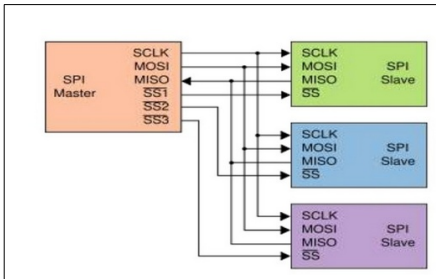

Mise en œuvre de la communication SPI STM32L432 ↔ RaspberryPi

Anthony Juton, janvier 2023

L'objet de ce guide est de mettre en œuvre la communication SPI entre le microcontrôleur STM32 et le nano-ordinateur Raspberry Pi.

On utilise ici l'environnement de développement gratuit, multiplateforme basé sur Eclipse STM32CubeIDE. On peut partir du projet avec les réglages par défaut pour la carte STM32L432KC.

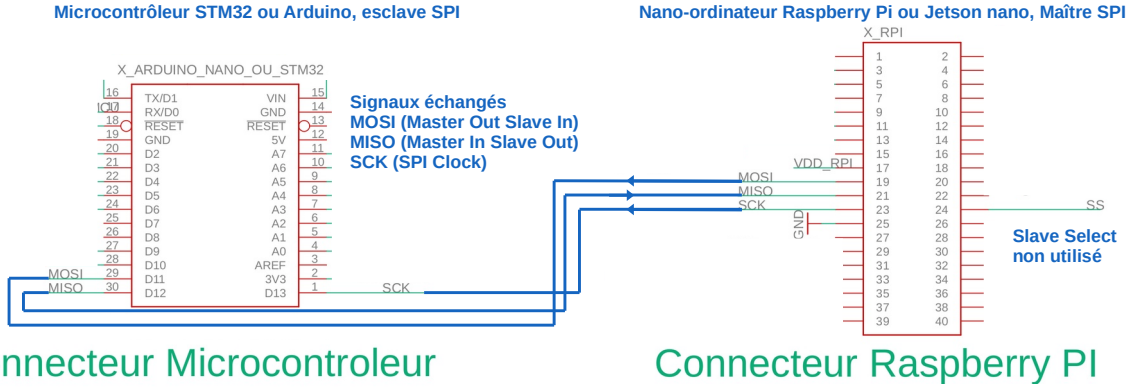

Figure 1: Connexions effectuées par la carte Hat entre la carte microcontrôleur et le nano-ordinateur

Le signal slave select n'est pas utile vu qu'il n'y a qu'un seul esclave, il n'a donc pas été câblé.

1 Caractéristiques des périphériques SPI

1.1 Périphérique SPI du microcontrôleur STM32L432

3.26 Serial peripheral interface (SPI)

Two SPI interfaces allow communication up to 40 Mbits/s in master and up to 24 Mbits/s slave modes, in half-duplex, full-duplex and simplex modes. The 3-bit prescaler gives 8 master mode frequencies and the frame size is configurable from 4 bits to 16 bits. The SPI interfaces support NSS pulse mode, TI mode and Hardware CRC calculation.

All SPI interfaces can be served by the DMA controller.

Figure 2: Extrait de la datasheet du microcontrôleur STM32L432

1.2 Périphérique SPI du nano-ordi. Raspberry Pi

On trouve des indications sur le périphérique SPI de la raspberry si le site officiel :

https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#serial-peripheral-interface-spi

Serial Peripheral Interface (SPI)

Edit this on GitHub

Raspberry Pi computers are equipped with a number of SPI buses. SPI can be used to connect a wide variety of peripherals - displays, network controllers (Ethernet, CAN bus), UARTs, etc. These devices are best supported by kernel device drivers, but the `spidev` API allows userspace drivers to be written in a wide array of languages.

SPI Hardware

Master modes

Signal name abbreviations

```
SCLK - Serial CLocK
CE   - Chip Enable (often called Chip Select)
MOSI - Master Out Slave In
MISO - Master In Slave Out
MOMI - Master Out Master In
```

Standard mode

In Standard SPI mode the peripheral implements the standard 3 wire serial protocol (SCLK, MOSI and MISO).

Transfer modes

- Polled
- Interrupt
- DMA

SPI Software

Linux driver

The default Linux driver is `spi-bcm2835`.

SPI0 is disabled by default. To enable it, use raspi-config, or ensure the line `dtparam=spi=on` is not commented out in `/boot/config.txt`. By default it uses 2 chip select lines, but this can be reduced to 1 using `dtoverlay=spi0-1cs`. `dtoverlay=spi0-2cs` also exists, and without any parameters it is equivalent to `dtparam=spi=on`.

Speed

The driver supports all speeds which are even integer divisors of the core clock, although as said above not all of these speeds will support data transfer due to limits in the GPIOs and in the devices attached. As a rule of thumb, anything over 50MHz is unlikely to work, but your mileage may vary.

Supported Mode bits

```
SPI_CPOL    - Clock polarity
SPI_CPHA    - Clock phase
SPI_CS_HIGH - Chip Select active high
SPI_NO_CS   - 1 device per bus, no Chip Select
SPI_3WIRE   - Bidirectional mode, data in and out pin shared
```

Using spidev from Python

There are several Python libraries that provide access to `spidev`, including `spidev` (`pip install spidev` - see https://pypi.org/project/spidev/) and `SPI-Py` (https://github.com/lthiery/SPI-Py).

Figure 3: Extraits de la documentation officielle de Raspberry

2 Au niveau du microcontrôleur STM32

2.1 Configuration du périphérique SPI3 de la STM32

Depuis le volet *Pinout View* et le volet de configuration *SPI3,* activer les entrées/sorties SPI du périphériques, en mode esclave sans signal NSS (Slave Select).

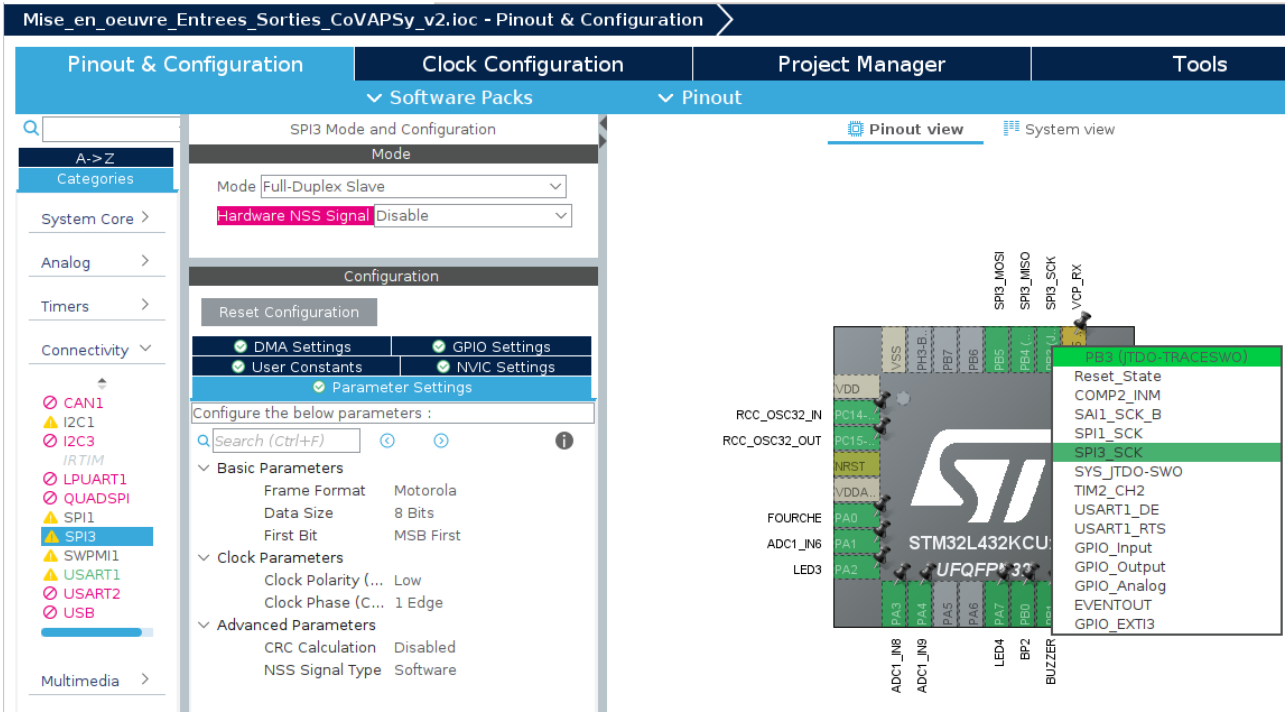

Figure 4: configuration des entrées TOR sous STM32CubeIDE

Il est possible de configurer les interruptions sous l'onglet NVIC

Une fois la configuration terminée, on génère le code associé : *Project* > *Generate Code*.

2.2 Programme

En SPI, l'esclave transmet des données lorsqu'il en reçoit. Ses données à transmettre doivent donc être prêtes à envoyer. Ici, 6 données on été choisies pour l'échange, avec 0xFF, 0xFF en début de trame.

Pour un premier programme simple, sans interruption, on définit un buffer de transmission et un buffer de réception.

```
/* USER CODE BEGIN 1 */
      uint8_t SPI_TxBuffer[6] = {0xFF,0xFF,0,0,0,0}; //buffer transmission
      uint8_t SPI_RxBuffer[6] ={};        // Buffer pour la réception SPI
  /* USER CODE END 1 */
```

Dans la boucle infinie, on attend les données (200 ms au maximum) et on les utilise pour commander une led. On modifie alors les données à transmettre.

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
```


```
  while (1)
  {
        if(HAL_SPI_TransmitReceive(&hspi3, (uint8_t *)SPI_TxBuffer, (uint8_t
*)SPI_RxBuffer, 6, 200) == HAL_OK)
        {
              //changement d'état de la led en fonction du 3ème octet reçu
              HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin,SPI_RxBuffer[3]%2);
              //incrémentation du 3ème octet des données à transmettre
              SPI_TxBuffer[3]++;
        }
```

3 Au niveau du nano-ordinateur Raspberry Pi4

3.1 Activation du périphérique SPI

Pour activer le périphérique SPI, cela peut se faire depuis la console (sudo raspi-config, comme pour l'i2c) ou depuis l'interface graphique de configuration :
Menu *Preferences > Raspberry Pi Configuration*.

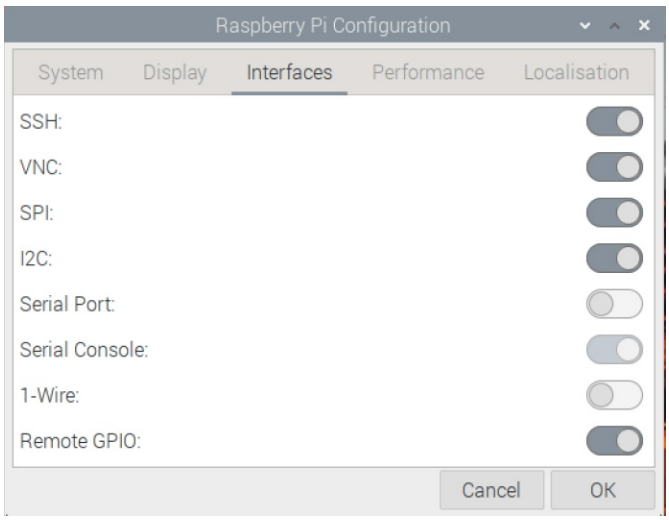

Figure 5: Activation du périphérique SPI depuis la fenêtre de configuration

Il faut ensuite redémarrer.

On peut ensuite vérifier que le périphérique SPI est disponible sur la raspberry, depuis la console :

```
$ ls /dev/*spi*
```

```
voitureenfant1@voitureenfant1:~ $ ls /dev/*spi*
/dev/spidev0.0  /dev/spidev0.1
```

3.2 Programme python

Le programme suivant envoie périodiquement 6 octets à la carte STM32 :


```
import time
import spidev

#utilisation de SPI0 avec un seul esclave
#device n'a pas d'importance, le signal SS n'est pas câblé
bus = 0
device = 1

# Activation de la communication SPI
spi = spidev.SpiDev()
spi.open(bus, 1)

# Configuration de la vitesse (1 Mbit/s) et du mode
spi.max_speed_hz = 1000000
spi.mode = 0
message_tx = [0x55,0x55,0,2,4,6]

while True:

    # envoi et réception simultanée de messages
    message_tx[3]+=1
    message_rx = spi.xfer(message_tx)
    print(message_rx)
    time.sleep(0.100) #attente 20 ms
```

3.3 Visualisation des échanges

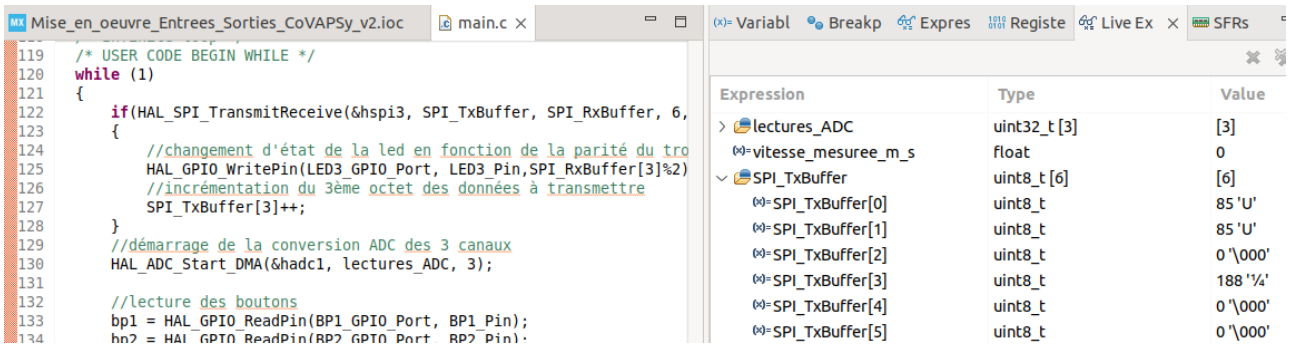

Figure 6: Surveillance du buffer de réception du STM32 via les Live Expressions

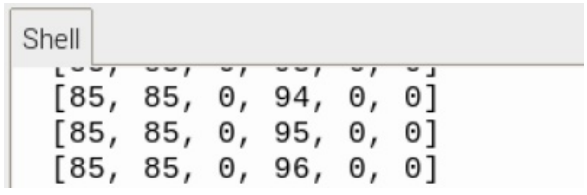

Figure 7: Affichage du buffer de réception de la raspberry Pi

4 Amélioration du programme

Pour faire mieux, on peut utiliser les interruptions sur réception des données.