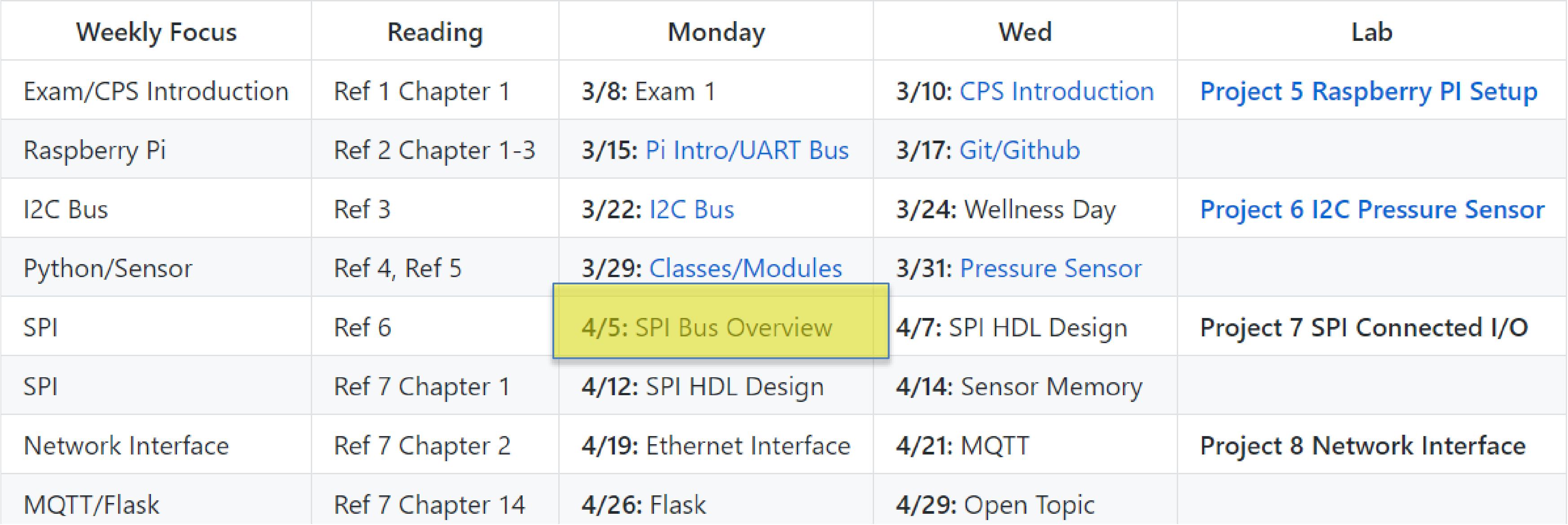
E210 Engineering Cyber-Physical Systems (Spring 2021)

Serial Peripheral Interface (SPI)

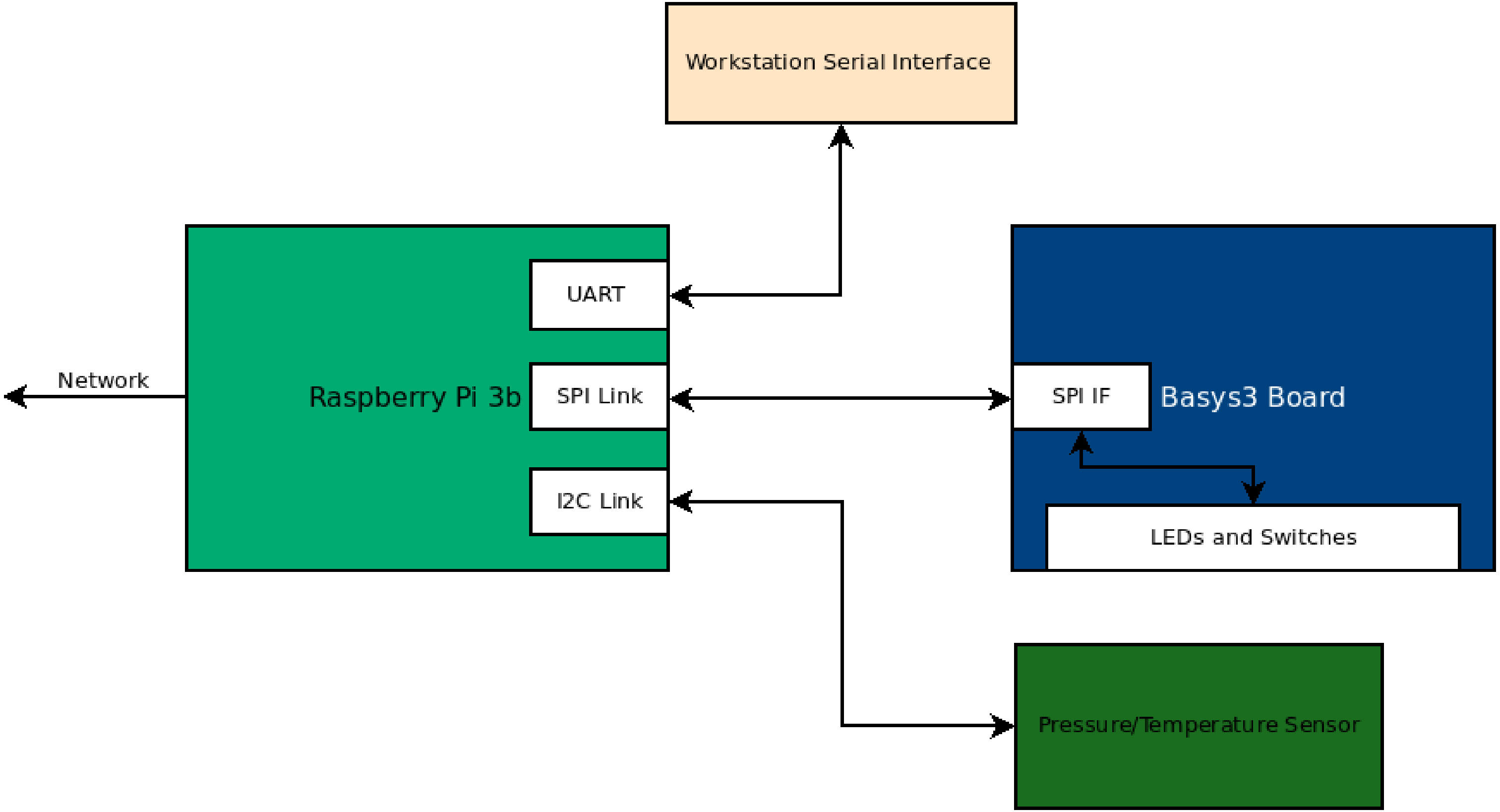
Bryce Himebaugh



Weekly Focus	Reading	Monday	Wed	Lab
Exam/CPS Introduction	Ref 1 Chapter 1	**3/8:** Exam 1	**3/10:** CPS Introduction	**Project 5 Raspberry PI Setup**
Raspberry Pi	Ref 2 Chapter 1-3	**3/15:** Pi Intro/UART Bus	**3/17:** Git/Github	
I2C Bus	Ref 3	**3/22:** I2C Bus	**3/24:** Wellness Day	**Project 6 I2C Pressure Sensor**
Python/Sensor	Ref 4, Ref 5	**3/29:** Classes/Modules	**3/31:** Pressure Sensor	
SPI	Ref 6	**4/5:** SPI Bus Overview	**4/7:** SPI HDL Design	**Project 7 SPI Connected I/O**
SPI	Ref 7 Chapter 1	**4/12:** SPI HDL Design	**4/14:** Sensor Memory	
Network Interface	Ref 7 Chapter 2	**4/19:** Ethernet Interface	**4/21:** MQTT	**Project 8 Network Interface**
MQTT/Flask	Ref 7 Chapter 14	**4/26:** Flask	**4/29:** Open Topic	

https://engr210.github.io/

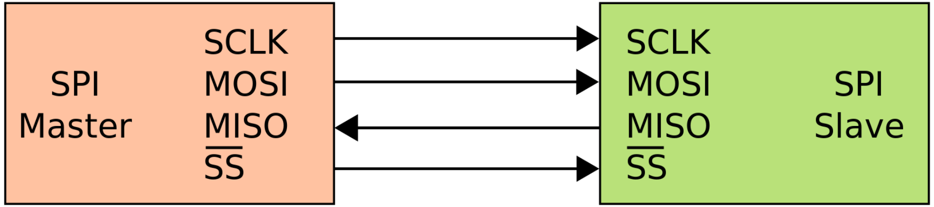
Raspberry SPI Link



SPI Overview

What is I2C?

1. Synchronous Serial Link
2. 4 Wire Bus
3. Devices Selected with Chip Select Pin
4. Full Duplex
5. Only One Bus Controller



History of I2C

- Developed by Motorola in mid-80s
- Short distance communication for embedded systems
- De facto standard
- Significantly faster than I2C or UART communication
  - Max Speeds Typically 20Mbps+ (no minimum speed)
- Used in SD Cards and Embedded LCD displays

Category	SPI	I2C	UART
Clock	Synchronous	Synchronous	Asynchronous
Speed	20 Mbps +	400K Baud (5M Baud Max)	115K Baud
Transmission Mode	Full Duplex	Half Duplex	Full Duplex
Number of Devices	Only Limited by CS Pins	112 (1024 possible)	2
Number of Pins	3 + CS	2 (Data, Clock)	2* (Rx, Tx), Optional (CTS,RTS)
Baud Rate Accuracy Requirement	N/A	N/A	~3% Baud Rate Accuracy
Development Complexity	Low	Med	Low

Device Naming Conventions

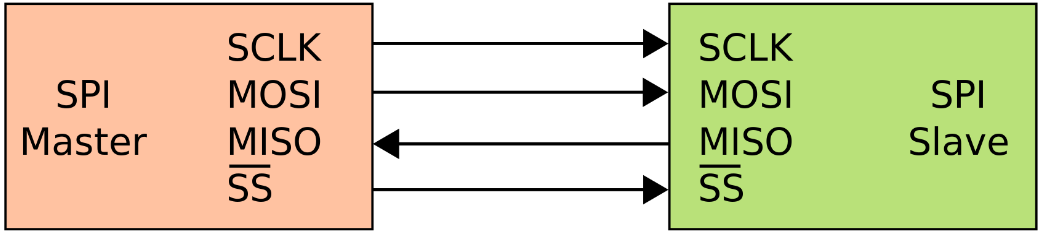
Open Source Hardware Association

New Name	Old Name
SDO – Serial Data Out	MOSI – Master Out Slave In
SDI – Serial Data In	MISO – Master In Slave Out
CS – Chip Select	SS – Slave Select
SCLK - Clock	SCLK – Clock

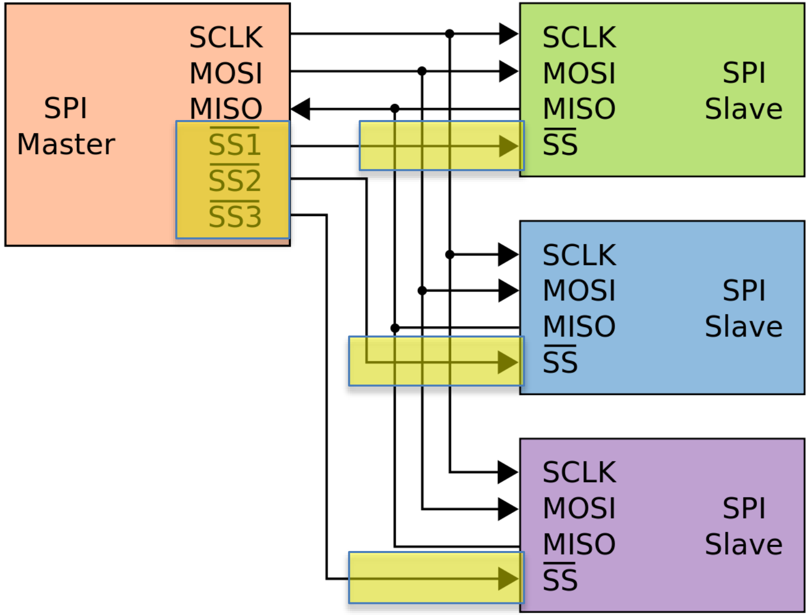
https://www.oshwa.org/a-resolution-to-redefine-spi-signal-names

Bus Connections

Single Peripheral

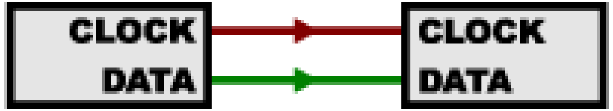


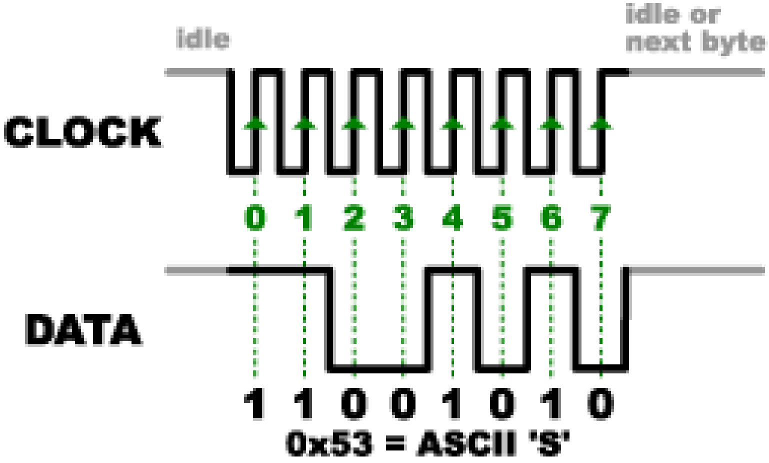
Multiple Independent Peripherals



Bus Protocol

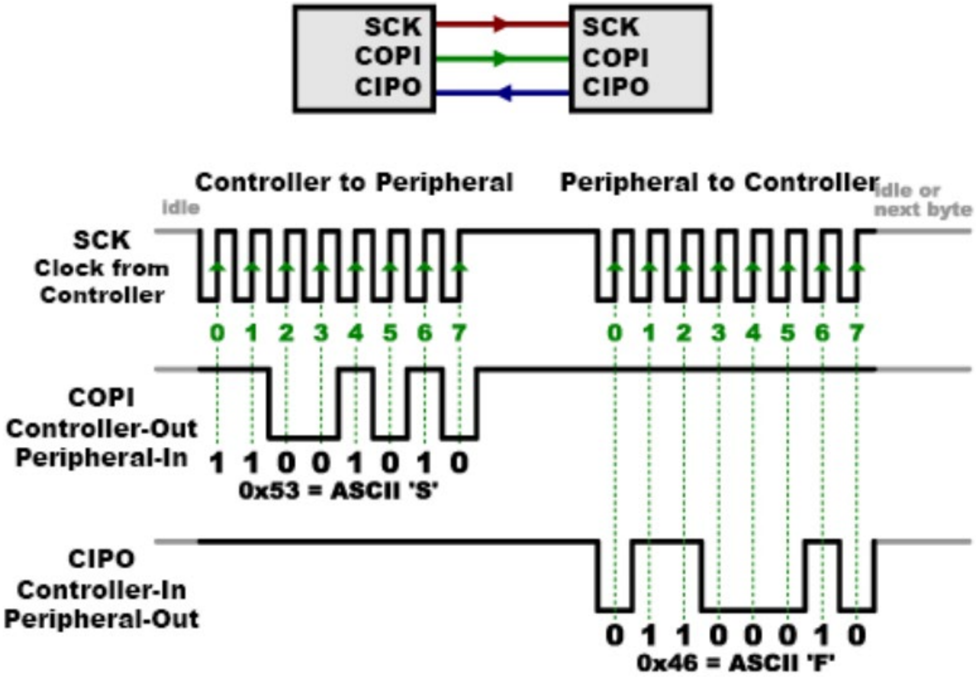
Sending Data





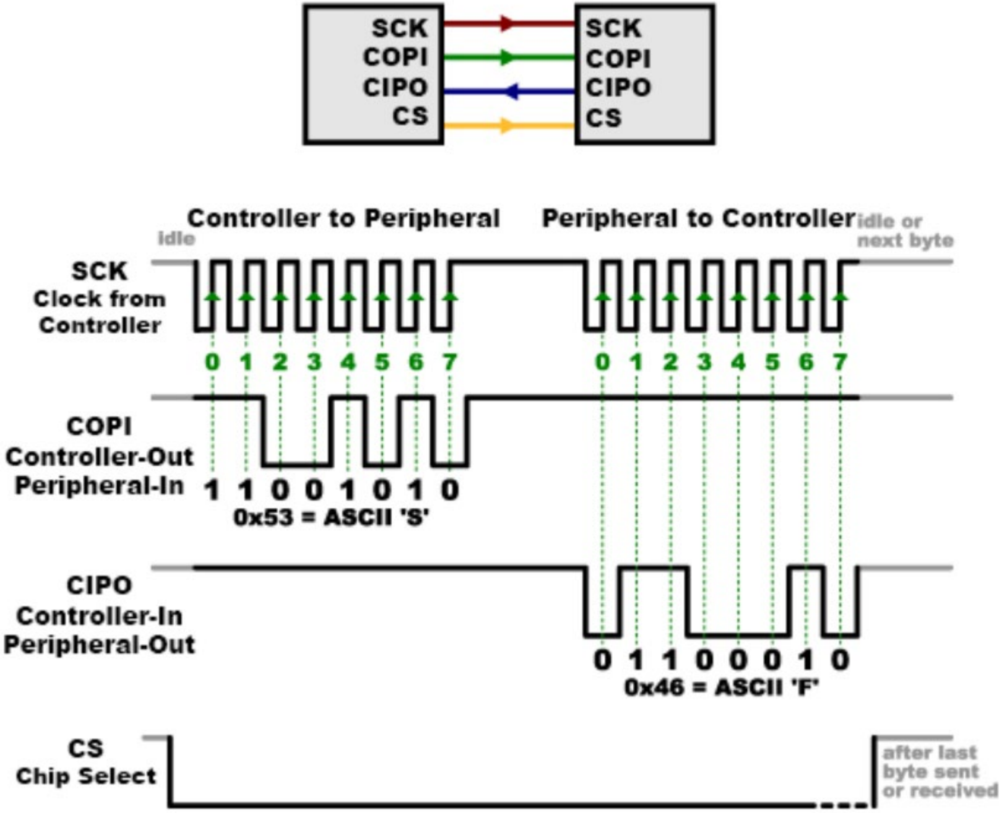
https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all

Receiving Data

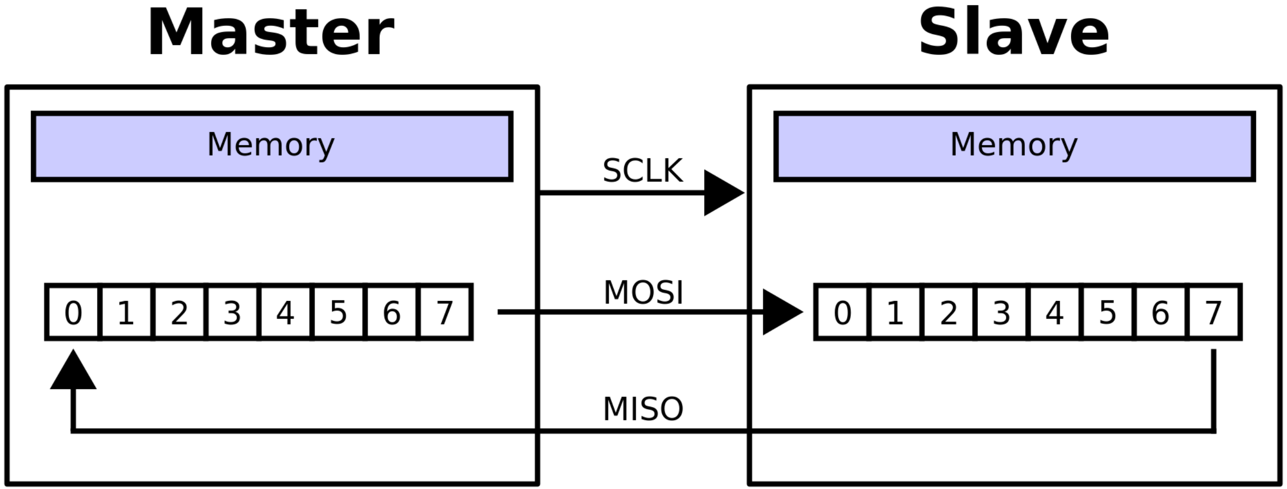


https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all

Chip Select


SCK
COPI
CIPO
CS
SCK
COPI
CIPO
CS
Controller to Peripheral
Peripheral to Controller
idle
idle or next byte
SCK Clock from Controller
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
COPI Controller-Out Peripheral-In
1 1 0 0 1 0 1 0
0x53 = ASCII 'S'
CIPO Controller-In Peripheral-Out
0 1 1 0 0 0 1 0
0x46 = ASCII 'F'
CS Chip Select
after last byte sent or received

16-bit Shift Register (split between two chips)



Swaps Registers

Waveform

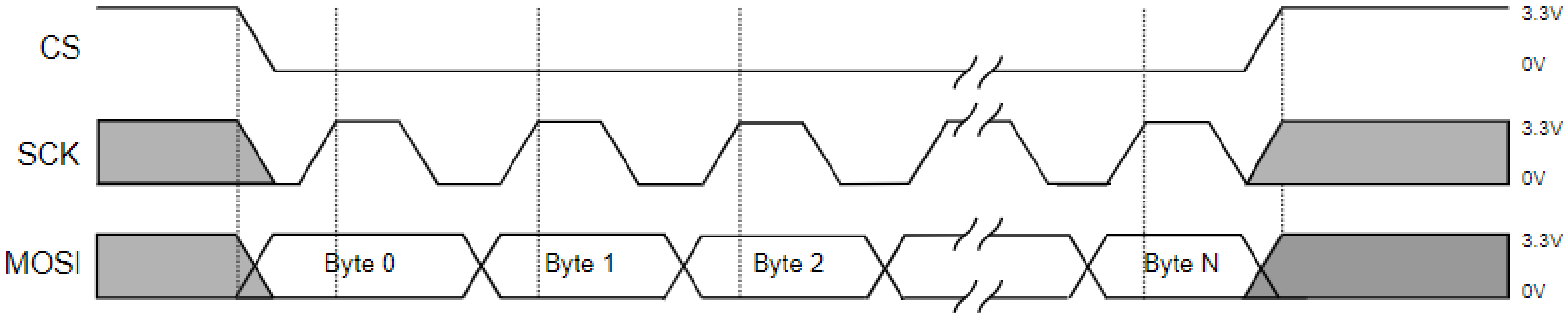
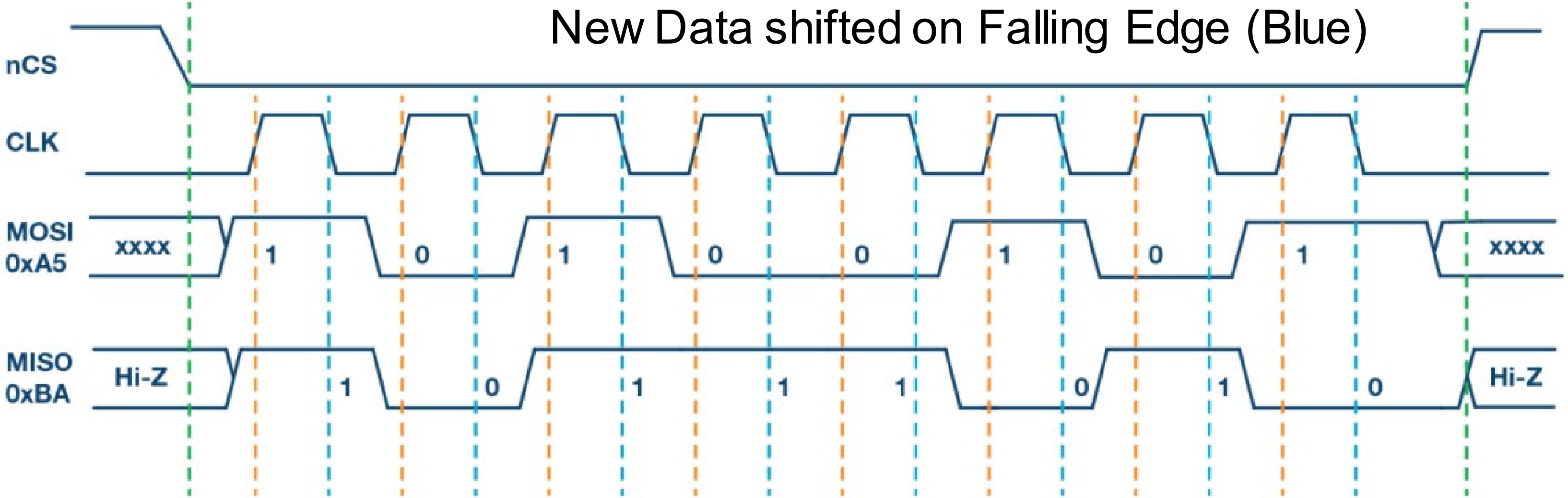
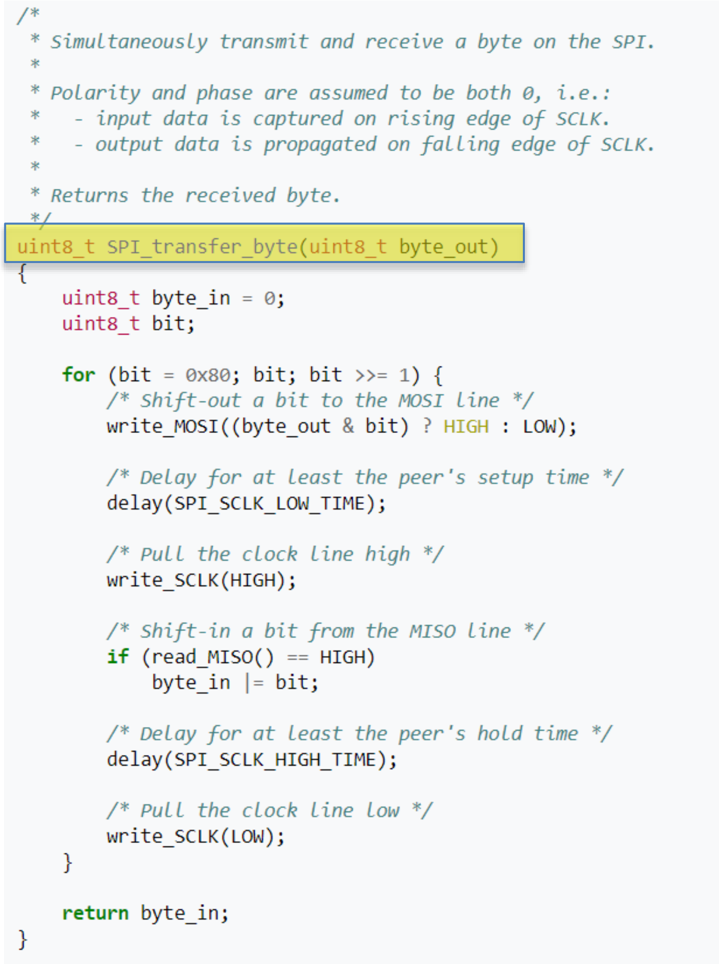


Figure 1. Example of a SPI transmission

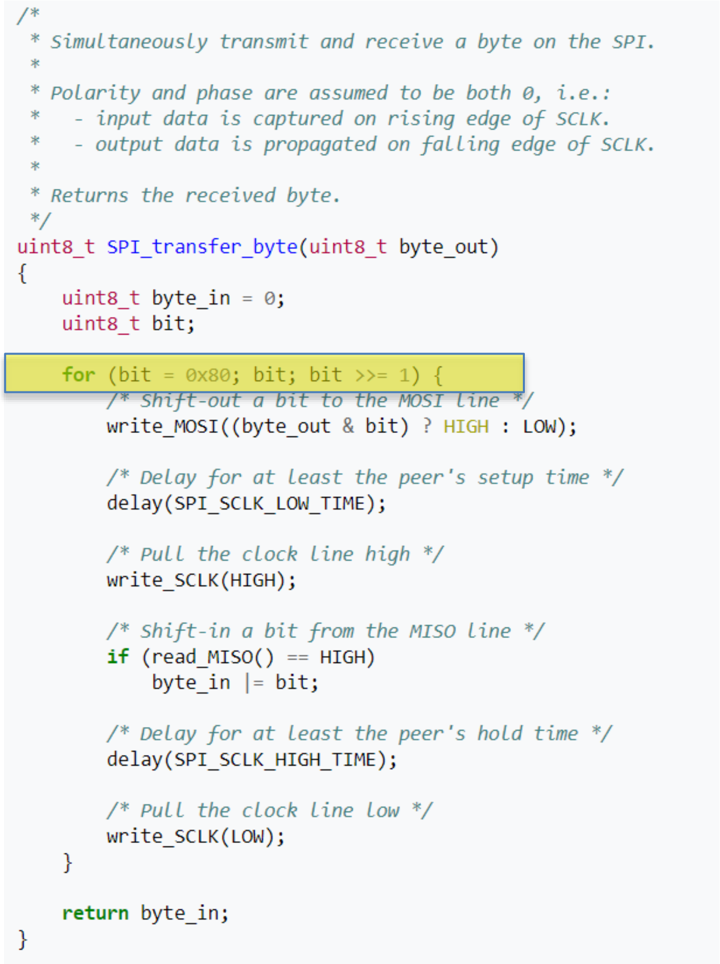
Waveform

Data sampled on Rising Edge (Orange)
New Data shifted on Falling Edge (Blue)

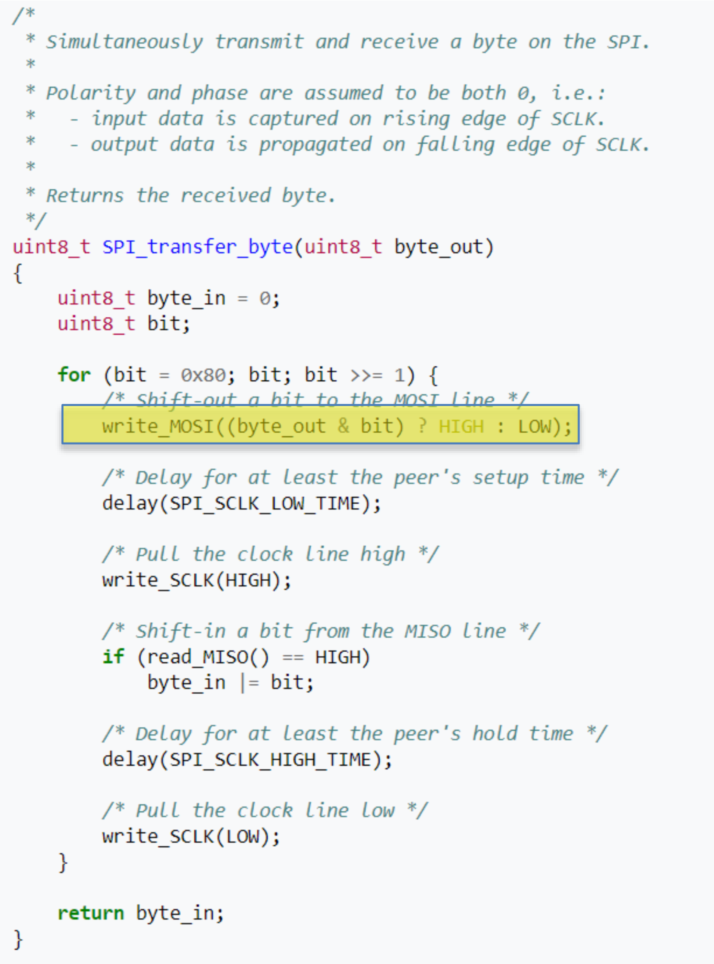




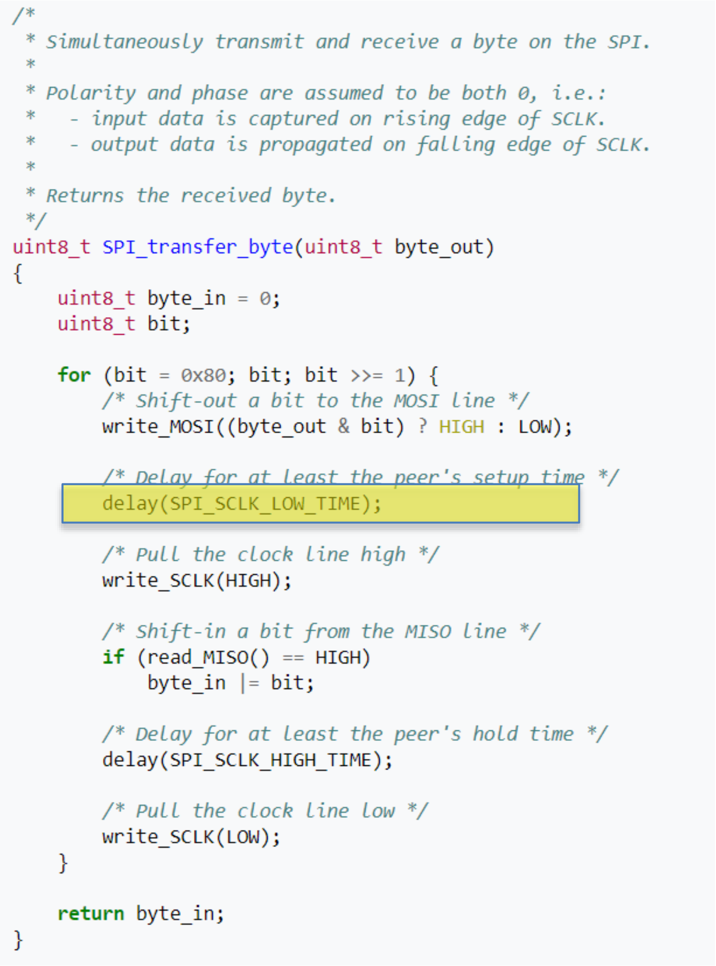
```
/*
 * Simultaneously transmit and receive a byte on the SPI.
 *
 * Polarity and phase are assumed to be both 0, i.e.:
 *   - input data is captured on rising edge of SCLK.
 *   - output data is propagated on falling edge of SCLK.
 *
 * Returns the received byte.
 */
uint8_t SPI_transfer_byte(uint8_t byte_out)
{
    uint8_t byte_in = 0;
    uint8_t bit;

    for (bit = 0x80; bit; bit >>= 1) {
        /* Shift-out a bit to the MOSI line */
        write_MOSI((byte_out & bit) ? HIGH : LOW);

        /* Delay for at least the peer's setup time */
        delay(SPI_SCLK_LOW_TIME);

        /* Pull the clock line high */
        write_SCLK(HIGH);

        /* Shift-in a bit from the MISO line */
        if (read_MISO() == HIGH)
            byte_in |= bit;

        /* Delay for at least the peer's hold time */
        delay(SPI_SCLK_HIGH_TIME);

        /* Pull the clock line low */
        write_SCLK(LOW);
    }

    return byte_in;
}
```



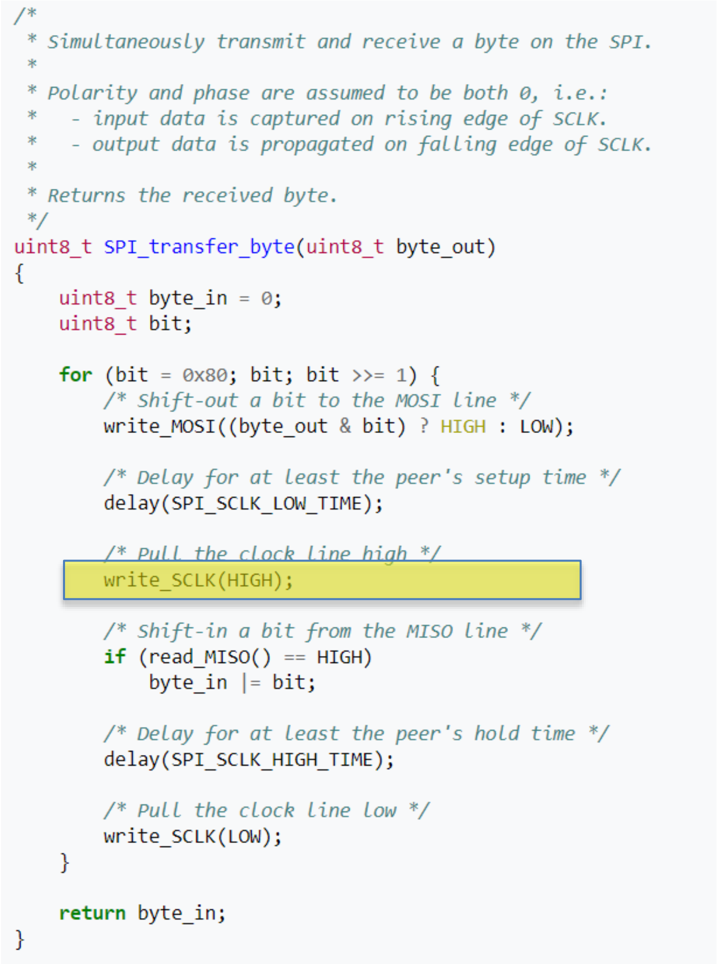
```c
/*
 * Simultaneously transmit and receive a byte on the SPI.
 *
 * Polarity and phase are assumed to be both 0, i.e.:
 *   - input data is captured on rising edge of SCLK.
 *   - output data is propagated on falling edge of SCLK.
 *
 * Returns the received byte.
 */
uint8_t SPI_transfer_byte(uint8_t byte_out)
{
    uint8_t byte_in = 0;
    uint8_t bit;

    for (bit = 0x80; bit; bit >>= 1) {
        /* Shift-out a bit to the MOSI line */
        write_MOSI((byte_out & bit) ? HIGH : LOW);

        /* Delay for at least the peer's setup time */
        delay(SPI_SCLK_LOW_TIME);

        /* Pull the clock line high */
        write_SCLK(HIGH);

        /* Shift-in a bit from the MISO line */
        if (read_MISO() == HIGH)
            byte_in |= bit;

        /* Delay for at least the peer's hold time */
        delay(SPI_SCLK_HIGH_TIME);

        /* Pull the clock line low */
        write_SCLK(LOW);
    }

    return byte_in;
}
```



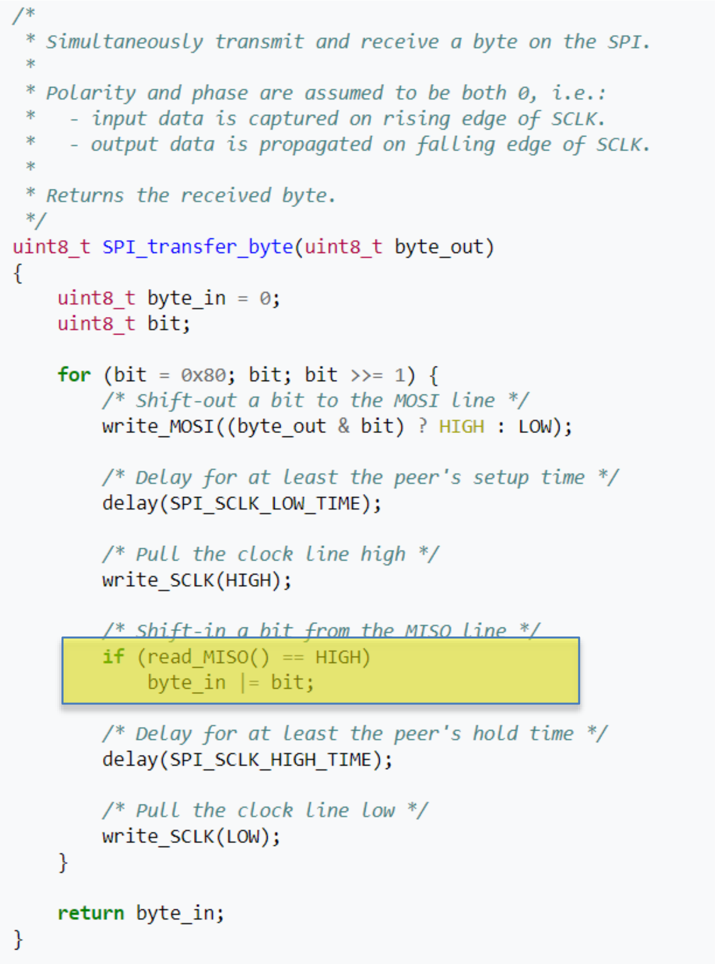
```c
/*
 * Simultaneously transmit and receive a byte on the SPI.
 *
 * Polarity and phase are assumed to be both 0, i.e.:
 *   - input data is captured on rising edge of SCLK.
 *   - output data is propagated on falling edge of SCLK.
 *
 * Returns the received byte.
 */
uint8_t SPI_transfer_byte(uint8_t byte_out)
{
    uint8_t byte_in = 0;
    uint8_t bit;

    for (bit = 0x80; bit; bit >>= 1) {
        /* Shift-out a bit to the MOSI line */
        write_MOSI((byte_out & bit) ? HIGH : LOW);

        /* Delay for at least the peer's setup time */
        delay(SPI_SCLK_LOW_TIME);

        /* Pull the clock line high */
        write_SCLK(HIGH);

        /* Shift-in a bit from the MISO line */
        if (read_MISO() == HIGH)
            byte_in |= bit;

        /* Delay for at least the peer's hold time */
        delay(SPI_SCLK_HIGH_TIME);

        /* Pull the clock line low */
        write_SCLK(LOW);
    }

    return byte_in;
}
```



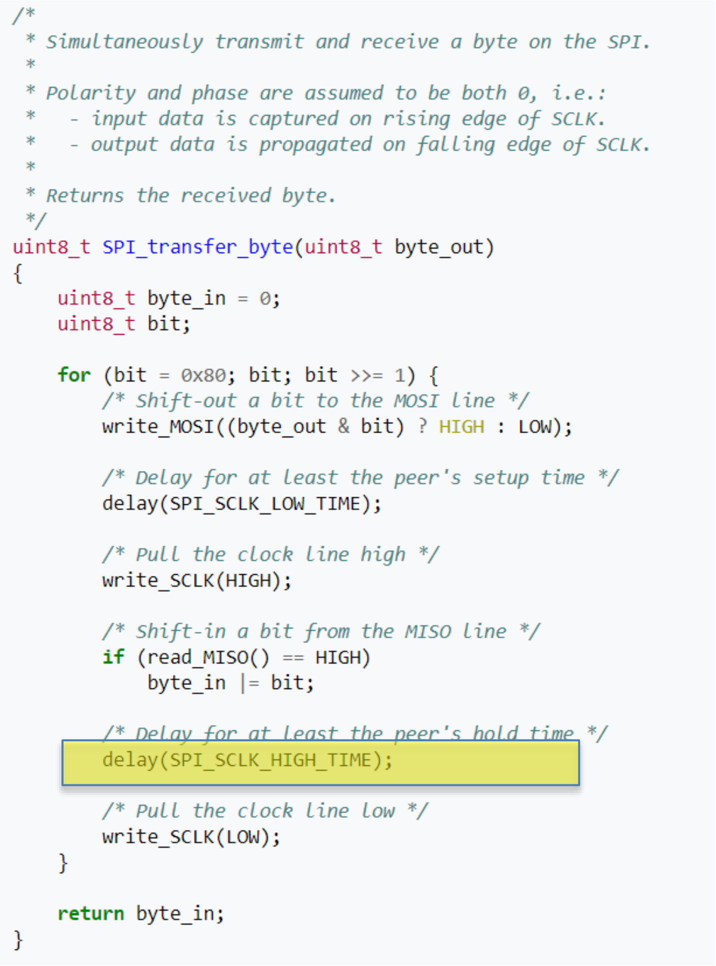
```c
/*
 * Simultaneously transmit and receive a byte on the SPI.
 *
 * Polarity and phase are assumed to be both 0, i.e.:
 *   - input data is captured on rising edge of SCLK.
 *   - output data is propagated on falling edge of SCLK.
 *
 * Returns the received byte.
 */
uint8_t SPI_transfer_byte(uint8_t byte_out)
{
    uint8_t byte_in = 0;
    uint8_t bit;

    for (bit = 0x80; bit; bit >>= 1) {
        /* Shift-out a bit to the MOSI line */
        write_MOSI((byte_out & bit) ? HIGH : LOW);

        /* Delay for at least the peer's setup time */
        delay(SPI_SCLK_LOW_TIME);

        /* Pull the clock line high */
        write_SCLK(HIGH);

        /* Shift-in a bit from the MISO line */
        if (read_MISO() == HIGH)
            byte_in |= bit;

        /* Delay for at least the peer's hold time */
        delay(SPI_SCLK_HIGH_TIME);

        /* Pull the clock line low */
        write_SCLK(LOW);
    }

    return byte_in;
}
```



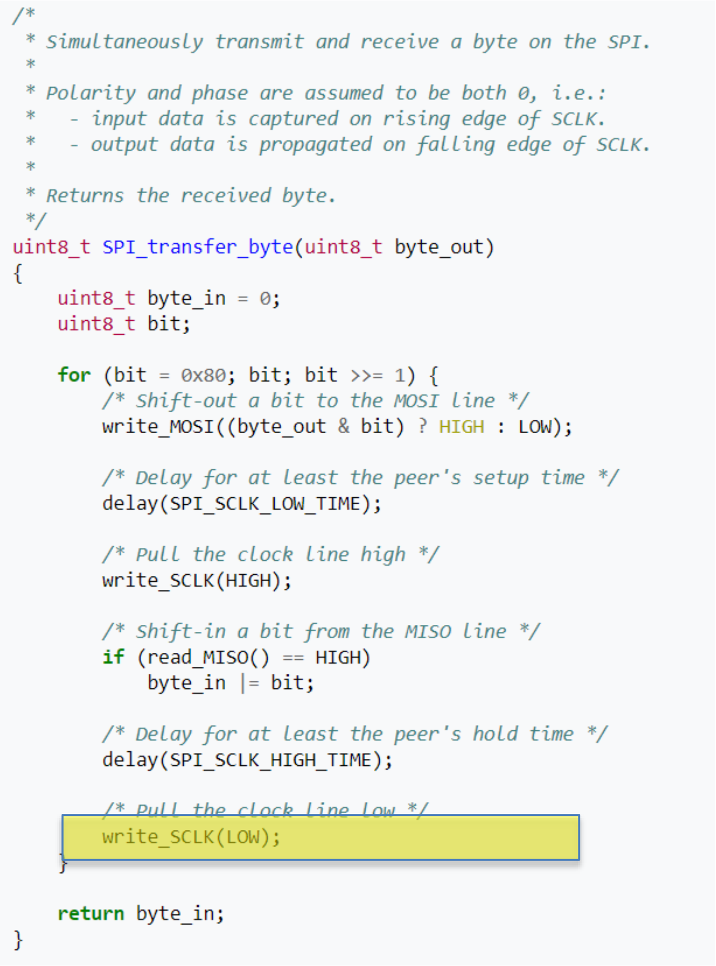
```
/*
 * Simultaneously transmit and receive a byte on the SPI.
 *
 * Polarity and phase are assumed to be both 0, i.e.:
 *   - input data is captured on rising edge of SCLK.
 *   - output data is propagated on falling edge of SCLK.
 *
 * Returns the received byte.
 */
uint8_t SPI_transfer_byte(uint8_t byte_out)
{
    uint8_t byte_in = 0;
    uint8_t bit;

    for (bit = 0x80; bit; bit >>= 1) {
        /* Shift-out a bit to the MOSI line */
        write_MOSI((byte_out & bit) ? HIGH : LOW);

        /* Delay for at least the peer's setup time */
        delay(SPI_SCLK_LOW_TIME);

        /* Pull the clock line high */
        write_SCLK(HIGH);

        /* Shift-in a bit from the MISO line */
        if (read_MISO() == HIGH)
            byte_in |= bit;

        /* Delay for at least the peer's hold time */
        delay(SPI_SCLK_HIGH_TIME);

        /* Pull the clock line low */
        write_SCLK(LOW);
    }

    return byte_in;
}
```



```c
/*
 * Simultaneously transmit and receive a byte on the SPI.
 *
 * Polarity and phase are assumed to be both 0, i.e.:
 *   - input data is captured on rising edge of SCLK.
 *   - output data is propagated on falling edge of SCLK.
 *
 * Returns the received byte.
 */
uint8_t SPI_transfer_byte(uint8_t byte_out)
{
    uint8_t byte_in = 0;
    uint8_t bit;

    for (bit = 0x80; bit; bit >>= 1) {
        /* Shift-out a bit to the MOSI line */
        write_MOSI((byte_out & bit) ? HIGH : LOW);

        /* Delay for at least the peer's setup time */
        delay(SPI_SCLK_LOW_TIME);

        /* Pull the clock line high */
        write_SCLK(HIGH);

        /* Shift-in a bit from the MISO line */
        if (read_MISO() == HIGH)
            byte_in |= bit;

        /* Delay for at least the peer's hold time */
        delay(SPI_SCLK_HIGH_TIME);

        /* Pull the clock line low */
        write_SCLK(LOW);
    }

    return byte_in;
}
```



```c
/*
 * Simultaneously transmit and receive a byte on the SPI.
 *
 * Polarity and phase are assumed to be both 0, i.e.:
 *   - input data is captured on rising edge of SCLK.
 *   - output data is propagated on falling edge of SCLK.
 *
 * Returns the received byte.
 */
uint8_t SPI_transfer_byte(uint8_t byte_out)
{
    uint8_t byte_in = 0;
    uint8_t bit;

    for (bit = 0x80; bit; bit >>= 1) {
        /* Shift-out a bit to the MOSI line */
        write_MOSI((byte_out & bit) ? HIGH : LOW);

        /* Delay for at least the peer's setup time */
        delay(SPI_SCLK_LOW_TIME);

        /* Pull the clock line high */
        write_SCLK(HIGH);

        /* Shift-in a bit from the MISO line */
        if (read_MISO() == HIGH)
            byte_in |= bit;

        /* Delay for at least the peer's hold time */
        delay(SPI_SCLK_HIGH_TIME);

        /* Pull the clock line low */
        write_SCLK(LOW);
    }

    return byte_in;
}
```



```c
/*
 * Simultaneously transmit and receive a byte on the SPI.
 *
 * Polarity and phase are assumed to be both 0, i.e.:
 *   - input data is captured on rising edge of SCLK.
 *   - output data is propagated on falling edge of SCLK.
 *
 * Returns the received byte.
 */
uint8_t SPI_transfer_byte(uint8_t byte_out)
{
    uint8_t byte_in = 0;
    uint8_t bit;

    for (bit = 0x80; bit; bit >>= 1) {
        /* Shift-out a bit to the MOSI line */
        write_MOSI((byte_out & bit) ? HIGH : LOW);

        /* Delay for at least the peer's setup time */
        delay(SPI_SCLK_LOW_TIME);

        /* Pull the clock line high */
        write_SCLK(HIGH);

        /* Shift-in a bit from the MISO line */
        if (read_MISO() == HIGH)
            byte_in |= bit;

        /* Delay for at least the peer's hold time */
        delay(SPI_SCLK_HIGH_TIME);

        /* Pull the clock line low */
        write_SCLK(LOW);
    }

    return byte_in;
}
```

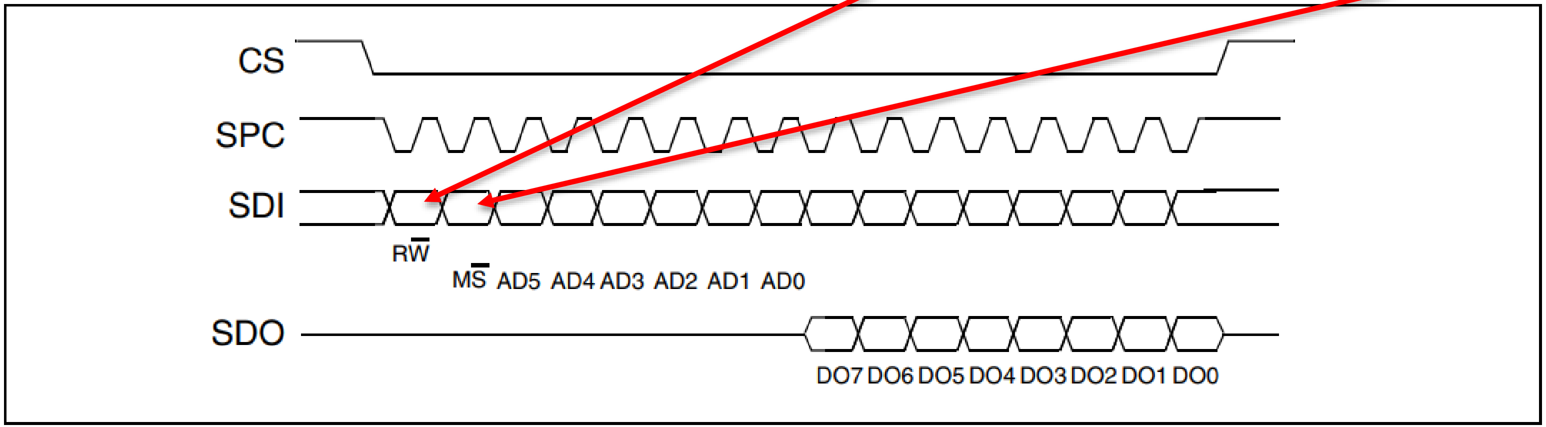
Register Addressing

Reading

Set to 1, for reading

Auto-Increment Address

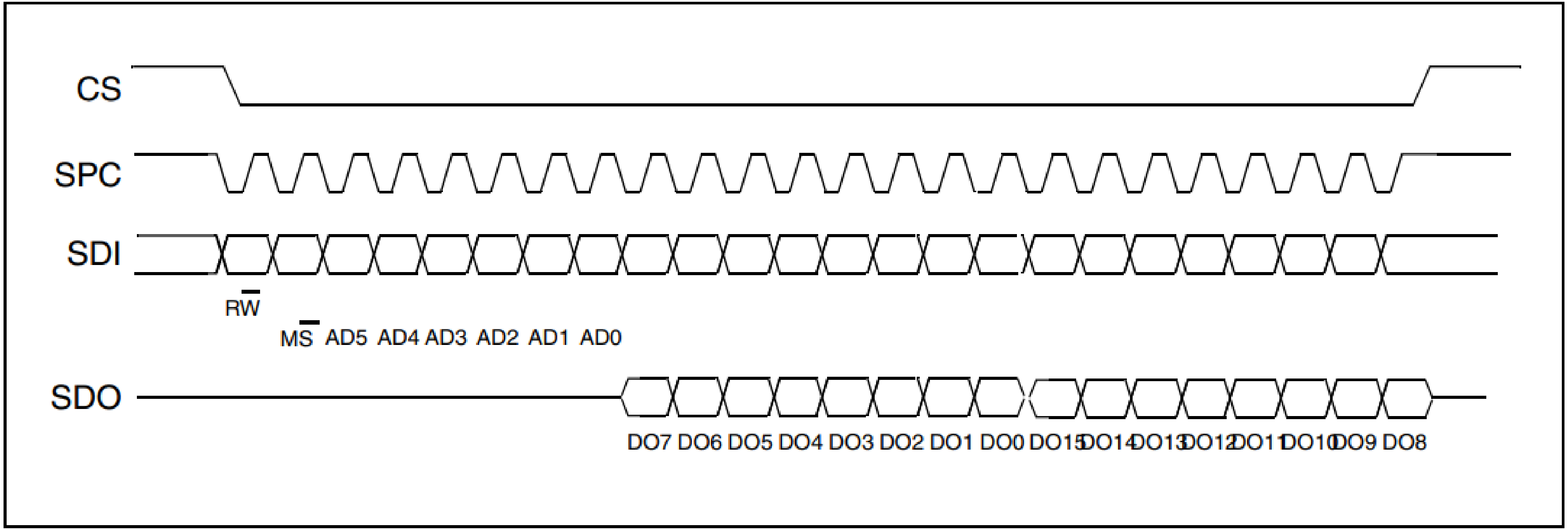
Figure 5. SPI read protocol


CS
SPC
SDI
R$\overline{W}$
M$\overline{S}$ AD5 AD4 AD3 AD2 AD1 AD0
SDO
DO7 DO6 DO5 DO4 DO3 DO2 DO1 DO0

The SPI Read command is performed with 16 clock pulses. The multiple byte read command is performed adding blocks of 8 clock pulses at the previous one.

Read Multiple Registers

Figure 6. Multiple bytes SPI read protocol (2 bytes example)

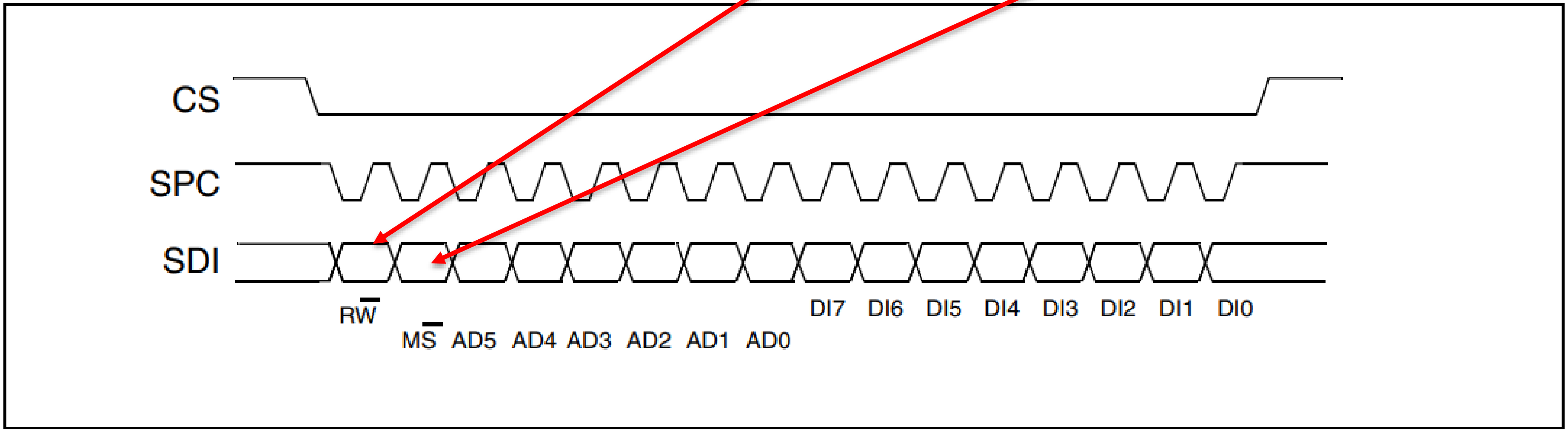

CS
SPC
SDI
SDO
RW
MS AD5 AD4 AD3 AD2 AD1 AD0
DO7 DO6 DO5 DO4 DO3 DO2 DO1 DO0 DO15 DO14 DO13 DO12 DO11 DO10 DO9 DO8

Writing

Set to 0, for writing

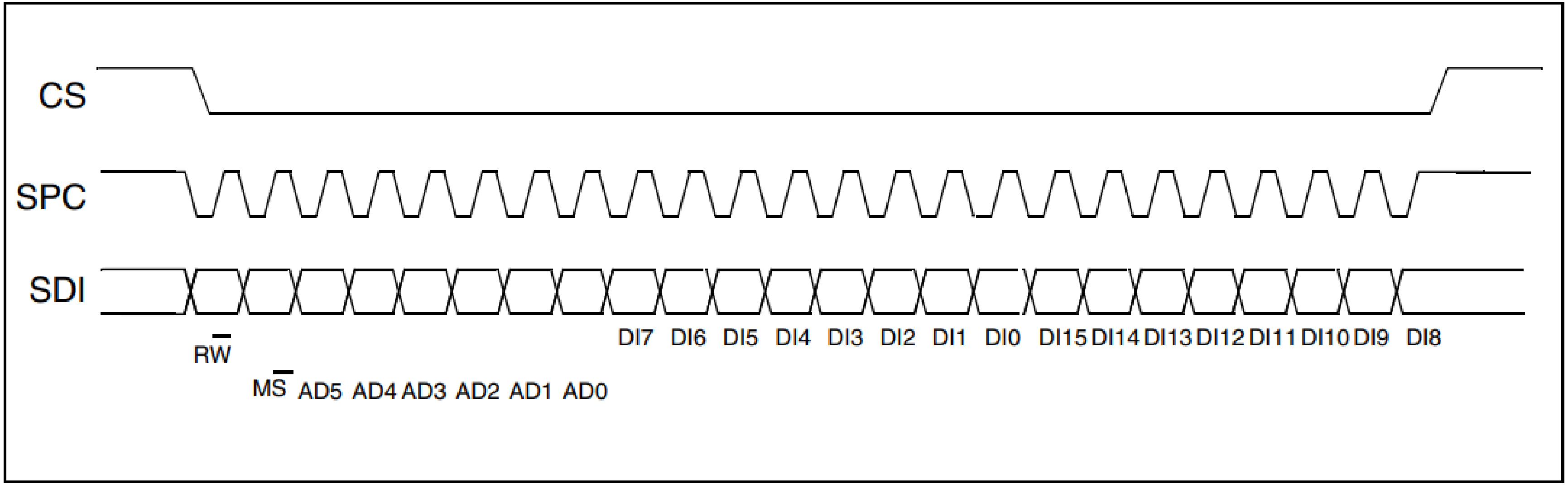
Multiple Register Write …

Figure 7. SPI write protocol

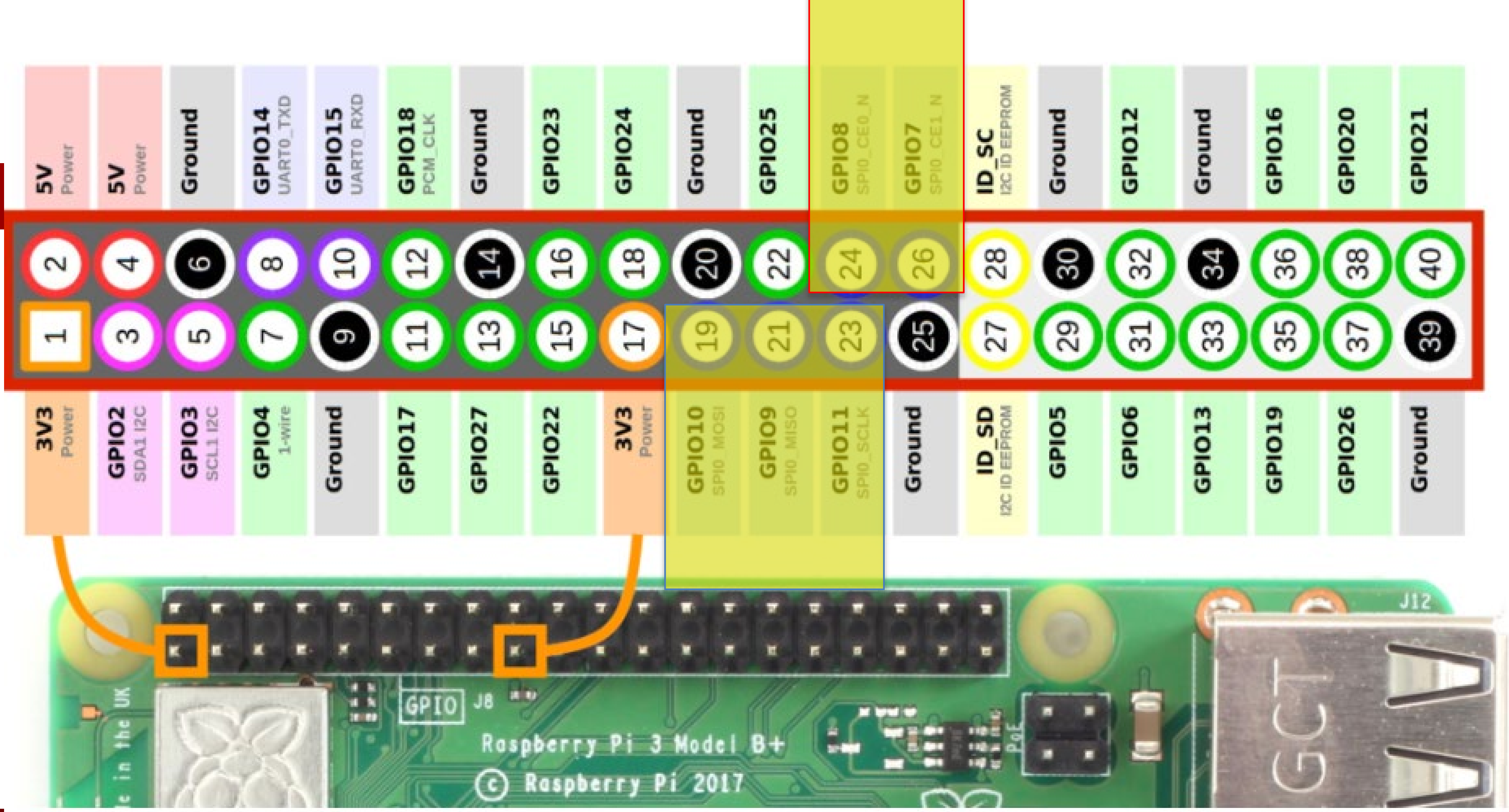


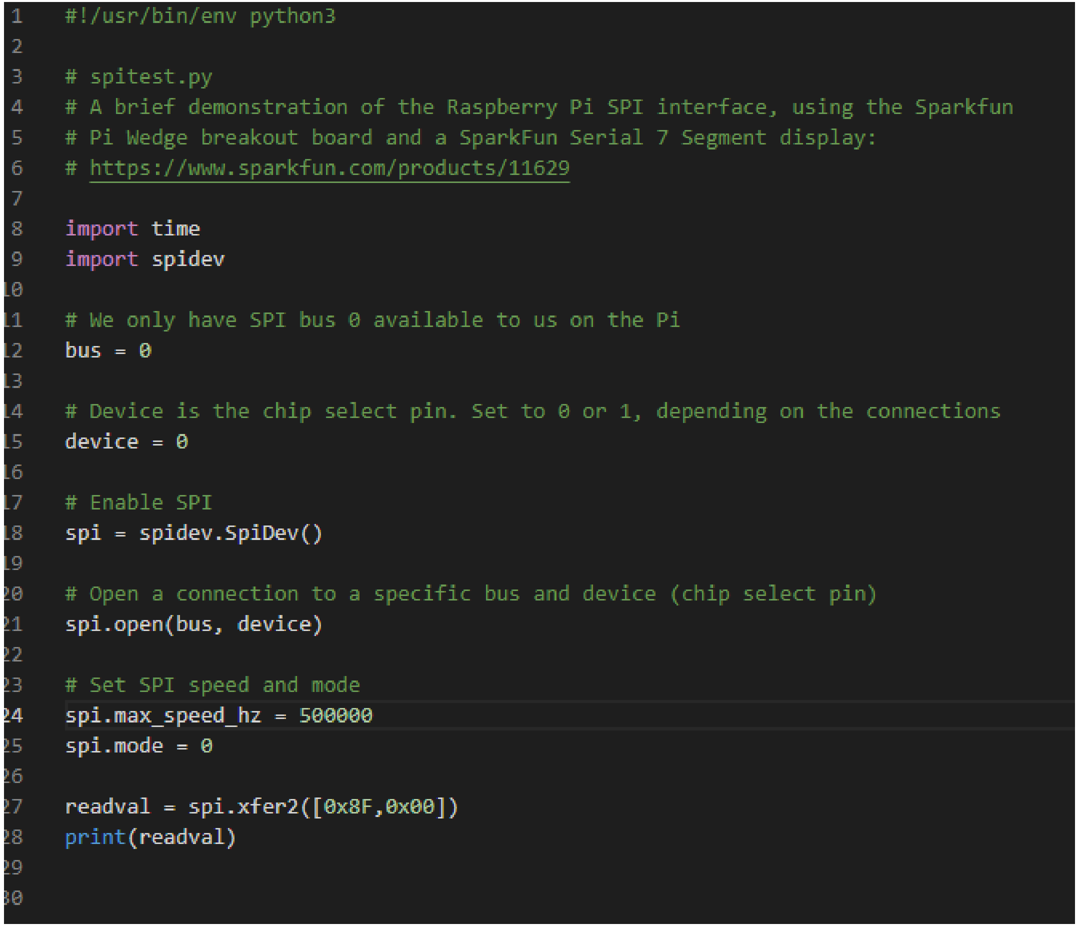
Write Multiple Registers

Figure 8. Multiple bytes SPI write protocol (2 bytes example)



Raspberry Pi Python SPI


5V Power
5V Power
Ground
GPIO14 UART0_TXD
GPIO15 UART0_RXD
GPIO18 PCM_CLK
Ground
GPIO23
GPIO24
Ground
GPIO25
GPIO8 SPI0_CE0_N
GPIO7 SPI0_CE1_N
ID_SC I2C ID EEPROM
Ground
GPIO12
Ground
GPIO16
GPIO20
GPIO21
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
3V3 Power
GPIO2 SDA1 I2C
GPIO3 SCL1 I2C
GPIO4 1-wire
Ground
GPIO17
GPIO27
GPIO22
3V3 Power
GPIO10 SPI0_MOSI
GPIO9 SPI0_MISO
GPIO11 SPI0_SCLK
Ground
ID_SD I2C ID EEPROM
GPIO5
GPIO6
GPIO13
GPIO19
GPIO26
Ground
J12
GPIO
J8
Raspberry Pi 3 Model B+
© Raspberry Pi 2017
PoE
GCT



```python
#!/usr/bin/env python3

# spitest.py
# A brief demonstration of the Raspberry Pi SPI interface, using the Sparkfun
# Pi Wedge breakout board and a SparkFun Serial 7 Segment display:
# https://www.sparkfun.com/products/11629

import time
import spidev

# We only have SPI bus 0 available to us on the Pi
bus = 0

# Device is the chip select pin. Set to 0 or 1, depending on the connections
device = 0

# Enable SPI
spi = spidev.SpiDev()

# Open a connection to a specific bus and device (chip select pin)
spi.open(bus, device)

# Set SPI speed and mode
spi.max_speed_hz = 500000
spi.mode = 0

readval = spi.xfer2([0x8F,0x00])
print(readval)


```

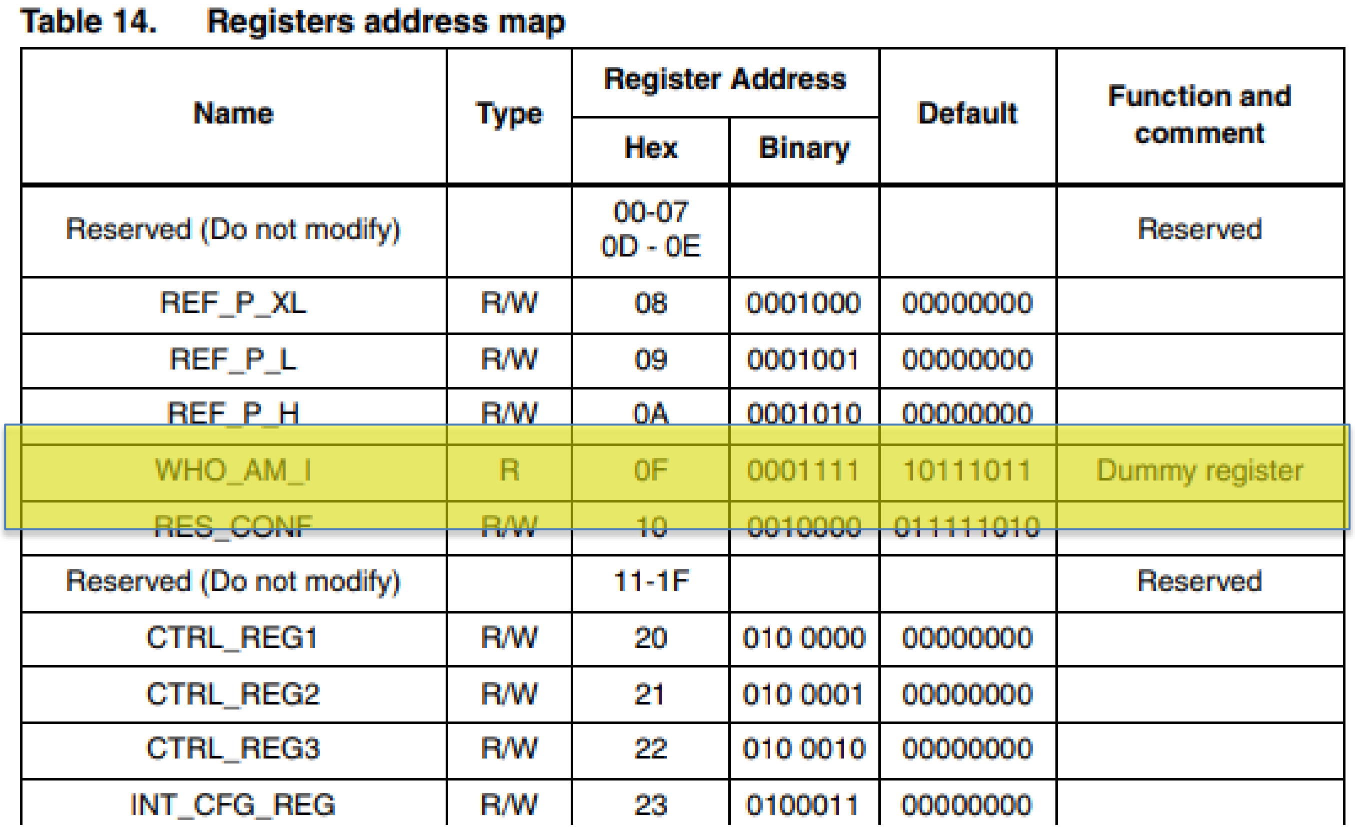
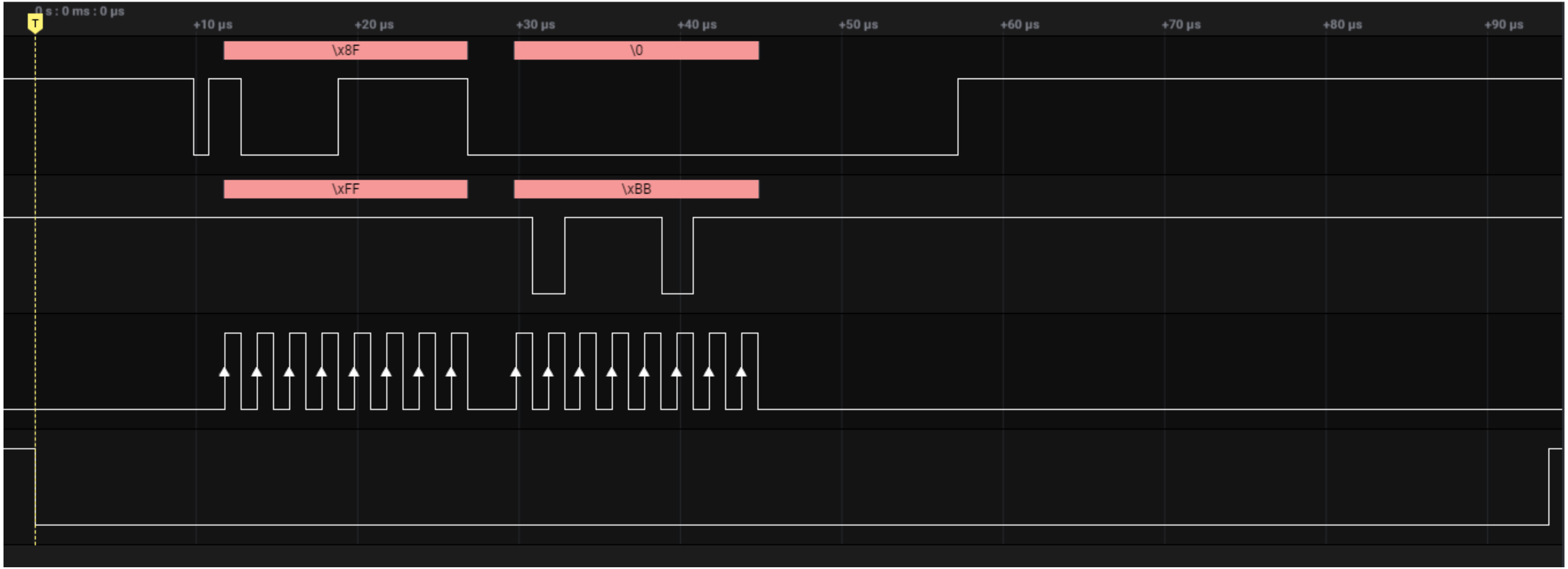


Table 14. Registers address map

Name	Type	Register Address		Default	Function and comment
		Hex	Binary		
Reserved (Do not modify)		00-07 0D - 0E			Reserved
REF_P_XL	R/W	08	0001000	00000000	
REF_P_L	R/W	09	0001001	00000000	
REF_P_H	R/W	0A	0001010	00000000	
WHO_AM_I	R	0F	0001111	10111011	Dummy register
RES_CONF	R/W	10	0010000	011111010	
Reserved (Do not modify)		11-1F			Reserved
CTRL_REG1	R/W	20	010 0000	00000000	
CTRL_REG2	R/W	21	010 0001	00000000	
CTRL_REG3	R/W	22	010 0010	00000000	
INT_CFG_REG	R/W	23	0100011	00000000	


0 s : 0 ms : 0 µs
T
+10 µs
+20 µs
+30 µs
+40 µs
+50 µs
+60 µs
+70 µs
+80 µs
+90 µs
\x8F
\0
\xFF
\xBB