NSY107
Cours 5 Le bus I^2C

Matthias Puech

Master 1 SEMS — Cnam

Introduction

Le protocole I^2C

Le module HAL I^2C

Le protocole SPI

Protocoles de communication inter-puces

Comment communiquent des puces sur une même carte ?

Exemple

MCU $\longleftrightarrow$ ADC/DACs externes, LCD, EEPROM, Radio…

Les standards

Communication par messages selon protocole établi au dessus de :

- UART ✓
- I^2C
- SPI

Présentation d'I^{2}C

Standard développé par Philips en 1982.
Royalty-free depuis 2006.

Nature des données transportées

paquets d'octets de 8 bits par défaut (comme UART)

Support du protocole

couche physique (définit communication au niveau électrique)

Topologie

Bus de données toutes les parties communiquent sur même canal

1. a aussi un mode *multi-master* qu'on n'étudiera pas ici

Présentation d'I^{2}C

Synchrone/Asynchrone

synchrone notion de temps commune à toutes les parties (un fil d'horloge)

Duplex

half-duplex "*à l'alternat*" (un fil de transmission des données)

Symétrie

client-serveur un *maître*, des *esclaves* [1]

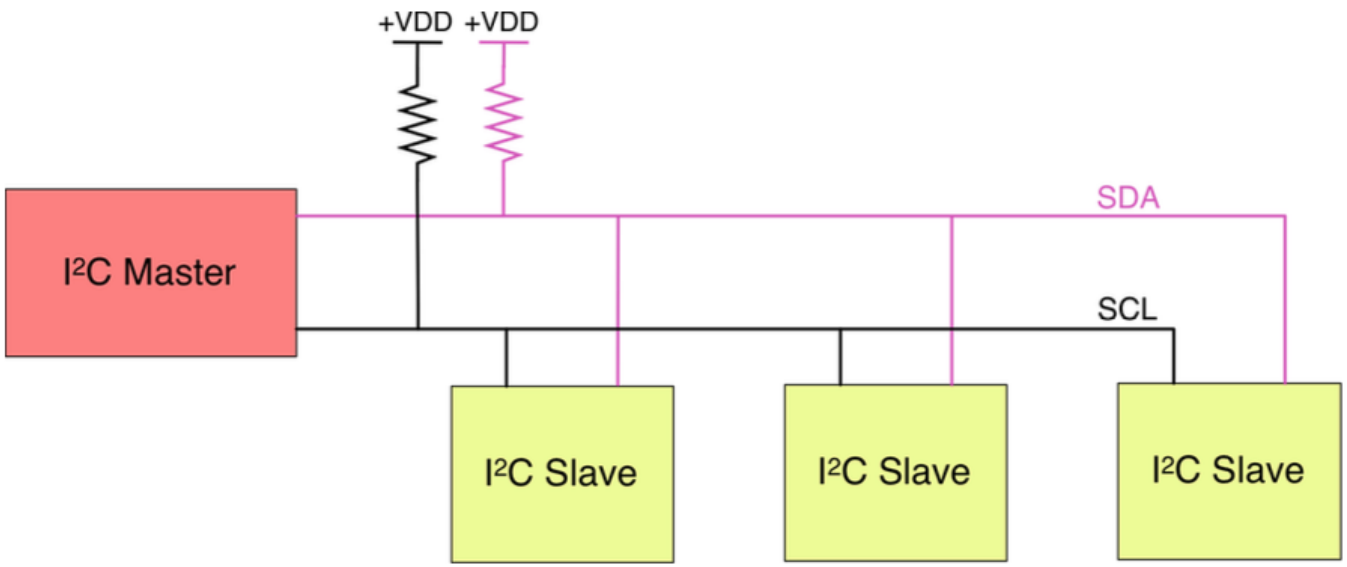
Modalité

série les informations se succèdent sur une voie (fil données)

1. a aussi un mode *multi-master* qu'on n'étudiera pas ici

Topologie

- deux fils :
  - SDA (*Serial Data Line* : les données)
  - SCL (*Serial Clock Line*)
- état haut par défaut (+VDD)
- chaque partie (maître/esclave) peut abaisser SDA ou SCL (communication bidirectionnelle)
- pour nous : maître=MCU, esclave=puce externe



Horloge

La ligne SCL est l'horloge. Elle est générée par le maître.
(signal carré, sa fréquence détermine le débit binaire)

Vitesses

- 100kHz *standard mode*
- 400kHz *fast mode*

1 MHz, 3.4MHz, 5MHz (peu communs)

Clock stretching

Un esclave peut abaisser SCL temporairement pour dire "pause".
(voir plus loin)

Adressage

Besoin d'identifier de façon unique les esclaves
⇝ chaque esclave a une *adresse* (un nombre)

Codage des adresses

7 bits (le plus commun)

10 bits (plus rarement)

Sur les puces esclaves (ADC), l'adresse est :

- soit fixée par le constructeur
- soit configurée par des pins
  (2-3 pins qui fixent les 2-3 derniers bits de l'adresse)

Adressage

Besoin d'identifier de façon unique les esclaves
⇝ chaque esclave a une *adresse* (un nombre)

Codage des adresses

7 bits (le plus commun)

10 bits (plus rarement)

Sur les puces esclaves (ADC), l'adresse est :

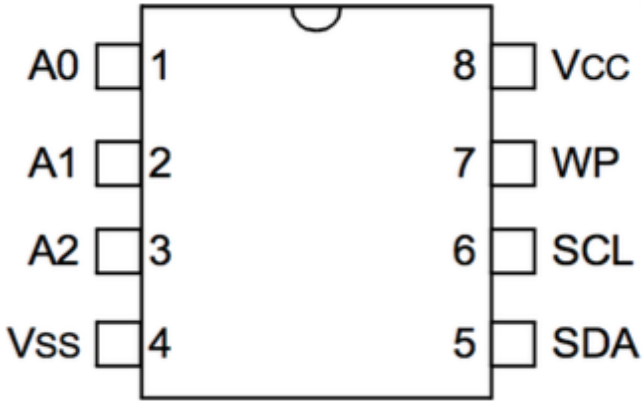
- soit fixée par le constructeur
- soit configurée par des pins
  (2-3 pins qui fixent les 2-3 derniers bits de l'adresse)

Limitation

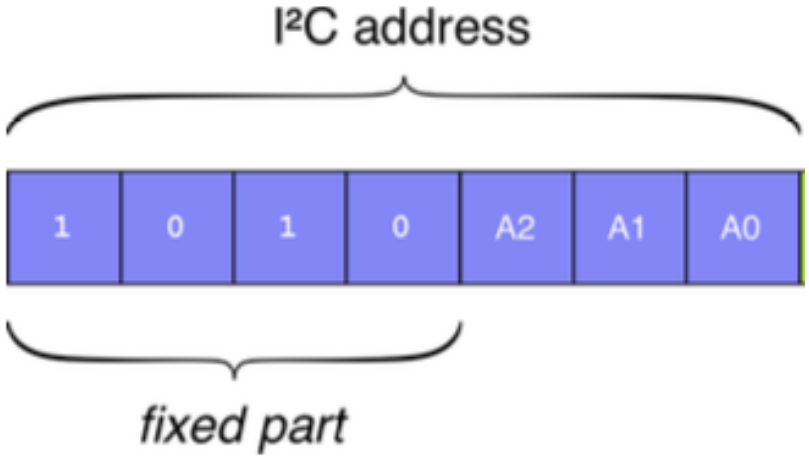
Sur un bus, l'adresse d'un esclave *doit être unique*
(⇝ max 4/8 puces sur le même bus I^2C)

Adressage

Exemple



Puce EEPROM 24LCxx



1010 est le préfixe fixé par le constructeur

Messages

Tout message est initié et terminé et par le maître

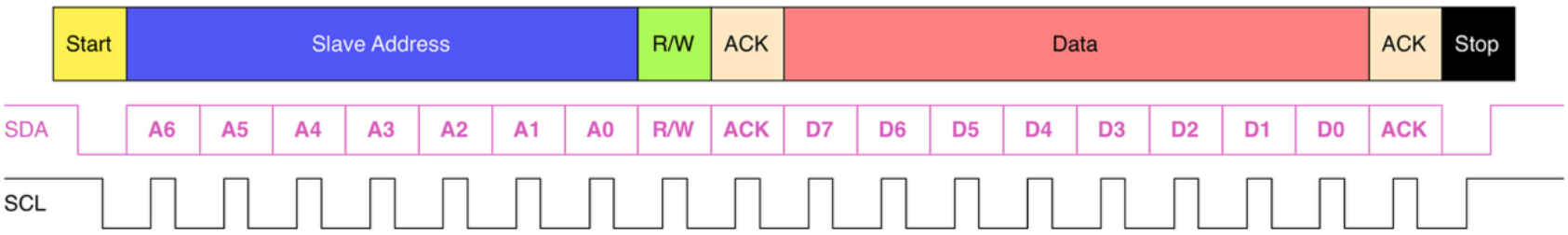
Messages

Tout message est initié et terminé et par le maître

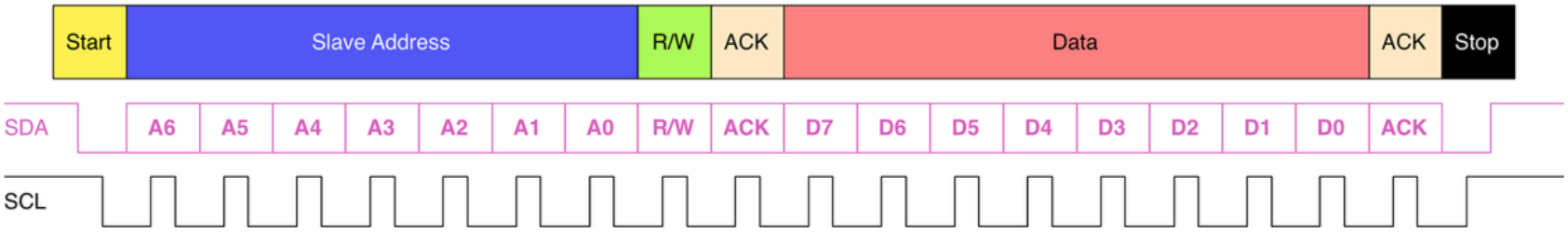
Structure d'un message

- commence par une condition START
- finit par une condition STOP
- constitué de plusieurs *frames* :
  (suite de 8 bits)
  - 1 ou 2 *frames* d'adresse (à qui on s'adresse)
  - 1 ou plusieurs *frames* de données (qu'est-ce qu'on lui dit)

Messages


Start
Slave Address
R/W
ACK
Data
ACK
Stop
SDA
A6
A5
A4
A3
A2
A1
A0
R/W
ACK
D7
D6
D5
D4
D3
D2
D1
D0
ACK
SCL

Messages



Conditions START/STOP

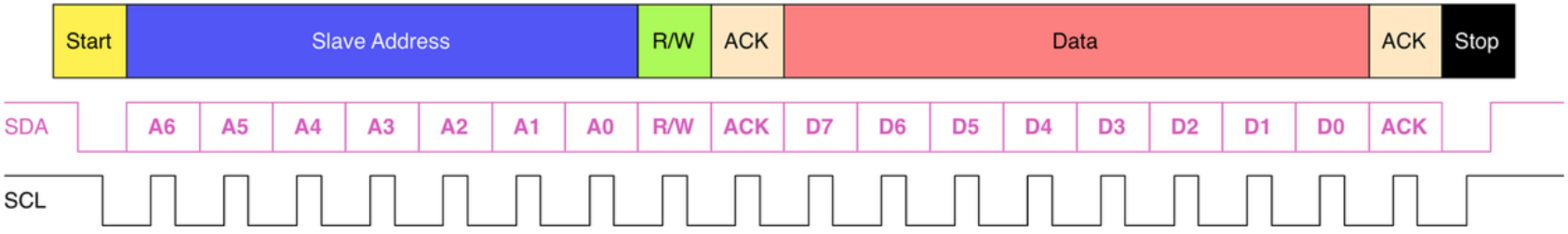
Émise par le maître

START front *descendant* sur SDA quand SCL est *haut*

STOP front *montant* sur SDA qand SCL est *haut*

Le reste du temps, on ne change SDA que quand SCL est *bas*
(pas de confusion possible)

Messages

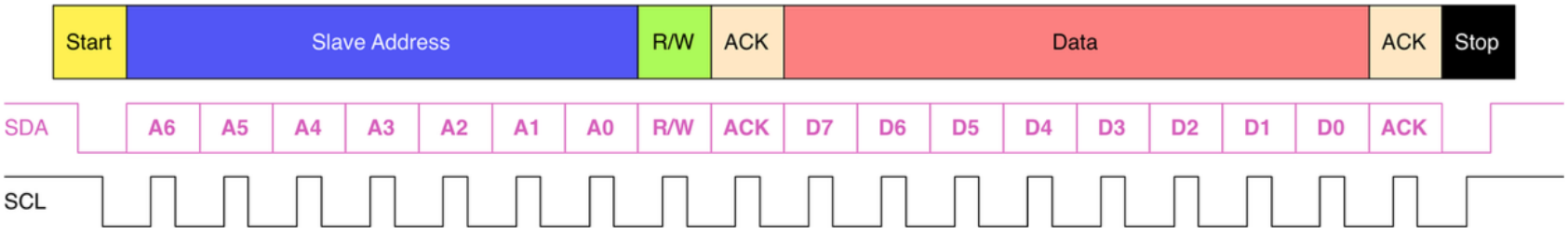


Frame

Suite de 9 bits émise par les deux parties :

- 8 bits transmis transmis de l'émetteur au récepteur
  (l'information, au format *most significant bit first* ou MSB)
- 1 bit transmis du récepteur à l'émetteur
  (ACK = acklowledgement = "bien reçu")
  - récepteur met SDA à 0 s'il a bien reçu le message.
  - si SDA reste à 1, le message n'a pas été reçu :
    - pas d'esclave présent sur le bus à l'adresse donnée
    - récepteur pas disponible (occupé, plus de mémoire)
    - message transmis incorrect

Messages

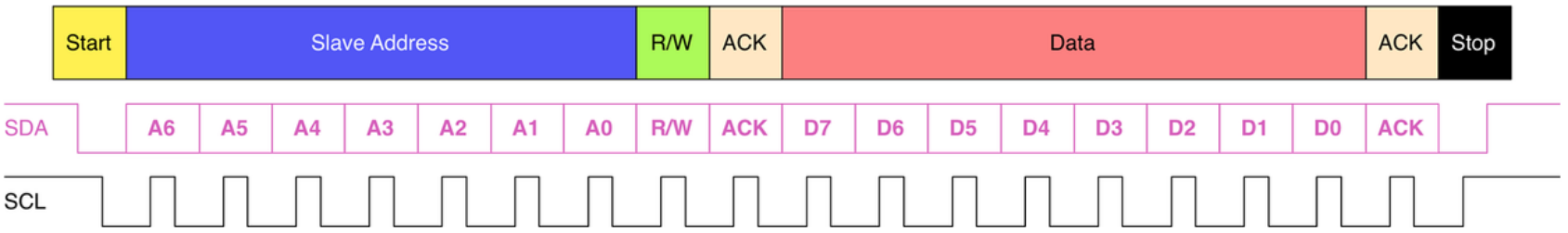


Frame d'adresse

Émise par le maître

- 7 bits de l'adresse de l'esclave à qui on veut parler
- 1 bit de direction (R/W) du reste du message :
  - 0 (W) maître → esclave
  - 1 (R) esclave → maître

Messages

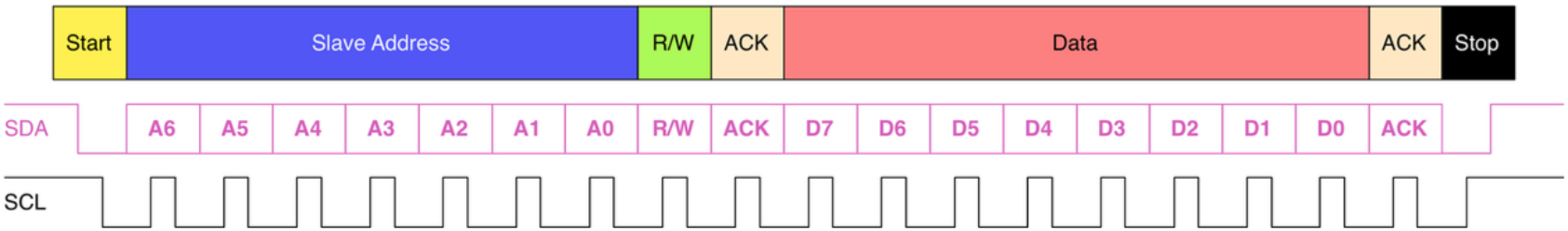


Frame de donnée

Émise par le maître (W) ou l'esclave (R)

- 8 bits
- I^2C ne spécifie pas de structure pour ces données

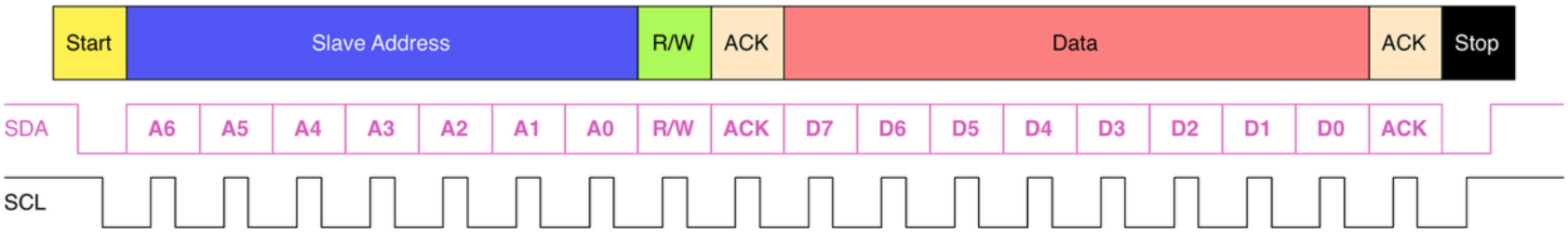
Messages



Exemple

Start 10101101 0 11110000 0 Stop

Messages

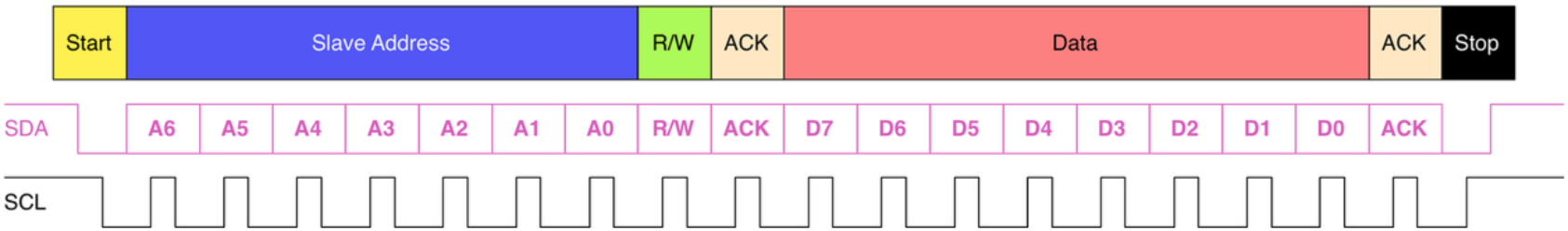


Exemple

Start 10101101 0 11110000 0 Stop

Maître “Bonjour esclave n. 1010 110, qu’as-tu à me dire ?”
Esclave 1010 110 “Bien reçu”
Esclave 1010 110 “J’ai à te dire 11110000”
Maître “Bien reçu, au revoir.”

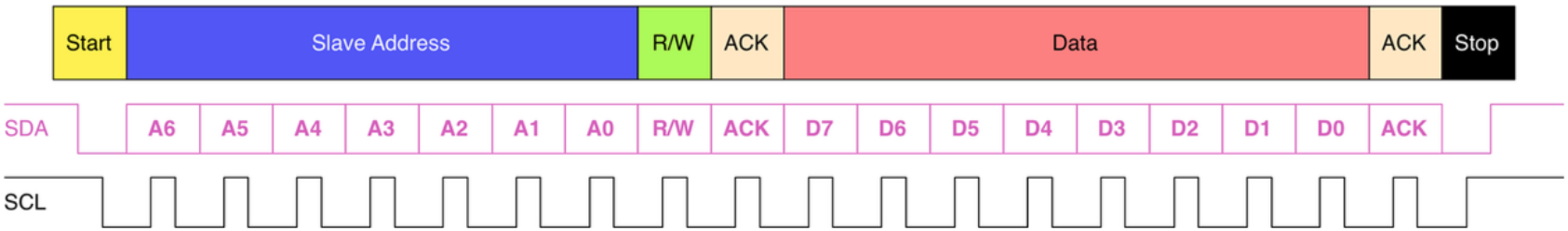
Messages



Exemple

Start 10101100 0 10001000 1 Stop

Messages



Exemple

Start 10101100 0 10001000 1 Stop

Maître “Bonjour esclave n. 1010 110, j’ai un message pour toi, es-tu prêt ?”
Esclave 1010 110 “Je suis prêt”
Maître “J’ai à te dire 10001000”
Esclave 1010 110 “Désolé je n’ai pas compris”
Maître “Au revoir.”

Messages

Tout message est initié et terminé et par le maître

Importante limitation

Impossible pour un esclave d'"alerter" le maître après un évenement
⇝ le maître doit faire du *polling*

Burst mode

Problème

Pour envoyer 4 octets de données, il faut en envoyer 8 (on doit réenvoyer l'adresse à chaque octet)

⇝ *overhead* important de communication

Burst mode

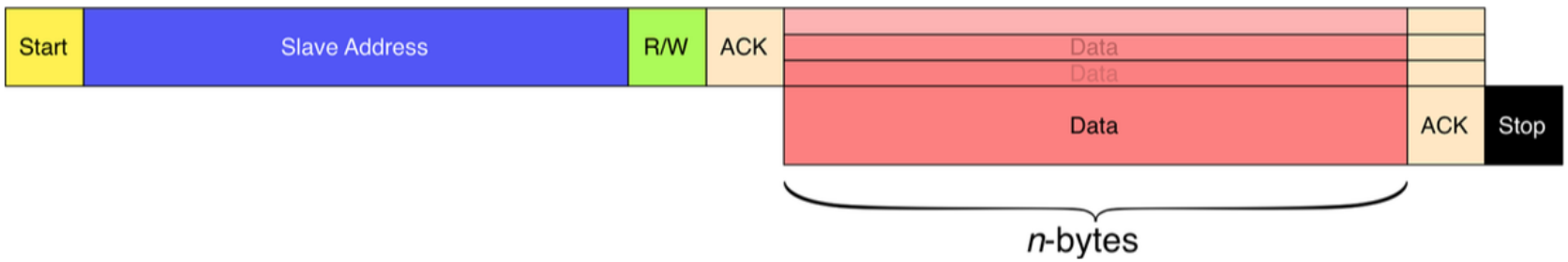
Problème

Pour envoyer 4 octets de données, il faut en envoyer 8 (on doit réenvoyer l'adresse à chaque octet)

⇝ *overhead* important de communication

Burst mode

En fait, le nombre de frames de données dans un message est arbitraire :



C'est Master qui dit STOP

⇝ c'est lui qui choisit combien de frames sont transférées.

Adressage 10 bits

Problème

Les adresses sont affectées par le comité I^2C.
(un constructeur = une plage d'adresses)

2^7 adresses distinctes ne sont pas assez aujourd'hui !
$\rightsquigarrow$ adresses sur 10 bits (1024 esclaves possibles)

Adressage 10 bits

Problème

Les adresses sont affectées par le comité I^2C.

(un constructeur = une plage d'adresses)

2^7 adresses distinctes ne sont pas assez aujourd'hui !

⇝ adresses sur 10 bits (1024 esclaves possibles)

Variante

On transfère l'adresse sur deux frames d'adresse :

Adressage 10 bits

Problème

Les adresses sont affectées par le comité I^2C.
(un constructeur = une plage d'adresses)

2^7 adresses distinctes ne sont pas assez aujourd'hui !
⇝ adresses sur 10 bits (1024 esclaves possibles)

Variante

On transfère l'adresse sur deux frames d'adresse :

Q Comment est-ce que le récepteur sait si la deuxième frame est le reste de l'adresse ou la donnée ?

Adressage 10 bits

Espace d'adressage I^2C

SLAVE ADDRESS	$\overline{R/W}$ BIT	DESCRIPTION
0000 000	0	General call address
0000 000	1	START byte
0000 001	X	CBUS address
0000 010	X	Reserved for different bus format
0000 011	X	Reserved for future purposes
0000 1XX	X	Hs-mode master code
1111 1XX	X	Reserved for future purposes
1111 0XX	X	10-bit slave addressing

Les 7 premiers bits d'addresse

⇝ si la frame d'adresse commence par 11110, alors la frame suivante est la suite de l'adresse.

Clock stretching

Problème

Parfois le master demande communication mais l'esclave a besoin de temps pour préparer sa donnée (calcul, mesure...)

Clock stretching

Problème

Parfois le master demande communication mais l'esclave a besoin de temps pour préparer sa donnée (calcul, mesure…)

Clock stretching = "pause !"

- l'esclave met SCL à 0 le temps dont il a besoin
- aucune communication n'est alors émise sur SDA
- le maître doit donc aussi échantillonner SCL pour détecter une éventuelle mise à 0

Clock stretching

Problème

Parfois le master demande communication mais l'esclave a besoin de temps pour préparer sa donnée (calcul, mesure…)

Clock stretching = "pause !"

- l'esclave met SCL à 0 le temps dont il a besoin
- aucune communication n'est alors émise sur SDA
- le maître doit donc aussi échantillonner SCL pour détecter une éventuelle mise à 0

Limitation

Un noeud "compromis" peut bloquer la communication sur le bus
$\rightsquigarrow$ I^2C est un protocole *coopératif*

Autres aspects d'I^2C

Messages broadcastés

En envoyant à l'adresse 0000 0000 on envoie à tous les esclaves

Multi-master

Plusieurs maîtres peuvent coexister sur le même bus
(arbitration nécessaire, possibles collisions...)

Transactions combinées/séquentielles

On peut :

- accélérer les transactions requêtes/réponses, et
- permettre des données de longueurs variables

Protocoles spécifiques

- I^2C ne fait aucune supposition sur les données véhiculées.
  $\rightsquigarrow$ protocole spécifique à une puce donnée, superposé à I^2C
- La specsheet de la puce spécifie ce protocole

Protocoles spécifiques

- I^2C ne fait aucune supposition sur les données véhiculées.
  ⇝ protocole spécifique à une puce donnée, superposé à I^2C
- La specsheet de la puce spécifie ce protocole

Norme de-facto

Modification de l'état de la puce par écriture dans des *registres*

écriture dans registre on compose un message maître → esclave (W) de $m+n$ octets en deux parties :

1. l'addresse du registre (n octets)
2. sa nouvelle valeur (m octets)

lecture d'un registre deux messages : *lecture*

1. message I^2C maître → esclave (W) :
   on envoie l'adresse du registre (n octets)
2. message I^2C esclave → maître (R) :
   on reçoit sa valeur (m octets)

Exemple : l'accéléromètre boussole LSM303DLHC

- détecte l'accélération et la direction du champs magnétique
- cliquez ici pour sa datasheet
- deux esclaves I^2C sur le même bus :
  - accéléromètre à l'adresse 0011001
  - boussole à l'adresse 0011110 (non configurables)
- possibilité de configurer des interruptions (chute libre, mouvement) sur pins séparées
- interface par lecture/écriture dans des registres (voir Table 17 p. 23)
  - adresses de registre sur 8 bits
  - valeurs des registres sur 8 bits
- les valeurs (accélération/position) sont échantillonnées sur 16 bits ; pour chacune il y a deux registres à lire :

  `OUT_*_L_*` les 8 bits de poids inférieur

  `OUT_*_H_*` les 8 bits de poids supérieur

  Exercice : les recomposer en un entier 16 bits

Sur notre carte

Deux options :

- coder le protocole nous même (*bit-banging*) :
  - utiliser les GPIOs pour mettre deux pins on/off
  - implémenter l'attente de 1/100000 par une boucle
  - implémenter fonctions `transmit(char c)` et `char receive()` en suivant le protocole.

Sur notre carte

Deux options :

- coder le protocole nous même (*bit-banging*) :
  - utiliser les GPIOs pour mettre deux pins on/off
  - implémenter l'attente de 1/100000 par une boucle
  - implémenter fonctions `transmit(char c)` et `char receive()` en suivant le protocole.
- utiliser un des périphériques I^2C embarqué sur le MCU :
  - \+ protocole implémenté en hardware
  - \+ le processeur est déchargé du travail de transmission
  - \+ on communique avec par lecture/écriture dans des registres (comme d'habitude)
  - – le périphérique est relié à des pins prédéfinies (voir specsheet)

HAL I²C Configuration

`lib/HAL/stm32f3xx_hal_i2c.{c,h}`

Périphériques I2C1 et I2C2

Les registres permettent un contrôle fin du protocole
On va les manipuler à haut niveau, grâce à HAL :

- on consulte la datasheet pour savoir à quelle pins correspondent les E/S
- on configure le GPIO des pins correspondantes : *Alternate Function* I2C*n*.
- on déclare et remplit une *handle* `I2C_HandleTypeDef` :
  - `Instance` = `I2C1` ou `I2C2`
  - `Mode` = `HAL_I2C_MODE_MASTER` (ou SLAVE)
  - `Init.Timing` = 0x00902025 (ne posez pas de questions)
  - `Init.AddressingMode` = `I2C_ADDRESSINGMODE_7BIT` (ou `10BIT`)
- on la passe à la fonction `HAL_I2C_Init(...)`

HAL I²C Configuration

`lib/HAL/stm32f3xx_hal_i2c.{c,h}`

Handle

Structure qui contient l'état et la configuration actuelle du périphérique, à passer avec toute communication. (ex : instance, taille de dernière donnée reçue...)

HAL I²C Utilisation en mode Maître

```
HAL_StatusTypeDef HAL_I2C_Master_Transmit(
  I2C_HandleTypeDef *hi2c,
  uint16_t DevAddress,
  uint8_t *pData,
  uint16_t Size,
  uint32_t Timeout);
```

Demande au périphérique d'envoyer un tableau d'octets, et boucle jusqu'à ce que ce soit fait. Renvoit un statut (OK/erreur).

`hi2c`	le handle
`DevAddress`	l'adresse de l'esclave
`pData`	tableau des octets à envoyer
`Size`	taille de `pData`
`Timeout`	timeout avant erreur ou `HAL_MAX_DELAY` pour attendre indéfiniment

HAL I²C Utilisation en mode Maître

```
HAL_StatusTypeDef HAL_I2C_Master_Receive(
  I2C_HandleTypeDef *hi2c,
  uint16_t DevAddress,
  uint8_t *pData,
  uint16_t Size,
  uint32_t Timeout);
```

Envoie l'adresse de l'esclave puis attend sa réponse sur `Size` octets. Renvoit un statut (OK/erreur).

- `hi2c` le handle
- `DevAddress` l'adresse de l'esclave
- `pData` buffer où seront écrites les données reçues
- `Size` taille des données à recevoir (doit être connue à l'avance)
- `Timeout` timeout avant erreur ou `HAL_MAX_DELAY` pour attendre indéfiniment

HAL I^2C Utilisation en mode Maître

Exemple

```
uint8_t request = 0x44;
/* Transmit data request */
if (HAL_I2C_Master_Transmit(&i2c_handle,
                            0x32, &request, 1,
                            HAL_MAX_DELAY) != HAL_OK)
  while(1);

uint8_t data[2];
/* Then we receive the data at this address */
if (HAL_I2C_Master_Receive(&i2c_handle,
                           0x32, &data, 2,
                           HAL_MAX_DELAY) != HAL_OK)
  while(1);

...
```

HAL I^{2}C Utilisation en mode Maître

Les fonctions précédentes étaient blocantes

API non-blocante

- activer l'interruption :

  `HAL_NVIC_EnableIRQ(I2C1_EV_IRQn);`

- utiliser les fonctions de transmission "_IT" :

  `HAL_I2C_Master_Transmit_IT(...)` transmission
  (la fonction retourne immédiatement)

  `HAL_I2C_Master_Receive_IT(...)` réception
  (la fonction retourne immédiatement)

- on définit :

```
void I2C1_EV_IRQHandler() {
    HAL_I2C_EV_IRQHandler(...);
}
```

  pour associer HAL au handler

HAL I²C Utilisation en mode Maître

API non-blocante

- à la fin de l'émission/réception, un "callback" est appelé :

  `HAL_I2C_MasterTxCpltCallback()` transmission OK
  `HAL_I2C_MasterRxCpltCallback()` réception OK
  `HAL_I2C_ErrorCallback()` erreur
  ...

HAL I^{2}C Utilisation en mode Esclave

Chaque périphérique I^{2}C peut faire office d'esclave I^{2}C
Il peut répondre à deux adresses distinctes

Exemple

Communication entre deux MCUs

HAL I²C Utilisation en mode Esclave

Chaque périphérique I²C peut faire office d'esclave I²C
Il peut répondre à deux adresses distinctes

Exemple

Communication entre deux MCUs

Marche à suivre

- on renseigne l'adresse I²C voulue dans la `I2C_HandleTypeDef` (champs `OwnAddress1/2`)
- on utilise les fonctions :

  HAL_I2C_Slave_Transmit(...) attend l'arrivée d'un message R du master, puis envoie les données

  HAL_I2C_Slave_Receive(...) attend l'arrivée d'un message W du master, puis reçois les données
- ...ou leurs variantes non-blocantes

Présentation de SPI

"Standard" développé par Motorola en 197x.

Nature des données paquets d'octets

Couche physique

Topologie "bus" (+ 1 fil SS par esclave)

Synchrone (fil clock)

Duplex full-duplex (2 fils de données)

Symétrie un client, plusieurs serveur

Modalité série

Présentation de SPI

"Standard" développé par Motorola en 197x.

Nature des données paquets d'octets

Couche physique

Topologie "bus" (+ 1 fil SS par esclave)

Synchrone (fil clock)

Duplex full-duplex (2 fils de données)

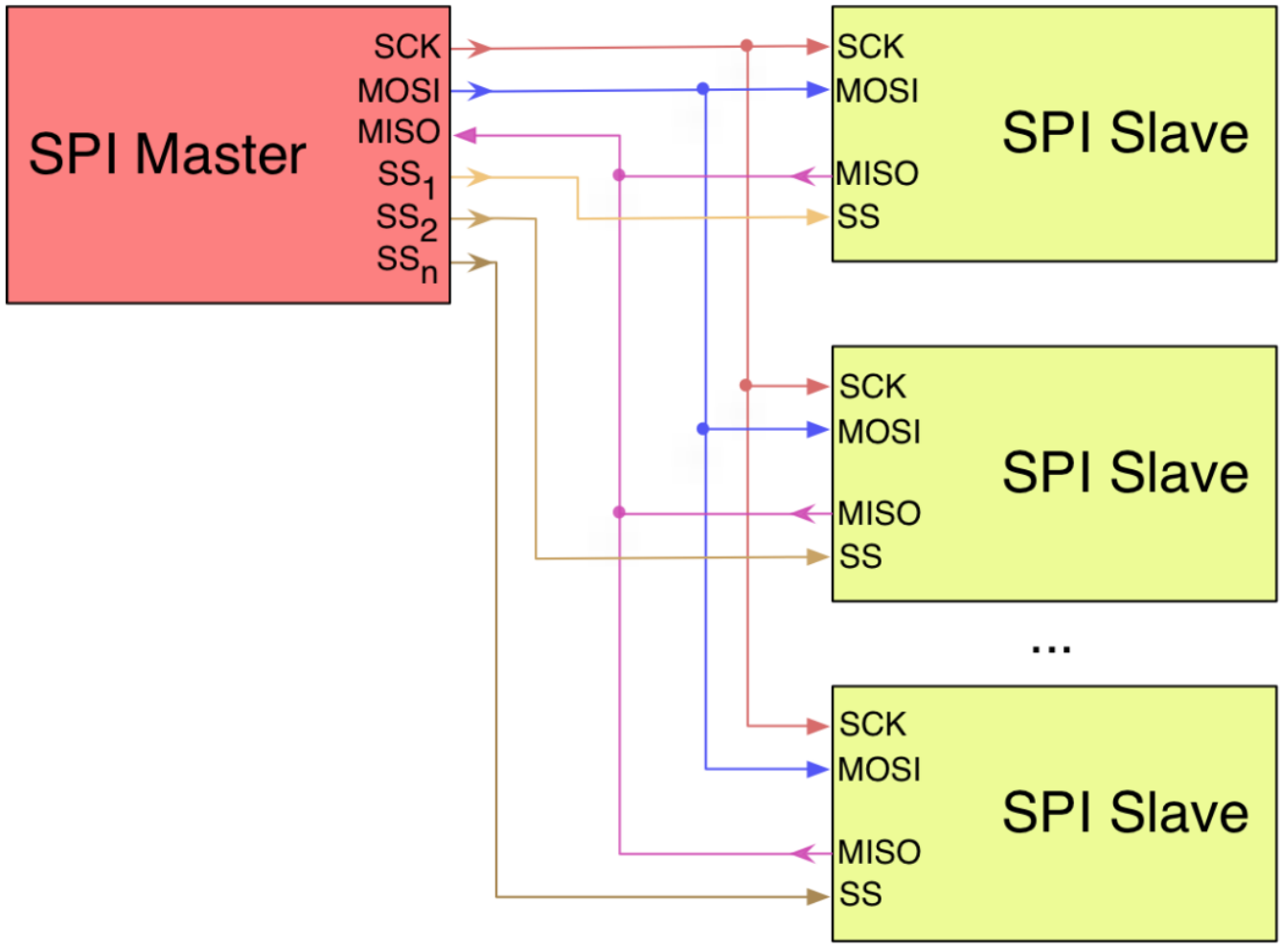
Symétrie un client, plusieurs serveur

Modalité série

Spécificités

- beaucoup plus rapide que I^2C(1-100MHz)
- pas de spécification sur la structure des messages (pas de *frames*, pas d'en-têtes)

Topologie


SPI Master
SCK
MOSI
MISO
SS1
SS2
SSn
SCK
MOSI
MISO
SS
SPI Slave
SCK
MOSI
MISO
SS
SPI Slave
...
SCK
MOSI
MISO
SS
SPI Slave

Topologie

Pour n esclaves

3+n fils :

- SCK (Clock)
- MOSI (*Master Out Slave In*)
- MISO (*Master In Slave Out*)
- $n\times$ SS (*Slave Select*)

Topologie

Pour n esclaves

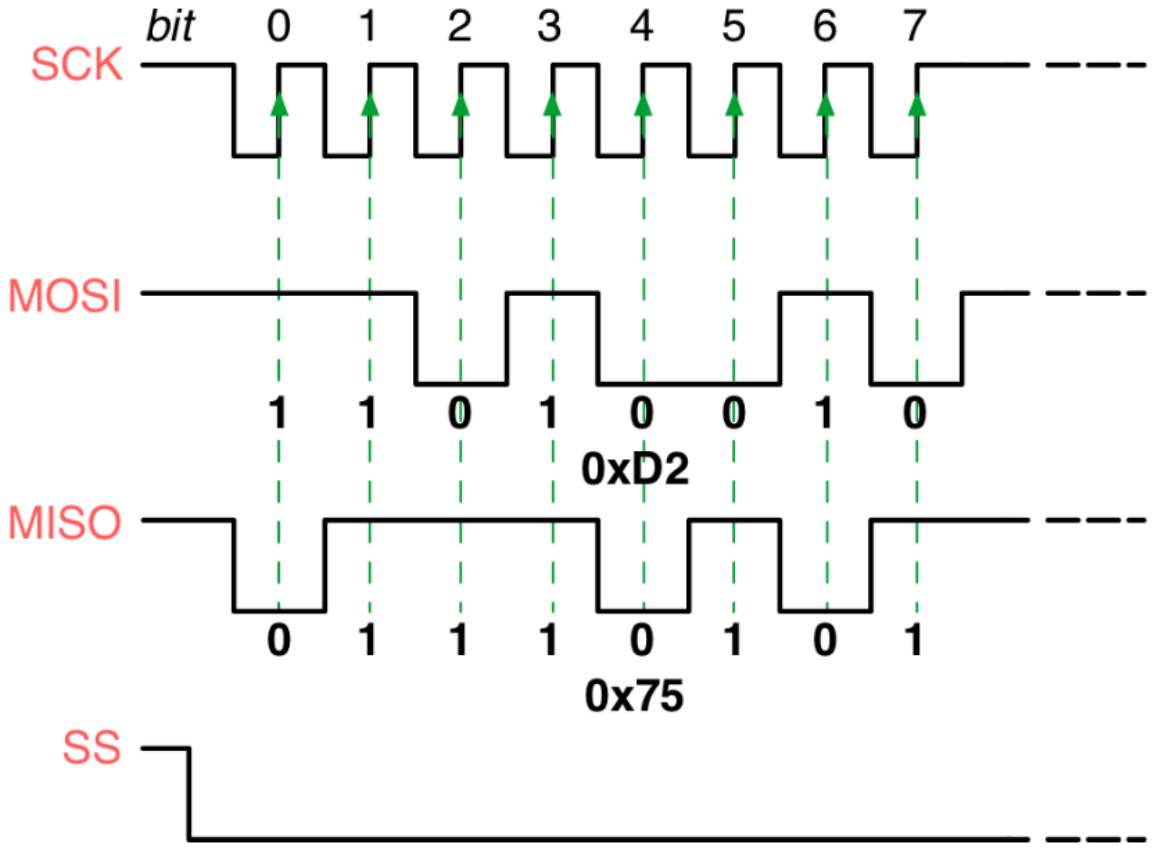
3+n fils :

- SCK (Clock)
- MOSI (*Master Out Slave In*)
- MISO (*Master In Slave Out*)
- $n\times$ SS (*Slave Select*)

Pour 1 esclave

- on se passe du fil SS
- communication unidirectionnelle $\rightsquigarrow$ on fusionne MISO et MOSI en SISO (*Slave In/Slave Out*)
  (on parle alors de *2-wire SPI* (SCK, SISO))

Le protocole, en deux mots



Le protocole, en deux mots

- le maître met SS*i* à 0 ⇝ START
- les données transitent alors simultanément par parquets d'octets

  MOSI maître vers esclave *i*
  MISO esclave *i* vers maître

  (*most significant bit first* ou MSB)
- le signal est échantillonné à chaque front montant de SCK (configurable)
- le maître met SS*i* à 1 ⇝ STOP