ENGR 498: Design for the Internet of Things - SPI

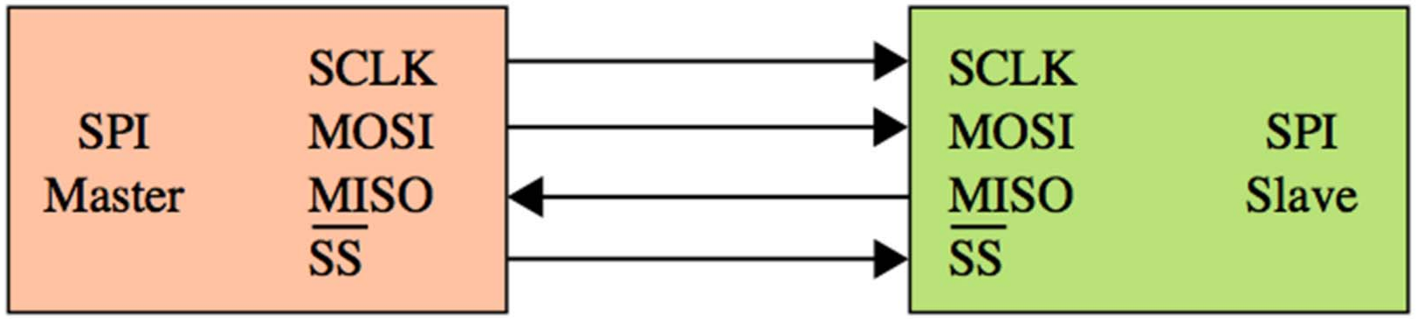
Dr. Jason Forsyth
Department of Engineering
James Madison University

General SPI Overview

Serial Peripheral Interface (SPI)

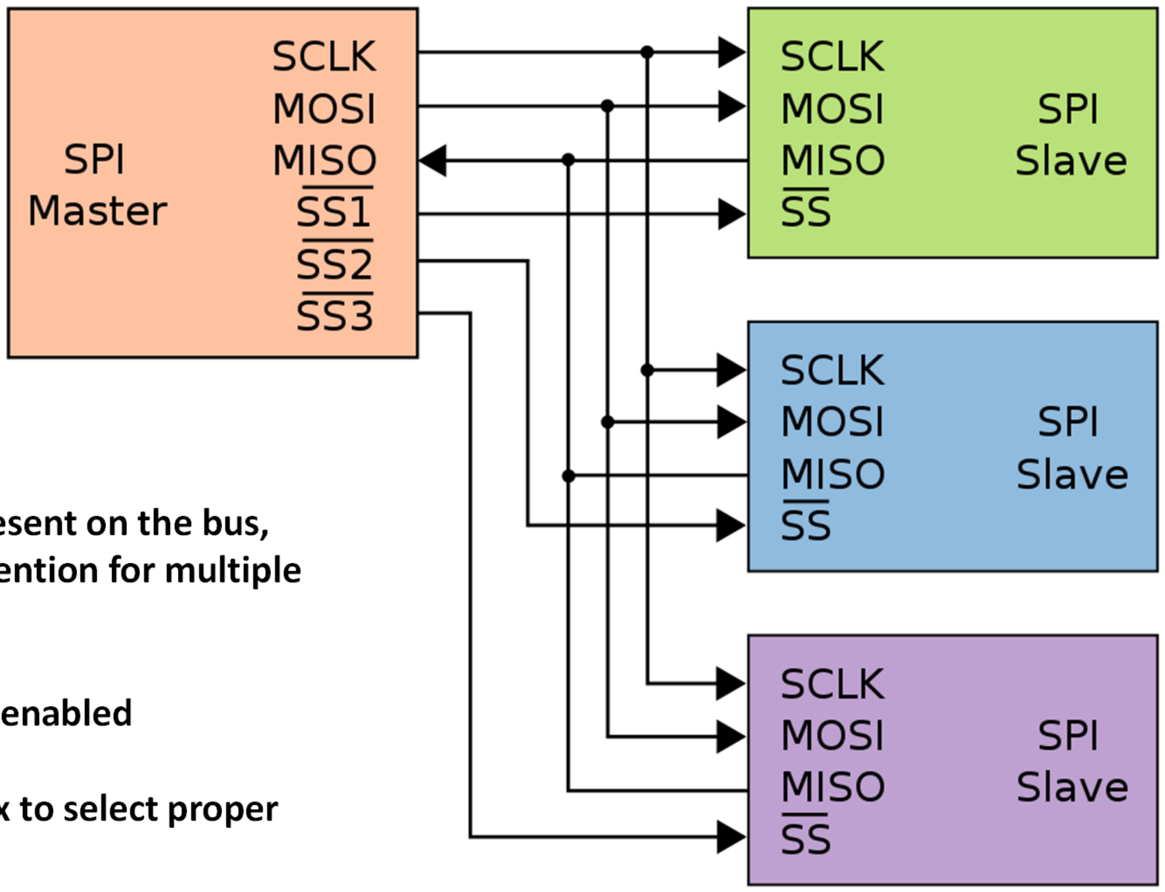
- SPI is three/four wire synchronous interface for external devices
- Communication between *master* and *slave* nodes controlled by master
- Shared clock line, individual data input and output lines
- Multiple devices available on bus and controlled through *chip select* line

SPI Bus – Single Master/Slave



- Supports Full-Duplex communication via individual serial data lines
  - MOSI: Master-Out Slave-In
  - MISO: Master-In Slave-Out
- Communication governed by shared clock line and chip enable/select
  - SCK: Serial Clock
  - CS/SS: Chip Select or Slave Select (generally active low)

SPI Bus – Multiple Slaves

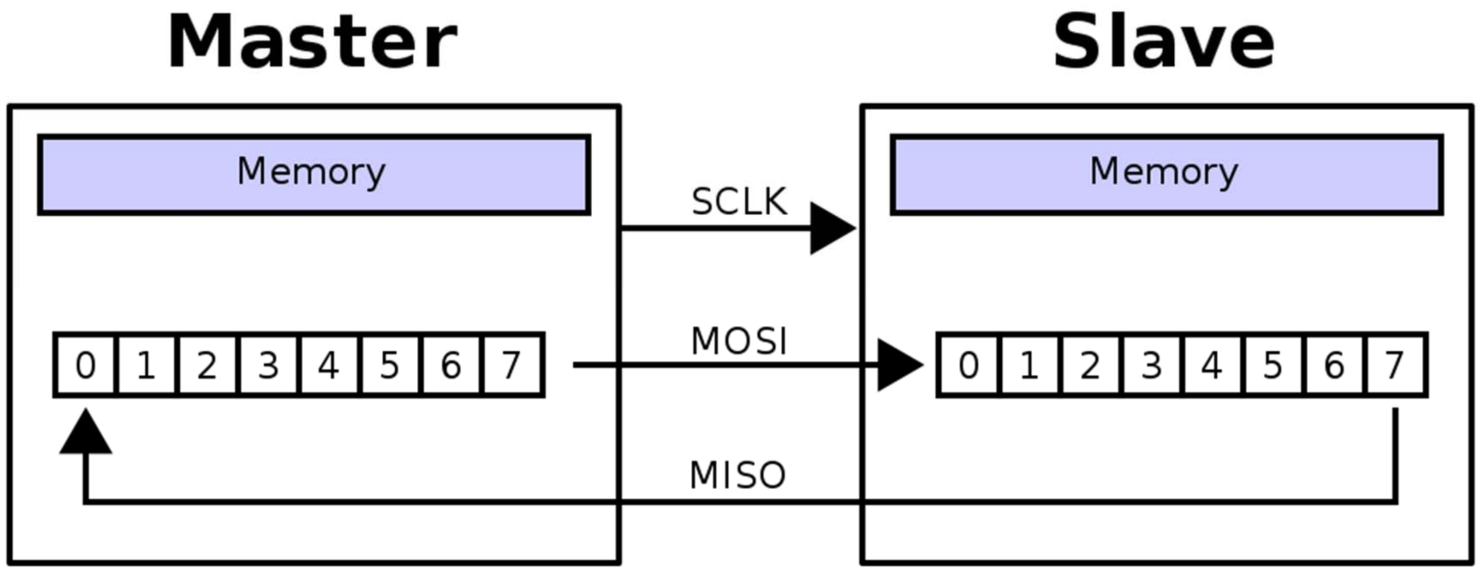


Multiple slaves can be present on the bus, however there is <u>no</u> convention for multiple masters

Slaves ignore MOSI if not enabled

CS lines must be multiplex to select proper slave

SPI Full Duplex Communication



- Master and slave share connection between shift registers
- Data is shifted OUT of Master and IN to Slave (MOSI) at the same time data is shifted OUT of Slave and IN to Master (MISO)

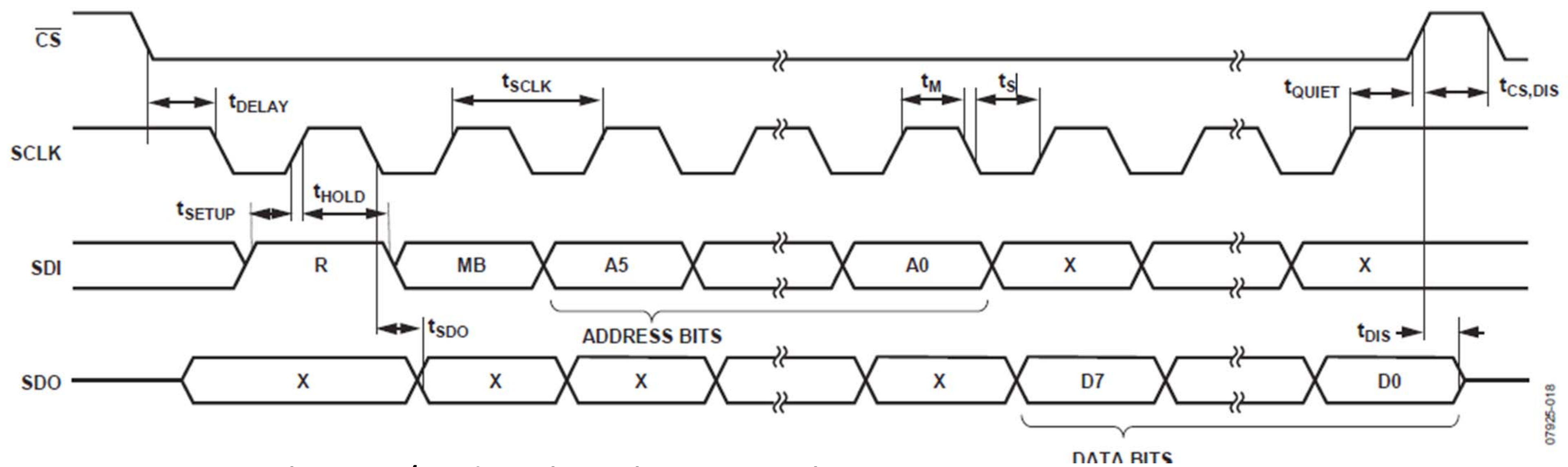
SPI Bus Transaction

1. Master asserts the Chip Select (CS) line.
2. Master drives clock and MOSI lines. Data transfer occurs.
3. Master de-asserts the Chip Select (CS)

Slave device configuration determines what data should be sent/received to make the peripheral *do something*.

Data will always be received. May be *bogus* information.

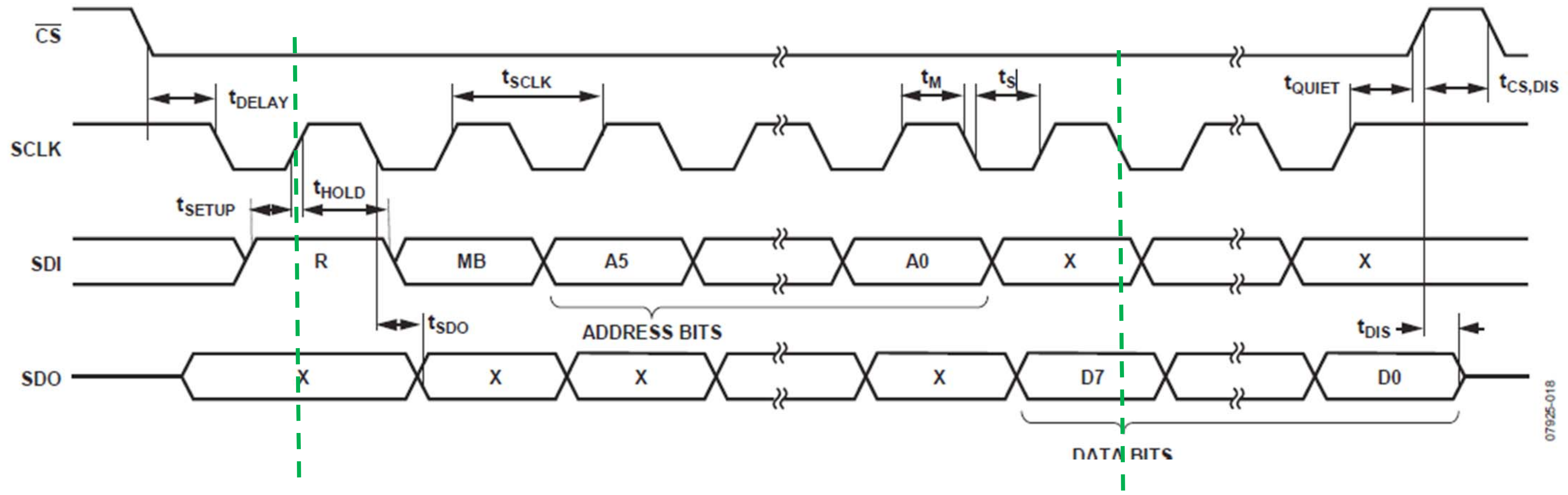
Sample SPI Transaction from ADXL345



- Transaction begins/end with CS line going low
- Data from Master shown on MOSI/SDI line; data from Slave on MISO/SDO line

- On which clock edge does is data read?

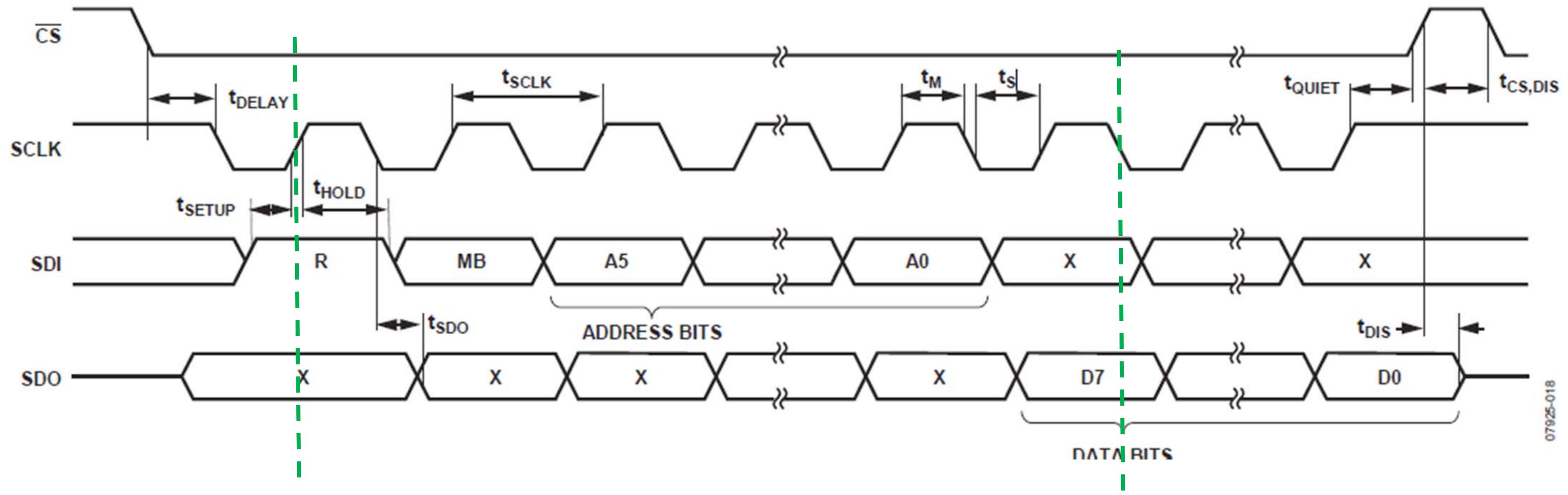
Which clock edge appears most “stable”?



Rising Edge?

Falling Edge?

Which clock edge appears most “stable”?



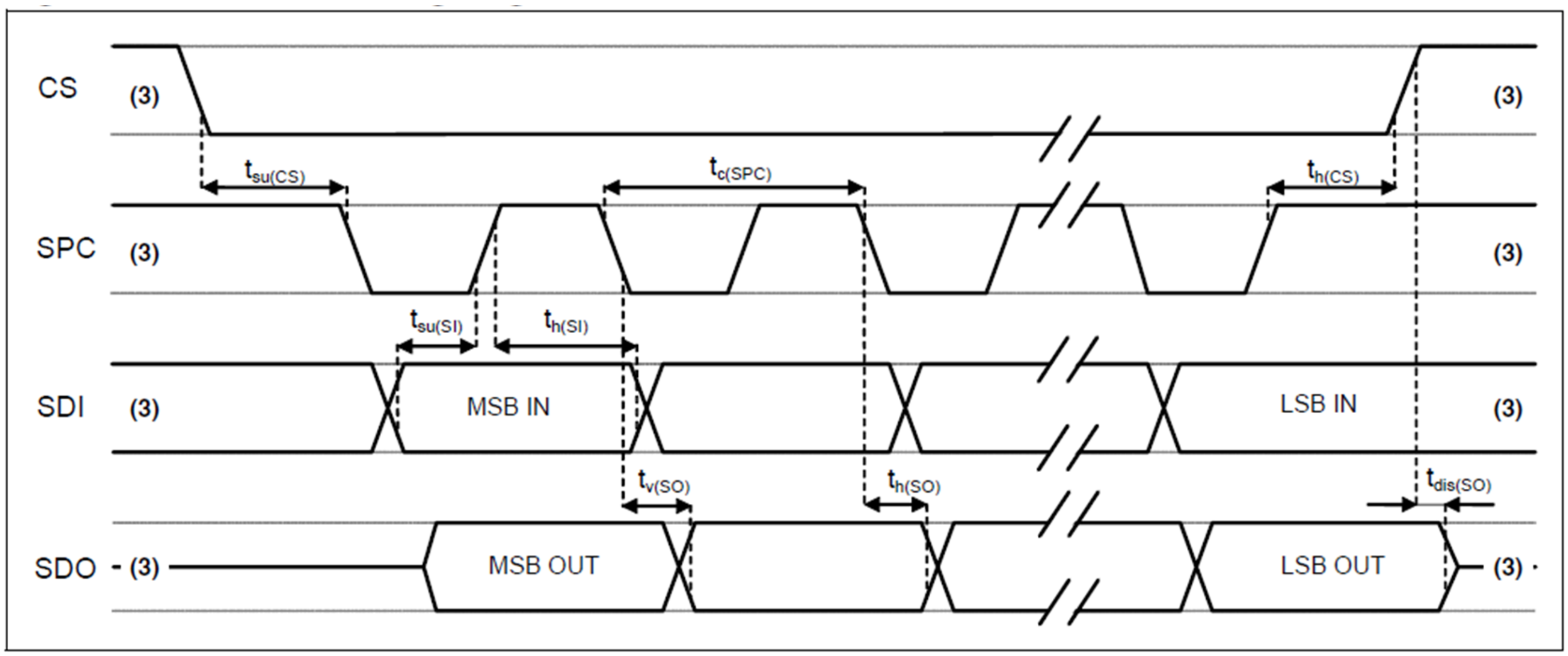
- Whether data is sampled on the rising/trailing or falling/leading edge is dependent upon the device.
- For this device (ADXL345) data is sampled on the TRAILING/RISING edge

Clock Phase and Polarity

- Whether data is sampled on the leading or trailing edge is called Clock Phase (CPHA)
  - CPHA = 0: sample on the leading edge
  - CPHA = 1: sample on the trailing edge

- Whether the leading/falling edge is rising/falling dependent upon Clock Polarity (CPOL)
  - CPOL = 0: clock is held LOW when idle
  - CPOL = 1: clock is held HIGH when idle

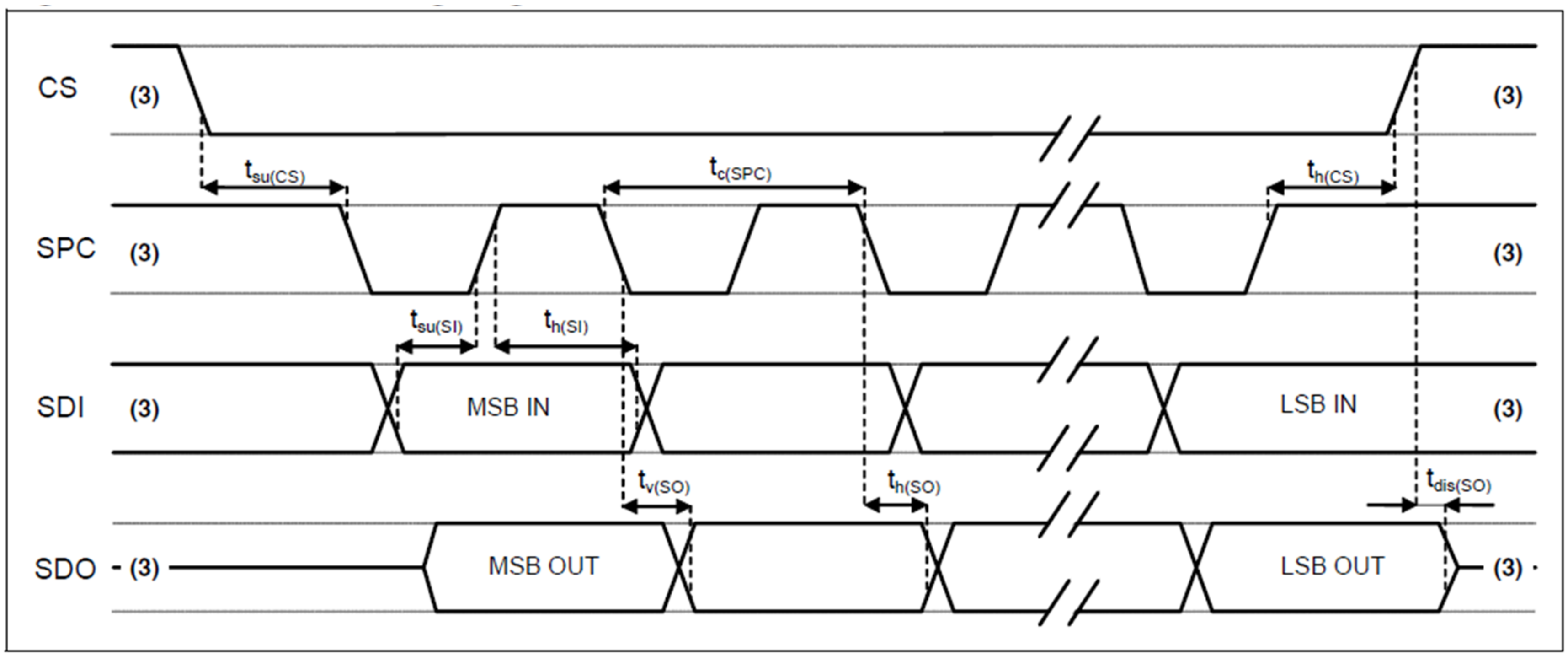
What is the CPOL of this transaction?



CPOL = 0: Clock is idle LOW

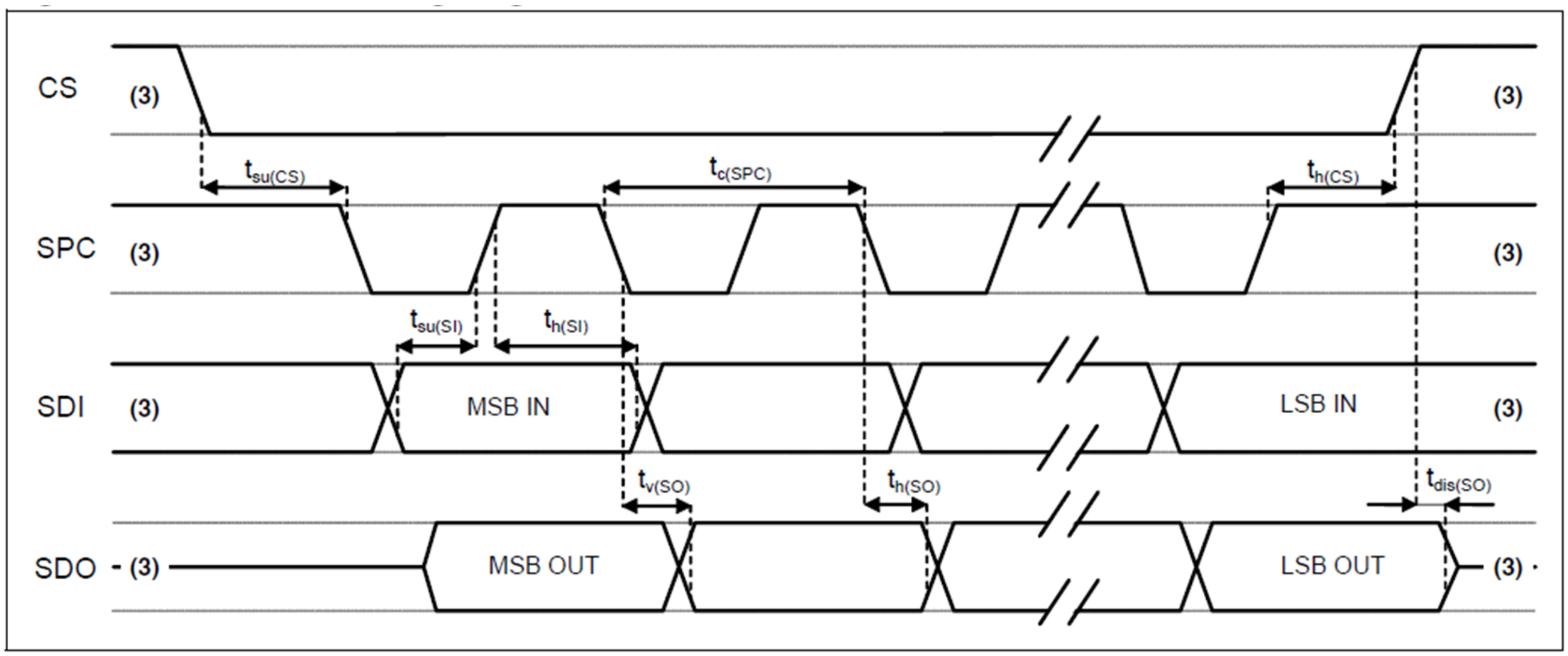
CPOL = 1: Clock is idle HIGH

What is the CPOL of this transaction?



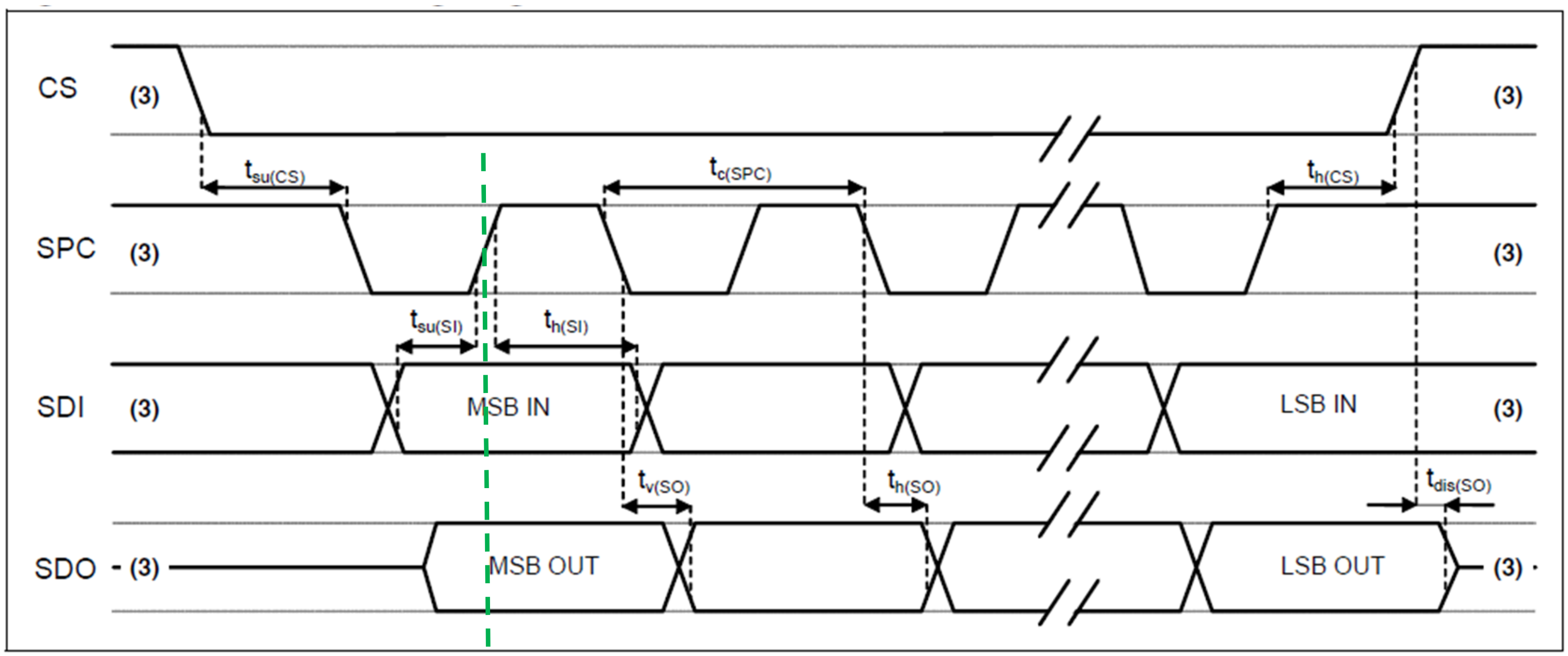
CPOL = 1 as clock is held HIGH while Idle

What is the CPHA of this transaction?



CPHA = 0: data is sampled on the leading edge
CPHA = 1: data is sampled on the trailing edge

What is the CPHA of this transaction?



CPHA = 1: data is read on the second edge (rising)

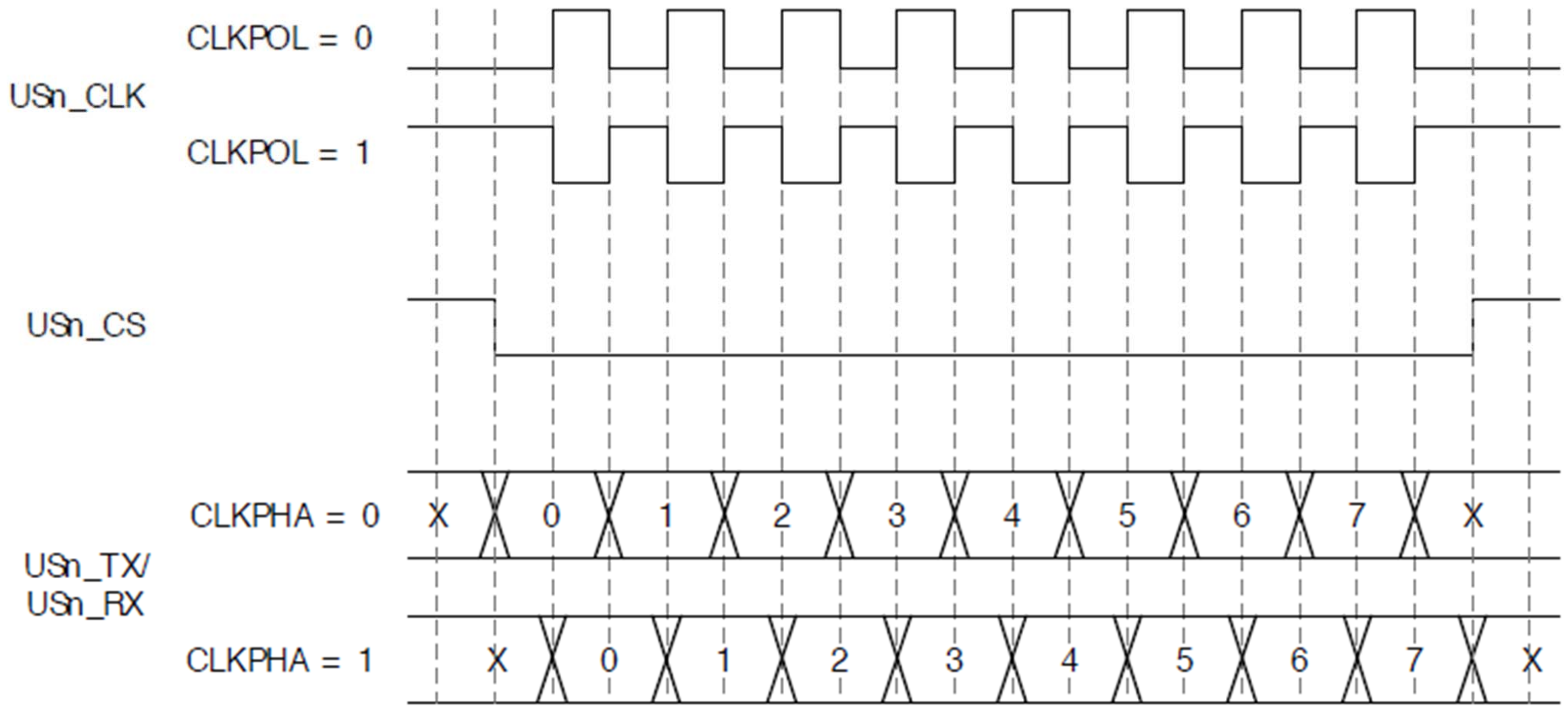
CPHA, CPOL, and SPI Mode

SPI Mode	CLKPOL	CPHA
0	0 – Idle LOW	0 – Sample on Leading
1	0 – Idle LOW	1 – Sample on Trailing
2	1 – Idle HIGH	0 – Sample on Leading
3	1 – Idle HIGH	1 – Sample on Trailing

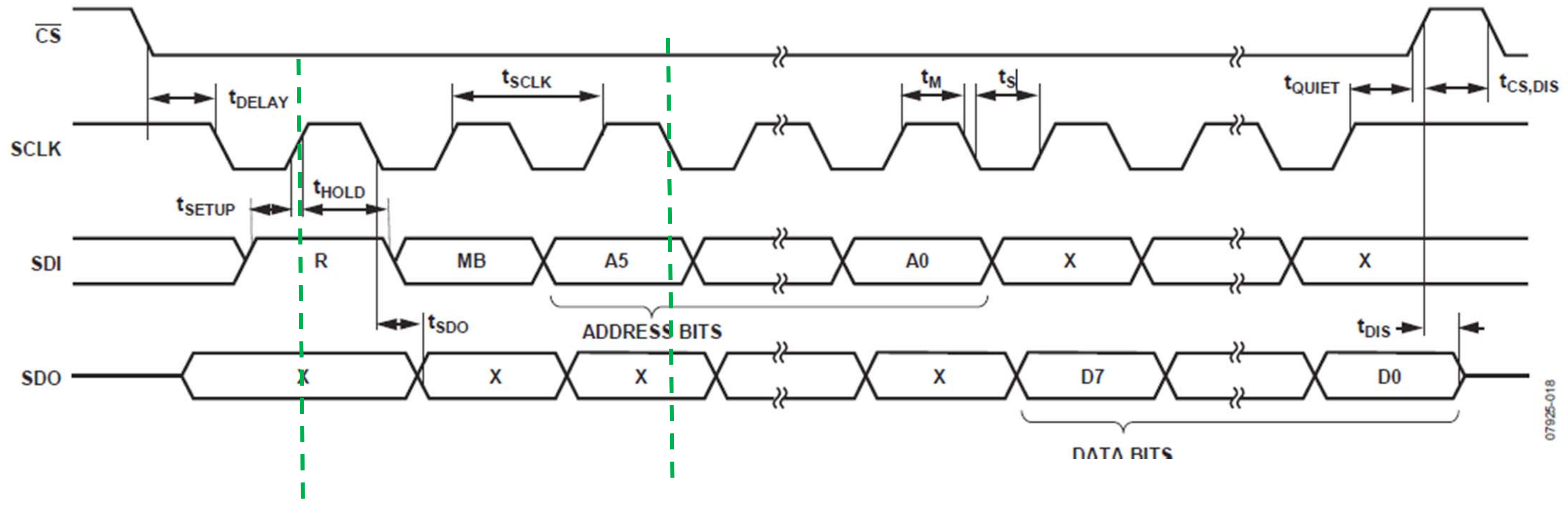
Typical

Combinations of CLKPOL and CPHA combined into SPI Modes

SPI Mode Diagram (not as helpful?)



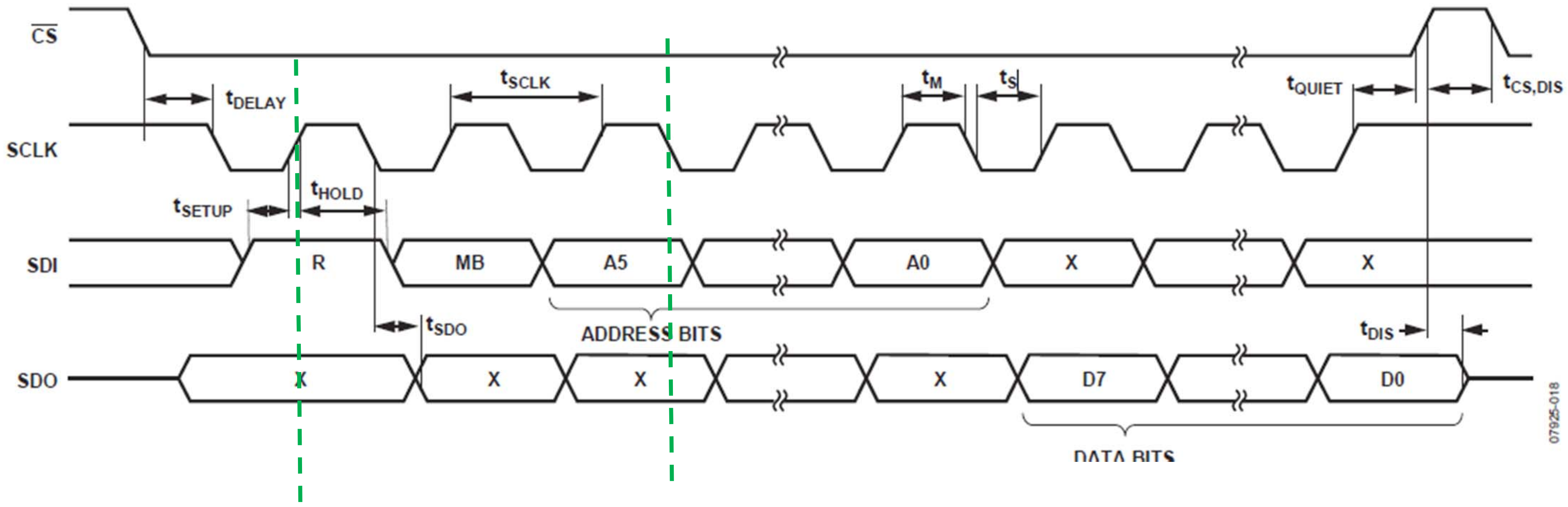
What is the SPI Mode of this transaction?



CPOL = 0: Clock is idle LOW
CPOL = 1: Clock is idle HIGH

CPHA = 0: data is sampled on the leading edge
CPHA = 1: data is sampled on the trailing edge

What is the SPI Mode of this transaction?



Overall would be in SPI Mode 3 (CPOL=1 and CPHA=1)

Interfacing with Peripherals

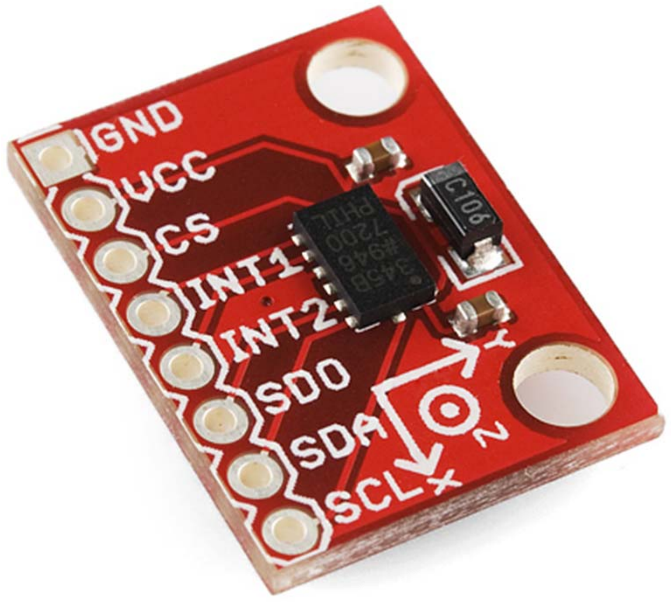
Interacting with Peripherals

- SPI specifies *how* to communicate with peripherals (voltage levels, signaling…etc.) it doesn't specific *what* to say (what's contained in a packet).

- Most devices have multiple functionalities that are accessed through on-board registers.

- Send specific SPI packets to read/write from these devices.

Analog Devices ADXL 345



- 3-axis accel with 13-bit resolution up to +/- 16g
- Supports SPI and I2C interfaces
- Provides interrupts to "wake up" processor on new data, tap, rotation…etc.

FUNCTIONAL BLOCK DIAGRAM

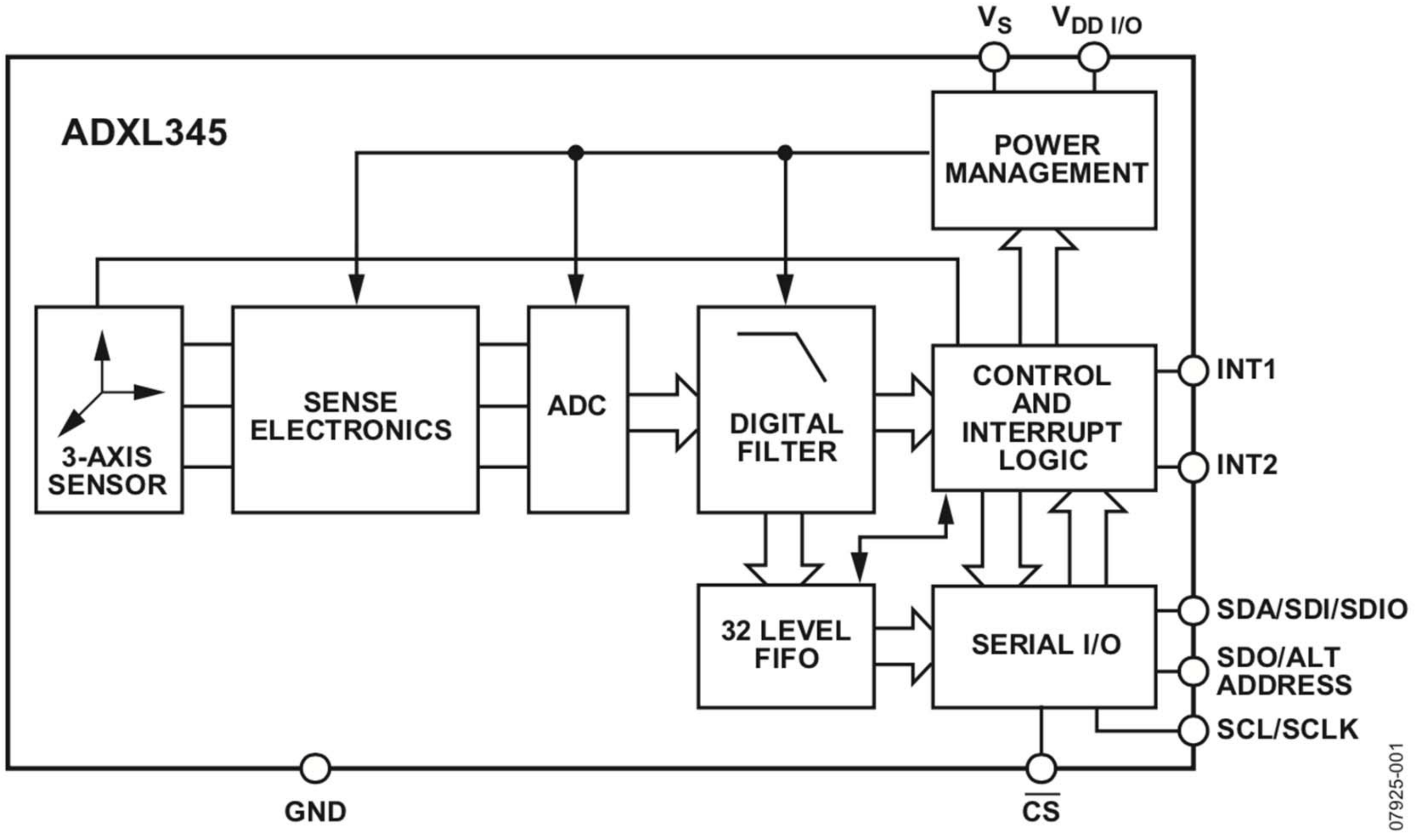


Figure 1.

ADXL345 – SPI Interfacing

- Maximum clock is 5MHz with CPOL=1 (clock is idle HIGH) and CHPA=1 (data sampled on rising edge)

- Read/write to on-board registers via standard message format

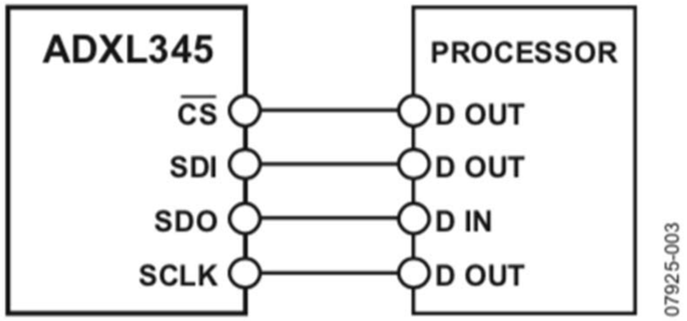


Figure 35. 4-Wire SPI Connection Diagram

$$[\underbrace{R/\overline{w}}_{\text{Read or write to address}},\ \underbrace{MB}_{\text{Multibyte operation}},\ \underbrace{A5\text{-}A0}_{\text{Internal reg. address}}]$$

ADXL345 – SPI Messaging (General)

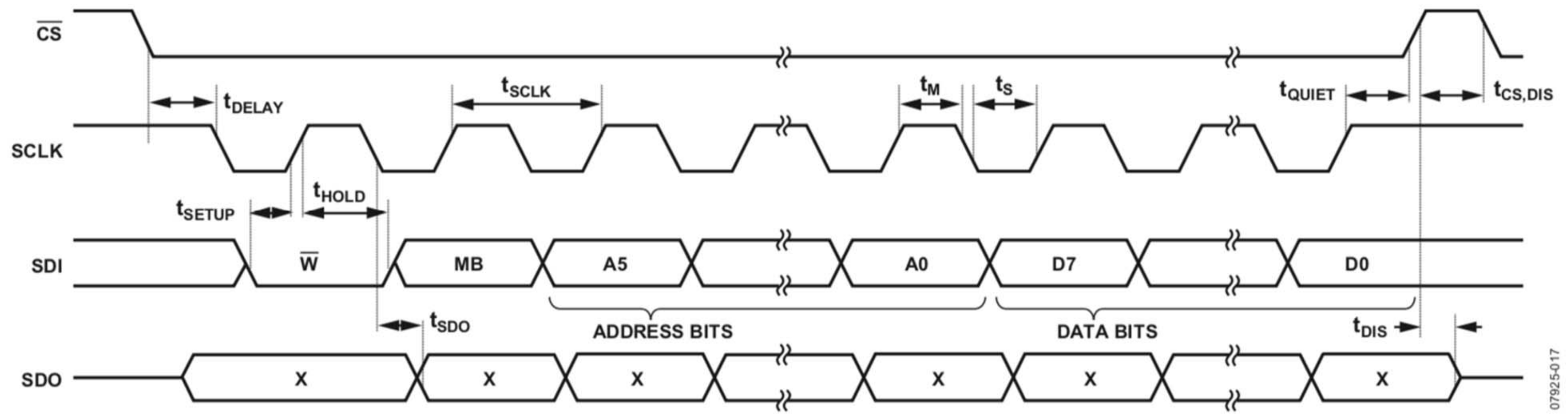


Figure 37. SPI 4-Wire Write

Send read/write packet then follow by data (in the case of a write) or 0x0 (in the case of a read).

Must assert/de-assert CS after the end of each transaction.

If MB used then can continually read/write to sequential internal addresses

ADXL345 – SPI Messaging (Read)

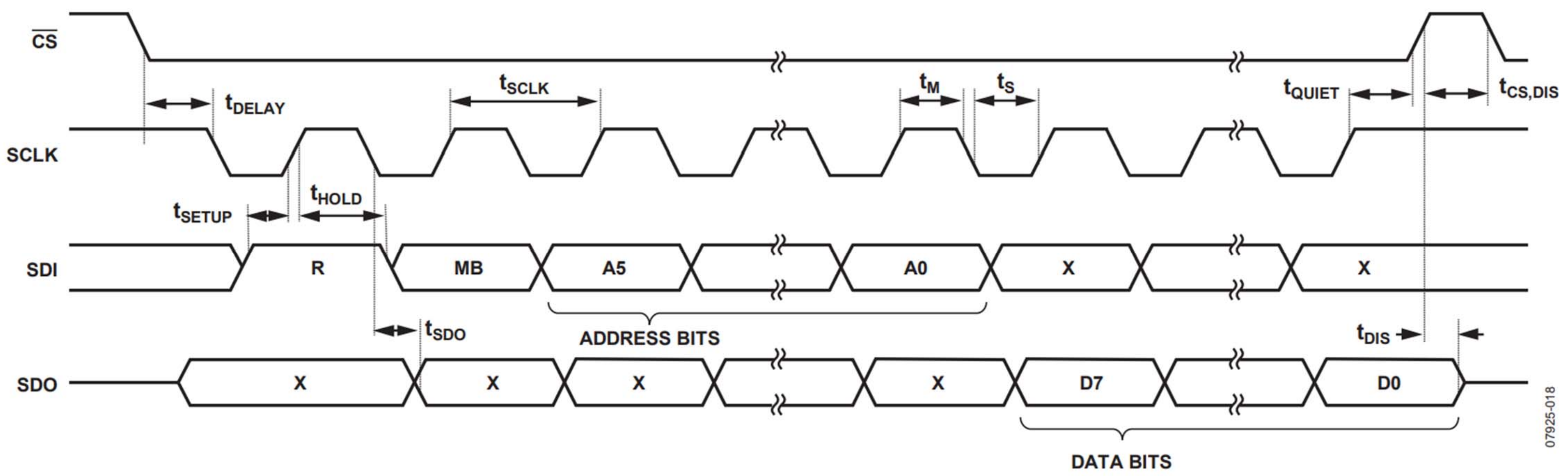


Figure 38. SPI 4-Wire Read

ADXL345 – SPI Messaging (Write)

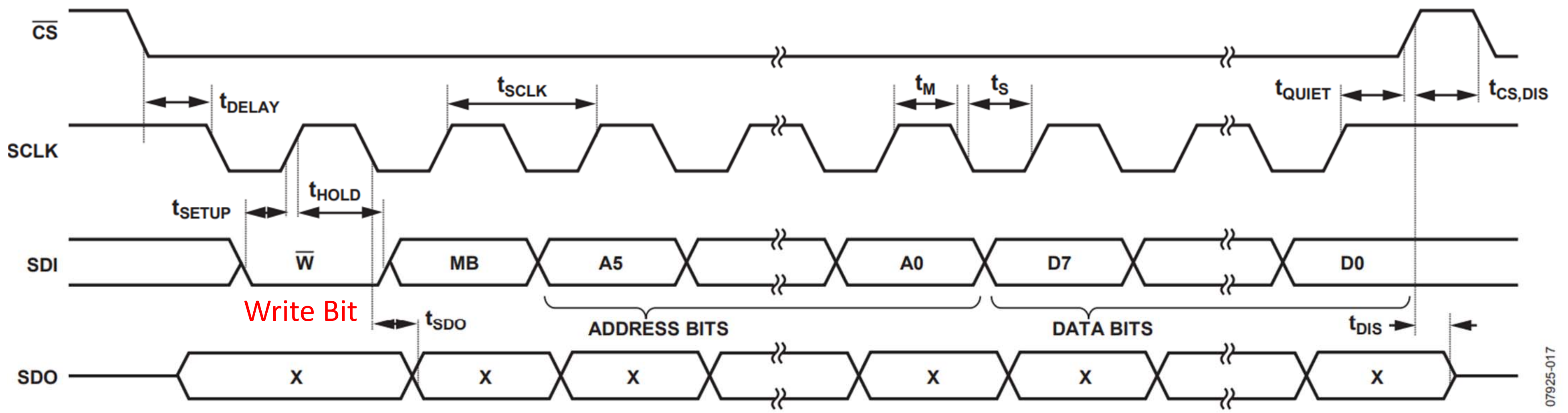


Figure 37. SPI 4-Wire Write

Communicating with the ADXL 345

- To communicate with the ADXL you must *read from* and *write to* different registers.

- Write to registers to: configure sample rate and resolution, set the power mode, control power levels…etc.

- Read from register to: read samples, check configuration…etc.

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

To Perform A Register READ

- Lower Chip Select pin to begin communication
- Transmit READ packet that specifies address to be read
- Transmit “bogus data” (0x0) so that the communication continues
- Received data will be placed into SPI buffer
- Raise Chip Select pin to end communication

To Perform A Register WRITE

- Lower Chip Select pin to begin communication
- Transmit WRITE packet that specifies address to be written
- Transmit new data that will be placed in the specific register
- SPI will “receive” data but it will be bogus/unknown
- Raise Chip Select pin to end communication

Construct a Packet to READ from Register 0x0

$R/\overline{w}$	MB	A5	A4	A3	A2	A1	A0
1	0	0	0	0	0	0	0

$[R/\overline{w}, \text{MB}, \text{A5-A0}]$

Read or write to address

Multibyte operation

Internal reg. address

Construct a Packet to READ from Register 0x5

R/$\overline{w}$	MB	A5	A4	A3	A2	A1	A0
1	0	0	0	0	1	0	1

[R/$\overline{w}$, MB, A5-A0]

Read or write to address

Multibyte operation

Internal reg. address

Construct a Packet to WRITE to Register 0x13

$R/\overline{w}$	MB	A5	A4	A3	A2	A1	A0
0	0	0	1	0	0	1	1

[$R/\overline{w}$, MB, A5-A0]

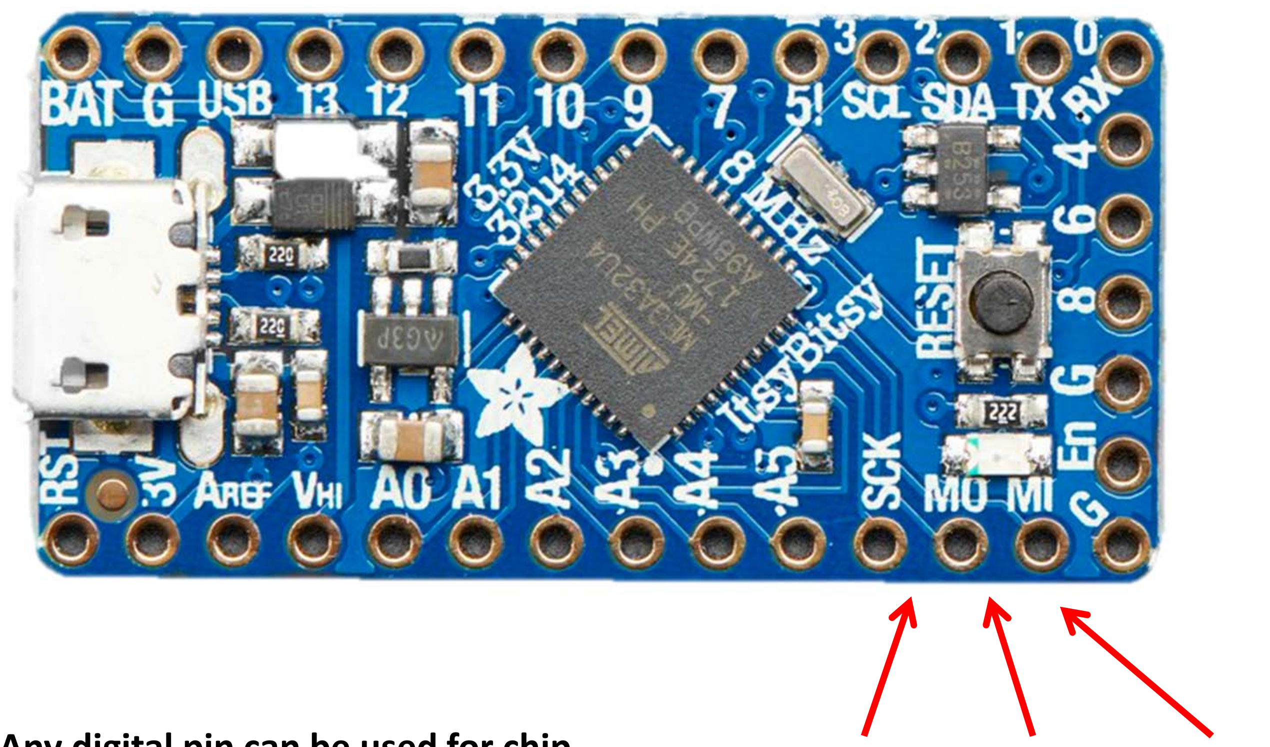
Read or write to address

Multibyte operation

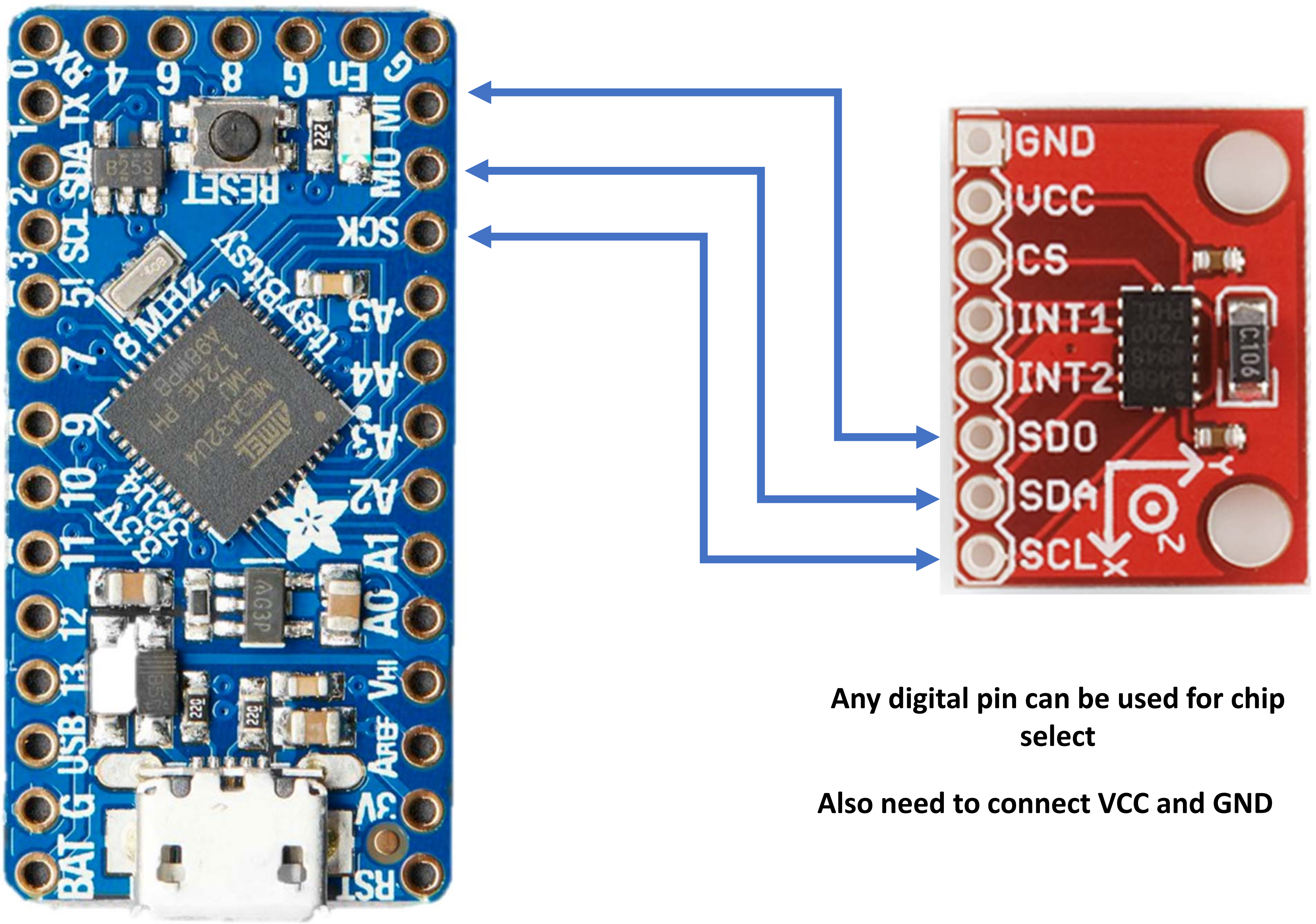
Internal reg. address

SPI on the Arduino

SPI Pins



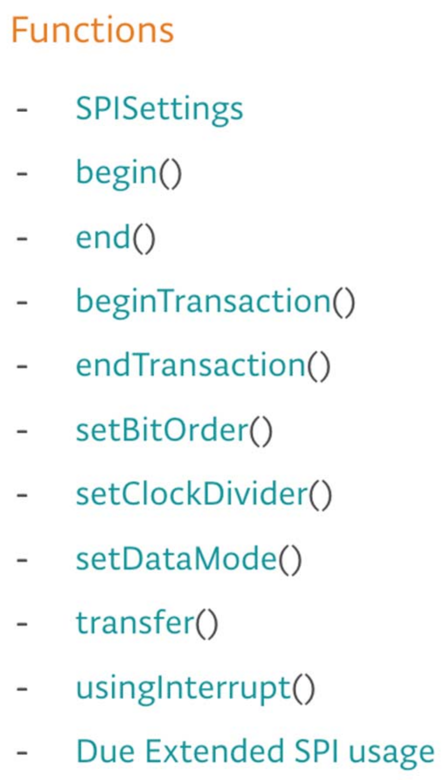
Any digital pin can be used for chip select



Any digital pin can be used for chip select

Also need to connect VCC and GND

Arduino SPI Library



Functions

- SPISettings
- begin()
- end()
- beginTransaction()
- endTransaction()
- setBitOrder()
- setClockDivider()
- setDataMode()
- transfer()
- usingInterrupt()
- Due Extended SPI usage

Arduino SPI Library

Functions

- ~~SPISettings~~
- begin() Use this to initialize the SPI to default values. Must occur first or device is disabled.
- ~~end()~~
- ~~beginTransaction()~~
- ~~endTransaction()~~
- ~~setBitOrder()~~
- ~~setClockDivider()~~
- setDataMode() Use this after begin() to set the CHPOL and CHPA modes
- transfer() Use this to send/receive data from the device
- ~~usingInterrupt()~~
- ~~Due Extended SPI usage~~

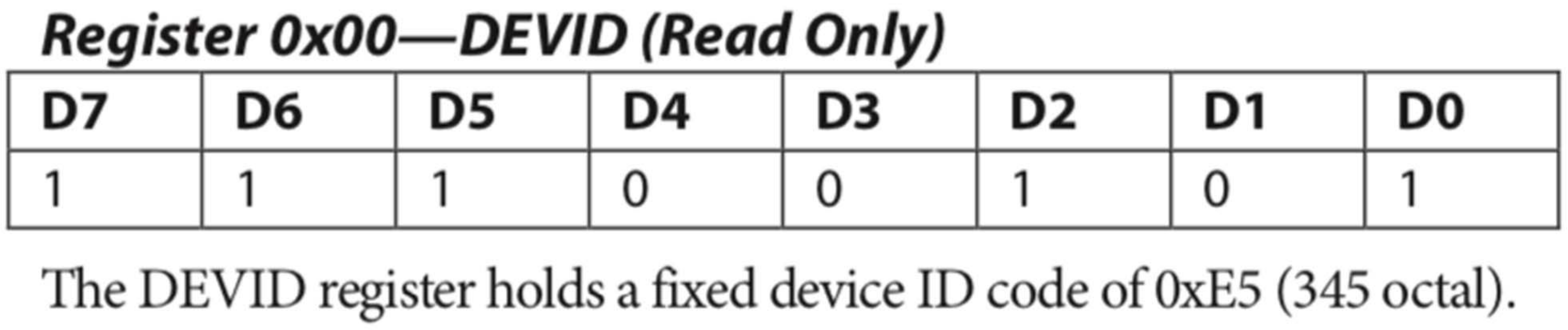
Configuring the ADXL345

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

Device ID Register



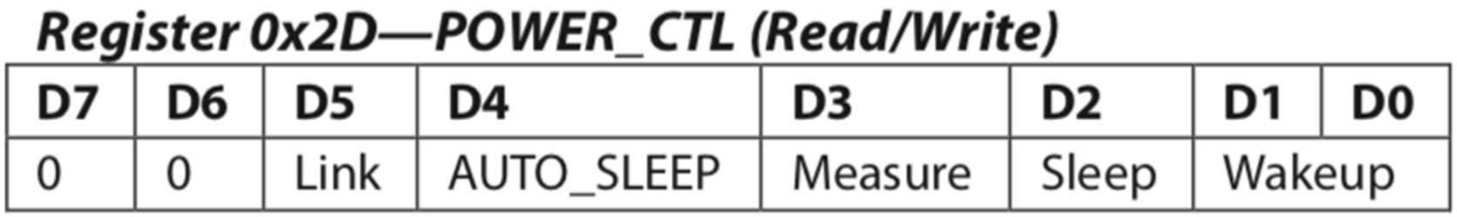
Register 0x00—DEVID (Read Only)

D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	0	0	1	0	1

The DEVID register holds a fixed device ID code of 0xE5 (345 octal).

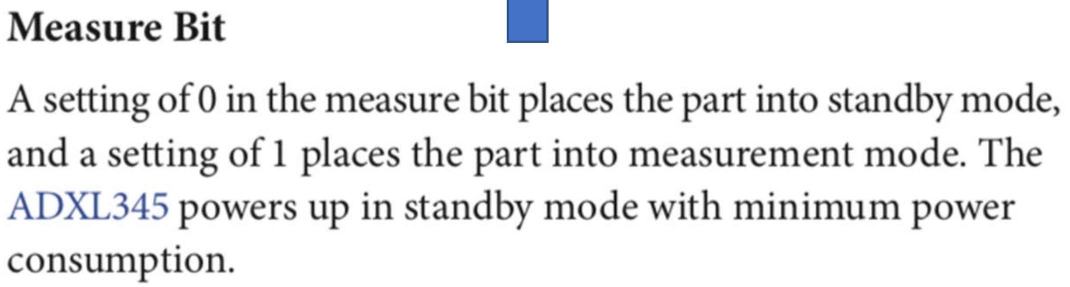
Try reading from this register to see if your SPI transactions work.
Known value of 11100101b or 0xE5

Power Control Register – Begin Sampling



Register 0x2D—POWER_CTL (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	AUTO_SLEEP	Measure	Sleep	Wakeup	

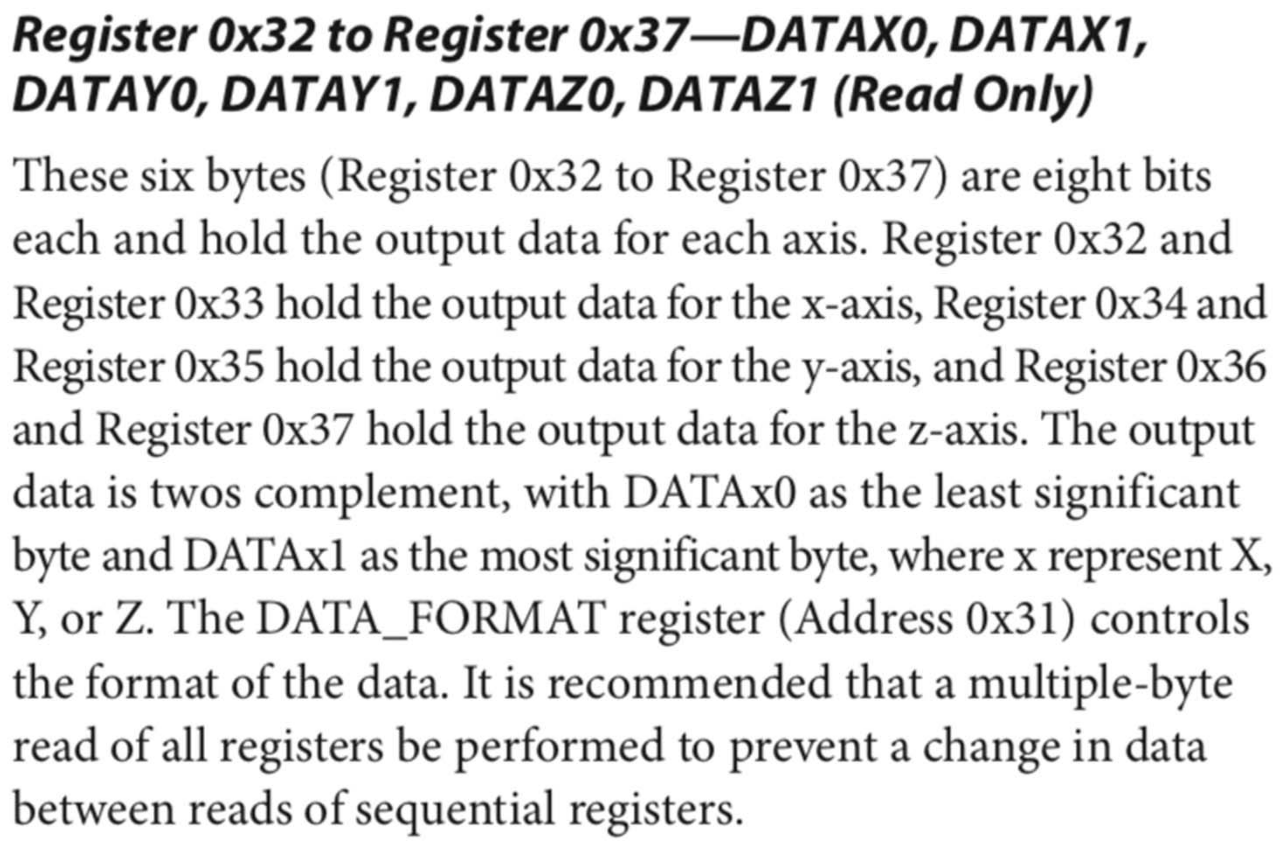


Measure Bit

A setting of 0 in the measure bit places the part into standby mode, and a setting of 1 places the part into measurement mode. The ADXL345 powers up in standby mode with minimum power consumption.

Default value is 0x0

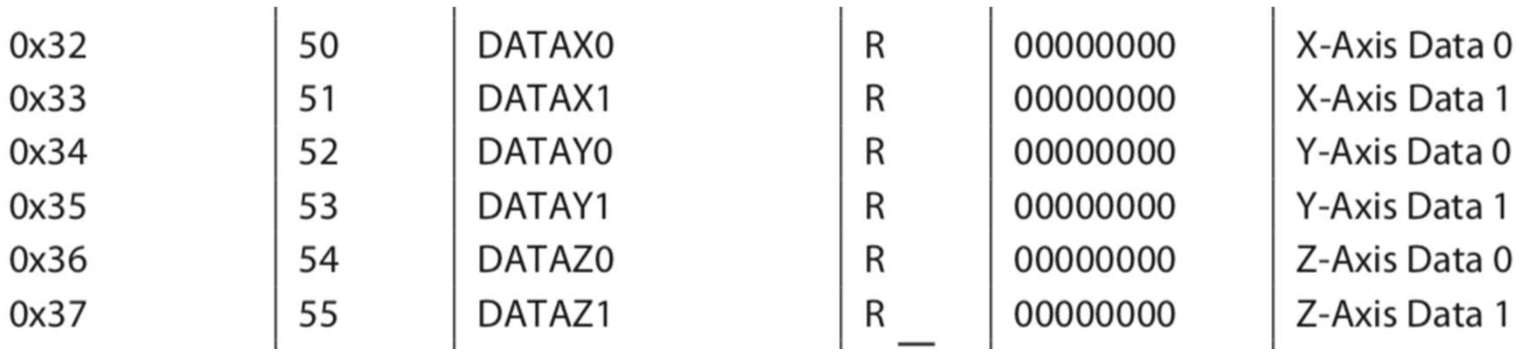
Reading Data from Axes



Register 0x32 to Register 0x37—DATAX0, DATAX1, DATAY0, DATAY1, DATAZ0, DATAZ1 (Read Only)

These six bytes (Register 0x32 to Register 0x37) are eight bits each and hold the output data for each axis. Register 0x32 and Register 0x33 hold the output data for the x-axis, Register 0x34 and Register 0x35 hold the output data for the y-axis, and Register 0x36 and Register 0x37 hold the output data for the z-axis. The output data is twos complement, with DATAx0 as the least significant byte and DATAx1 as the most significant byte, where x represent X, Y, or Z. The DATA_FORMAT register (Address 0x31) controls the format of the data. It is recommended that a multiple-byte read of all registers be performed to prevent a change in data between reads of sequential registers.

Read Data from the Axes

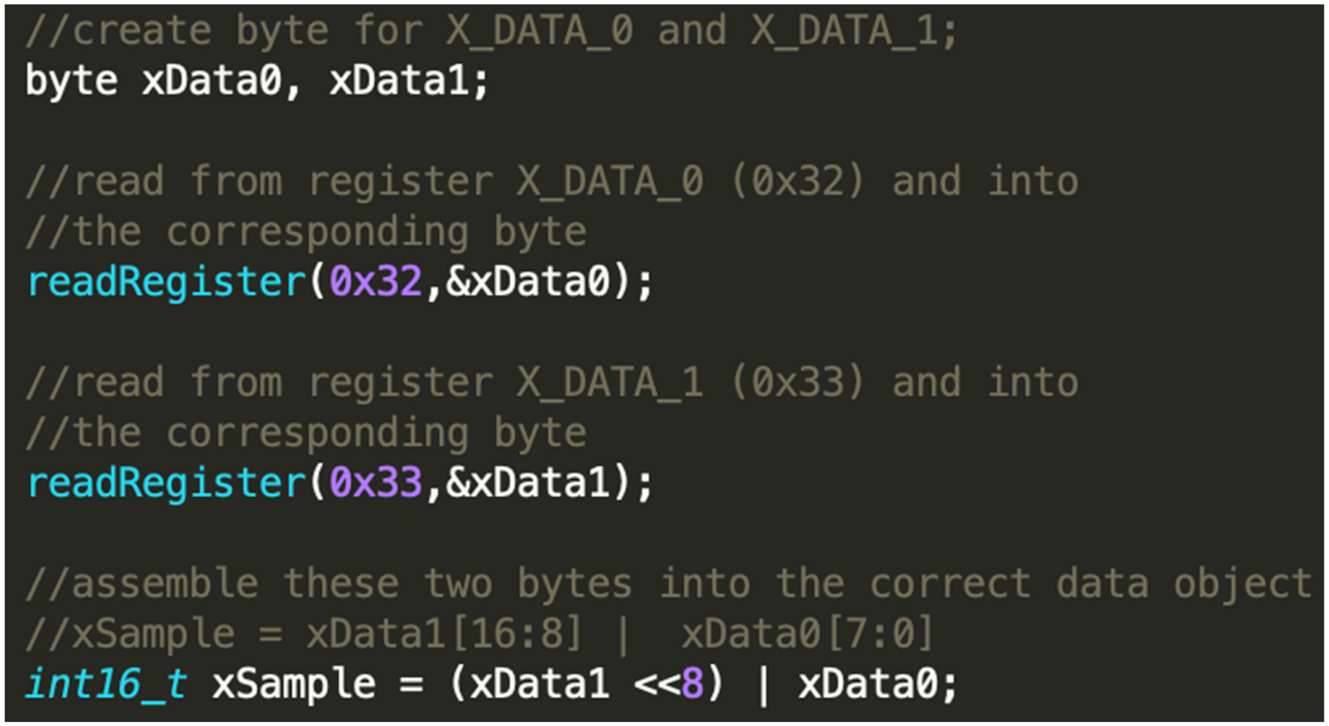


0x32	50	DATAX0	R	00000000	X-Axis Data 0
0x33	51	DATAX1	R	00000000	X-Axis Data 1
0x34	52	DATAY0	R	00000000	Y-Axis Data 0
0x35	53	DATAY1	R	00000000	Y-Axis Data 1
0x36	54	DATAZ0	R	00000000	Z-Axis Data 0
0x37	55	DATAZ1	R	00000000	Z-Axis Data 1

Lower Byte for X-axis

Upper Byte for X-axis

Reading from DATA Register in Practice



```
//create byte for X_DATA_0 and X_DATA_1;
byte xData0, xData1;

//read from register X_DATA_0 (0x32) and into
//the corresponding byte
readRegister(0x32,&xData0);

//read from register X_DATA_1 (0x33) and into
//the corresponding byte
readRegister(0x33,&xData1);

//assemble these two bytes into the correct data object
//xSample = xData1[16:8] |  xData0[7:0]
int16_t xSample = (xData1 <<8) | xData0;
```

Optional / Additional ADXL Configurations

Data Format Register – Setting the Range

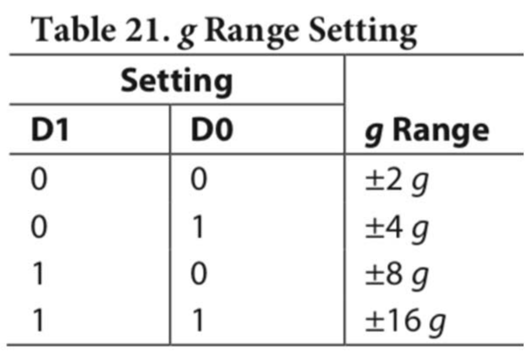
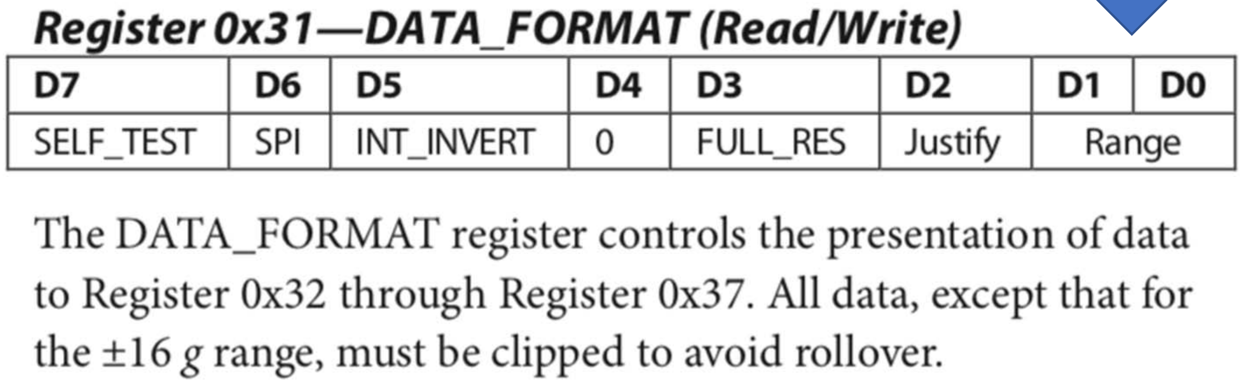


Table 21. *g* Range Setting

Setting		
D1	**D0**	***g* Range**
0	0	±2 *g*
0	1	±4 *g*
1	0	±8 *g*
1	1	±16 *g*



Register 0x31—DATA_FORMAT (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

The DATA_FORMAT register controls the presentation of data to Register 0x32 through Register 0x37. All data, except that for the ±16 *g* range, must be clipped to avoid rollover.

Default value is 0x0

Data Rate Register – Setting the Sample Rate

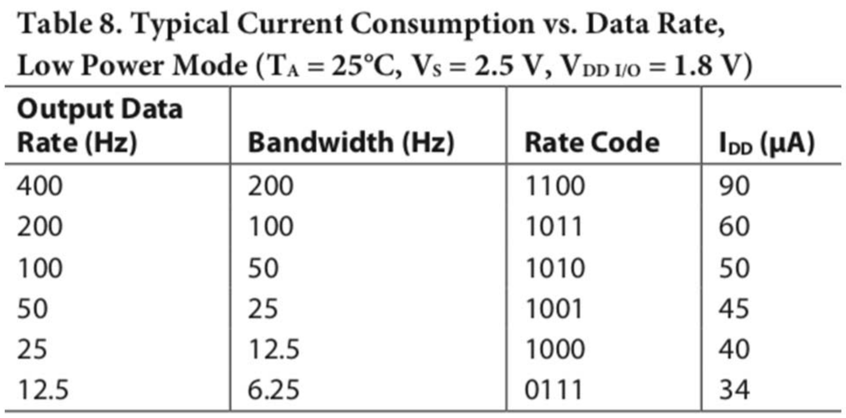
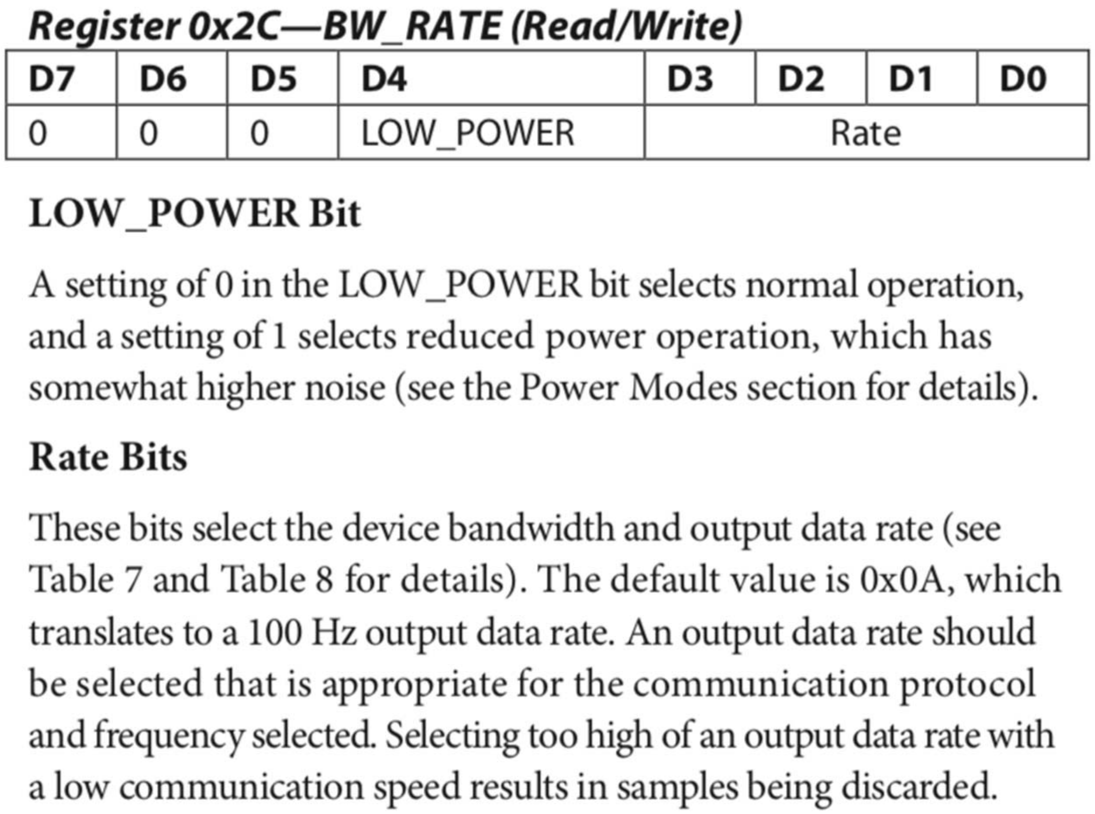


Table 8. Typical Current Consumption vs. Data Rate, Low Power Mode ($T_A = 25°C$, $V_S = 2.5$ V, $V_{DD\ I/O} = 1.8$ V)

Output Data Rate (Hz)	Bandwidth (Hz)	Rate Code	I_{DD} (µA)
400	200	1100	90
200	100	1011	60
100	50	1010	50
50	25	1001	45
25	12.5	1000	40
12.5	6.25	0111	34



Register 0x2C—BW_RATE (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	LOW_POWER	Rate			

LOW_POWER Bit

A setting of 0 in the LOW_POWER bit selects normal operation, and a setting of 1 selects reduced power operation, which has somewhat higher noise (see the Power Modes section for details).

Rate Bits

These bits select the device bandwidth and output data rate (see Table 7 and Table 8 for details). The default value is 0x0A, which translates to a 100 Hz output data rate. An output data rate should be selected that is appropriate for the communication protocol and frequency selected. Selecting too high of an output data rate with a low communication speed results in samples being discarded.

Default value is 0x0A

Setting Up Your ADXL345

- When powered on all registers have their default. See Table 19. Default values can/are acceptable and will allow the device to work.
- Set the desired range in the DATA_FORMAT register (optional)
- Set the desired sample rate in the BW_RATE register (optional)
- Set the MEASURE bit in the POWER_CTRL register (required)
- Read samples from X_DATA, Y_DATA, and Z_DATA (required)

SPI Milestone

- Create your own ADXL345 C/C++ class that will implement basic functionality
  - Power on; set data range; set sampling rate….etc.
- Core SPI functionality will be provided readRegister() and writeRegister(). Can expand as you desire.
- Will implement reading from DEVICE_ID in next lecture session.