Bài 5: SPI Software & SPI Hardware

# 1. SPI Software

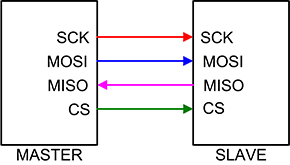
Trên mỗi dòng vi điều khiển khác nhau module SPI sẽ được tích hợp, điều khiển bởi các thanh ghi,phần cứng, IO khác nhau, đấy gọi là SPI cứng (*hardware SPI*). Như vậy bản chất chuẩn truyền thông SPI giống nhau trên mỗi chip nhưng lại được cài đặt và sử dụng không giống nhau. Điều này gây thêm khó khăn cho người sử dụng khi bạn bắt đầu tìm hiểu một dòng vi điều khiển mới, bạn sẽ phải nhớ các chân MISO, SS, MOSI, SCK mỗi chip khác nhau, nhớ các thanh ghi, các chế độ hoạt động và cách cài đặt trên các dòng vi điều khiển khác nhau.

Để khắc phục nhược điểm trên, có 1 cách lập trình giả lập SPI cứng đó là “SPI mềm”. Thực chất SPI mềm là cách “mô phỏng” bằng cách tạo ra một giao thức truyền thông giống SPI nhưng chỉ sử dụng các cổng vào ra của vi điều khiển. Như vậy chỉ với việc điều khiển GPIO của chip hoàn toàn có thể thực hiện giao thức SPI.

Điều này cũng có nghĩa là có thể sử dụng giao thức này trên bất kì vi điều khiển nào mà không cần phải nhớ thanh ghi hay các chân phần cứng.

## 1.1 Cấu hình GPIO cho SPI mềm.

SPI dùng 4 chân để truyền nhận, gồm MISO, MOSI, CS và SCK.



* MISO: (Master In Slave Out) Chân nhận tín hiệu của Master nối với chân truyền của Slave, vì vậy được cấu hình Input ở Master và Output ở Slave.
* MOSI: (Master Out SLave In) Ngược lại với MISO, cấu hình Input cho Slave và Output cho Master.
* SCK: (Clock) Chân truyền tín hiệu xung đồng bộ từ Master cho Slave, được cấu hình Output cho Master và Input cho Slave.
* CS: (Chip Select) Chân gửi tín hiệu chọn Slave của Master tới các Slave. Cấu hình Output cho Master và Input cho Slave. Có thể có nhiều CS để Master điều khiển nhiều Slave.

Bước đầu, ta định nghĩa cho 4 chân sử dụng cho SPI:

#define SPI\_MISO\_Pin GPIO\_Pin\_1

#define SPI\_MOSI\_Pin GPIO\_Pin\_2

#define SPI\_CS\_Pin GPIO\_Pin\_3

#define SPI\_GPIO GPIOA

#define SPI\_RCC RCC\_APB2Periph\_GPIOA

Tạo hàm khởi tạo cho SPI:

void SPI\_Init(){

GPIO\_WriteBit(SPI\_PORT, SPI\_SCK, 0);

GPIO\_WriteBit(SPI\_PORT, SPI\_CS, 1);

GPIO\_WriteBit(SPI\_PORT, SPI\_MISO, 0);

GPIO\_WriteBit(SPI\_PORT, SPI\_MOSI, 0);

}

Hàm SPI\_Init khởi tạo các giá trị cho các chân khi chưa có sự kiện truyền nhận.

* SCK = 0: Clock sẽ ở mức thấp, tùy vào cấu hình CPHA, CPOL mà giá trị này có thể sẽ khác. Ở đây ta mô phỏng đơn giản, khi clock ở mức 1 sẽ lấu mẫu dữ liệu, mức 0 sẽ là chế độ nghỉ.
* CS = 1: Khi Master chưa truyền nhận, CS ở mức 1 cho tất cả các Slave.
* Các chân MISO, MOSI được kéo về 0 ở chế độ nghỉ.

## 1.2 SPI software code cho master

Clock sẽ được tạo bằng hàm delay kết hợp với set/reset tín hiệu trên đường SCK:

void Clock()

{

GPIO\_WriteBit(SPI\_PORT, SPI\_SCK, 1);

delay(1000);

GPIO\_WriteBit(SPI\_PORT, SPI\_SCK, 0);

delay(1000);

}

Mỗi khi hàm Clock được gọi, sẽ tạo 1 xung vuông, tín hiệu sẽ được truyền nhận theo clock này.



| Clock(); | | Clock(); | | Clock(); | |
| --- | --- | --- | --- | --- | --- |
| Truyền |  | truyền |  | truyền |  |

Hàm truyền ở master:

void SPI\_Master\_Transmit(uint8\_t u8Data){

uint8\_t u8Mask = 0x80;

uint8\_t tempData;

GPIO\_WriteBit(SPI\_GPIO, SPI\_CS\_Pin, Bit\_RESET);

delay(10);

for(int i=0; i<8; i++){

tempData = u8Data & u8Mask;

if(tempData){

GPIO\_WriteBit(SPI\_GPIO, SPI\_MOSI\_Pin, Bit\_SET);

delay(10);

} else{

GPIO\_WriteBit(SPI\_GPIO, SPI\_MOSI\_Pin, Bit\_RESET);

delay(10);

}

u8Data<<=1;

Clock();

}

GPIO\_WriteBit(SPI\_GPIO, SPI\_CS\_Pin, Bit\_SET);

delay(10);

}

Mô phỏng quá trình truyền nhận:

while(1)

{

for(int i=0; i<7; i++){

SPI\_Master\_Transmit(DataTrans[i]);

delay(5000000);

}

}

Các byte trong 1 chuỗi DataTrans sẽ được truyền lần lượ bằng hàm SPI\_Master\_Transmit.

uint8\_t SPI\_SlaveReceive()

{

uint8\_t i = 0; uint8\_t dataReceive=0x00;bool st=0;

while (digitalRead(SS));

while (!digitalRead(SCK));

for(i=0; i<8;i++)

{ if(digitalRead(SCK)){

while (digitalRead(SCK)) st = digitalRead(MISO);

dataReceive=dataReceive<<1;

dataReceive=dataReceive|st;

}

while (!digitalRead(SCK));

}

return dataReceive;

}

///in main loop

void loop(){

if(!(digitalRead(SS))){

dat=SPI\_SlaveReceive();

Serial.println(dat,DEC);

}

}

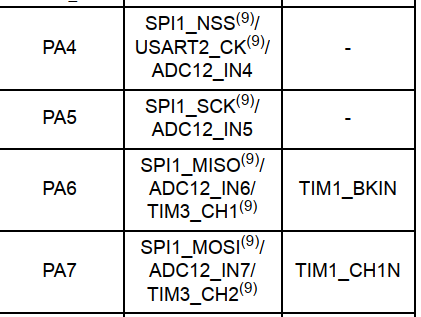
Ở slave sẽ dựa vào tín hiệu CS và SCK để xác định lệnh truyền nhận từ Master cũng như thời diểm đọc dữ liệu.

# 2. SPI Hardware

STM32F1 có 2 khối SPI, SPI1 ở APB2 và SPI2 ở PAB1. Các khối này được xây dựng các kết nối, driver và các hàm riêng trong bộ thư viện chuẩn.

## 2.1 Cấu hình GPIO cho SPI.

STM32 cấu hình sẵn các chân dành cho chức năng SPI.



Ở đây, ví dụ sử dụng SPI1, các chân đã được thiết lập sẵn. Ta sẽ cấu hình cho các chân này. Đầu tiên define các chân với các tên riêng biệt:

#define SPI1\_NSS GPIO\_Pin\_4

#define SPI1\_SCK GPIO\_Pin\_5

#define SPI1\_MISO GPIO\_Pin\_6

#define SPI1\_MOSI GPIO\_Pin\_7

#define SPI1\_GPIO GPIOA

Hàm GPIO config cấu hình các chân của SPI theo từng mode khác nhau:

void GPIO\_Cofig(){

GPIO\_InitTypeDef GPIO\_InitStructure;

GPIO\_InitStructure.GPIO\_Pin = SPI1\_NSS| SPI1\_SCK| SPI1\_MISO| SPI1\_MOSI;

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AF\_PP;

GPIO\_Init(SPI1\_GPIO, &GPIO\_InitStructure);

}

## 2.2 Cấu hình SPI.

Tương tự các ngoại vi khác, các tham số SPI được cấu hình trong Struct SPI\_InitTypeDef:

* SPI\_Mode: Quy định chế độ hoạt động của thiết bị SPI.
* SPI\_Direction: Quy định kiểu truyền của thiết bị.
* SPI\_BaudRatePrescaler: Hệ số chia clock cấp cho Module SPI.
* SPI\_CPOL: Cấu hình cực tính của SCK . Có 2 chế độ:
  + SPI\_CPOL\_Low: Cực tính mức 0 khi SCK không truyền xung.
  + SPI\_CPOL\_High: Cực tính mức 1 khi SCK không truyền xung.
* SPI\_CPHA: Cấu hình chế độ hoạt động của SCK. Có 2 chế độ:
  + SPI\_CPHA\_1Edge: Tín hiệu truyền đi ở cạnh xung đầu tiên.
  + SPI\_CPHA\_2Edge: Tín hiệu truyền đi ở cạnh xung thứ hai.
* SPI\_DataSize: Cấu hình số bit truyền. 8 hoặc 16 bit.
* SPI\_FirstBit: Cấu hình chiều truyền của các bit là MSB hay LSB.
* SPI\_CRCPolynomial: Cấu hình số bit CheckSum cho SPI.
* SPI\_NSS: Cấu hình chân SS là điều khiển bằng thiết bị hay phần mềm.

Hàm SPI\_Config() cấu hình tham số có SPI:

void SPI\_Config(){

SPI\_InitTypeDef SPI\_InitStructure;

SPI\_InitStructure.SPI\_BaudRatePrescaler = SPI\_BaudRatePrescaler\_32;

SPI\_InitStructure.SPI\_CPHA = SPI\_CPHA\_2Edge;

SPI\_InitStructure.SPI\_CPOL = SPI\_CPOL\_Low;

SPI\_InitStructure.SPI\_DataSize = SPI\_DataSize\_8b;

SPI\_InitStructure.SPI\_Direction = SPI\_Direction\_2Lines\_FullDuplex;

SPI\_InitStructure.SPI\_FirstBit = SPI\_FirstBit\_MSB;

SPI\_InitStructure.SPI\_Mode = SPI\_Mode\_Slave;

SPI\_InitStructure.SPI\_NSS = SPI\_NSS\_Soft;

SPI\_Init(SPI1, &SPI\_InitStructure);

SPI\_Cmd(SPI1, ENABLE);

}

* Hàm **SPI\_I2S\_SendData(SPI\_TypeDef\* SPIx, uint16\_t Data)**, tùy vào cấu hình datasize là 8 hay 16 bit sẽ truyền đi 8 hoặc 16 bit dữ liệu. Hàm nhận 2 tham số là bộ SPI sử dụng và data cần truyền.
* Hàm **SPI\_I2S\_ReceiveData(SPI\_TypeDef\* SPIx)** trả về giá trị đọc được trên SPIx. Hàm trả về 8 hoặc 16 bit data.
* Hàm **SPI\_I2S\_GetFlagStatus(SPI\_TypeDef\* SPIx, uint16\_t SPI\_I2S\_FLAG)** trả về giá trị 1 cờ trong thanh ghi của SPI. Các cờ thường được dùng:
  + **SPI\_I2S\_FLAG\_TXE**: Cờ báo truyền, cờ này sẽ set lên 1 khi truyền xong data trong buffer.
  + **SPI\_I2S\_FLAG\_RXNE**: Cờ báo nhận, cờ này set lên 1 khi nhận xong data.
  + **SPI\_I2S\_FLAG\_BSY**: Cờ báo bận,set lên 1 khi SPI đang bận truyền nhận.

Các hàm truyền nhận có thể viết nhứ sau:

\*Vì cấu hình NSS soft nên khi truyền, ta phải chủ động kéo SS xuống Low bằng phần mềm:

void SPI\_Send1Byte(uint8\_t data){

GPIO\_WriteBit(SPI1\_GPIO, SPI1\_NSS, Bit\_RESET);

SPI\_I2S\_SendData(SPI1, data);

while(SPI\_I2S\_GetFlagStatus(SPI1, SPI\_I2S\_FLAG\_TXE)==0);

GPIO\_WriteBit(SPI1\_GPIO, SPI1\_NSS, Bit\_SET);

}

uint8\_t SPI\_Receive1Byte(void){

uint8\_t temp;

while(SPI\_I2S\_GetFlagStatus(SPI1, SPI\_I2S\_FLAG\_BSY)==1);

temp = (uint8\_t)SPI\_I2S\_ReceiveData(SPI1);

while(SPI\_I2S\_GetFlagStatus(SPI1, SPI\_I2S\_FLAG\_RXNE)==0);

return temp;

}