

## Séance 7

# Bus et périphériques de communication



Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons Attribution – Pas d'Utilisation Commerciale – Pas de Modification 4.0 International.

# Objectifs

- **Communication** entre un système embarqué et le monde
  - Principes généraux de la communication
  - Notion de protocole de communication
- **Bus et périphériques** de communications
  - Port série, USB
  - I<sup>2</sup>C, SPI, UART, CAN, 1-Wire

# Communication



# Communication

- Connexion entre le système embarqué et le monde extérieur
  - Entre le processeur et les périphériques on-board
  - Avec des périphériques distants hors de la carte
  - Avec d'autres systèmes embarqués ou machines
- Différents éléments à considérer pour établir la communication
  - Type de bus, connectique utilisé
  - Protocole de communication, définition des messages échangés

# Série ou parallèle

- Nombre de **bits transmis** en même temps

*Ajustement du taux de transfert, débit de la connexion*

- Transmission **série** échange un bit à la fois



*Plus lent, mais moins de fils, distances plus longue*

- Plusieurs bits en même temps sur transmission **parallèle**

*Plus rapide, plus de fils et interférences, distances courtes*

# Synchrone ou asynchrone

- **Synchronisation** émetteur/récepteur pour avoir même données

*Même taux de transfert de données et protocole*

- Mode asynchrone utilise des **bits de cadre** comme UART

*Délimitation des données à lire, plus petit débit, moins de fil*

- Mode synchrone utilise une **horloge partagée** comme SPI

*Distance plus courte, plus gros débit, un fil de plus*

# Half-duplex ou full-duplex

- Détermine les **sens de communication** supportés
  - Transmission et réception, simultanément ou non*
- Alternance pour communication bidirectionnelle en **half-duplex**
  - Un seul fil et un seul sens à la fois*
- Communication bidirectionnelle simultanée en **full-duplex**
  - Plus de hardware et fils, meilleur taux d'échange*

# Communication sur la BBB

- Plusieurs **bus de communication** existants sur la BBB  
*Connectés pour certains d'entre eux sur les pins headers*
- Connexions **locales ou distantes** prévues de base
  - Port série, UART, I<sup>2</sup>C et SPI
  - CAN, USB et Ethernet



# Interface d'extension

- Interface d'extension de la BBB grâce aux header pins

*Manipulée à l'aide du device tree overlay (DTO)*

- DT est une structure de données représentant le hardware
  - Description hardware d'un processeur spécifique
  - Lu au moment du boot pour connaître la machine
  - Peut être modifié à chaud pendant l'exécution

# Device Tree (1)

- Le **device tree** est un arbre binaire avec composants hardware

*Au maximum deux fils par nœud, racine seul nœud sans parent*

- **Fichier texte** décrivant un device tree

- Racine de l'arbre est /
- Déclaration de compatibilité avec processeur hardware
- Liste de tous les nœuds décrivant hardware hors processeur

- Nœuds caractérisés par **deux éléments**

- Déclaration de compatibilité du device représenté
- Paramètres d'adressage, notamment registres utilisés

# Device Tree (2)

## ■ Format de base du device tree

*Chargé au boot, portions modifiables à chaud*



```
//*****
/{
    compatible = ??<manufacturer>,<model>;
    <node name>[@<device address>]{
        compatible = ??<manufacturer>,<model>;
        reg = <start address    length>;
    };
//Additional device node descriptions
:
:
:
};

//*****
```

# BBB Device Tree (1)

- Gestion dynamique des DT avec **cape manager** (capemgr)

*Fragments de DT se retrouvent dans /lib/firmware*

- Vérification du **statut de la configuration** des slots et des pins

*À l'aide de deux fichiers se trouvant dans sysfs*

```
$ cat /sys/devices/platform/bone_capemgr/slots
0: PF---- -1
1: PF---- -1
2: PF---- -1
3: PF---- -1
4: P-0-L- 0 Override Board Name ,00A0 ,Override Manuf ,cape-
universaln

$ cat /sys/kernel/debug/pinctrl/44e10800.pinmux/pins | grep 964
pin 89 (44e10964.0) 00000027 pinctrl-single
```



# Mode des pins

- Configuration du **mode de chacune des pins** séparément

*Écriture d'une valeur dans le mux map dans un fichier kernel*



- Exemple avec **P9.11** qui est gpi0[30] donc  $0 \times 32 + 20 = 30$

*Bits 0–2 de la configuration représentent le mode (de 0 à 7)*

```
$ cd /sys/kernel/debug/pinctrl/44e10800.pinmux  
  
$ cat pins | grep 878  
pin 30 (44e10878.0) 00000037 pinctrl-single  
  
$ cat pinmux-pins | grep 878  
pin 30 (44e10878.0): ocp:P9_12_pinmux (GPIO UNCLAIMED) function  
pinmux_P9_12_default_pin group pinmux_P9_12_default_pin
```



Port série

# Port série

- Port série très utilisé dans industrie **automatisation de contrôle**

*L'une des classes de périphérique la plus importante*

- **Interface de communication** série bit à bit

*Port de communication le plus important dans système embarqué*

- Possibilité de contrôle total via une **console série**

*Exécutée sur un port série, ou émulée depuis un périphérique USB*

# Ligne électrique

- **Trois signaux** obligatoires pour une communication série
  - TxD pour les données transmises
  - RxD pour les données reçues
  - GND pour la masse commune
- Six signaux optionnels pour **contrôler la communication**  
*DTR, DCD, DSR, RI, RTS et CTS*
- **Deux extrémités** différentes à une connexion série
  - *Data Terminal Equipment* (DTE), PC mâle
  - *Data Communication Equipment* (DCE), périphérique femelle

# Implémentation

- Port série se réfère au **standard RS-232** (ou RS-422/RS-485)  
*Utilisation de +12 V et -12 V pour les valeurs logiques*
- Remplacement sur ordinateur par **périphérique USB émulateur**  
*Comme ce que fait la BBB par défaut, d'ailleurs*
- Très utilisé dans le **monde embarqué et industriel**
  - Facile à utiliser et à implémenter
  - Hardware et software léger ne surchargeant pas le CPU

# UART

- Universal Asynchronous Receiver/Transmitter (UART)

*Communication niveau TTL, pouvant être sur interface RS-232*

- Il s'agit d'un **dispositif physique** dans microcontrôleur ou IC

*Implémentation via TTL avec deux fils pour communiquer*

- **Caractéristiques**

*Série, asynchrone, full-duplex*

# Port série dans Linux

- Port série représenté par un **fichier `/dev/ttyXXX`**

*Où `xxx` identifie le type de port comme `S0, S1, USBO, ACM0...`*
- Driver Linux **tty core** contrôle un port série
  - Driver de type caractères
  - Contrôle du flux de données et son format
- Utilisation d'une **line discipline (LDISC)** pour l'interaction
  - Intermédiaire entre user space et kernel space (code driver)
  - LDISC standard agit comme terminal Unix

*CTRL+C pour interrompre, LF remplacés par CR/LF...*

# Configuration

- **Vitesse** de transfert des bits sur le port en bit/s

*Vitesses autorisées parmi, par exemple, 75, 9600, 115200... bit/s*

- Nombre de **bits de donnée** utilisé pour échanger l'information

*Typiquement des octets (8 bits), mais bytes de 7 bits possible*

- Deux paramètres liés au **protocole de communication**

- **Parité** pour contrôler les erreurs de transmission avec 1 bit

- **Bit d'arrêt** définit le type de bit utilisé pour marquer le stop

- **Exemple** : « 115200 ,8N1 »

(8 bits, pas de parité, stop 1)



# Exemple serveur echo (1)

- Possibilité d'**émuler un port série** à partir de la connexion USB

*Permet notamment d'avoir une console via connexion USB*
- **Connexion série** via USB entre l'hôte et la BBB
  - /dev/ttyUSB0 côté hôte
  - /dev/ttyGS0 côté BBB
- Exemple d'un simple **serveur echo** qui renvoie ce qu'il reçoit

*Par exemple, Python avec utilisation du module serial*

## Exemple serveur echo (2)

- **Serveur** écoute port série et renvoie ce qu'il reçoit

*Utilisation de serial.readline() pour lire un texte*

```
1 def server(serial):
2     while True:
3         line = serial.readline()
4         print('echoing: ', line.decode().strip())
5         serial.write(line)
```

# Exemple serveur echo (3)

- Envoi d'une chaîne par le **client** et affichage réponse serveur

*Utilisation de serial.write() pour écrire un texte*

```
1 def client(serial):
2     while True:
3         line = input('>> ')
4         if line == 'exit':
5             break
6
7         serial.write(line.encode() + b'\n')
8         line = serial.readline()
9         print('got: ', line.strip())
```

# Exemple serveur echo (4)

- Démarrage programme principale après **configuration** du port

*Création d'un objet de type serial.Serial*

```
1 if __name__ == '__main__':
2     mode = sys.argv[1]
3     device = sys.argv[2]
4
5     serial = serial.Serial(
6         port      = device,
7         baudrate = 115200,
8         bytesize = 8,
9         parity   = 'N',
10        stopbits = 1,
11        timeout   = None,
12        xonxoff   = 0,
13        rtscts    = 0
14    )
15
16    if mode == "client":
17        client(serial)
18    elif mode == "server":
19        server(serial)
20
21    serial.close()
```

# Port série sur BBB

- Six ports séries sur la BBB, un seul activé par défaut

*/dev/tty00 couplé console série, et /dev/tty03 unidirectionnel*

Nom	Périphérique	TxD	RxD	RTS	CTS
<b>UART1</b>	/dev/tty01	P9.24	P9.26		
<b>UART2</b>	/dev/tty02	P9.21	P9.22	P8.38	P8.37
<b>UART3</b>	/dev/tty03	P9.42		P8.34	P8.36
<b>UART4</b>	/dev/tty04	P9.13	P9.11	P8.33	P8.35
<b>UART5</b>	/dev/tty05	P8.37	P8.38		

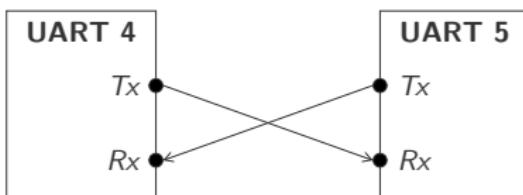
```
$ ls /dev/tty0*
/dev/tty00  /dev/tty01  /dev/tty02  /dev/tty04

$ dtc -I dtb -O dts /lib/firmware/BB-UART4-00A0.dtbo | grep
exclusive-use
    exclusive-use = "P9.13", "P9.11", "uart4";
```

# Exemple serveur echo (5)

- Serveur echo entre deux ports UART physique réel

*Même exemple que précédemment, lancé avec les bons chemins*



```
$ echo BB-UART5 > /sys/devices/platform/bone_capemgr/slots
$ ls /dev/tty0*
/dev/tty00  /dev/tty01  /dev/tty02  /dev/tty04  /dev/tty05
```

**Bus USB**



# Bus USB



- Bus **polyvalent** pour connecter un périphérique électronique

*Communication entre le CPU et des périphériques*

- Au moins **deux acteurs** impliqués dans communication USB

- **Hôte** dirige le trafic vers les périphériques

- **Périphérique** répond aux requêtes de l'hôte

- Hôte **interroge régulièrement** tous les périphériques USB

*Vérification de s'ils veulent lui envoyer un message*

# Processus d'énumération

- L'hôte comprend quelle **sorte de périphérique** a été connecté  
*Reconfiguration automatique du système pour pouvoir le gérer*
- **Processus d'énumération** lors 1<sup>re</sup> connexion d'un périphérique
  - Envoi d'un signal de reset au périphérique
  - Lecture des informations du périphérique USB par l'hôte
  - Identification sans équivoque du périphérique USB
- Chargement d'un **driver** pour contrôler le périphérique USB  
*Une fois fait, le périphérique est dans l'état configuré*



# Branchement périphérique USB

- Branchement périphérique USB et monitoring **messages kernel**

*Insertion d'une clé WiFi*

```
$ dmesg
[ 5797.828445] usb 1-1: new high-speed USB device number 2 using musb-hdrc
[ 5797.959670] usb 1-1: New USB device found, idVendor=7392, idProduct=7811
[ 5797.959709] usb 1-1: New USB device strings: Mfr=1, Product=2,
SerialNumber=3
[ 5797.959726] usb 1-1: Product: 802.11n WLAN Adapter
[ 5797.959741] usb 1-1: Manufacturer: Realtek
[ 5797.959756] usb 1-1: SerialNumber: 00e04c000001
[ 5798.565101] rtl8192cu: Chip version 0x10
[ 5798.658479] rtl8192cu: MAC address: 74:da:38:2e:1f:bc
[ 5798.658517] rtl8192cu: Board Type 0
[ 5798.658645] rtl_usb: rx_max_size 15360, rx_urb_num 8, in_ep 1
[ 5798.658835] rtl8192cu: Loading firmware rtlwifi/rtl8192cu_fw_TMSC.bin
[ 5798.771670] ieee80211 phy0: Selected rate control algorithm 'rtl_rc'
[ 5798.796870] usbcore: registered new interface driver rtl8192cu
[ 5798.804345] rtl8192cu: MAC auto ON okay!
[ 5798.876593] rtl8192cu: Tx queue select: 0x05
[ 5799.498199] IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready
```

# Ligne électrique

- Quatre signaux dans les standards USB 1.0 et USB 2.0
  - D+ pour les données positives
  - D- pour les données négatives
  - VCC fournit une alimentation à 5V
  - GND pour la masse commune
- Ajout de nouveaux signaux dans le standard USB 3.0

*Superspeed transmitter/receiver dual pairs*
- Possibilité d'alimenter le périphérique grâce à la ligne vcc

*De 5 V et 1,5 A jusque 20 V et 5 A en puissance max fournie*

# Bus USB dans Linux



- Support des modes **hôte ou périphérique** dans le kernel Linux  
*Périphérique appelés “USB gadgets” dans le kernel*
- **Système embarqué** peut agir dans les deux modes  
*La BBB possède notamment un port USB pour relier l'hôte*
- **Communication** avec un master qui poll les périphérique  
*Poll sur endpoint IN ou OUT, selon sens communication*

# Type de endpoint



- Configuration et récupération d'information via le **contrôle**

*Obligatoire et utilisé avec transfert de données asynchrone*

- Émulation de la **ligne d'interruption** des CPUs

*Transfert garanti de petite quantité de données synchrone*

- Large transfert de données avec **bulk**

*Transfert asynchrone, sans perte, mais pas de garantie temps*

- Transfert **isochrone** de larges quantités de données

*Application temps-réel comme vidéos, mais perte de données*

# Hôte USB

- Branchement périphérique provoque **chargement d'un driver**
  - Driver spécifique pour le périphérique branché
  - Driver générique pour la classe du périphérique branché
- **Lecteur de codes-barres** USB agit comme un clavier USB

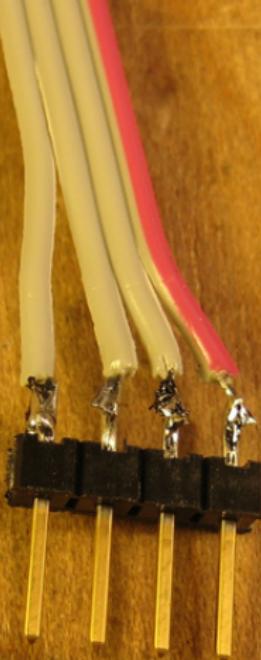
*Renvoie le code lu comme une chaîne de caractères*

# Périphérique USB

- Système embarqué se comporte comme **périphérique USB**  
*Utilisation du sous-système USB gadget du kernel*
- Plusieurs **comportements** possibles pour le système embarqué
  - Accès à une console série via /dev/ACM0
  - Communication série/Ethernet via /dev/usb0
  - Stockage d'un système de fichier comme une clé USB

# Accès direct au bus USB

- Accès direct aux **données brutes** sans passer par driver  
*Seule possibilité si Linux n'a rien trouvé après énumération*
- Utilisation de la **librairie libusb** pour accéder au bus  
*Offre des fonctions de bas niveau d'accès au bus USB*



Bus I<sup>2</sup>C

# Bus I<sup>2</sup>C (1)

- Bus **Inter-Integrated Circuit (I<sup>2</sup>C)** connecte éléments on-board  
*Communication entre périphériques on-board et le CPU*
- Réseau entre **éléments proches** sur petite surface  
*Comme un CPU master et plusieurs senseurs/actuateurs esclaves*
- Possibilité d'une **ligne d'interruption** directe avec le CPU  
*Indique qu'un message doit être lu immédiatement par le master*
- **Caractéristiques**  
*Série, synchrone, des maîtres/des esclaves*

## Bus I<sup>2</sup>C (2)

- Création d'un réseau avec des **nœuds** connectés sur bus I<sup>2</sup>C

*Chacun étant configuré comme étant un maître ou un esclave*

- Connexion de **sous-systèmes** d'une même board entre eux

*Mémoire de type EEPROM, convertisseurs DAC et ADC...*

# Ligne électrique

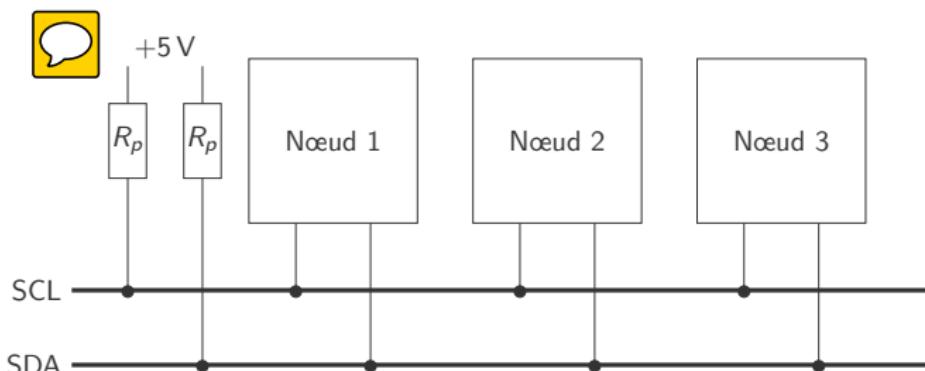
- Trois fils utilisés pour un bus I<sup>2</sup>C
  - SCL fournit un signal d'horloge
  - SDA fournit le bus des données
  - GND pour la masse commune
- Possibilité d'une ligne d'interruption directe sur le CPU

*Souvent implémentée comme ligne dédiée reliée à un GPIO*

# Connexion

- Connexion parallèle de plusieurs périphériques I<sup>2</sup>C

*Ajout de résistances pull-up, sauf si intégrées dans le contrôleur*



# Bus I<sup>2</sup>C dans Linux

- Périphérique I<sup>2</sup>C identifié par une **adresse sur 7 bits**

*Adresse pas affectée au runtime, mais par designer du board*

- Master utilise adresse pour **atteindre un périphérique** sur le bus

*Et indication de si le master veut lire (1) ou écrire (0)*

- Deux modes **master et slave**, de manière similaire à l'USB

*Et donc deux types de drivers supportés dans le kernel*

# Bus I<sup>2</sup>C sur la BBB (1)

- Trois bus I<sup>2</sup>C sur la BBB, chacun avec un master dédié

*Une pas exportée, une dédiée au cape EEPROM et une libre*

Nom	SDA	SCL	Adresse mémoire
i2c0	Pas exporté		0x44E0B000
i2c1	P9.18 ou P9.26	P9.17 ou P9.24	0x4802A000
i2c2	P9.20 ou P9.22	P9.19 ou P9.21	0x4819C000

```
& ls /sys/bus/i2c/devices/
0-0024  0-0034  0-0050  0-0070  2-0054  2-0055  2-0056  2-0057
i2c-0  i2c-1  i2c-2

& ls -l /sys/bus/i2c/devices/i2c-2
lrwxrwxrwx 1 root root 0 Mar 26 16:29 /sys/bus/i2c/devices/i2c-2
-> ../../../../../../devices/platform/ocp/4819c000.i2c/i2c-2
```

# Bus I<sup>2</sup>C sur la BBB (2)

- Trois bus I<sup>2</sup>C sur la BBB, chacun avec un master dédié

*Une pas exportée, une dédiée au cape EEPROM et une libre*

Nom	SDA	SCL	Adresse mémoire
i2c0	Pas exporté		0x44E0B000
i2c1	P9.18 ou P9.26	P9.17 ou P9.24	0x4802A000
i2c2	P9.20 ou P9.22	P9.19 ou P9.21	0x4819C000

```
& dtc -I dtb -O dts /lib/firmware/BB-I2C2-00A0.dtbo | grep  
exclusive-use  
    exclusive-use = "P9.19", "P9.20", "i2c2";
```



# Accès direct au bus I<sup>2</sup>C

- Possibilité d'**accès direct au bus** depuis le user space  
*Sauf l'éventuelle ligne d'interruption à gérer dans le kernel*
- **Trois étapes** à suivre pour accéder à un périphérique I<sup>2</sup>C
  - Créer le bus de périphérique I<sup>2</sup>C pour pouvoir y accéder
  - Ouvrir le fichier représentant le bus
  - Définir l'adresse de l'esclave à contacter avec ioctl
  - Lecture et écriture sur le bus avec read et write

# Exemple accéléromètre (1)

- Libérer puis activer le bus I<sup>2</sup>C en changeant l'overlay
  - Écrire dans un fichier

```
$ echo BB-I2C2 > /sys/devices/platform/bone_capemgr/slots
```

- ou utiliser config-pin

```
$ config-pin overlay BB-I2C2
```

# Exemple accéléromètre (2)

- Connexion de l'accéléromètre sur le bus **i2c-2** (P9.19 et P9.20)

*Retrouver que la connexion est bien faite avec i2cdetect*

```
$ i2cdetect -r 2
WARNING! This program can confuse your I2C bus, cause data
loss and worse!
I will probe file /dev/i2c-2 using read byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n] Y
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- -- --
10:          -- -- -- -- -- -- -- -- -- -- -- -- -- --
20:          -- -- -- -- -- -- -- -- -- -- -- -- --
30:          -- -- -- -- -- -- -- -- -- -- -- -- --
40:          -- -- -- -- -- -- -- -- -- -- -- --
50:          -- -- -- 53 UU UU UU UU -- -- -- -- --
60:          -- -- -- -- -- -- -- -- -- -- -- --
70:          -- -- -- -- -- -- -- -- -- -- -- --
```



# Exemple accéléromètre (3)

- **Interrogation** de l'accéléromètre pour récupérer son ID

*Lire la valeur du registre 0x00*

```
$ i2cget -y 2 0x53 0x00  
0xe5
```



- **Configuration** de l'accéléromètre pour activer le mode mesure

*Écrire 0x08 dans le registre 0x2D*

```
$ i2cset -y 2 0x53 0x2D 0x08
```

# Exemple C (1)

- Ouverture du fichier représentant le bus **i2c-2**

*En lecture et en écriture, pour toute la transaction sur le bus*

```
1 #include <stdio.h>
2 #include <stdint.h>
3 #include <stdlib.h>
4 #include <linux/i2c-dev.h>
5 #include <fcntl.h>
6
7 int main() {
8     int file;
9     char *filename = "/dev/i2c-2";
10
11     if ((file = open(filename, O_RDWR)) < 0) {
12         perror("Failed to open the I2C bus");
13         exit(1);
14 }
```

# Exemple C (2)

## ■ Configuration de l'adresse de l'esclave



*Juste donner l'adresse 7 bits, valable toute la transaction*

```
1 int addr = 0x53;
2 if (ioctl(file, I2C_SLAVE, addr) < 0) {
3     perror("Failed to acquire access or talk to slave.\n");
4     exit(1);
5 }
```

## ■ Écriture de données vers l'esclave sur le bus

*Valeur du registre dont on veut lire la valeur*

```
1 uint8_t buff[10] = {0};
2 buff[0] = 0x00;
3 if (write(file, buff, 1) != 1) {
4     perror("Failed to write register address to read.\n");
5     exit(1);
6 }
```

# Exemple C (3)

- **Lecture** de données depuis l'esclave sur le bus

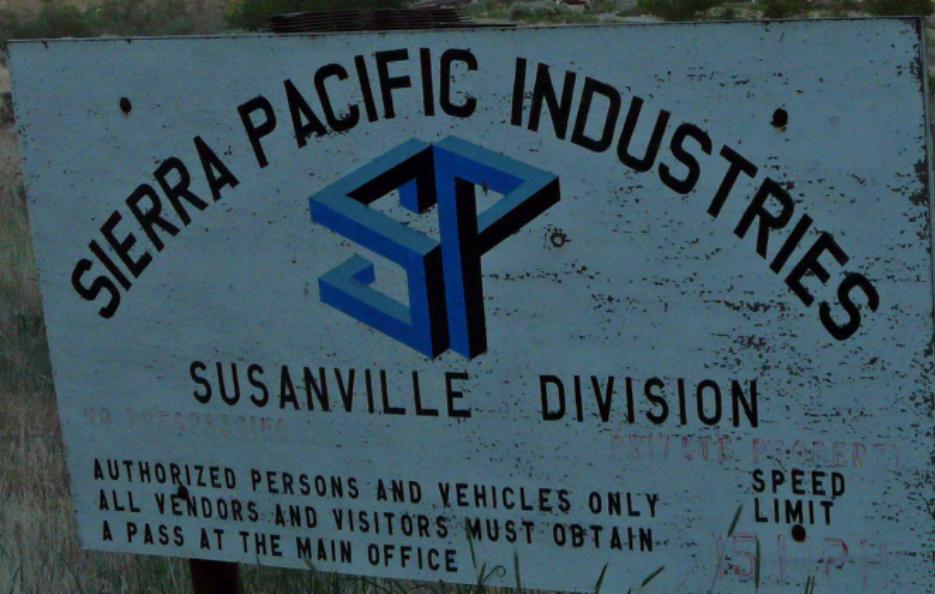
*Valeur du device ID qui est de 0xe5, soit 229 en decimal*

```
1 if (read(file, buff, 1) != 1) {
2     perror("Failed to read register from I2C bus.\n");
3     exit(1);
4 }
5 printf("Data read: (%u)\n", buff[0]);
```

- **Fermeture** du fichier représentant le bus I<sup>2</sup>C

```
1 close(file);
2
3 return 0;
4 }
```

Bus SPI



# Bus SPI

- Bus **Serial Peripheral Interface** (SPI) similaire à I<sup>2</sup>C

*Débits plus grands que I<sup>2</sup>C et UART et transferts bidirectionnels*

- Partage d'une **clock commune** de synchronisation comme I<sup>2</sup>C

*Choix par le master d'une fréquence supportée par les esclaves*

- **Caractéristiques**

*Série, full-duplex, synchrone et un maître/des esclaves*

## Bus SPI (2)

- Un ou plusieurs fils de **sélection de l'esclave**

*Ces lignes sont typiquement actives lorsqu'elles sont basses*

- Deux fils pour **échanger les données** dans les deux sens
  - Communication dans les deux sens de manière simultanée
  - Tel un registre glissant, envoi d'un bit par chaque côté
- Flux de données **efficaces** pour application multimédia

*LCDs, vidéos, traitement de signal, télécommunication...*

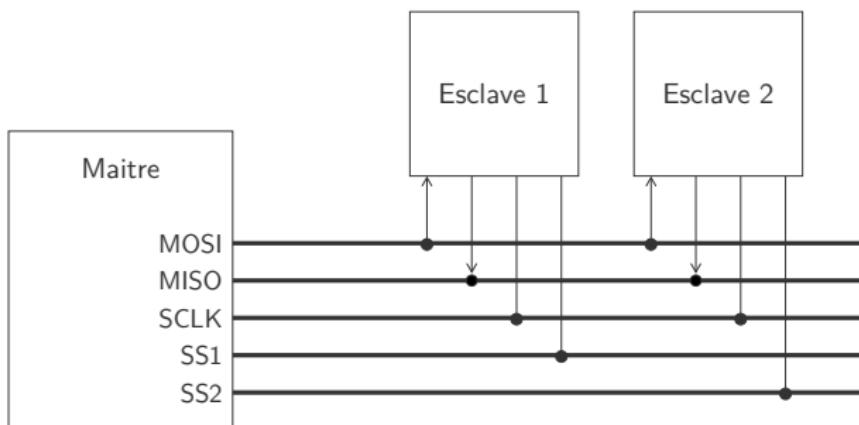
# Ligne électrique

- **Cinq signaux** dans le protocole SPI
  - SCLK fournit un signal d'horloge
  - MOSI fournit le bus des données Master-Out/Slave-In
  - MISO fournit le bus des données Master-In/Slave-Out
  - SS pour sélectionner l'esclave
  - GND pour la masse commune
- Une **ligne SS** par esclave qu'il est possible de supporter  
*Possibilité de générer le signal SS par une GPIO classique*

# Connexion

- Connexion parallèle de plusieurs périphériques SPI

*Mais connexions séparées des différentes lignes SS*



# Bus SPI dans Linux

- Bus SPI représenté par le dossier `/sys/bus/spi/devices`

*Contient tous les périphériques disponibles sur le système*

- Accès brut au bus SPI similaire à l'I<sup>2</sup>C

- Créer le bus de périphérique SPI pour pouvoir y accéder
- Ouvrir le fichier représentant le bus
- Lecture et écriture sur le bus avec `read` et `write`

# Bus SPI sur la BBB (1)

- Deux bus SPI sur la BBB, avec deux lignes SS

*Un réservé pour le HDMI, l'autre libre*

Nom	MISO	MOSI	SCLK	SS0	SS1
spi1	P9.21	P9.18	P9.22	P9.17	Pas disponible
spi2	P9.29	P9.30	P9.31	P9.20 ou P9.28	P9.19 ou P9.42

```
& ls /sys/bus/spi/devices/
spi1.0    spi1.1    spi2.0    spi2.1

& ls /sys/bus/spi/devices/spi2.0/
driver    modalias   of_node   power   spidev   statistics   subsystem
uevent
```

# Bus SPI sur la BBB (2)

- Deux bus SPI sur la BBB, avec deux lignes SS

*Un réservé pour le HDMI, l'autre libre*

Nom	MISO	MOSI	SCLK	SS0	SS1
<b>spi1</b>	P9.21	P9.18	P9.22	P9.17	Pas disponible
<b>spi2</b>	P9.29	P9.30	P9.31	P9.20 ou P9.28	P9.19 ou P9.42

```
& dtc -I dtb -O dts /lib/firmware/BB-SPIDEV1-00A0.dtbo | grep  
exclusive-use  
    exclusive-use = "P9.31", "P9.29", "P9.30", "P9.28", "spi1";
```

# Bus 1-Wire



# Bus 1-Wire (1)



- Bus **1-Wire** (W1 ou OW) moins connu, mais important

*Lent par rapport aux autres, mais un seul fil de données*

- **Simplification** de la connexion entre CPU et périphérique

*Identification, authentification, donnée de calibration...*

- Alimentation par un **condensateur à recharger** (80 pF)

*Rechargée par le fil de données avant la transmission effective*

- **Caractéristiques**

*Half-duplex, asynchrone, un master/des esclaves*

## Bus 1-Wire (2)

- Similaire à I<sup>2</sup>C mais avec **plus longue portée**  
*Et avec un débit de données beaucoup plus faible*
- Communication avec **petits périphériques** peu onéreux  
*Thermomètre digital, instruments météorologiques...*
- Utilisé dans **alimentation Dell** pour vérifier l'adaptateur  
*Informations de puissance, courant et tension via 3<sup>e</sup> fil*

# Ligne électrique

- Deux ou trois fils utilisés pour un bus 1-Wire
  - Data pour les données échangées
  - GND pour la masse commune
  - VCC fournit une alimentation (optionnel)
- VCC optionnel si mécanisme de stockage d'énergie in-built

*Utilisation de Data pour alimentation une capacité*

# Mode parasite

- Mode d'utilisation avec Data pour l'**alimentation**

*Ralentissement de la communication le temps de la charge*

- Il faut **pull-up Data** suffisamment longtemps

*Pour charger complètement la capacité interne*

Mode parasite



Mode normal



Data



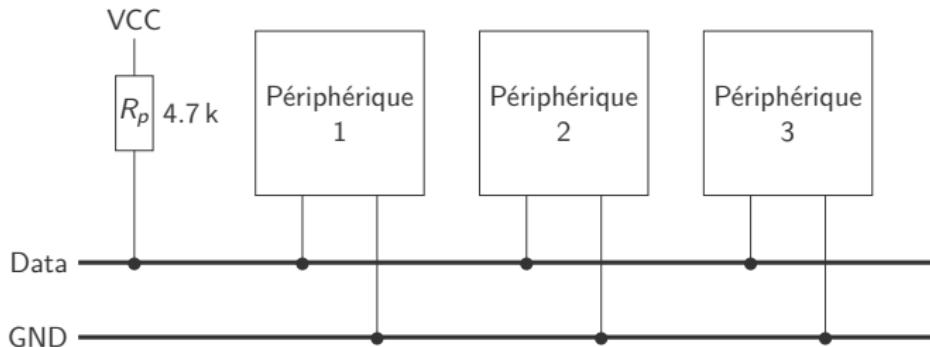
# Connexion (1)

- Un microcontrôleur est **le master** connecté à **des esclaves**  
*Numéro de série sur 64 bits (avec type de périphérique et CRC)*
- Trouver esclaves par protocole énumération appelé **singulation**
  - Diffusion d'un message broadcast particulier
  - Commande de sélection avant de contacter un périphérique
- **Connexion et déconnexion** du bus très facile

*Périphérique stocke sa propre configuration et est de suite prêt*

# Connexion (2)

- Connexion de plusieurs périphériques **en parallèle**  
*Utilisation d'une résistance pull-up*
- Réseau de périphériques avec un master appelé **MicroLAN**



# Bus 1-Wire dans Linux

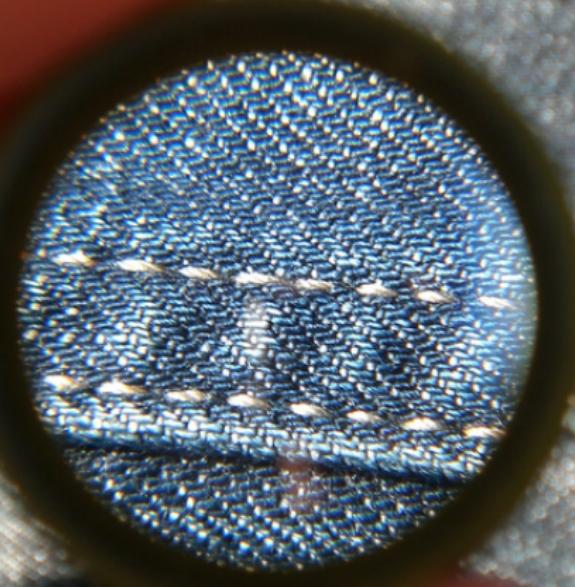
- Pas de contrôleur 1-Wire disponible de base sur Linux

*Solution software adaptée vu la lenteur du bus*

- Software pour émuler le bus 1-Wire

*Il suffit de choisir une pin GPIO comme Data*

# Comparaison



# Communication série

- **Échange d'informations** entre processeur et périphériques

*Différents ports et bus en fonction des propriétés désirées*

- **Trois principaux** types de communications utilisés

- Port série RS-232
- Universal Asynchronous Receiver/Transmitter (UART)
- Serial Peripheral Interface (SPI)

# Mise en réseau

- Mise en réseau de plusieurs systèmes embarqués

*Différent bus en fonction de la distance et niveau de réseautage*

- Trois principaux types de communications utilisés

- Inter-Integrated Circuit (I<sup>2</sup>C) (niveau board)
- Controller Area Network (CAN) (niveau système)
- Ethernet (LAN)

# Comparaison

Bus	Transmission	Sens	Synchronisation		Débit max
<b>RS-232</b>	série	duplex	asynchrone		de 75 bits/s à 115200 bit/s
<b>UART</b>	série	full-duplex	asynchrone	M/S	3686,4 kbit/s
<b>USB</b>	série				10 Gbit/s
<b>I<sup>2</sup>C</b>	série		synchrone	Ms/Ss	de 100 kbit/s à 5 Mbit/s
<b>SPI</b>	série	full-duplex	synchrone	M/Ss	10 Mbits/s
<b>1-Wire</b>		half-duplex	asynchrone	M/Ss	16.3 kbit/s

# Crédits

- <https://www.flickr.com/photos/30003006@N00/2439637326>
- <https://www.flickr.com/photos/17289090@N07/5817880561>
- [https://www.flickr.com/photos/emil\\_kabanov/6862588971](https://www.flickr.com/photos/emil_kabanov/6862588971)
- <https://www.flickr.com/photos/hoeken/3042677810>
- <https://www.flickr.com/photos/ncbob/7423723004>
- <https://www.flickr.com/photos/goingslo/4067593977>
- [https://www.flickr.com/photos/phm\\_sinan/1364979311](https://www.flickr.com/photos/phm_sinan/1364979311)