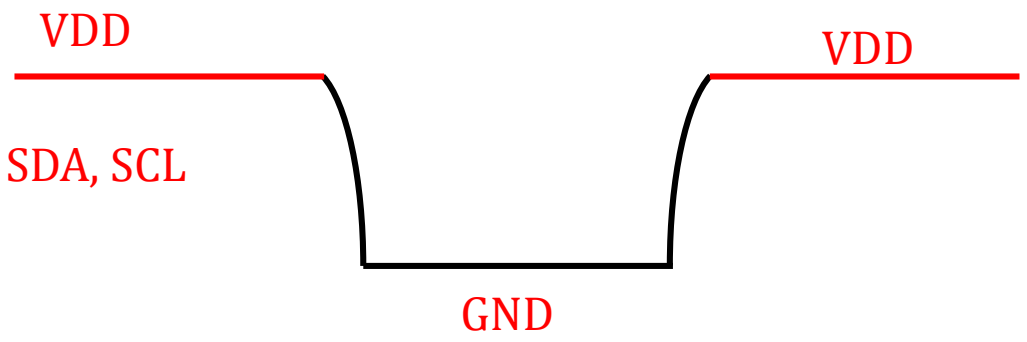
- Le bus I2C repose sur deux lignes pour assurer une communication bidirectionnelle de type HALF-DUPLEX
- Un seul nœud envoi les données à un instant donnée (contrairement au FULL DUPLEX ou un nœud peut envoyer et recevoir les données à la fois comme le SPI par exemple)
- Les résistances de pull-up sont nécessaires au fonctionnement du bus. Ses résistances sont utilisées entre les lignes SDA, SCL et VDD

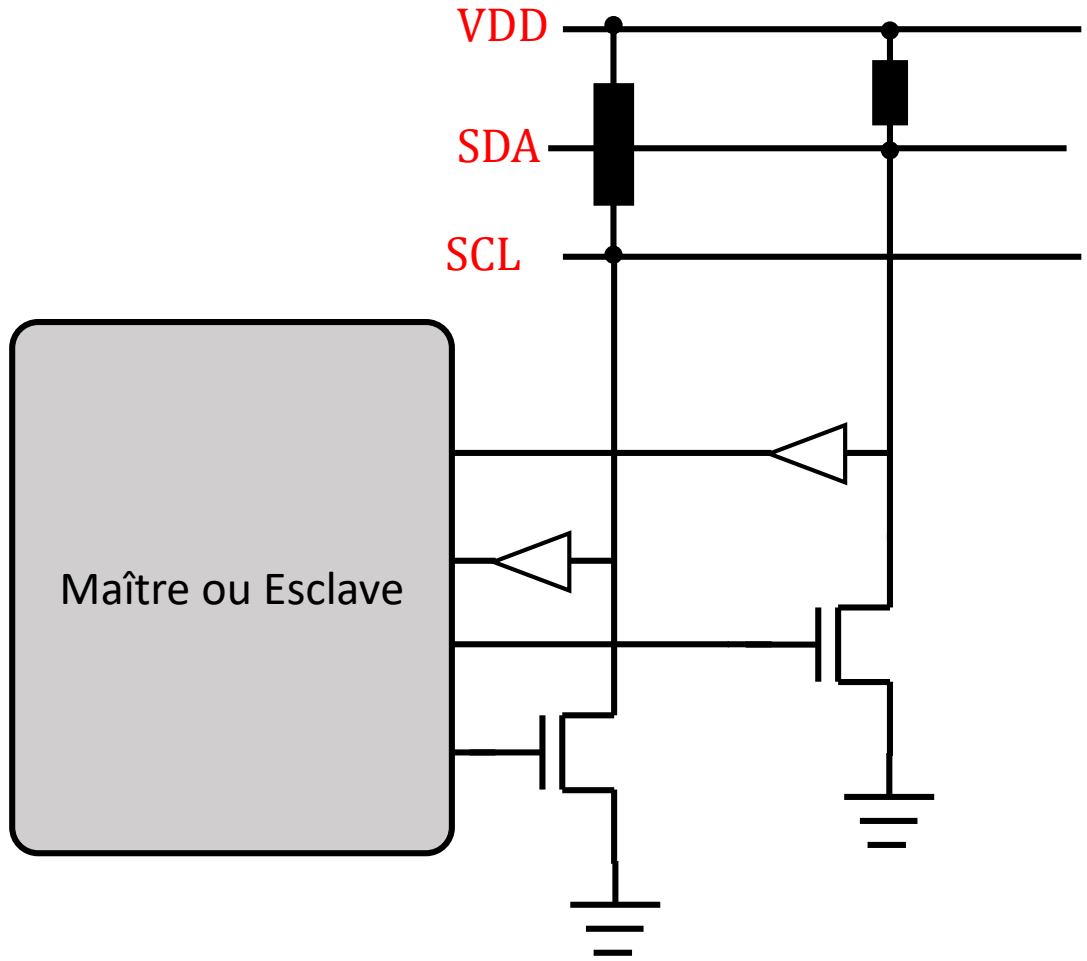


La couche physique du bus I2C

- La figure suivante illustre le principe de la connexion entre les nœuds (Maîtres ou esclaves) et le bus I2C
- Les résistances de pull-up sont utilisées entre les lignes SDA et SCL et l'alimentation VDD

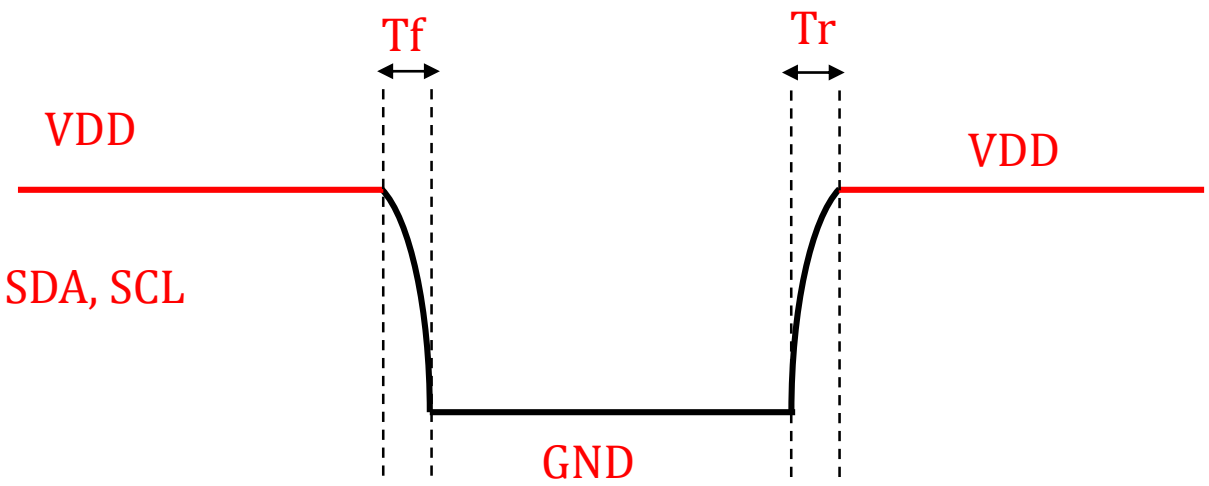
- Cette structure est appelée drain-ouvert (Open-Drain)
  - Elle permet de forcer la ligne (SDA ou SCL) à la masse si le transistor NMOS est ON
  - Ou bien mettre la ligne (SDA ou SCL) à VDD si le NMOS est à l'état bloquée.





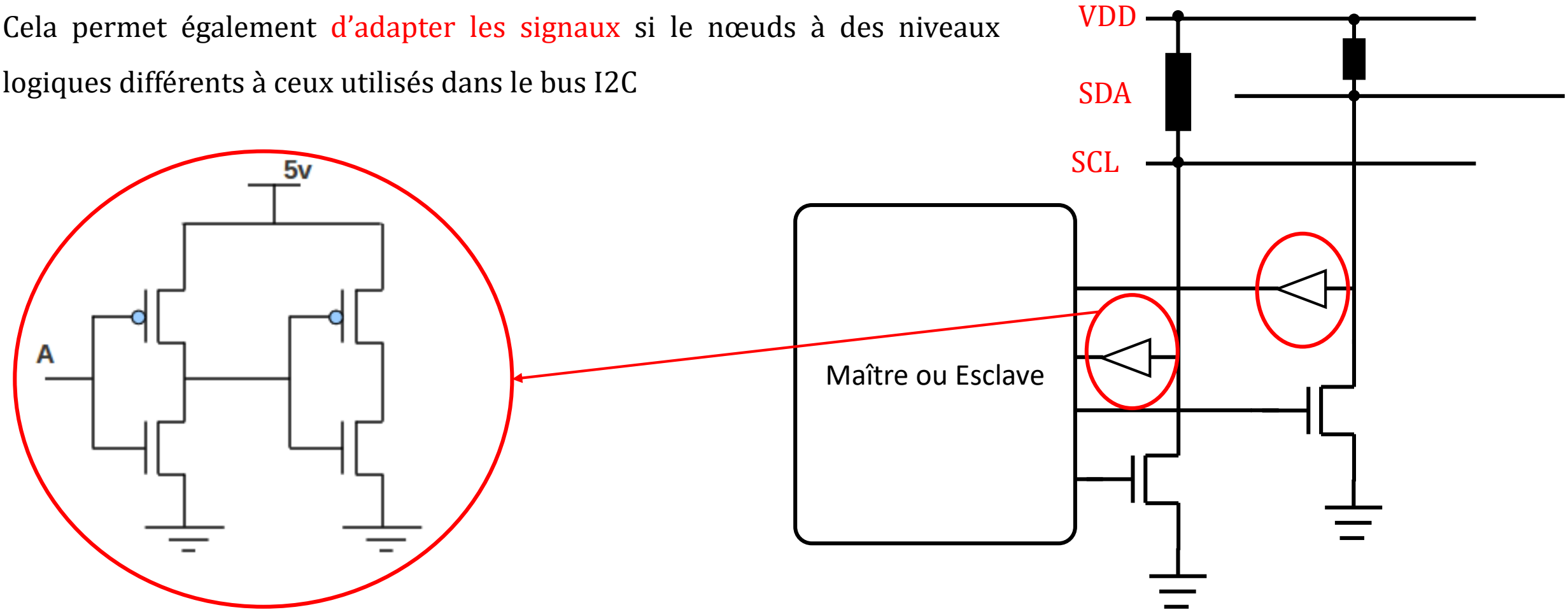
La couche physique du bus I2C

- Les facteurs influençant les temps de montés et de la descentes sont:
- La charge capacitive du bus I2C: la longueur des fils ainsi que les nœuds connectés au bus peuvent influencer la charge capacitive du bus
- Les valeurs des résistances de pull-up: des valeurs plus faibles de résistances engendres des temps de montés et de descentes rapide (décharge rapide de la capacité du bus) mais nécessitent des puissances élevées
- Les performances du transistor NMOS: les NMOS avec des temps de commutations courts sont utilisés pour permettre des vitesses de transmissions élevées



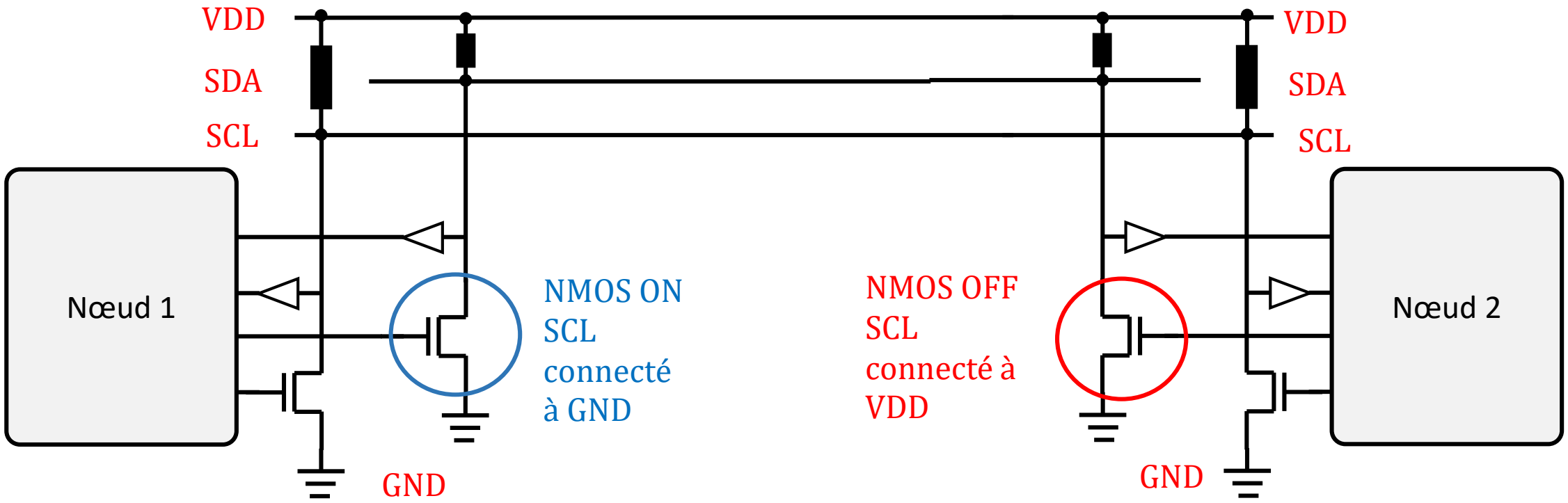
La couche physique du bus I2C

- Pour amplifier les signaux reçus sur le bus I2C, des buffers sont utilisés
- Il s'agit de deux inverseurs CMOS montés en série
- Cela permet également d'adapter les signaux si le nœuds à des niveaux logiques différents à ceux utilisés dans le bus I2C



La couche physique du bus I2C

- Cette structure permet de protéger le bus I2C contre un état destructive (court-circuit entre VDD et GND (Push-Pull))
- Dans ce cas, nous avons le NMOS du nœud 1 qui est ON alors et le NMOS du nœud 2 OFF alors la ligne SCL est mise au niveau logique 0.
- Elle s'agit d'une connexion de type AND: la sortie est le AND de toutes les entrées



Le protocole I2C: état du bus au repos

Lorsque le bus **I2C** est **au repos** (c’est-à-dire qu'aucune communication n'est en cours), il présente les caractéristiques suivantes :

- **Ligne SDA (Serial Data) = Niveau Haut (1 logique, tiré vers VCC via une résistance de pull-up).**
- **Ligne SCL (Serial Clock) = Niveau Haut (1 logique, également tiré vers VCC via une résistance de pull-up).**

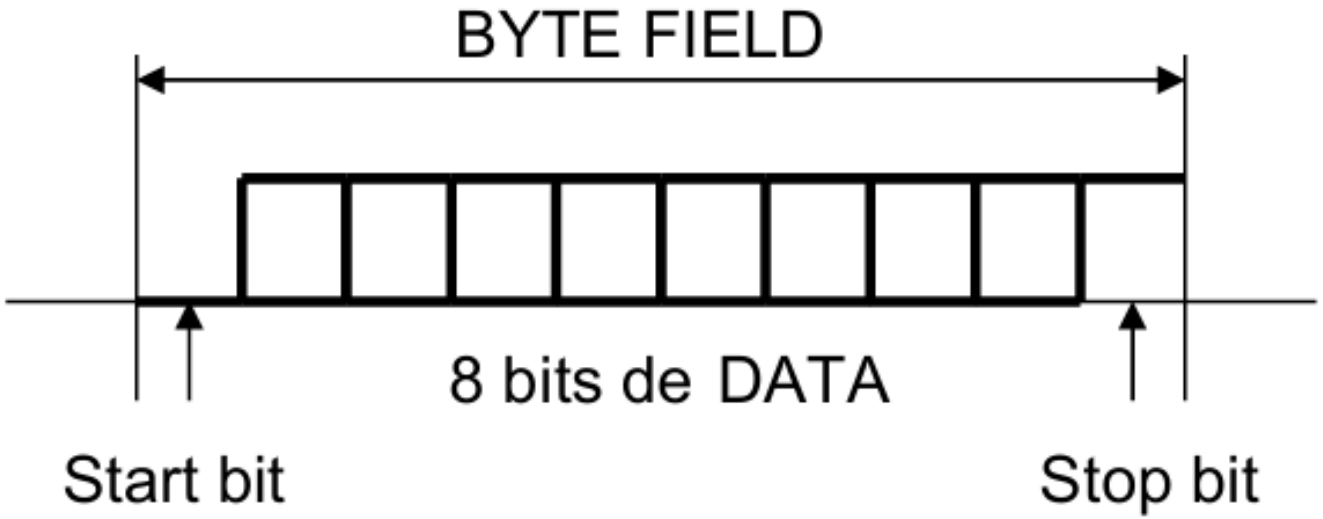
Le protocole I2C: les bits START and STOP

- Les bits **START** et **STOP** jouent un rôle crucial pour gérer le début et la fin d'une communication entre les dispositifs maîtres et esclaves.

*Le protocole **I2C** utilise deux conditions spéciales pour gérer la communication :*

- ***Start Condition (bit de départ)***
- ***Stop Condition (bit d'arrêt)***

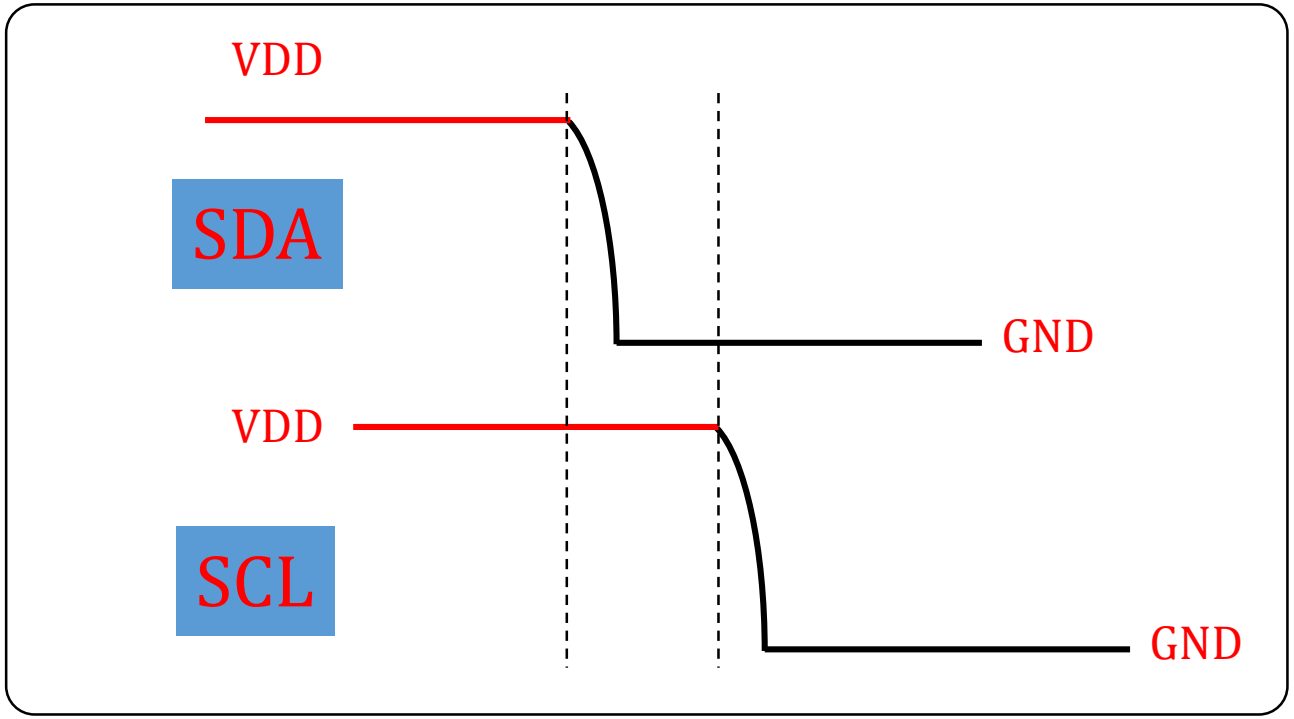
*Ces bits sont générés par le **maître** pour indiquer le début et la fin d'une transmission.*



Le protocole I2C: les bits START and STOP

1. Bit START :

- Le bit START est utilisé pour **initier une communication** sur le bus I2C.
- Lorsqu'un dispositif maître veut communiquer avec un ou plusieurs esclaves, il commence par envoyer une condition de départ (START condition).
- Ce bit est généré par une transition de **niveau haut à bas sur la ligne SDA pendant que SCL (Serial Clock Line) reste haut**.

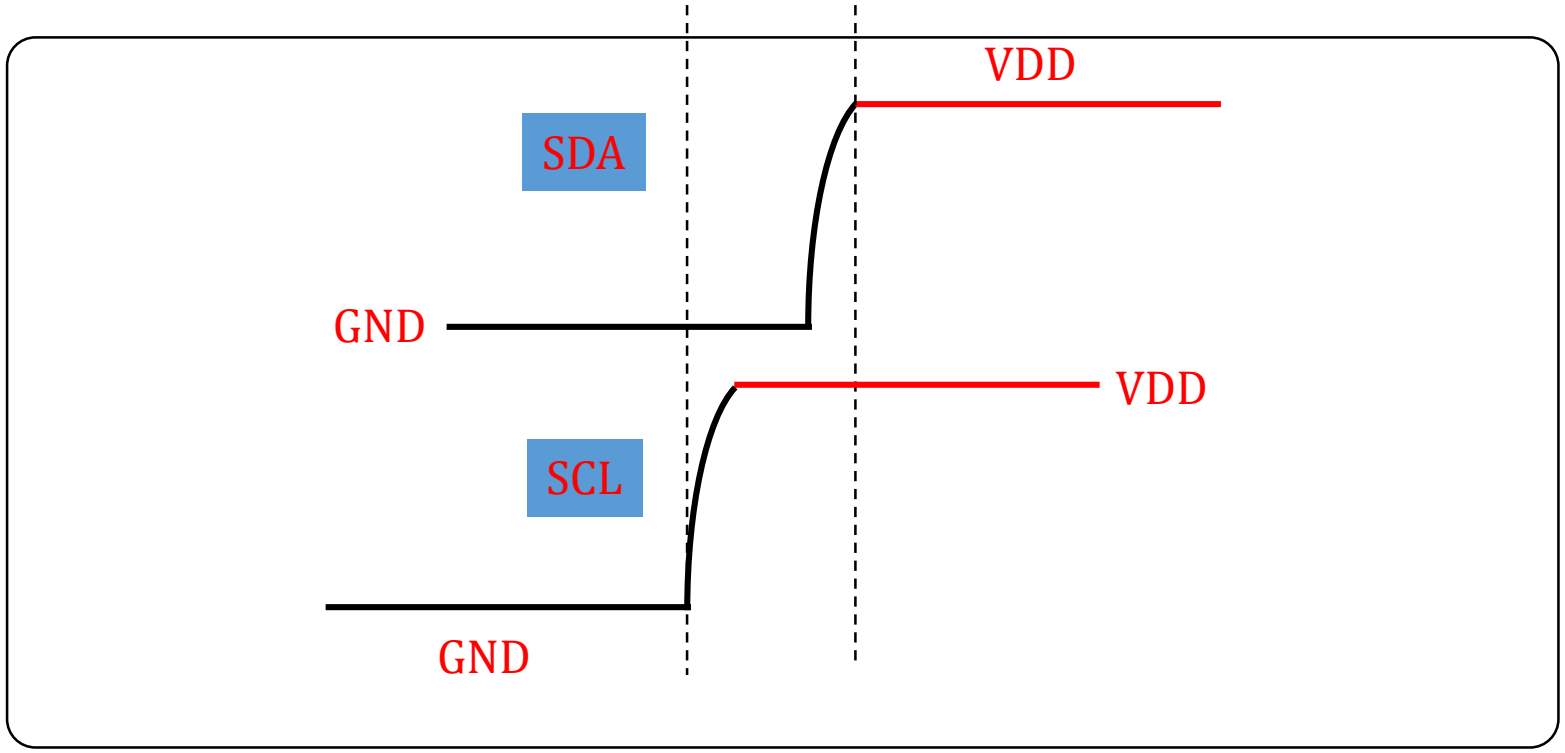


- ***Cette transition est utilisée pour indiquer aux nœuds esclaves que le maître veut initier une communication***
- ***Tous les esclaves mettent fin à leurs communications***

Le protocole I2C: les bits START and STOP

2. Bit STOP

- Le bit STOP indique la fin de la communication.
- Ce bit est généré par une transition de niveau bas à haut sur la ligne SDA, pendant que SCL reste haut.
- Après la condition STOP, le bus I2C est libre, et un autre maître (s'il y en a plusieurs) peut l'utiliser pour démarrer une nouvelle communication.

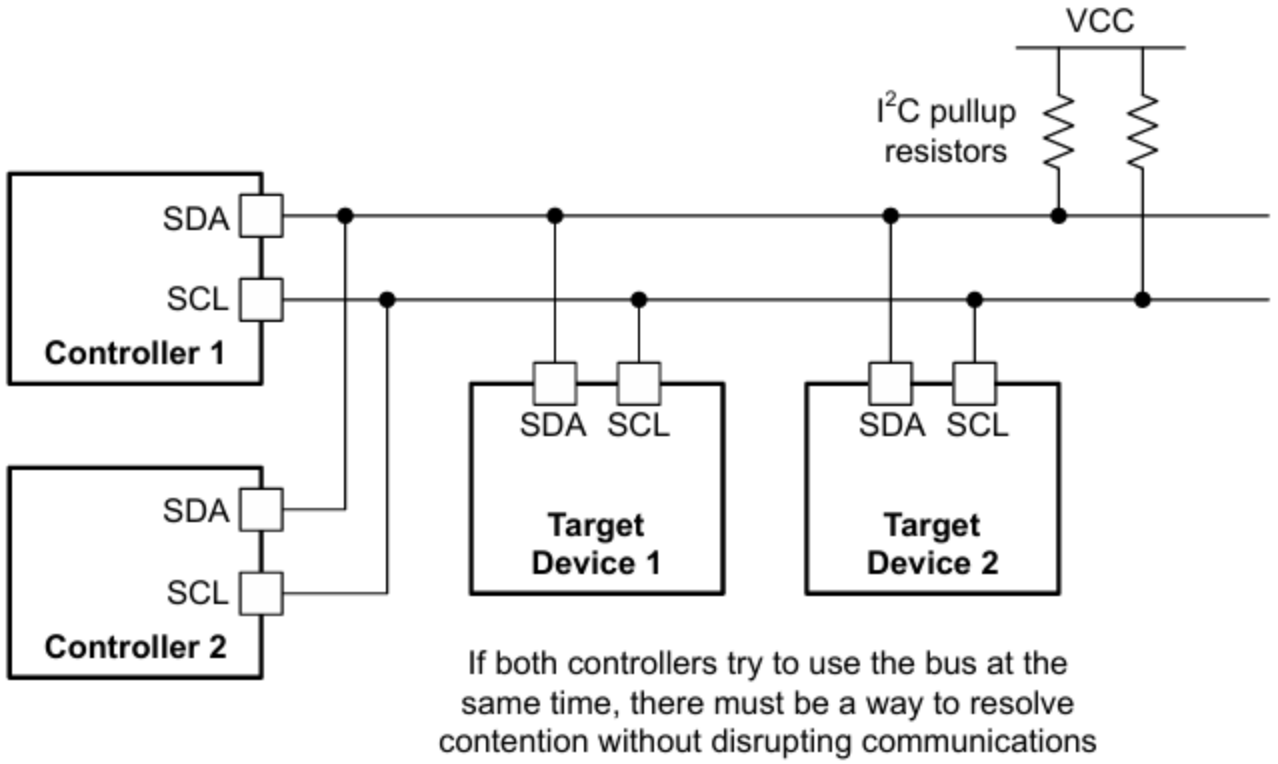


- ***Cette transition est utilisée pour indiquer aux nœuds esclaves que le maître a terminé l'envoie de la donnée***
- ***Un autre maître peut alors accéder au bus pour envoyer une requête ou un autre esclave peut envoyer une réponse à un maître***

Le protocole I2C: le protocole d'arbitrage

- Problématique :

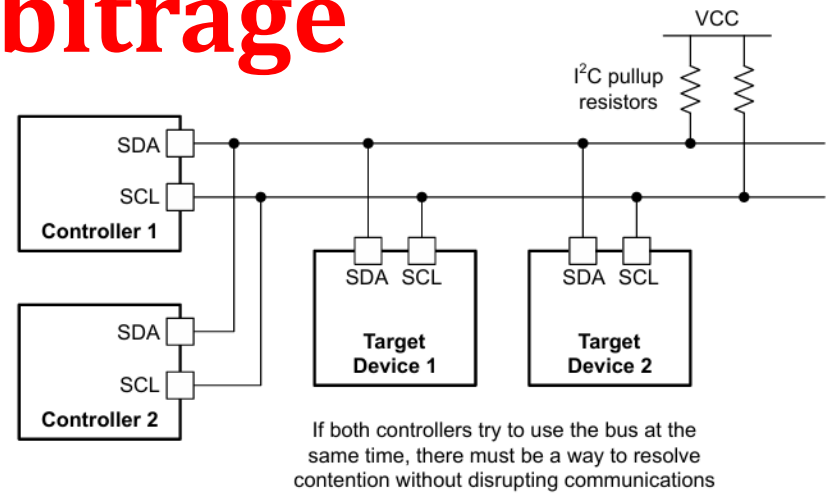
Supposons que nous avons plusieurs contrôleurs du bus I2C. Dans ce cas, il se peut que deux ou plusieurs contrôleurs cherchent à accéder au bus en même temps. Donc il faut une technique d'arbitrage permettant de donner l'accès à un seul contrôleur.



Source: https://www.ti.com/lit/an/sbaa565/sbaa565.pdf

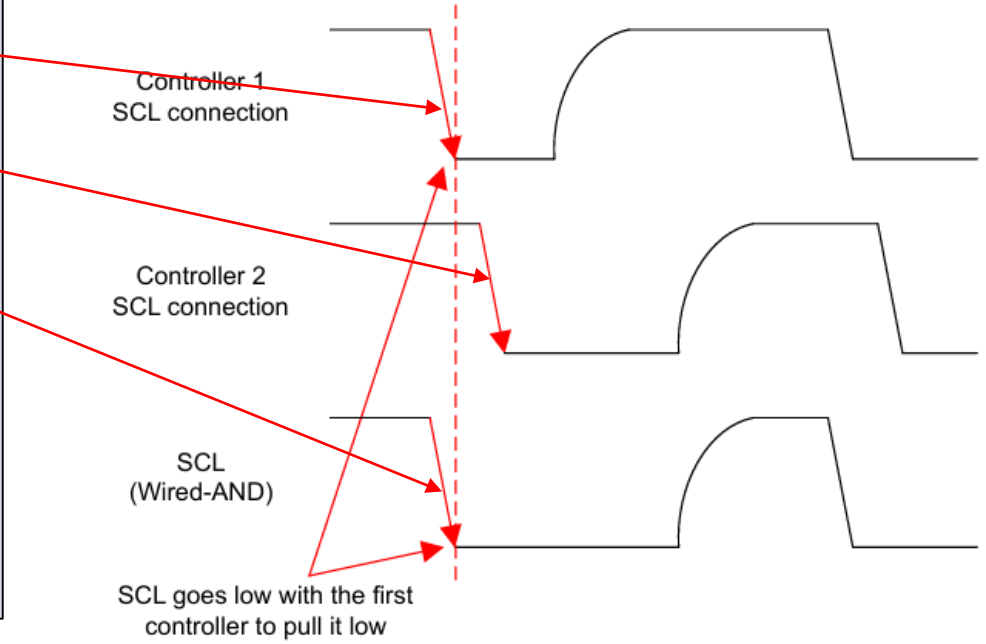
Le protocole I2C: le protocole d'arbitrage

- Solution:
- *Le bus I2C utilise une technique d'arbitrage se basant premièrement sur la synchronisation du signal d'horloge SCL*



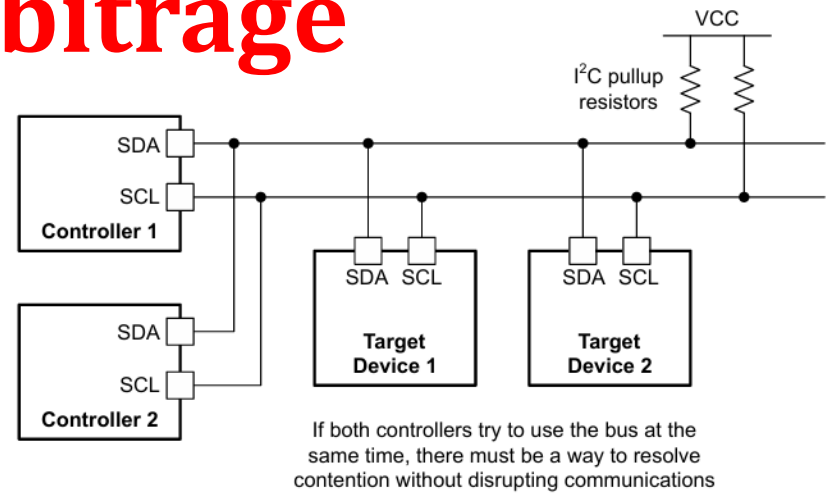
Source: https://www.ti.com/lit/an/sbaa565/sbaa565.pdf

- *Dans la figure ci-contre, le Contrôleur N1 force le signal SCL à zéro*
- *Après un certains temps le contrôleur n2 force aussi ce signal à zéro.*
- *Le signal SCL passe automatiquement à zéro dès que le premier contrôleur le force par ce que il s'agit d'une ligne appelée AND-wire.*
- *De-même, le contrôleur N1 cherche à forcer le signal CLK à nouveau vers 1 mais le contrôleur N2 reste toujours à 0 => Résultat : SCL = 0 jusqu'à ce que le contrôleur N2 change d'état.*

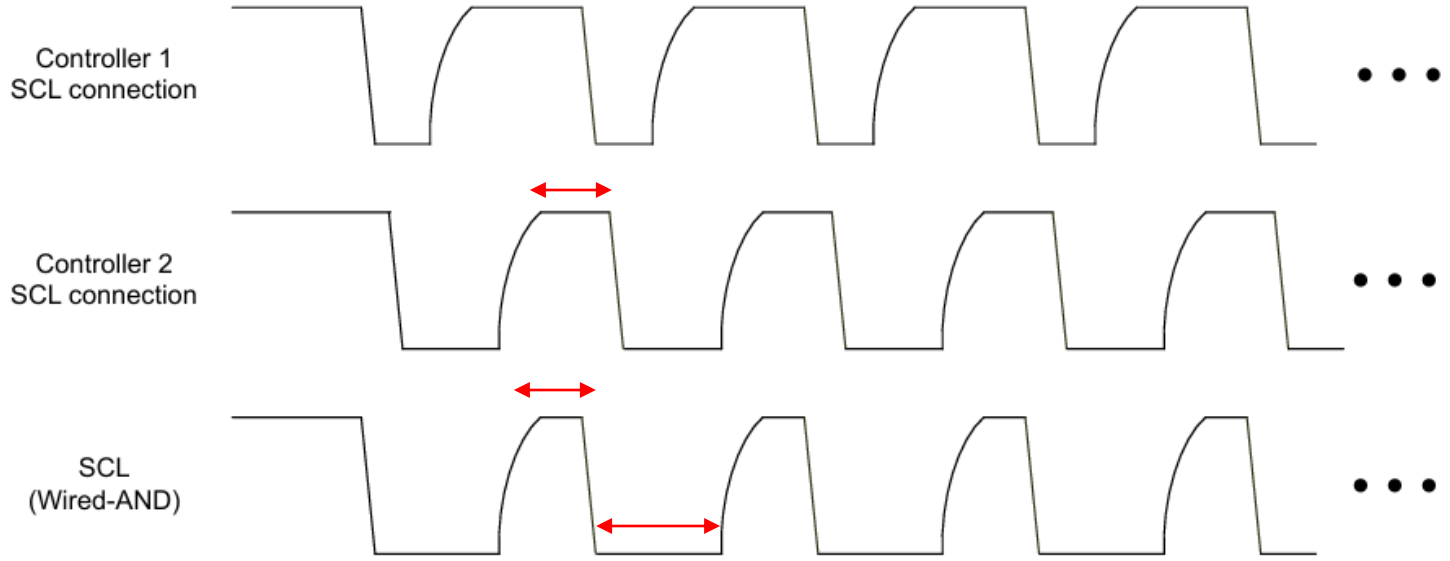


Le protocole I2C: le protocole d'arbitrage

- Le processus de synchronisation continue comme indiqué dans la figure suivante.
  - *La durée T-High du signal SCL est fixée par le contrôleur ayant la plus faible T-High (dans ce cas Contrôleur N2)*
  - *La durée T-Low du SCL est fixée par le contrôleur ayant la plus grand T-Low (dans ce cas le contrôleur n1)*



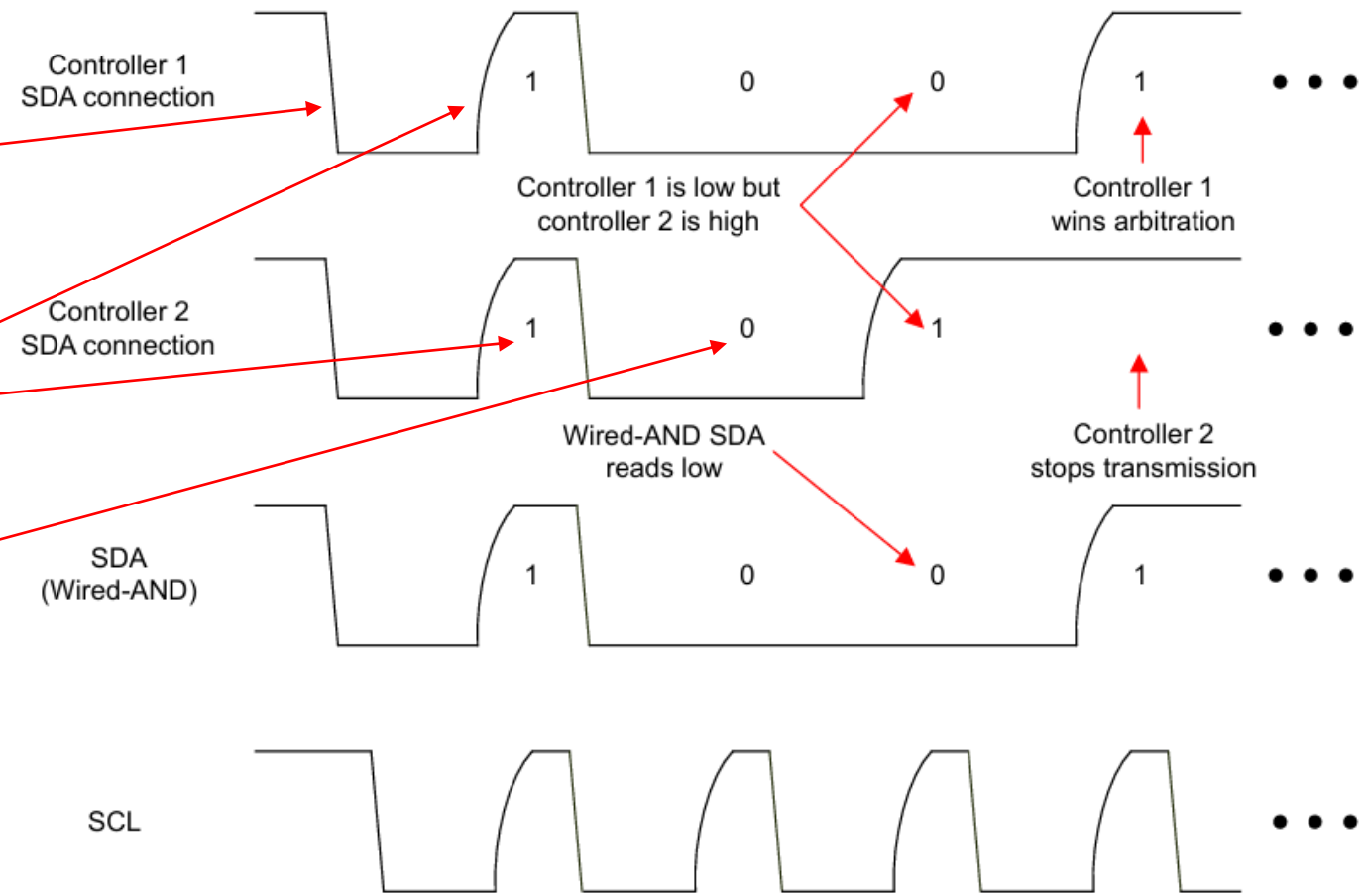
Source: https://www.ti.com/lit/an/sbaa565/sbaa565.pdf



- *Le résultat est la fonction AND appliquée sur les deux SCL*

Le protocole I2C: le protocole d'arbitrage

- Une fois la synchronisation a été effectuée, les deux contrôleurs passent le signal SDA vers 0 quand SCL = 1 pour indiquer le bit de START
  - *Dans la figure suivante, les deux contrôleurs commencent par envoyer le premier Bit =1 sur SDA (aucun conflit)*
  - *Les deux contrôleurs envoient aussi le deuxième Bit = 0 sans conflit*
  - *Le premier contrôleur envoie le troisième Bit = 0, mais le deuxième envoie Bit = 1. dans ce cas, le contrôleur 1 gagne l'accès au bus.*



Source: https://www.ti.com/lit/an/sbaa565/sbaa565.pdf

Le protocole I2C: la séquence de transmission

Séquence d'une Transmission I2C :

1. Le maître envoie une condition START pour initier la communication.
2. Il envoie l'adresse de l'esclave et attend un accusé de réception (ACK) de l'esclave.
3. Le maître envoie les données ou reçoit les données de l'esclave.
4. Une fois la transmission terminée, le maître envoie une condition STOP pour libérer le bus.

[Start] → [Adresse de l'Esclave + R/W] → [ACK] → [Données] → [ACK] → … → [Stop]

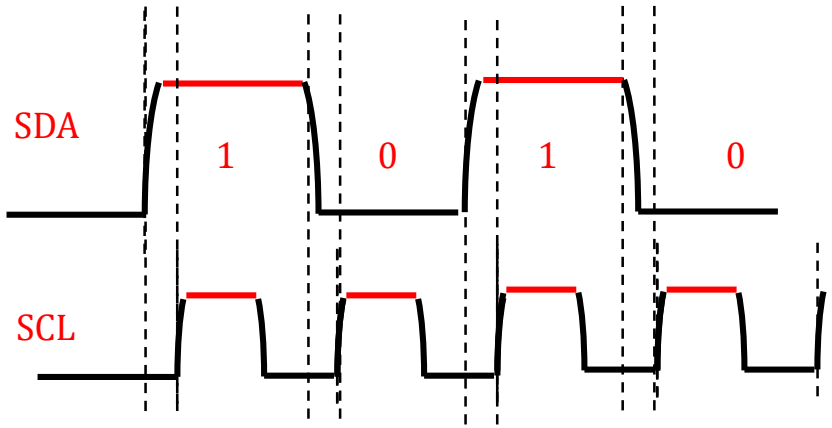
Ces conditions sont essentielles pour synchroniser les dispositifs connectés au bus I2C et pour garantir que la communication s'effectue correctement, sans interférence.

Le protocole I2C: l'envoie des données sur SDA en fonction du signal SCL

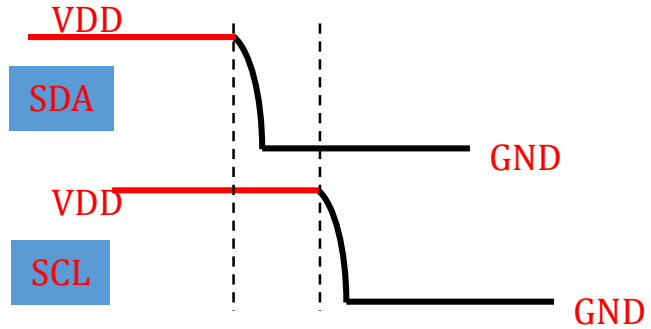
Pour assurer la stabilité des Données :

- La ligne **SDA** doit être **stable** pendant que **SCL est à l'état haut**.
- Cela signifie que les bits de données ne doivent changer que lorsque **SCL est à l'état bas**.
- Si **SDA** change lorsque **SCL est haut**, cela est interprété comme une condition de **START** ou **STOP**.

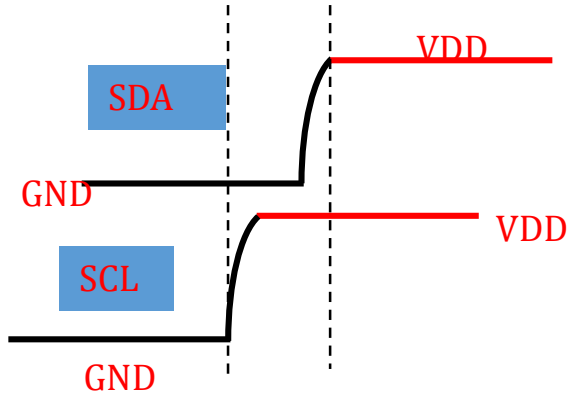
Data Transmission



START condition



STOP condition

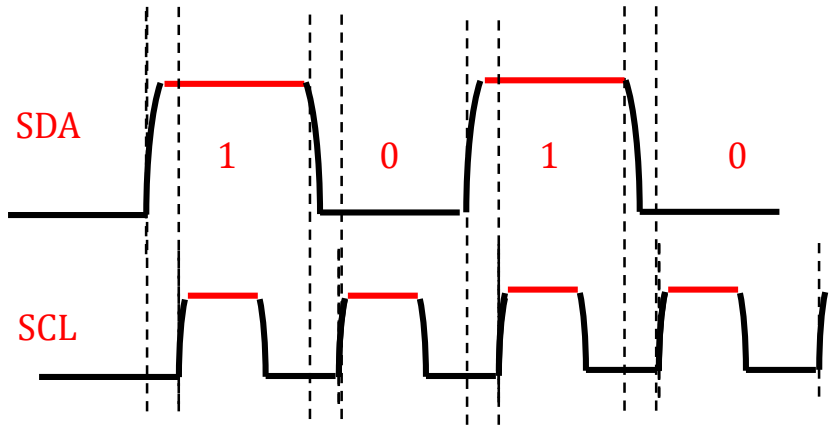


Le protocole I2C: l'envoie des données sur SDA en fonction du signal SCL

Prenons l'envoi d'un bit 1 par exemple:

1. Le maître place 1 sur S**DA** lorsque **SCL est bas**.
2. **SCL** passe à l'état haut, et le récepteur lit le bit 1 sur **SDA**.
3. **SCL** revient à l'état bas, et le maître est alors prêt à placer le prochain bit de données sur **SDA**.
4. …..

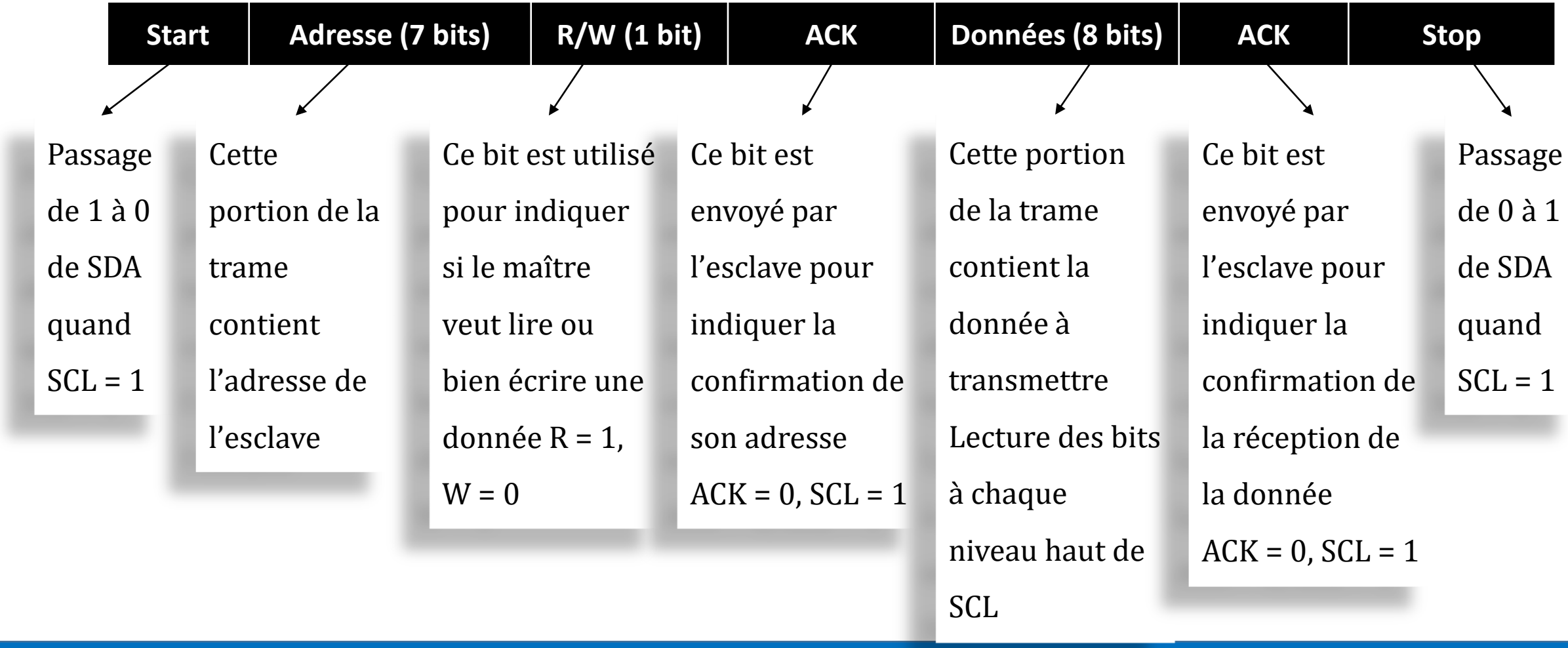
Data Transmission



Cette coordination entre SDA et SCL est fondamentale pour que toutes les transmissions sur le bus I2C soient fiables et que les périphériques puissent lire les bits correctement sans confusion.

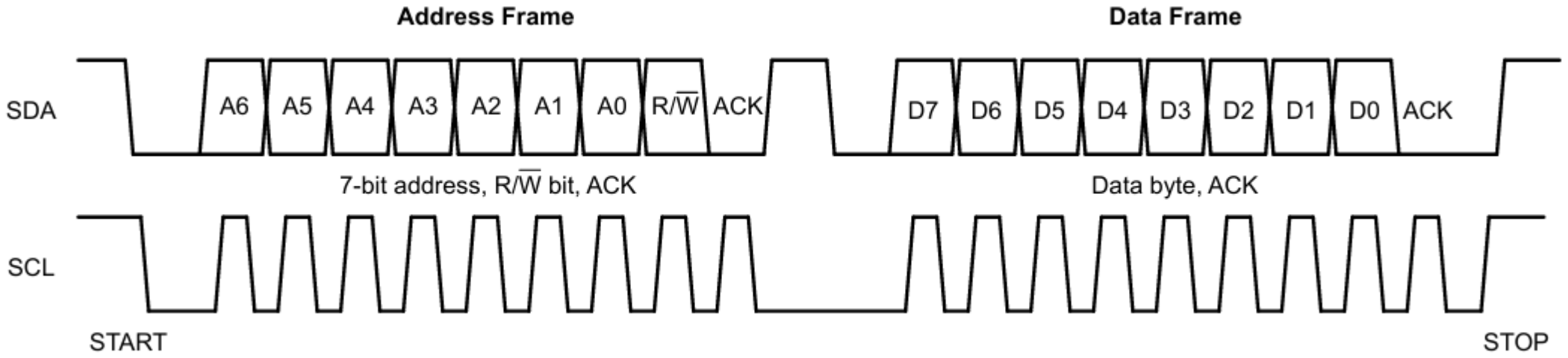
Le protocole I2C: la trame de donnée

La trame de donnée du bus I2C est constituée de la séquence suivante:



Le protocole I2C: la trame de donnée

1. Le maitre met SDA = 0 quand SCL = 1 puis met SCL = 0 (le bus est libéré)
2. L'adresse de l'esclave concerné (7bits soit 128 adresses) est envoyée bit par bit
3. Un bit R/W est spécifié par le maître
4. L'esclave met ACK = 0 quand CLK = 1 pour confirmer la réception de son adresse (ACK = 1 erreur de communication)
5. Les données sont alors transmises par paquet de 8bits (Byte) suivi d'un bit ACK
6. Enfin, le bit stop permet de mettre fin à la communication en lâchant le signal SCL = 1 et en mettant SDA = 1



https://www.ti.com/lit/an/sbaa565/sbaa565.pdf

Le protocole I2C: exemple de code en langage MikroC

1. Le maître envoie les données à l'esclave

```
I2C1_Start();          // Condition Start
I2C1_Wr(0x50);         // Envoi de l'adresse de l'esclave (0x50)
I2C1_Wr(0x3A);         // Envoi de la première donnée (0x3A)
I2C1_Wr(0x7F);         // Envoi de la deuxième donnée (0x7F)
I2C1_Stop();           // Condition Stop
```

1. Le maître reçoit les données de l'esclave

```
I2C1_Start();
I2C1_Wr(0x50 | 1);   // Envoi de l'adresse en mode lecture
data1 = I2C1_Rd(1);  // Lire 1er octet avec ACK
data2 = I2C1_Rd(0);  // Lire 2ème octet avec NACK (fin de communication)
I2C1_Stop();
```