E210 Engineering Cyber-Physical Systems

Serial Peripheral Interface (SPI)

Bryce Himebaugh

Weekly Focus	Reading	Monday	Wed	Lab
CPS Intro/UART		1/10: CPS Introduction	1/12: Pi Intro/UART Bus	Project 0 Raspberry PI Setup
I2C Bus		1/17: MLK Day	1/19: I2C Bus Overview	Project 1 I2C Pressure/Temperature Sensor
I2C and SPI Bus		1/24: Pressure Sensor	1/26: SPI Bus Overview	Project 2 SPI Accelerometer
SPI/Networking		1/31: Accelerometer	2/2: Networking Overview	Project 3 Flask Web Server
Networking		2/7: Flask	2/9: Redis/matplotlib	Project 3 Continued
Web Server		2/14: CPS Wrapup	2/16: Exam Review	P5 Demultiplexer

https://engr210.github.io/

Raspberry SPI Link

Workstation Serial Interface

UART

Network

Raspberry Pi 3b

SPI

SPI

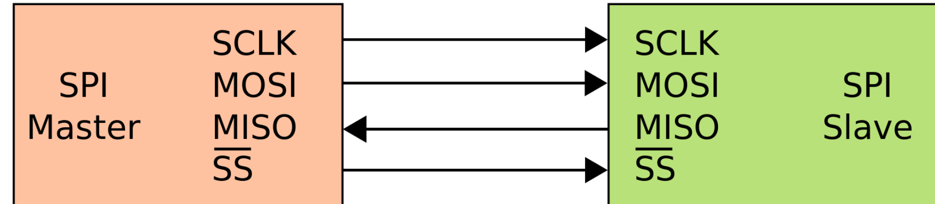
Accelerometer

I2C Link

Pressure/Temperature Sensor

SPI Overview

What is SPI?



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 SPI

- 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

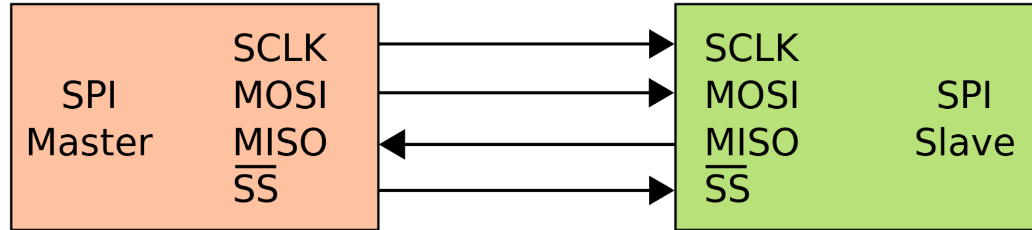
Open Source Hardware Association

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

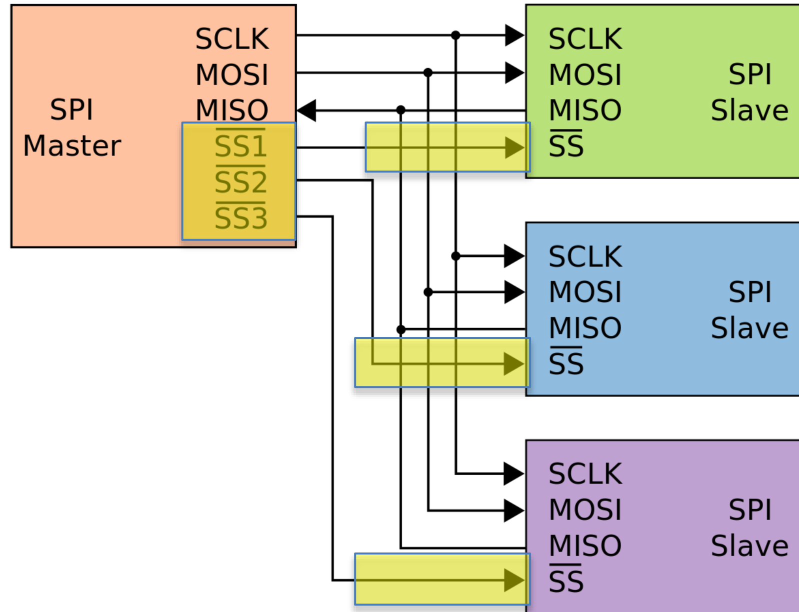
https://www.oshwa.org/a-resolution-to-redefine-spi-signal-names

Bus Connections

Single Peripheral

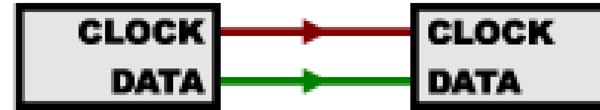


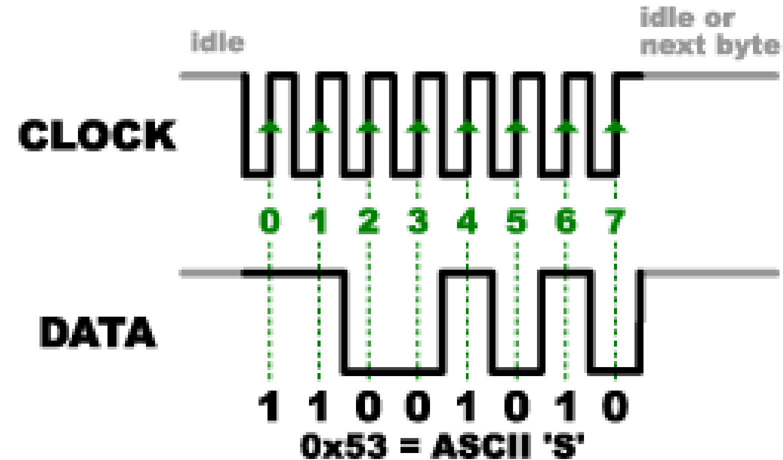
Multiple Independent Peripherals



Bus Protocol

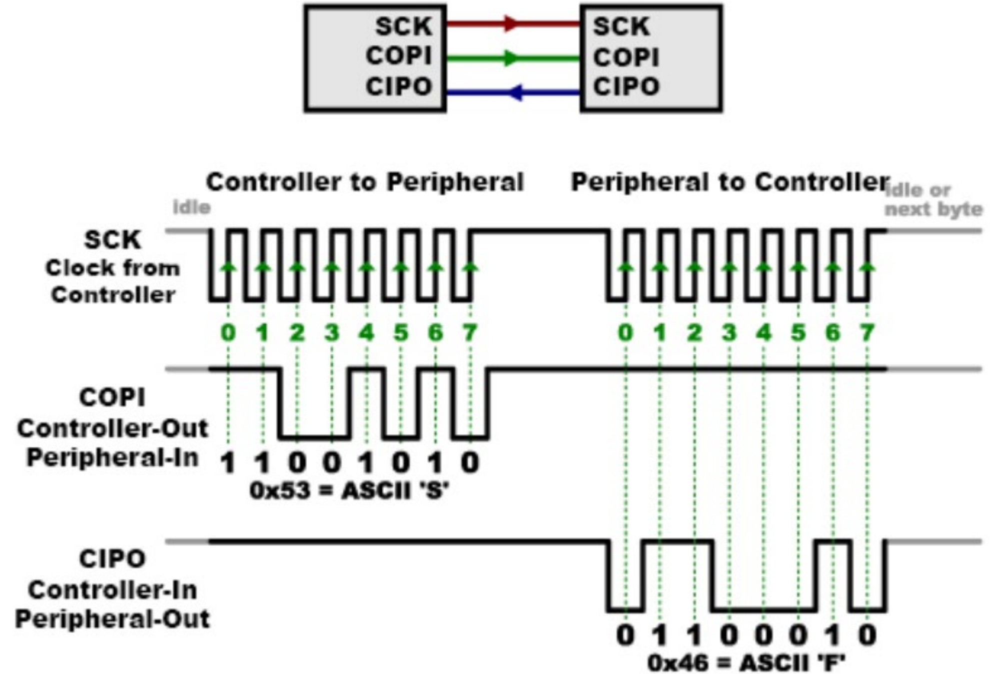
Sending Data





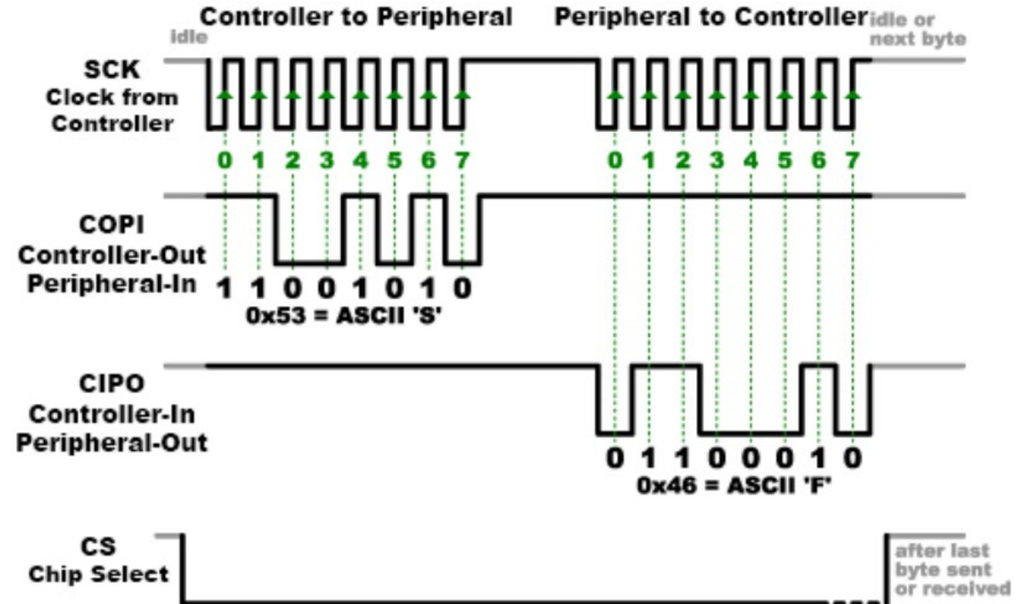
https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all

Receiving Data

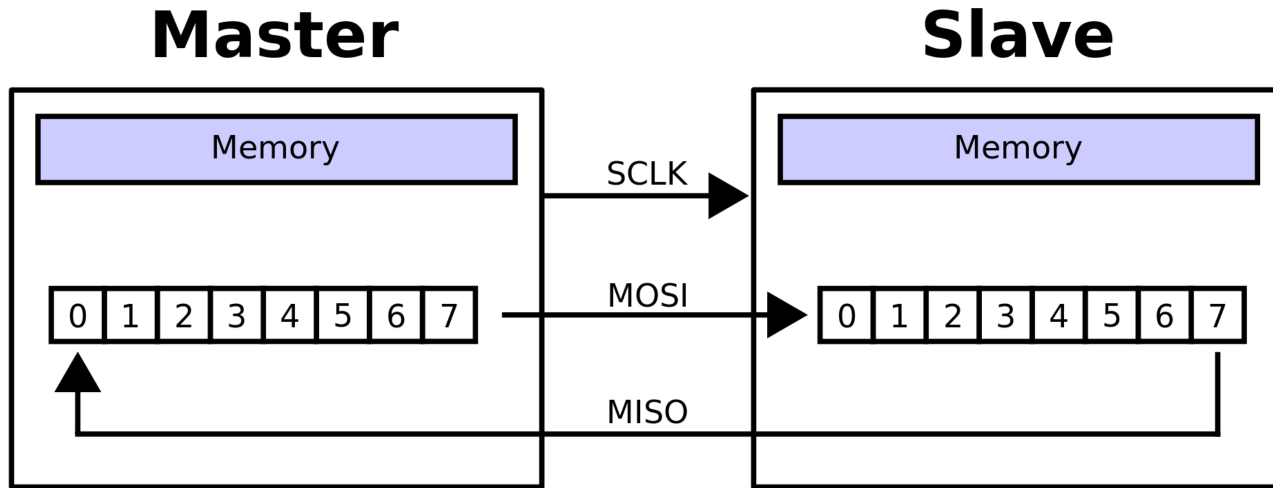


https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all

Chip Select



16-bit Shift Register (split between two chips)



Swaps Registers

Waveform

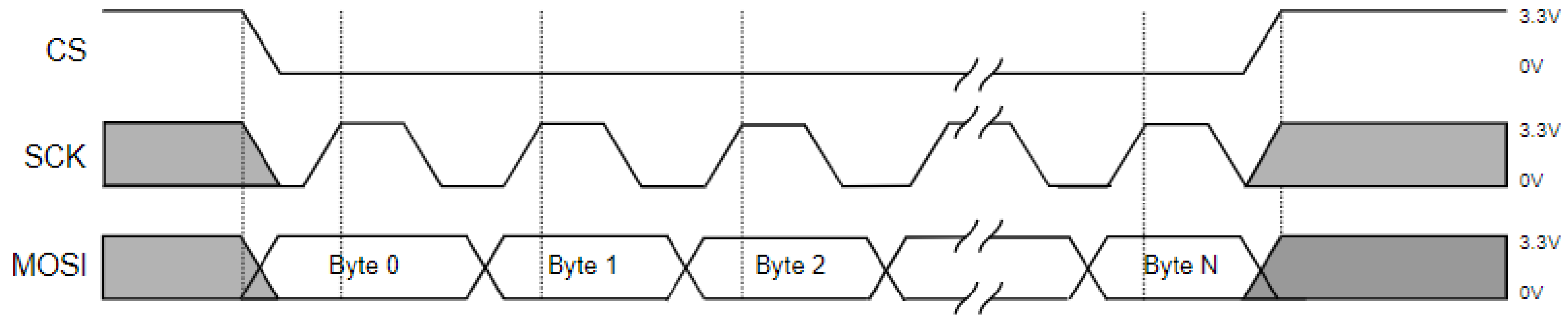


Figure 1. Example of a SPI transmission

Waveform

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

nCS
CLK
MOSI 0xA5: xxxx 1 0 1 0 0 1 0 1 xxxx
MISO 0xBA: Hi-Z 1 0 1 1 1 0 1 0 Hi-Z

```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;
}
```

```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;
}
```

```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

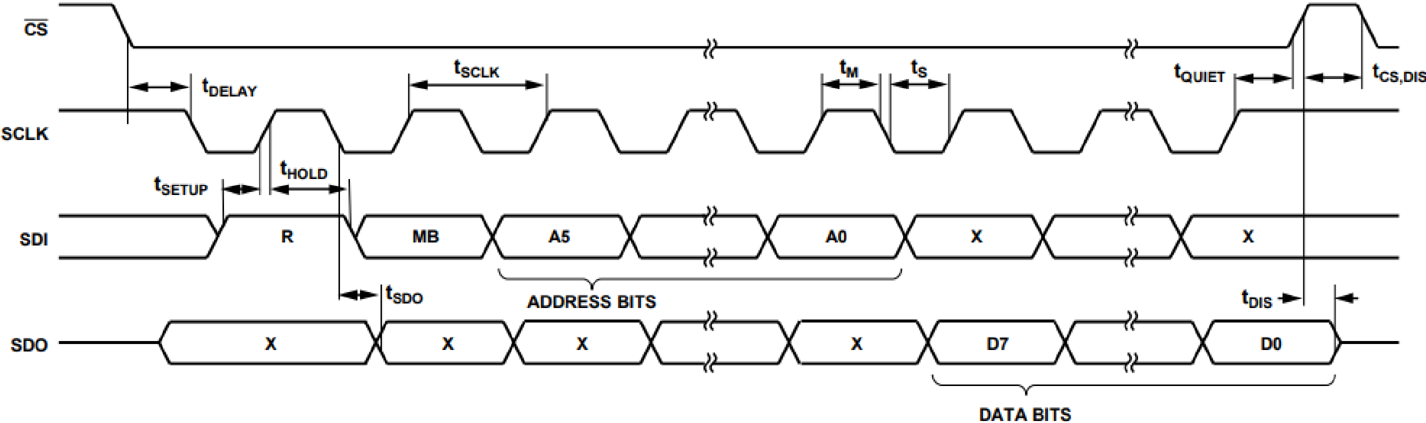


Figure 28. SPI 4-Wire Read

Writing

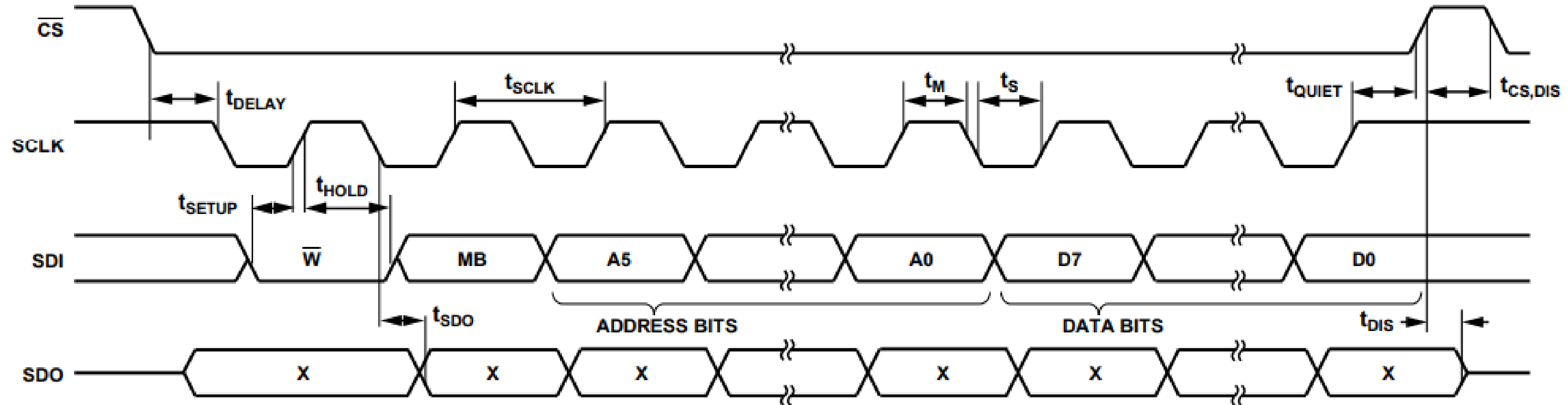


Figure 27. SPI 4-Wire Write

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

ADXL343 Accelerometer

Pi Model B/B+

	Pin	Pin	
3V3 Power	1	2	5V Power
GPIO2 SDA1 I2C	3	4	5V Power
GPIO3 SCL1 I2C	5	6	Ground
GPIO4	7	8	GPIO14 UART0_TXD
Ground	9	10	GPIO15 UART0_RXD
GPIO17	11	12	GPIO18 PCM_CLK
GPIO27	13	14	Ground
GPIO22	15	16	GPIO23
3V3 Power	17	18	GPIO24
GPIO10 SPI0_MOSI	19	20	Ground
GPIO9 SPI0_MISO	21	22	GPIO25
GPIO11 SPI0_SCLK	23	24	GPIO8 SPI0_CE0_N
Ground	25	26	GPIO7 SPI0_CE1_N

ADXL343 Digital Accelerometer

VIN
3V3
GND
SCL
SDA
SDO
CS
INT1
INT2

3-Axis, ±2 *g*/±4 *g*/±8 *g*/±16 *g* Digital MEMS Accelerometer

ANALOG DEVICES

Data Sheet **ADXL343**

FEATURES

- Multipurpose accelerometer with 10- to 13-bit resolution for use in a wide variety of applications
- Digital output accessible via SPI (3- and 4-wire) and I²C
- Built-in motion detection features make tap, double-tap, activity, inactivity, and free-fall detection trivial
  - User-adjustable thresholds
  - Interrupts independently mappable to two interrupt pins
- Low power operation down to 23 µA and embedded FIFO for reducing overall system power
- Wide supply voltage range: 2.0 V to 3.6 V
  - I/O voltage 1.7 V to V_S
- Wide operating temperature range (−40°C to +85°C)
- 10,000 *g* shock survival
- Small, thin, Pb free, RoHS compliant 3 mm × 5 mm × 1 mm LGA package

APPLICATIONS

- Handsets
- Gaming and pointing devices
- Hard disk drive (HDD) protection

GENERAL DESCRIPTION

The ADXL343 is a versatile 3-axis, digital-output, low *g* MEMS accelerometer. Selectable measurement range and bandwidth, and configurable, built-in motion detection make it suitable for sensing acceleration in a wide variety of applications. Robustness to 10,000 *g* of shock and a wide temperature range (−40°C to +85°C) enable use of the accelerometer even in harsh environments.

The ADXL343 measures acceleration with high resolution (13-bit) measurement at up to ±16 *g*. Digital output data is formatted as 16-bit twos complement and is accessible through either an SPI (3- or 4-wire) or I²C digital interface. The ADXL343 can measure the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (3.9 m*g*/LSB) enables measurement of inclination changes less than 1.0°.

Several special sensing functions are provided. Activity and inactivity sensing detect the presence or lack of motion. Tap sensing detects single and double taps in any direction. Free-fall sensing detects if the device is falling. These functions can be mapped individually to either of two interrupt output pins.

An integrated memory management system with a 32-level first in, first out (FIFO) buffer can be used to store data to minimize host processor activity and lower overall system power consumption.

The ADXL343 is supplied in a small, thin, 3 mm × 5 mm × 1 mm, 14-terminal, plastic package.

FUNCTIONAL BLOCK DIAGRAM

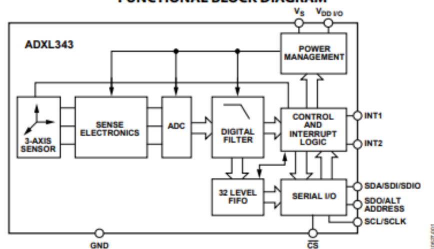


Figure 1.

ADXL343 Data Sheet

PIN CONFIGURATION AND FUNCTION DESCRIPTIONS

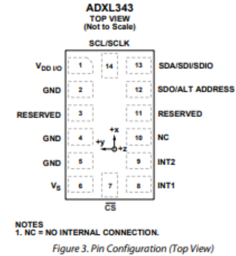


Figure 3. Pin Configuration (Top View)

Table 5. Pin Function Descriptions

Pin No.	Mnemonic	Description
1	$V_{DD I/O}$	Digital Interface Supply Voltage.
2	GND	This pin must be connected to ground.
3	RESERVED	Reserved. This pin must be connected to V_S or left open.
4	GND	This pin must be connected to ground.
5	GND	This pin must be connected to ground.
6	V_S	Supply Voltage.
7	$\overline{CS}$	Chip Select.
8	INT1	Interrupt 1 Output.
9	INT2	Interrupt 2 Output.
10	NC	Not Internally Connected.
11	RESERVED	Reserved. This pin must be connected to ground or left open.
12	SDO/ALT ADDRESS	Serial Data Output (SPI 4-Wire)/Alternate I²C Address Select (I²C).
13	SDA/SDI/SDIO	Serial Data (I²C)/Serial Data Input (SPI 4-Wire)/Serial Data Input and Output (SPI 3-Wire).
14	SCL/SCLK	Serial Communications Clock. SCL is the clock for I²C, and SCLK is the clock for SPI.

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

import spidev
import time
import numpy as np

class adxl343:
    ''' Enables communication between the raspberry pi and the ADXL343 board from Sparkfun '''
    def __init__(self,spi_device=0, ce_pin=0, speed=1000000):
        """
        spi_device: there are two spi ports. It is most common to use port 0.
        ce_pin: there are two CE pins that are automatically controlled by the pi.
        speed: the speed for the spi clock is specified here. used to limit spi speed.
        """
        self.spi = spidev.SpiDev()
        self.spi.open(spi_device, ce_pin)
        self.spi.max_speed_hz = speed   # Sets the maximum speed of the SPI link
        self.spi.mode = 0b11            # Sets the clock polarity and the phase respectively
        time.sleep(0.5)
        if self.get_device_id() == '0xe5':
            self.enable()
            print("found ADXL343")
        else:
            print("Device ID Incorrect")
```

spidev Documentation

1. https://pypi.org/project/spidev/

Max Clock Speed

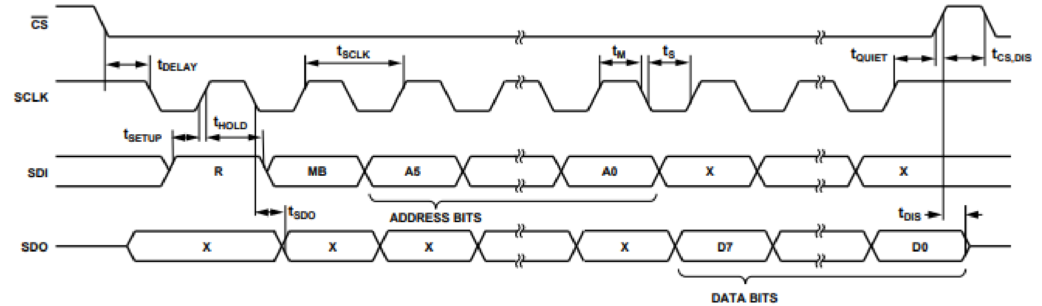


Figure 28. SPI 4-Wire Read

Table 10. SPI Timing ($T_A = 25°C$, $V_S = 2.5$ V, $V_{DD\,I/O} = 1.8$ V)[1]

Parameter	Limit[2, 3] Min	Max	Unit	Description
f_{SCLK}		5	MHz	SPI clock frequency
t_{SCLK}	200		ns	1/(SPI clock frequency) mark-space ratio for the SCLK input is 40/60 to 60/40
t_{DELAY}	5		ns	$\overline{CS}$ falling edge to SCLK falling edge
t_{QUIET}	5		ns	SCLK rising edge to $\overline{CS}$ rising edge
t_{DIS}		10	ns	$\overline{CS}$ rising edge to SDO disabled
$t_{CS,DIS}$	150		ns	$\overline{CS}$ deassertion between SPI communications
t_S	$0.3 \times t_{SCLK}$		ns	SCLK low pulse width (space)
t_M	$0.3 \times t_{SCLK}$		ns	SCLK high pulse width (mark)
t_{SETUP}	5		ns	SDI valid before SCLK rising edge
t_{HOLD}	5		ns	SDI valid after SCLK rising edge
t_{SDO}		40	ns	SCLK falling edge to SDO/SDIO output transition
t_R[4]		20	ns	SDO/SDIO output high to output low transition
t_F[4]		20	ns	SDO/SDIO output low to output high transition

SPI Polarity and Phase (mode=0b00)

CPOL=0, CPHA=0

CPOL=0 SCLK

$\overline{SS}$

Sample

MISO

CPHA=0

MOSI

SPI Polarity and Phase (mode=0b01)

CPOL=0, CPHA=1

CPOL=0

SCLK

$\overline{SS}$

Sample

MISO

CPHA=1

MOSI

SPI Polarity and Phase (mode=0b10))

CPOL=1, CPHA=0

CPOL=1

SCLK

SS

Sample

MISO

CPHA=0

MOSI

SPI Polarity and Phase (mode=0b11)

CPOL=1, CPHA=1

CPOL=1

SCLK

SS

Sample

MISO

CPHA=1

MOSI

What the polarity and phase of this waveform?

Figure 28. SPI 4-Wire Read

Registers

REGISTER MAP

Table 19.

Address					
Hex	Dec	Name	Type	Reset Value	Description
0x00	0	DEVID	R	11100101	Device ID
0x01 to 0x1C	1 to 28	Reserved			Reserved; do not access
0x1D	29	THRESH_TAP	R/$\overline{W}$	00000000	Tap threshold
0x1E	30	OFSX	R/$\overline{W}$	00000000	X-axis offset
0x1F	31	OFSY	R/$\overline{W}$	00000000	Y-axis offset
0x20	32	OFSZ	R/$\overline{W}$	00000000	Z-axis offset
0x21	33	DUR	R/$\overline{W}$	00000000	Tap duration
0x22	34	Latent	R/$\overline{W}$	00000000	Tap latency
0x23	35	Window	R/$\overline{W}$	00000000	Tap window
0x24	36	THRESH_ACT	R/$\overline{W}$	00000000	Activity threshold
0x25	37	THRESH_INACT	R/$\overline{W}$	00000000	Inactivity threshold
0x26	38	TIME_INACT	R/$\overline{W}$	00000000	Inactivity time
0x27	39	ACT_INACT_CTL	R/$\overline{W}$	00000000	Axis enable control for activity and inactivity detection
0x28	40	THRESH_FF	R/$\overline{W}$	00000000	Free-fall threshold
0x29	41	TIME_FF	R/$\overline{W}$	00000000	Free-fall time
0x2A	42	TAP_AXES	R/$\overline{W}$	00000000	Axis control for single tap/double tap
0x2B	43	ACT_TAP_STATUS	R	00000000	Source of single tap/double tap
0x2C	44	BW_RATE	R/$\overline{W}$	00001010	Data rate and power mode control
0x2D	45	POWER_CTL	R/$\overline{W}$	00000000	Power-saving features control
0x2E	46	INT_ENABLE	R/$\overline{W}$	00000000	Interrupt enable control
0x2F	47	INT_MAP	R/$\overline{W}$	00000000	Interrupt mapping control
0x30	48	INT_SOURCE	R	00000010	Source of interrupts
0x31	49	DATA_FORMAT	R/$\overline{W}$	00000000	Data format control
0x32	50	DATAX0	R	00000000	X-Axis Data 0
0x33	51	DATAX1	R	00000000	X-Axis Data 1
0x34	52	DATAY0	R	00000000	Y-Axis Data 0
0x35	53	DATAY1	R	00000000	Y-Axis Data 1
0x36	54	DATAZ0	R	00000000	Z-Axis Data 0
0x37	55	DATAZ1	R	00000000	Z-Axis Data 1
0x38	56	FIFO_CTL	R/$\overline{W}$	00000000	FIFO control
0x39	57	FIFO_STATUS	R	00000000	FIFO status

Write Register

```
def write_register(self, address, data):
        self.spi.xfer2([address,data])
        return(0)
```

Read Register

```
def read_register(self, address):
        address = address | 0x80
        read_bytes = self.spi.xfer2([address,0x00])
        return (read_bytes[1])
```