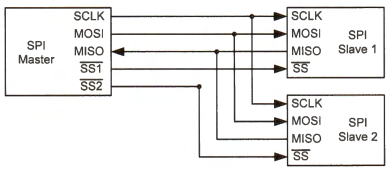
**Interfacing a 3-axis MEMS Accelerometer using SPI Communication Protocol in ARM Cortex-M microcontroller**

**1. Aim:**

1. Write embedded C program to study about Serial Peripheral Interface (SPI) communication protocol in STM32F407 microcontroller.
2. Understand ADXL345 datasheet and interface it with microcontroller to capture the 3-axis data.

**2. Introduction:**

Serial peripheral interface (SPI) is a synchronous serial communication interface widely used to exchange data between a microprocessor and peripheral devices using four wires. For example, a digital camera often uses SPI to control its lens and save photos to a MMC or SD media. SPI is simple, has low power requirements, and supports high throughput. Disadvantages of SPI include that it does not support multiple masters, and slaves cannot start the communication or control data transfer speed. The master initiates and controls all communications. A SPI interface consists of four lines: a master-in-slave-out data line (MISO), a master-out-slave-in data line (MOSI), a serial clock line (SCLK), and an active-low slave select line (SS), as shown in the figure below.



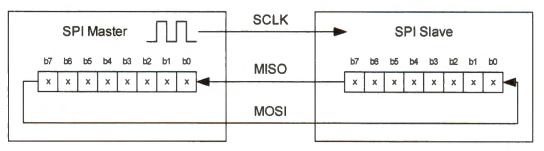
SPI only supports a single master communicating with multiple slave devices. As shown in Figure, when the master wishes to exchange data with a slave, it pulls down the corresponding select line (SSN). The master then generates clock pulses to coordinate the data transmission on the MOSI and MISO lines.

Data exchange can take place in both directions simultaneously, and this two-way serial channel is often called full duplex. Data bits are transmitted on both the MOSI line and the MISO line synchronously, with the flow directions opposite to each other. Note the SCLK line has only one direction, and only the master can generate the clock signal. The slave devices cannot control the clock line. When there are multiple slave devices, the master decides which slave device it wants to communicate. There is a dedicated Slave Select (SS) line for each slave device. The master selects the target slave device by pulling the corresponding SS line to a low voltage prior to data transfer. The selected slave device then listens for the clock and MOSI signals. When there is only one slave device, the SS line can be directly connected to ground physically, or the program can make the slave continuously selected.

**3.1 Data Exchange:**

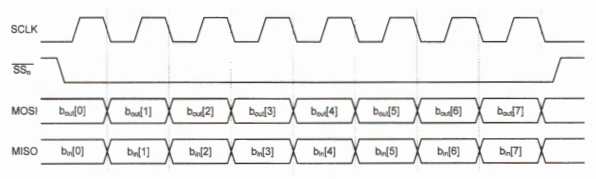
SPI is a synchronous protocol, and the slave devices must send and receive data based on the clock provided by the master. It differs from an asynchronous protocol in which no clock signal is provided physically. SPI devices must exchange data at the same speed. The master and a slave perform data exchange at synchronized time steps based on the clock signal generated by the master.

When a bit is shifted out on the MISO line from the slave's data register during a clock period, a new data bit is shifted into this register from the MOSI line in the same clock period, as shown in Figure. When one device writes a bit to the data line at the rising or falling edge of the clock, the other device then reads the bit at the opposite edge of the same clock period. The data transfer size is usually a byte or halfword (16 bits).



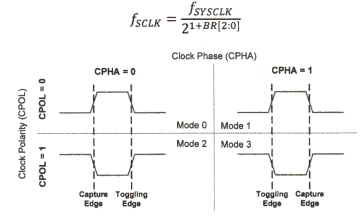
Communication from the master to a slave and communication from a slave to the master are always taking place concurrently. In each communication link (either MISO or MOSI), each device sends out a data item and at the same time receives a new data item. No devices can just be a transmitter or a receiver. Therefore, when a slave wants to send data to the master via the MISO line, the slave must wait for the clock signal. At the same time, the master must send some dummy data out via the MOSI line to generate the clock signal to initiate the data transfer.

When a master exchanges data with slave n, the master must set SSn low to select slave n, as shown in Figure. During the communication, the most significant bit of both data registers is sent out first.

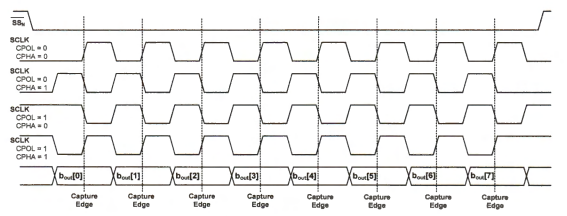


**3.2 Clock Configuration:**

The clock speed determines the data transfer rate. The data rate ranges from 1 to 20 megabits per second. The master can change the clock speed by programming the clock prescaler register. For STM32L processors, the baud rate control factor is stored in the BR[2:0] bits of the SPI control register (CR1). The SCLK clock frequency is programmed by setting the baud rate control factor.

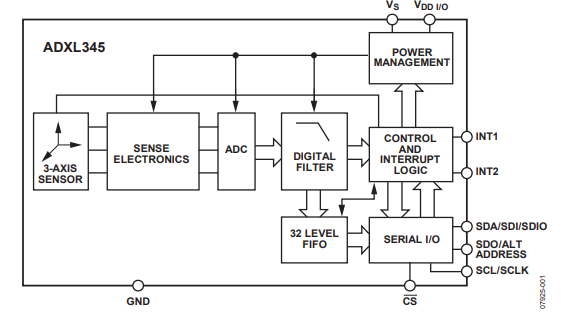


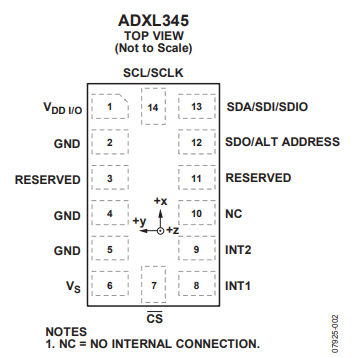
Four possible clock modes are available to program the clock edge used for data sampling and data toggling, as shown in Figure. The clock modes depend on two parameters: clock phase (CPHA) and clock polarity (CPOL). When CPOL is 0, the SCLK line is pulled low during idle time. When CPOL is 1, the SCLK line is pulled high during idle. When CPHA is 0, the first clock transition (either rising or falling) is the first data capture edge. When CPHA is 1, the second clock transition is the first capture edge. The combination of CPOL and CPHA selects the clock edge for transmitting and capturing data. The capture of the first bit is delayed a half cycle in mode 0 and 2.

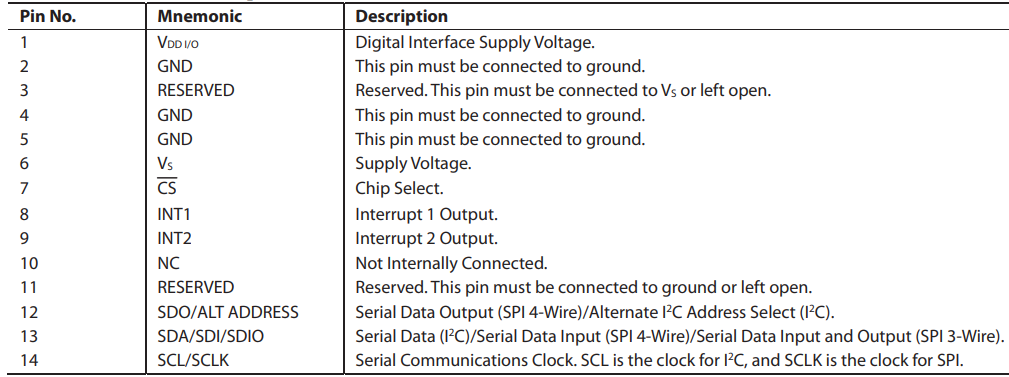


**4. ADXL345 General Description:**

The ADXL345 is a small, thin, ultralow power, 3-axis accelerometer 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 a SPI (3- or 4-wire) or I2C digital interface. The ADXL345 is well suited for mobile device applications. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (3.9 mg/LSB) enables measurement of inclination changes less than 1.0°. The ADXL345 is supplied in a small, thin, 3 mm × 5 mm × 1 mm, 14-lead, plastic package. The functional block diagram is shown below.

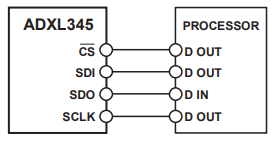


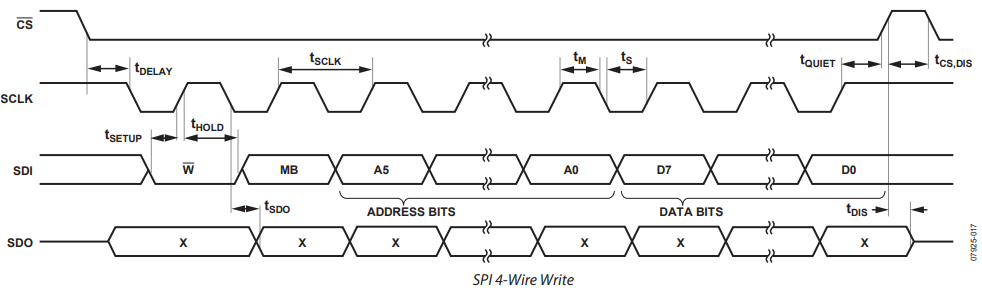


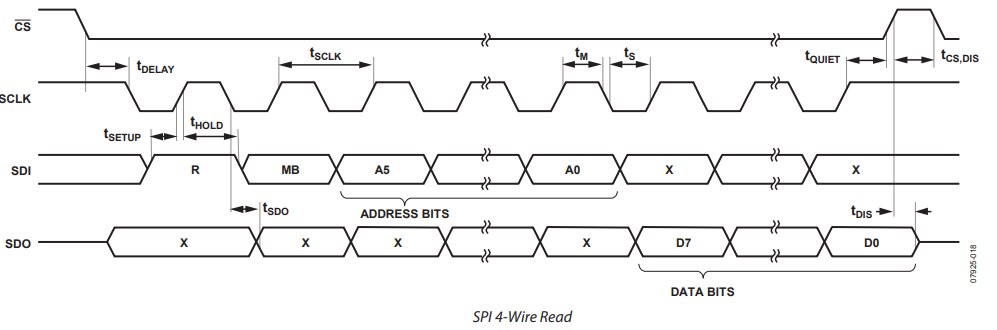


The ADXL345 automatically modulates its power consumption in proportion to its output data rate. If additional power savings is desired, a lower power mode is available. In this mode, the internal sampling rate is reduced, allowing for power savings in the 12.5 Hz to 400 Hz data rate range at the expense of slightly greater noise. To enter low power mode, set the LOW\_POWER bit (Bit 4) in the BW\_RATE register (Address 0x2C).

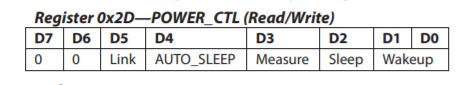
For SPI, either 3- or 4-wire configuration is possible, as shown in the connection diagrams. Clearing the SPI bit (Bit D6) in the DATA\_FORMAT register (Address 0x31) selects 4-wire mode, whereas setting the SPI bit selects 3-wire mode. The maximum SPI clock speed is 5 MHz with 100 pF maximum loading, and the timing scheme follows clock polarity (CPOL) = 1 and clock phase (CPHA) = 1. If power is applied to the ADXL345 before the clock polarity and phase of the host processor are configured, the CS pin should be brought high before changing the clock polarity and phase. When using 3-wire SPI, it is recommended that the SDO pin be either pulled up to VDD I/O or pulled down to GND via a 10 kΩ resistor.



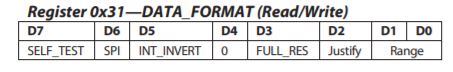




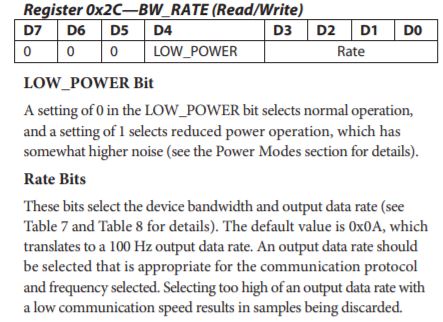
* According to the timing diagram, in order to read/write multiple bits in same operation, the 6th bit should be set. That’s why in the code, the register addresses are OR’d with 0x40, which sets the 6th bit high.
* Also, according to the timing diagram, the MSB defines whether it is a read or write operation. During write operation, the MSB should be 0 and for read operation, it should be HIGH. That’s why for read operations, the register addresses are OR’d with 0x80, setting the MSB.
* Now, the appropriate registers in the sensor should be configured, so that it can be used in data transfer. First, the device should be activated. So, the measure bit (D3) should be set to high in order to start the measurement. So, 0x08 should be written into register 0x2D to start the device.

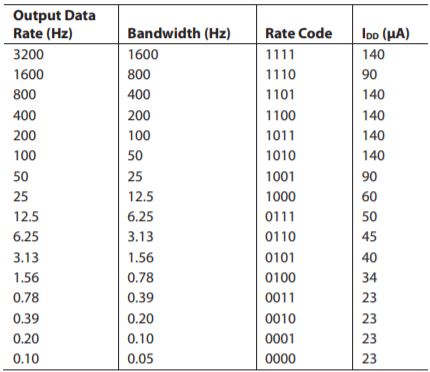


* Next, range of measurement has to be selected. Writing 01 in the LSB sets it to +/-4g. Also, the interrupt configuration by default is active high. So, setting the bit D5 makes it active low. So, if interrupt is needed in active low, 0x21 has to be written in register 0x31.

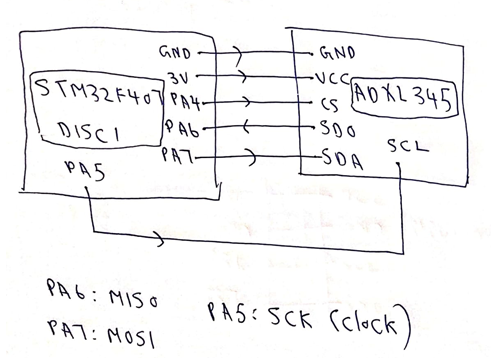


* The register corresponding to sampling rate is 0x2C. Bits D0 to D3 correspond to sampling rate. So, from the below table taken from the datasheet, we can set the lower nibble so as to get the required sampling rate.





**5. Hardware Schematic Diagram:**



**6. Program for ADXL345 Interfacing:**

#include "stm32f4xx.h"

#define add 0x32

int8\_t array[6],buf;

int16\_t x,y,z;

float xacc,yacc,zacc;

//ADXL345 Analog Devices

void GPIO\_Config(void)

{

//PA5,6,7-CLK,MISO,MOSI

RCC->AHB1ENR |=(1UL<<0);//Enable Port A clock

GPIOA->MODER |=(2UL<<10);//Set PA5,6,7 to AF

GPIOA->MODER |=(2UL<<12);

GPIOA->MODER |=(2UL<<14);

GPIOA->AFR[0] |=(5UL<<20); //enable SPI CLK to PA5

GPIOA->AFR[0] |=(5UL<<24); //enable MISO to PA6

GPIOA->AFR[0] |=(5UL<<28); //enable MOSI to PA7

//PA4-CS

GPIOA->MODER |=(1UL<<8);//PA4 as output

}

void SPI\_Config(void)

{

RCC->APB2ENR |=(1UL<<12);//Enable SPI clock

SPI1->CR1 |=(2UL<<3);//Set baud rate as 2Mbit/s

SPI1->CR1 |=(1UL<<2);//Set as Master mode

SPI1->CR1 |=(1UL<<1);//Set clock polarity as HIGH

SPI1->CR1 |=(1UL<<0);//Set clock phase as HIGH

SPI1->CR1 |=(3UL<<8);//Should be set HIGH

SPI1->CR1 |=(1UL<<6);//Start SPI

SPI1->CR2 = 0x0000;//Motorola format

}

int8\_t SPI\_Send (uint8\_t byte)

{

SPI1->DR = byte;

while ((SPI1->SR) & (1<<7)); /\* Wait for send to finish \*/

buf=SPI1->DR;

return buf;

}

void Accel\_Write(uint8\_t address,uint8\_t val)

{

GPIOA->ODR |=(1UL<<21);//Select accelerometer

SPI\_Send(address);

SPI\_Send(val);

GPIOA->ODR |=(1UL<<5);//Disconnect accelerometer

}

void ADXL\_Config(void)

{

Accel\_Write(0x2D,0x00);// Reset before configuring power register

Accel\_Write(0x2D,0x08);//Configure the power register, and turn on the device

// Configuring the data format register

//The 5th bit corresponds to setting interrupt to active low if set

Accel\_Write(0x31,0x01);//Lower nibble:Selecting +/- 4g range and 4 wire SPI mode

//Configuring sampling rate to 100hz

Accel\_Write(0x2C,0x0A);

}

void Accel\_Read(uint8\_t address)

{

//X0

GPIOA->ODR |=(1UL<<21);//Select accelerometer

address |=0x80;//Read operation

SPI\_Send(address);

array[0]=SPI\_Send(0);

GPIOA->ODR |=(1UL<<5);//Disconnect accelerometer

//X1

GPIOA->ODR |=(1UL<<21);//Select accelerometer

++address;

SPI\_Send(address);

array[1]=SPI\_Send(0);

GPIOA->ODR |=(1UL<<5);//Disconnect accelerometer

//Y0

GPIOA->ODR |=(1UL<<21);//Select accelerometer

++address;

SPI\_Send(address);

array[2]=SPI\_Send(0);

GPIOA->ODR |=(1UL<<5);//Disconnect accelerometer

//y1

GPIOA->ODR |=(1UL<<21);//Select accelerometer

++address;

SPI\_Send(address);

array[3]=SPI\_Send(0);

GPIOA->ODR |=(1UL<<5);//Disconnect accelerometer

//Z0

GPIOA->ODR |=(1UL<<21);//Select accelerometer

++address;

SPI\_Send(address);

array[4]=SPI\_Send(0);

GPIOA->ODR |=(1UL<<5);//Disconnect accelerometer

//Z1

GPIOA->ODR |=(1UL<<21);//Select accelerometer

++address;

SPI\_Send(address);

array[5]=SPI\_Send(0);

GPIOA->ODR |=(1UL<<5);//Disconnect accelerometer

x=((array[1]<<8)|array[0]);

y=((array[3]<<8)|array[2]);

z=((array[5]<<8)|array[4]);

xacc=x\*0.0078;

yacc=y\*0.0078;

zacc=z\*0.0078;

}

int main()

{

GPIO\_Config();

SPI\_Config();

ADXL\_Config();

while(1)

{

Accel\_Read(add);

}

}

**Conclusion:**

SPI communication protocol was used to read data from ADXL345 sensor.