BÀI 4: GIAO TIẾP NỐI TIẾP

1. MỤC TIÊU BÀI TN

Trong bài TN này SV sẽ thực hành những phần sau:

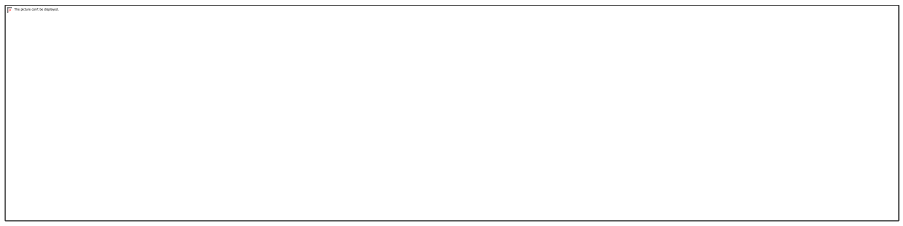
- Viết chương trình điều khiển các modun giao tiếp nối tiếp SPI, I2C, UART
- Kết nối vật lý raspberry với các ic ngoại vi như: Arduino, DAC MCP4921, modun thời gian thực DS3231

2. GIAO TIẾP SPI

SPI (Serial Peripheral Bus) là một chuẩn giao tiếp nối tiếp tốc độ cao do hãng Motorola phát triển trong những năm 1980. Trong giao tiếp SPI, luôn có một thiết bị được đặt là Master và điều phối toàn bộ quá trình tuyền nhận tín hiệu giữa Master và các Slave. SPI đôi khi được gọi là chuẩn truyền thông "4 dây" vì có 4 đường giao tiếp trong chuẩn này đó là

- SCK (Serial Clock),
- MISO (Master Input Slave Output),
- MOSI (Master Ouput Slave Input)
- CS (Slave Select).

Raspberry được tích hợp 2 bộ SPI. Driver cho modun SPI được cài đặt sẵn trong Raspbian và có thể kiểm tra bằng cách gõ câu lệnh **ls –l /dev** trên terminal, sau đó tìm node **spidev0.0**. Các đường giao tiếp của bộ SPI có thể kết nối trên header tại các chân:



2.1 GIAO TIẾP SPI TRÊN RASPBERRY VỚI THƯ VIỆN WRINGPI

Tham khảo thêm tại website: http://wiringpi.com/reference/spi-library/

WiringPi tích hợp thư viện cho modun SPI. Để thiết lập và sử dụng modun SPI, thực hiện theo các bước sau:

Bước 1: tải SPI driver vào kernel với câu lệnh

gpio load spi

Bước 2: Trong file code C, include file header của thư viện

#include <wiringPiSPI.h>

Bước 3: Cấu hình giao tiếp SPI với:

wiringPiSPISetup (int channel, int speed)

Bước 4: Truyền nhận data bằng câu lệnh:

wiringPiSPIDataRW (int channel, unsigned char *data, int len);

2.2 Bài tập thực hành: Kết nối SPI sử dụng thư viện WiringPi (Raspberry Pi (master) Arduino Uno (slave))

Trong hướng bài thực hành này, chúng ta sẽ thực hành thiết lập giao tiếp giữa Raspberry Pi và Arduino (Uno), sử dụng giao thức SPI.

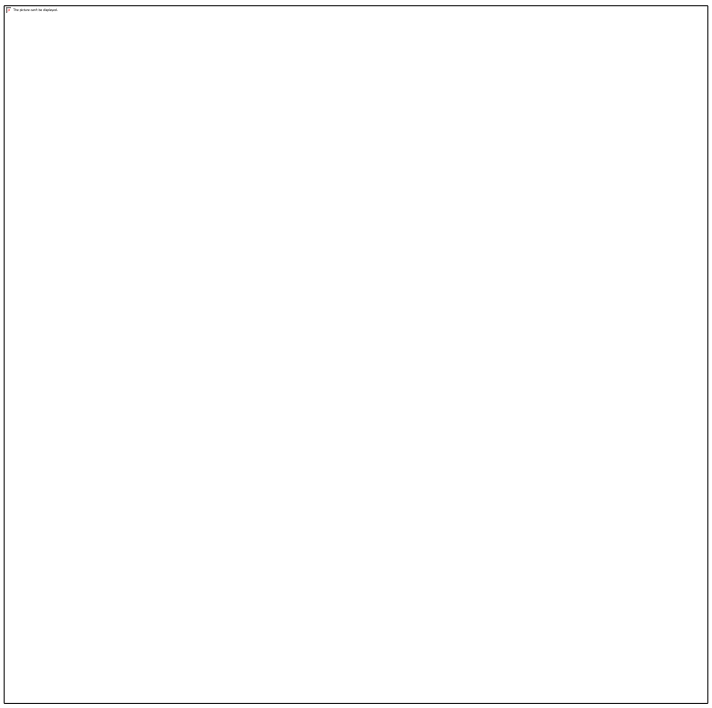
Raspberry Pi đóng vai trò thiết bị Master và Arduino là Slave. Trên Raspberry Pi, chúng ta sẽ sử dụng thư viện WiringPi, cụ thể thể là các hàm WiringPiSPI trong thư viện.

Mục tiêu của bài thực hành này là gửi một byte từ Raspberry Pi tới Arduino, xử lý byte này và nhận giá trị mới trên Raspberry Pi.

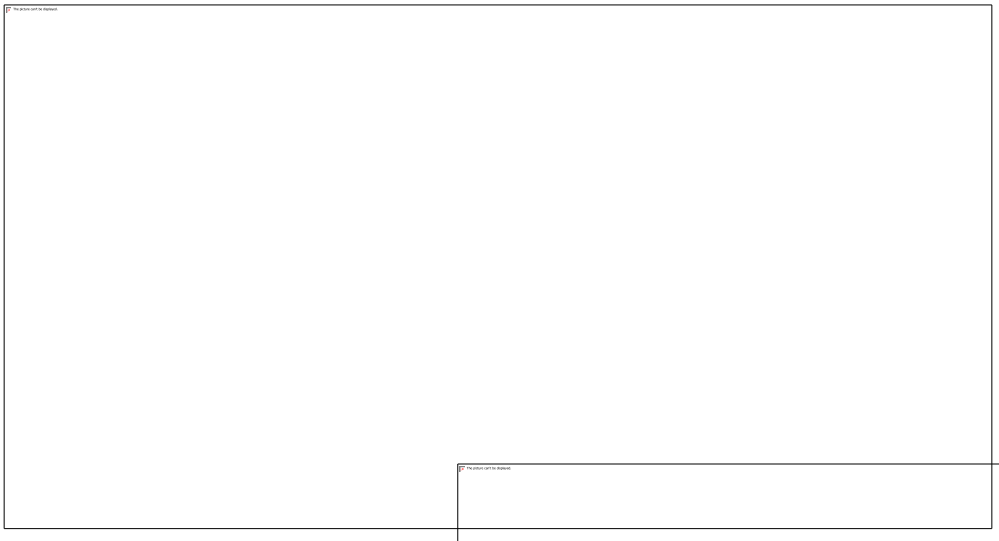
2.2.1 Thiết lập phần cứng

Nối các chân MISO với MISO và MOSI với MOSI, **không phải MISO với MOSI**. Khi viết code chúng ta sẽ xử lý các chân MISO và MOSI tùy thuộc vào thiết bị được đặt làm Master hay Slave.

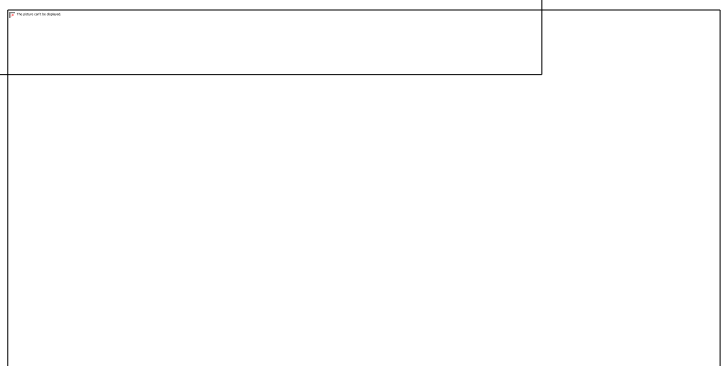
Lưu ý rằng đối với SPI, ta thường có một dây tín hiệu được kết nối với CS (Chip Select) hoặc SS (Slave Select). Điều này rất hữu ích để chọn Slave đang cần truyền/nhận. Ở đây, vì chỉ có một Slave là Arduino, nên không cần dây này, giao tiếp vẫn hoạt động.



Ngoài ra, cần chú ý: **Raspberry Pi hoạt động ở 3.3V, trong khi Arduino Uno hoạt động ở 5V**. Đối với ví dụ này, kết nối trực tiếp như hình vẽ có thể chấp nhận được. Nhưng với những ứng dụng phức tạp hơn, chúng ta **phải sử dụng bộ chuyển đổi mức logic (3.3V - 5V)** để tránh chập cháy.



Mạch Chuyển Đổi Logic 3.3V-5.0V 4 Kênh



2.2.2 Thiết lập phần mềm

Với Arduino (Slave)

Sử dụng đoạn code cơ bản sau để thiết lập bo Arduino như một SPI slave và xử lý dữ liệu nhận được qua bus SPI.

```
#include <SPI.h>

void setup() {
  // have to send on master in, *slave out*
  pinMode(MISO, OUTPUT);

  // turn on SPI in slave mode
  SPCR |= _BV(SPE);

  // turn on interrupts
  SPI.attachInterrupt();
}

// SPI interrupt routine
ISR (SPI_STC_vect)
{
  byte c = SPDR;

  SPDR = c+10;
}  // end of interrupt service routine (ISR) for SPI

void loop () { }
```

Tải code này lên Arduino để thiết lập một SPI Slave.

Khi một byte được nhận qua SPI, Arduino sẽ đọc nó từ SPDR. Sau khi xử lý nó (cộn thêm 10), Arduino sẽ đặt thanh ghi SPDR với giá trị mới, do đó, master có thể đọc nó trong lần chuyển SPI tiếp theo.

Với Raspberry (Master)

```
#include <iostream>
#include <wiringPiSPI.h>

#define SPI_CHANNEL 0
#define SPI_CLOCK_SPEED 1000000

int main(int argc, char **argv)
```

```
{
    int fd = wiringPiSPISetupMode(SPI_CHANNEL, SPI_CLOCK_SPEED, 0);
    if (fd == -1) {
        std::cout << "Failed to init SPI communication.\n";
        return -1;
    }
    std::cout << "SPI communication successfully setup.\n";

    unsigned char buf[2] = { 23, 0 };
    wiringPiSPIDataRW(SPI_CHANNEL, buf, 2);

    std::cout << "Data returned: " << +buf[1] << "\n";
    return 0;
}
```

```
#include <iostream>
#include <wiringPiSPI.h>
```

Đầu tiên, thêm thư viện WiringPiSPI, là một phần của thư viện WiringPi.

```
#define SPI_CHANNEL 0
#define SPI_CLOCK_SPEED 1000000
```

Có 4 chế độ SPI khác nhau: 0, 1, 2 và 3 – khác nhau cơ bản về cực và pha của tín hiệu. Chú ý nếu Slave sử dụng chế độ 2, thì Master cần phải định cấu hình SPI với chế độ 2, để chúng có thể được “đồng bộ hóa”. Ở đây chúng ta đang sử dụng chế độ 0, đây là chế độ mặc định trên Arduino.