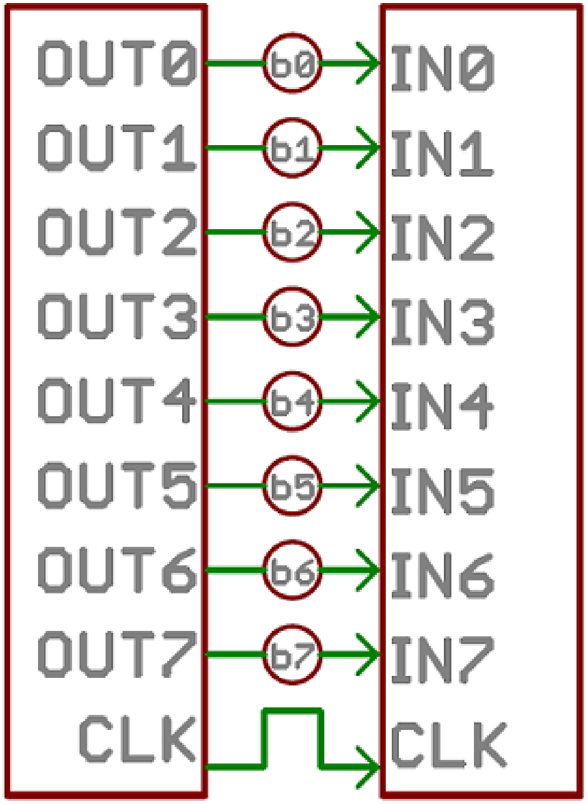
COMMUNICATION

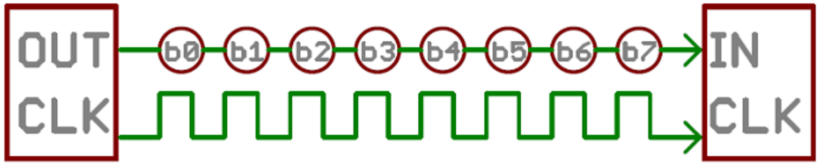
EL-GY 6483 Real Time Embedded Systems

CHARACTERIZING COMMUNICATION PROTOCOLS

- Serial (one data bit at a time) or parallel?
- Synchronous (explicit clock) or asynchronous (implicit clock)?
- Full duplex (send and receive at the same time) or half duplex?
- Asymmetric (e.g. master/slave configuration) or symmetric?
- Does it need addressing? If so, how does it do it?
- How many wires (hardware lines) does it need?
- What is its throughput (data bits transferred/unit time)?
- Advantages/disadvantages?

PARALLEL (LEFT) VS SERIAL (RIGHT)



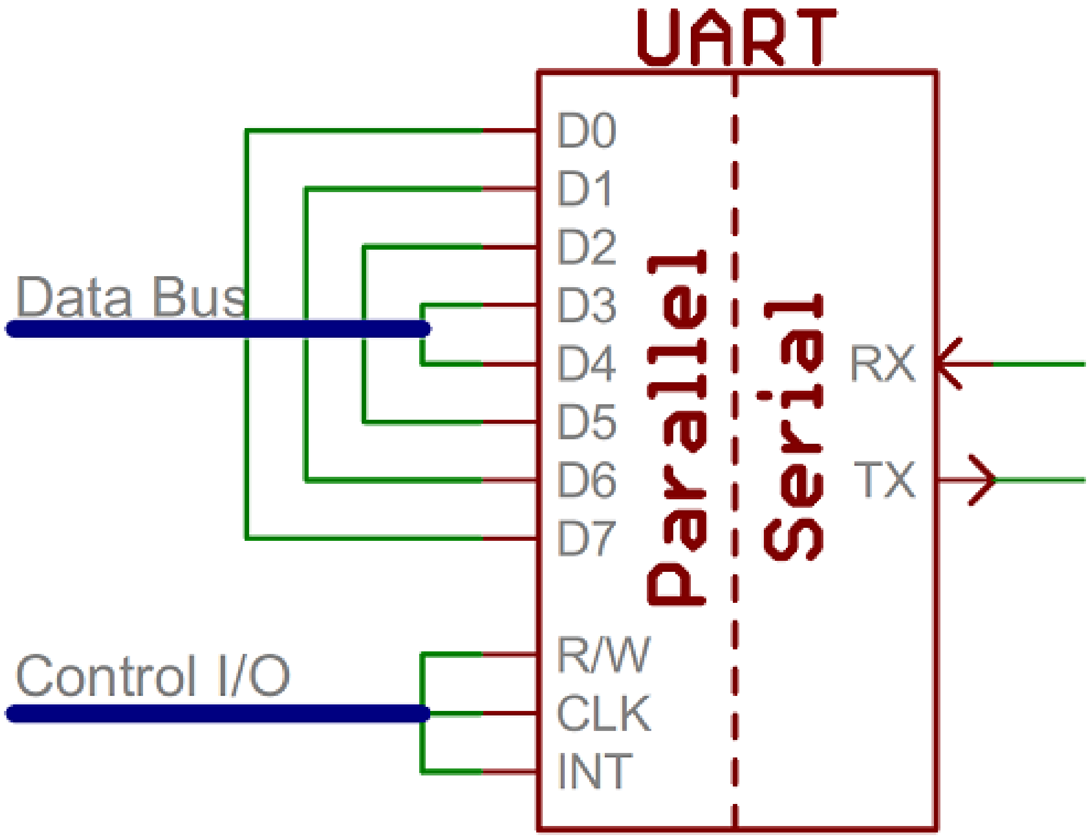


UART

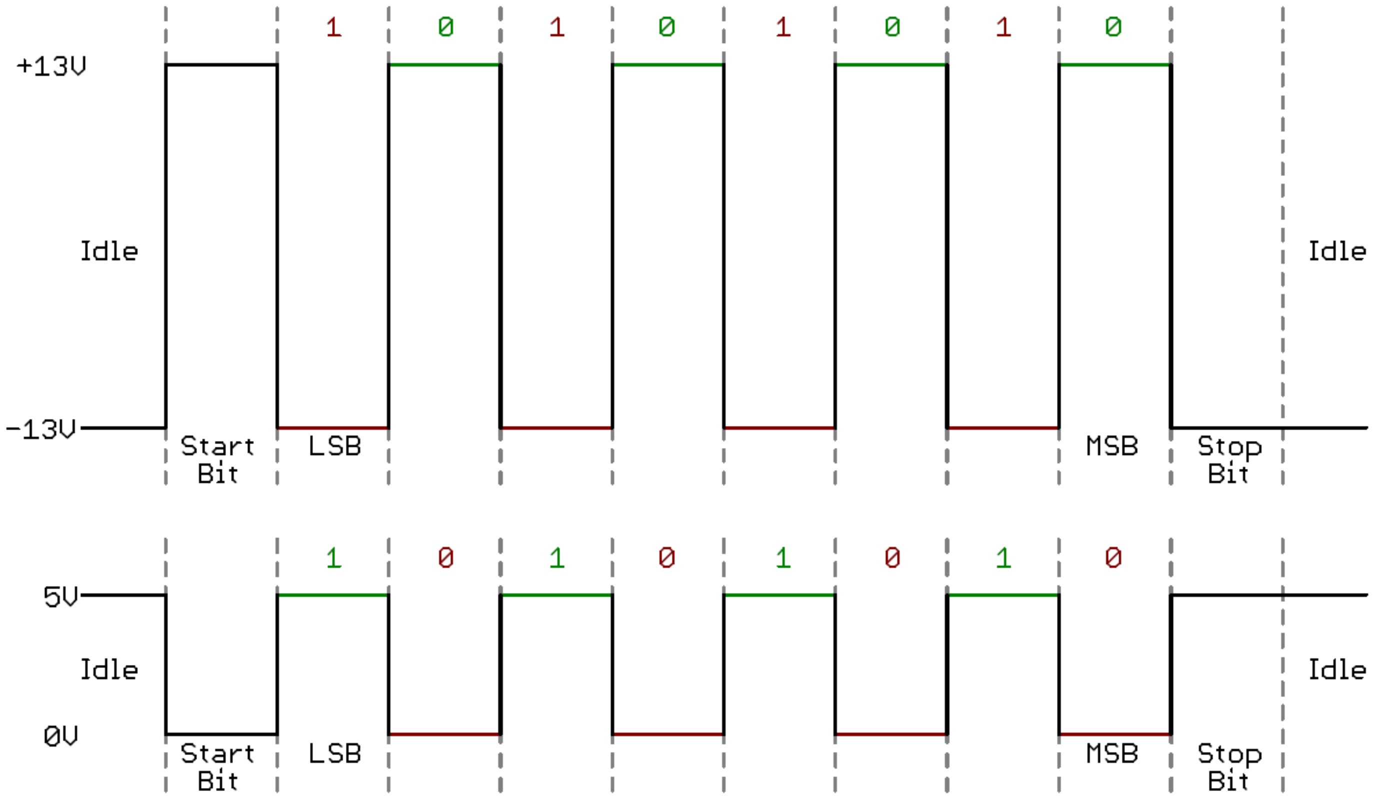
UART, TTL, RS-232

- **UART**: Universal asynchronous receiver/transmitter, implemented in processor.
- **RS-232**: Physical (electrical) protocol that sends signals at ± 12 volts
- **TTL**: Physical (electrical) protocol that sends signals at 0 - V_{cc} (usually 3.3 or 5V)

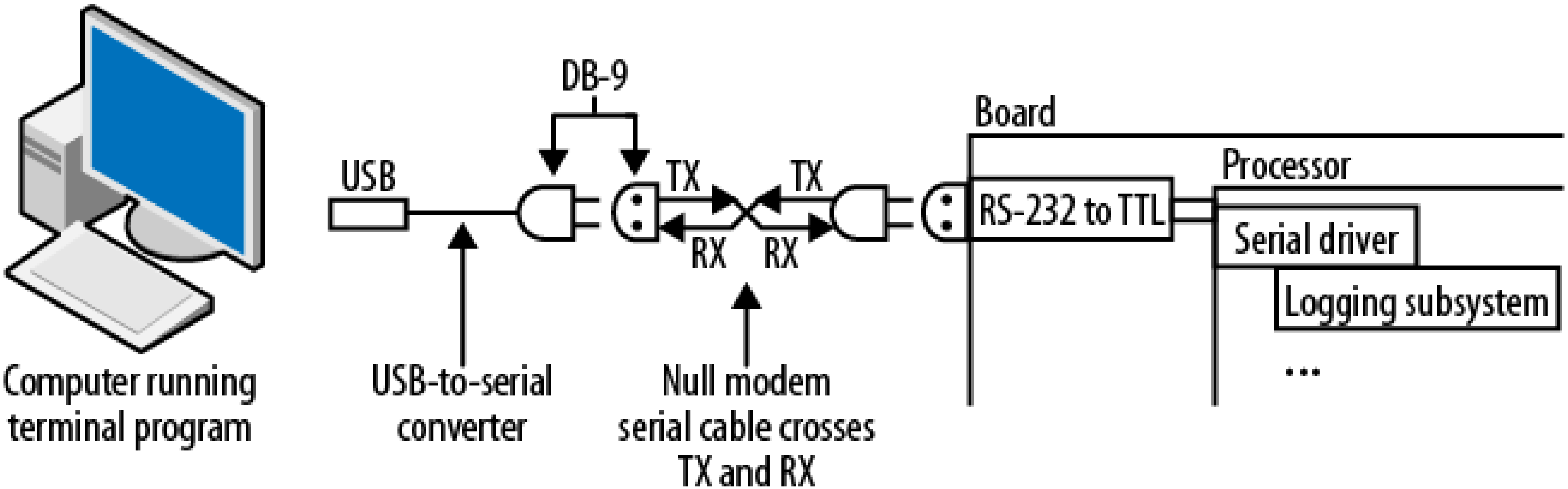
UART


UART
Data Bus
Control I/O
D0
D1
D2
D3
D4
D5
D6
D7
R/W
CLK
INT
Parallel
Serial
RX
TX

RS-232 AND TTL

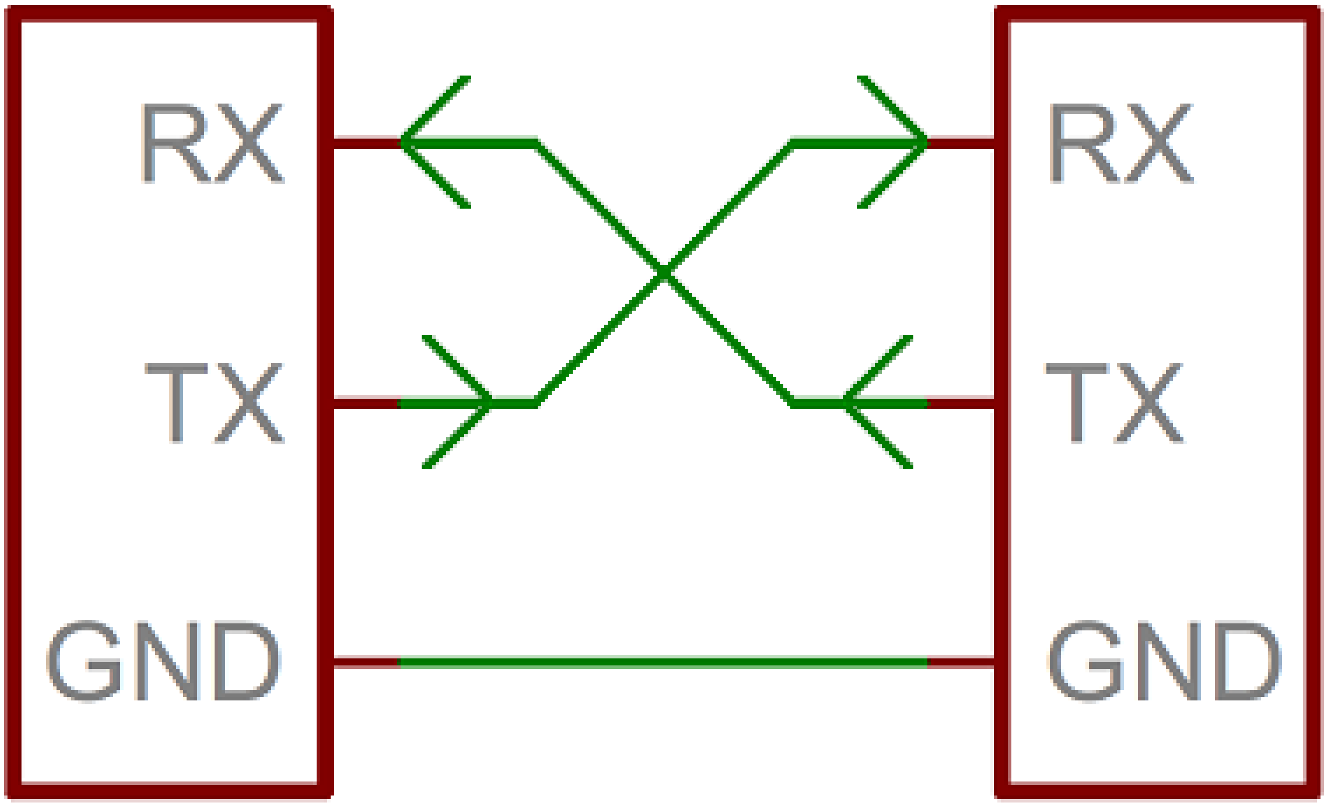


TYPICAL USE



DB-9
USB
TX
TX
RX
RX
Board
Processor
RS-232 to TTL
Serial driver
Logging subsystem
...
Computer running terminal program
USB-to-serial converter
Null modem serial cable crosses TX and RX

SERIAL BUS



RX TX CONNECTION

ESTABLISH PRIOR AGREEMENT ON

- Baud rate (often 9.6k, 19.2k, 38.4k, 115.2k)
- Data size and endianness (usually lsb first)
- Synchronization bits (one start bit, one or two stop bits)
  - Start bit: idle data line going from 1 to 0
  - Stop bit: goes back to idle state by holding line at 1
- Parity: add a bit so that
  - even parity: number of bits is even
  - odd parity: number of bits is odd
- Flow control

EXAMPLE

9600 8N1: 9600 baud, 8 data bits, no parity, 1 stop bit

Let's send "OK": "O" is 0b01001111 and "K" is 0b01001011.

- How long (time) is the line held for each bit?
- How many data bits were sent?
- How many bits were sent?
- What is the throughput in bits? Bytes?
- How much overhead?

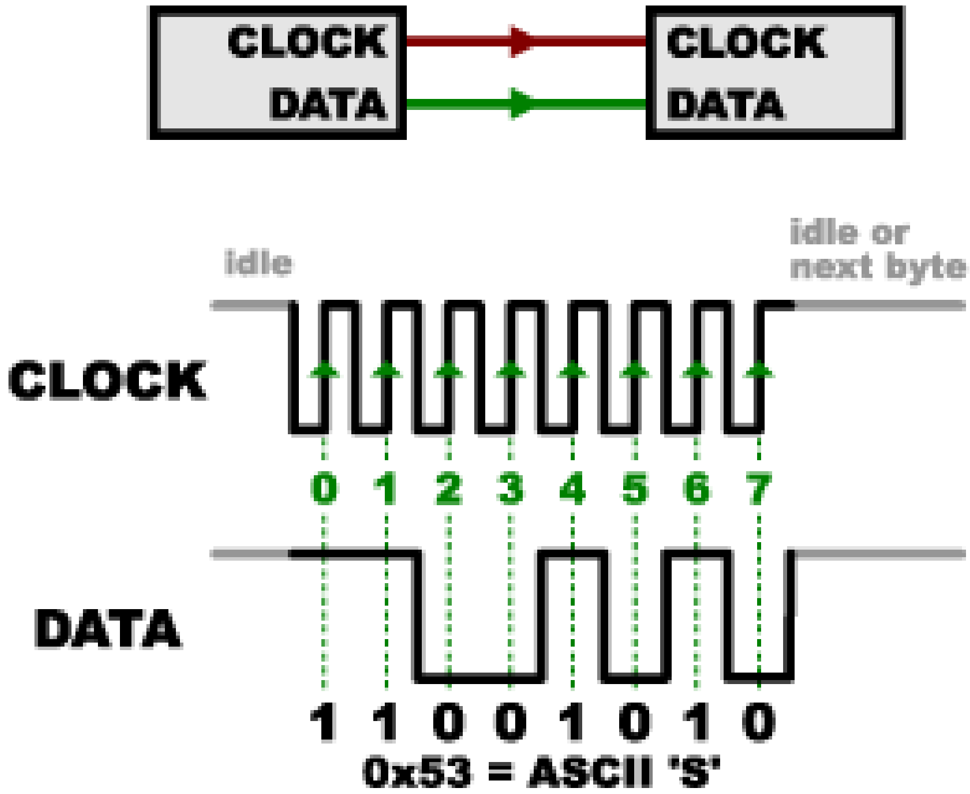
NOTE

Since there is no clock signal, the pair must synchronize by the start bit and maintain tight synchronization.

- Requires expensive clocks with very little error
- Higher baud rates need even more exact clocks

SPI: SERIAL PERIPHERAL INTERFACE

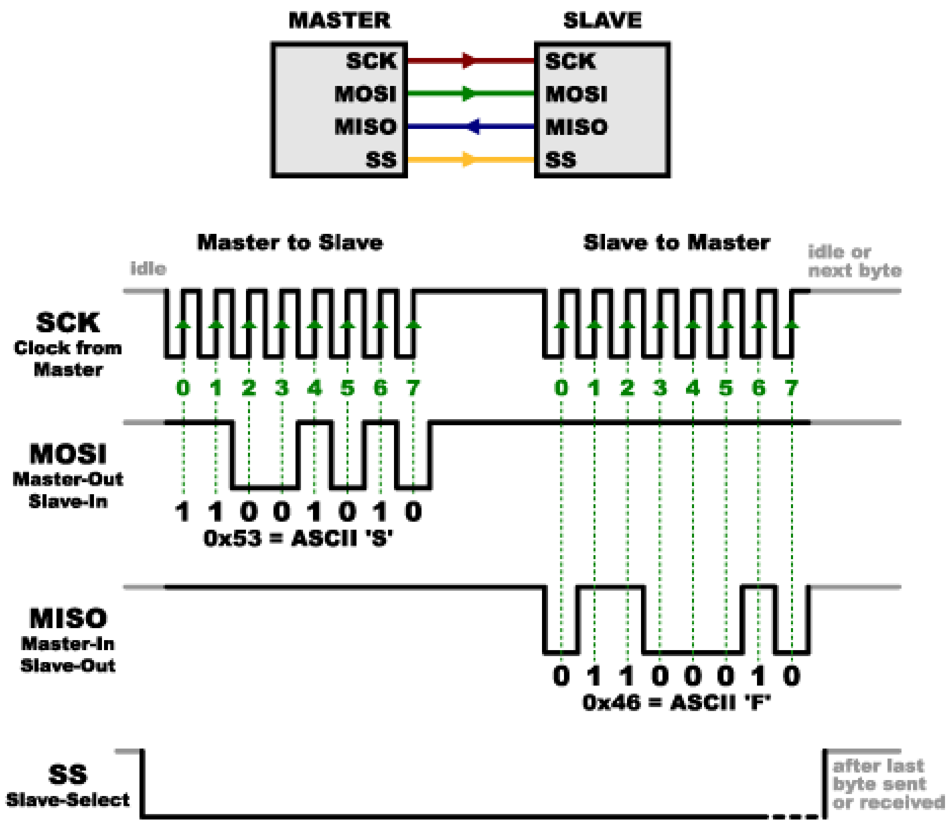
A SYNCHRONOUS COMMUNICATION PROTOCOL



FOUR LINES IN SPI

- Master-In Slave-Out (MISO), also known as Serial Data In (SDI)
- Master-Out Slave-In (MOSI), sometimes known as Serial Data Out (SDO)
- Clock (SCK, CLK, or SCLK)
- Chip Select (CS) or Slave Select (SS)

FOUR LINES IN SPI



CONFIGURATION

- Endianness: lsb or msb first?
- Clock polarity/phase: Idle clock signal is high or low? data valid on rising edge or falling edge?

SS is held high (active low)

DATASHEET TIMING DIAGRAM

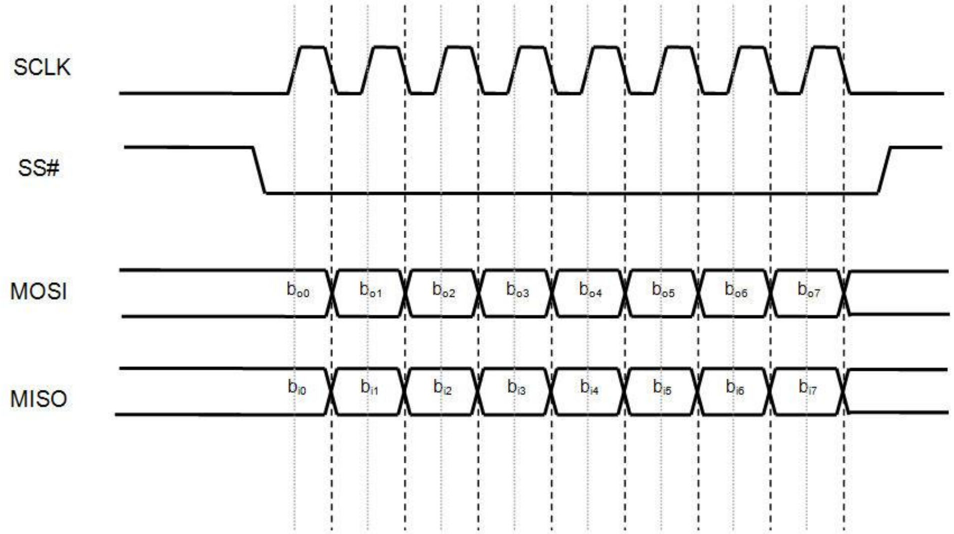
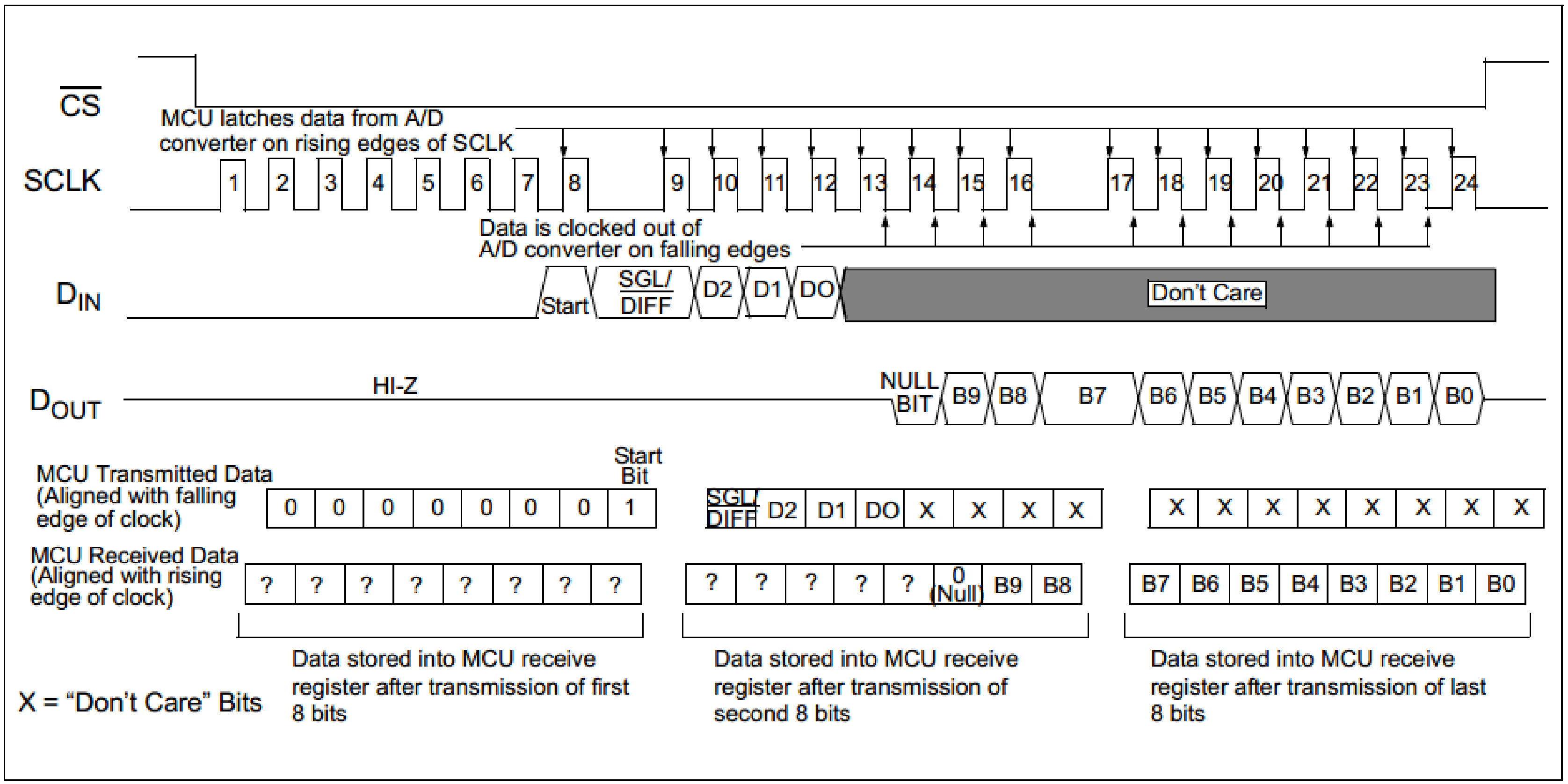
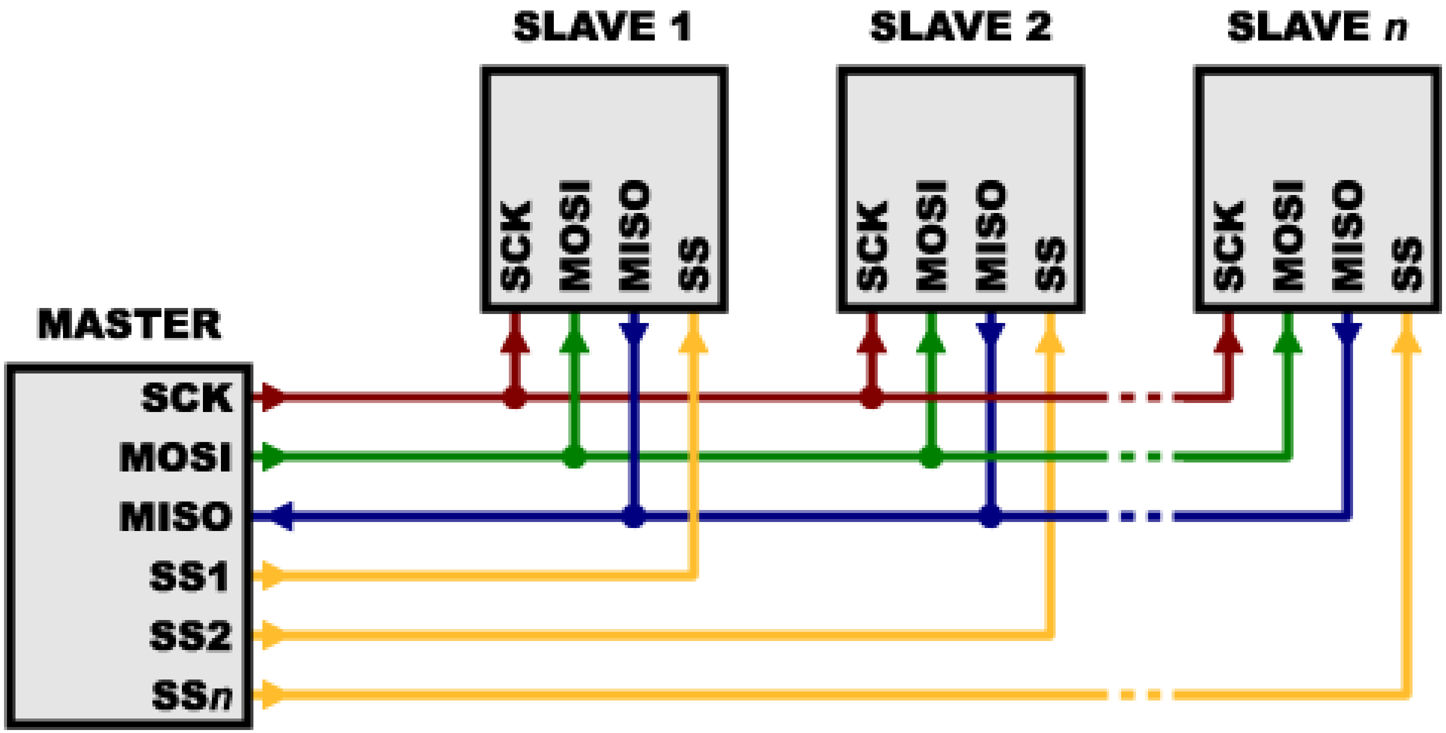


Figure 2 : A simple SPI communication. Data bits on MOSI and MISO toggle on the SCLK falling edge and are sampled on the SCLK rising edge. The SPI mode defines which SCLK edge is used for toggling data and which SCLK edge is used for sampling data.

DATASHEET TIMING DIAGRAM



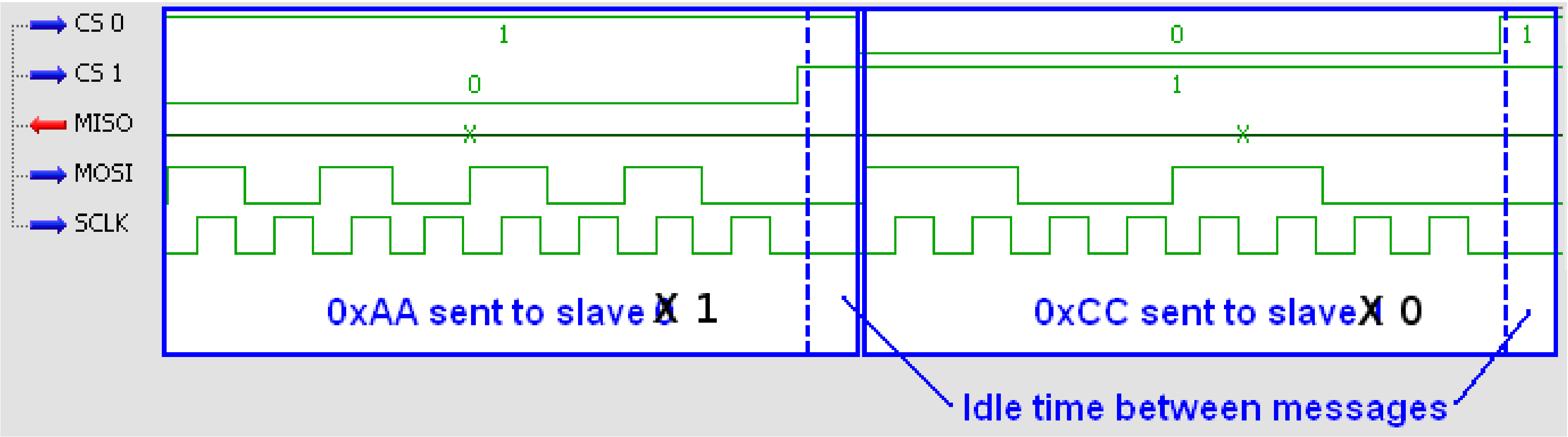
MULTIPLE SLAVES



CLOCK FORCING

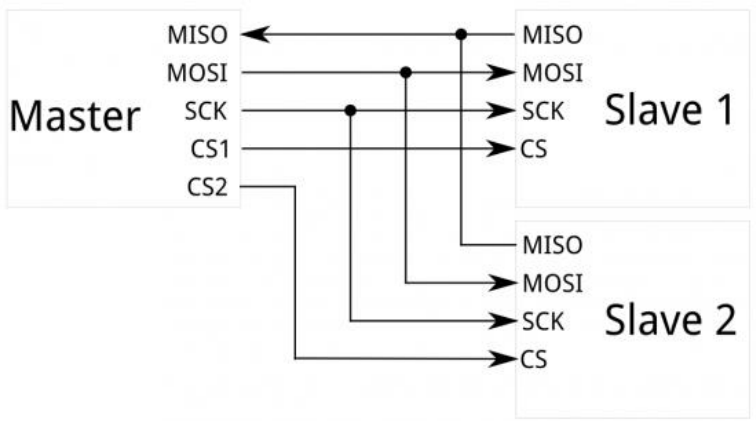
SPI requires full-duplex, i.e., for each bit sent from one side (master or slave), the other side (master or slave) has to send a bit. (Typically 0xFF by convention.)

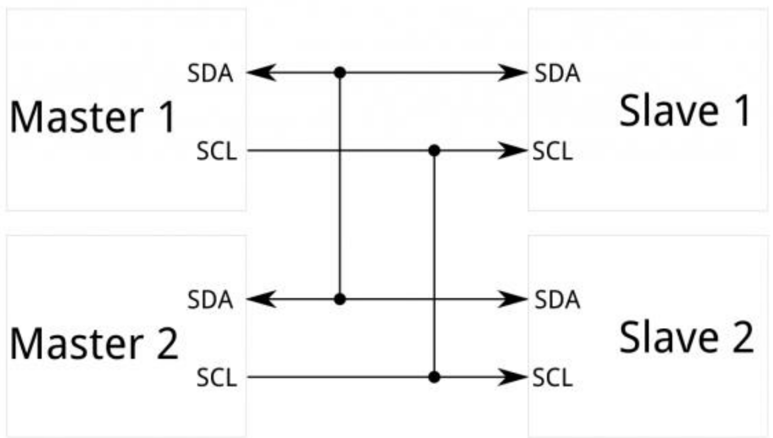
SPI EXAMPLE


CS 0
CS 1
MISO
MOSI
SCLK
1
0
x
0
1
x
1
0xAA sent to slave X 1
0xCC sent to slaveX 0
Idle time between messages

I2C: INTER-INTEGRATED CIRCUIT PROTOCOL

HOW MANY WIRES DO YOU REALLY NEED?





I2C: 2 WIRES

More slaves, fewer wires:

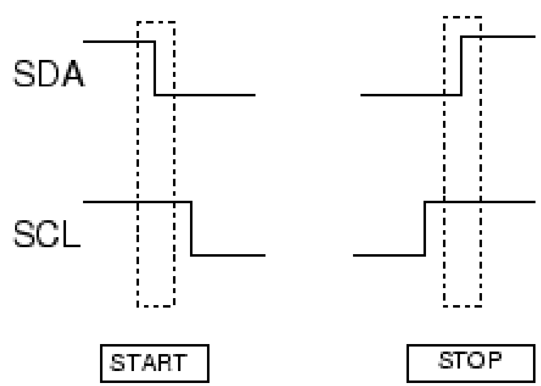
- SCL is the clock signal
- SDA is the data signal

I2C BASIC PROTOCOL

- Master starts communication:
  - Send a 7-bit address
  - Indicate whether it wants to read from or write to the slave
- Slave with that address then sends ACK
- Master writes to/reads from slave
- Master sends a stop bit

START CONDITION

- Master leaves SCL high and pulls SDA low
- If two master devices wish to take ownership of the bus at one time, there's an arbitration procedure
- Except for the start and stop signals, the SDA line only changes while the clock is low



ADDRESS FRAME

- 7-bit address, most significant bit (msb) first
- R/W bit indicating whether this is a read (1) or write (0) operation
- NACK/ACK bit (receiver pulls SDA low to ACK)

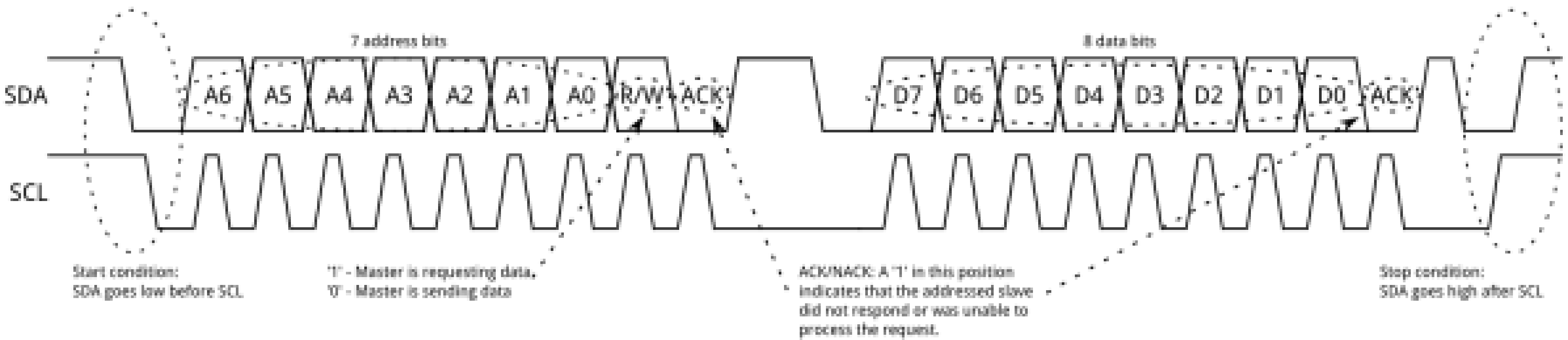
A NACK indicates:

- Master TX: The slave is unable to accept the data. No such slave, command not understood, or unable to accept any more data.
- Slave TX: The master wishes the transfer to stop after this data byte.

DATA FRAMES

- Data placed on SDA by master (W) or slave (R)
- NACK/ACK bit sent by receiver after every 8 bits (receiver pulls SDA low to ACK)

BASIC PROTOCOL

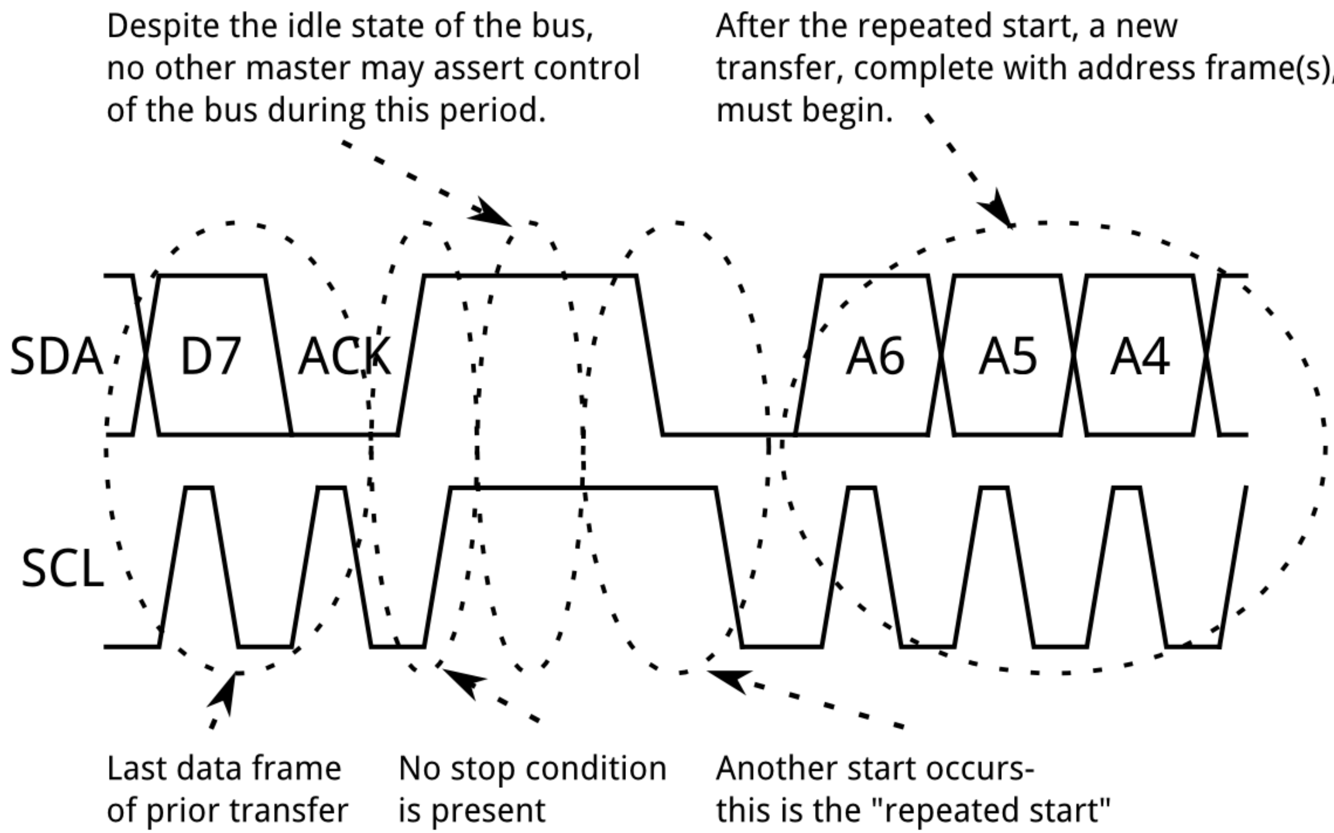


AFTER ACK/NACK

After ACK, master may do one of three things:

- Prepare to transfer another byte of data: master pulses SCL high, transmitter sets SDA
- Send a stop sequence: Set SDA low, let SCL go high, then let SDA go high (releases bus)
- Send a repeated start: Set SDA high, let SCL go high, and pull SDA low again (starts new transaction without releasing bus)

REPEATED START



I2C MODES

SINGLE-BYTE WRITE								
MASTER	START	SLAVE ADDRESS + WRITE		REGISTER ADDRESS		DATA		STOP
SLAVE			ACK		ACK		ACK	

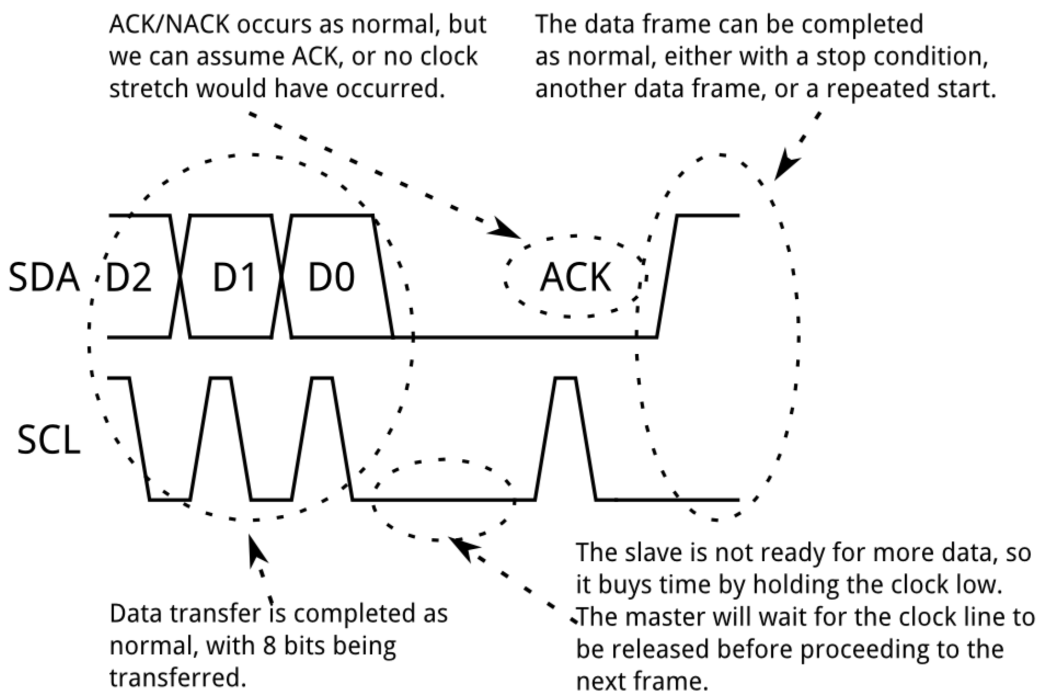
MULTIPLE-BYTE WRITE										
MASTER	START	SLAVE ADDRESS + WRITE		REGISTER ADDRESS		DATA		DATA		STOP
SLAVE			ACK		ACK		ACK		ACK	

SINGLE-BYTE READ										
MASTER	START	SLAVE ADDRESS + WRITE		REGISTER ADDRESS		START[1]	SLAVE ADDRESS + READ			NACK	STOP
SLAVE			ACK		ACK			ACK	DATA		

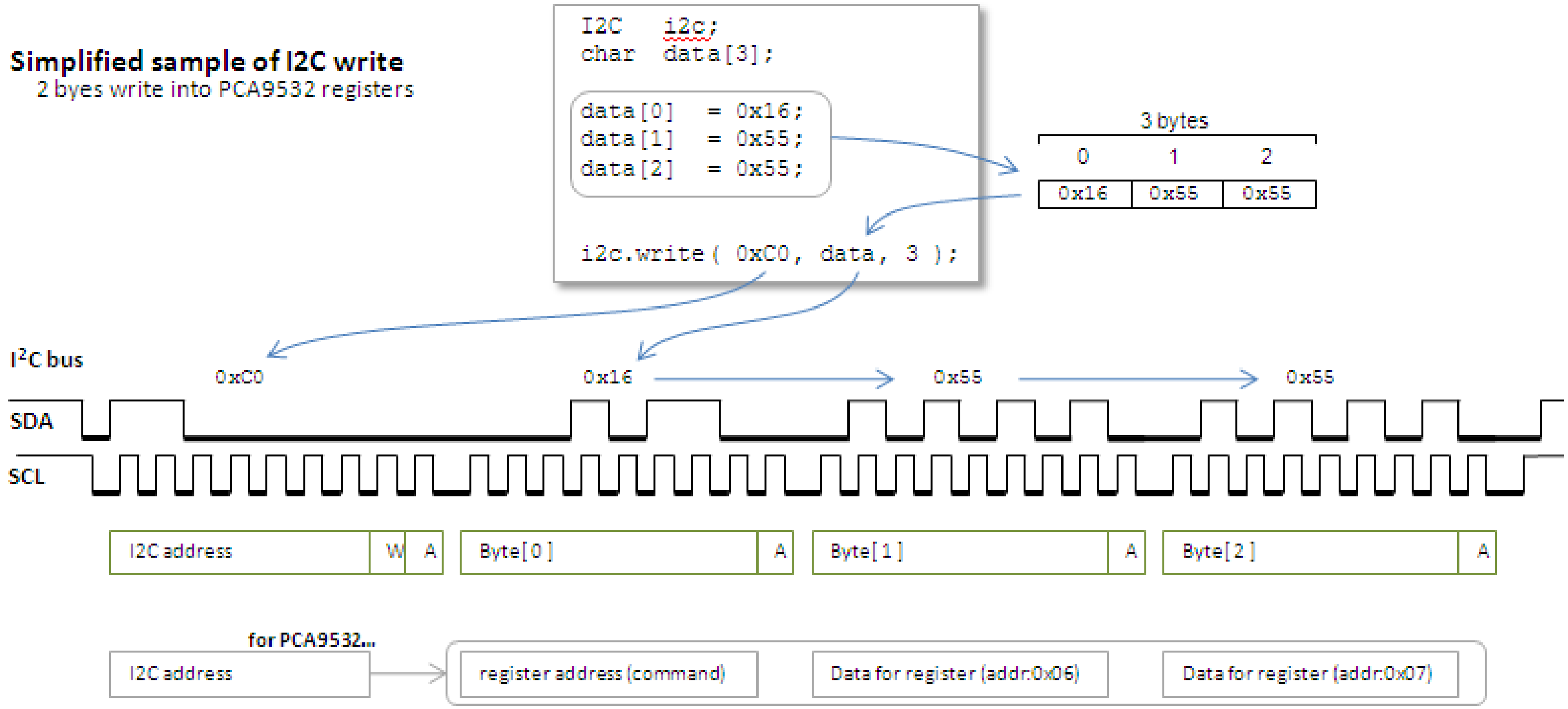
MULTIPLE-BYTE READ													
MASTER	START	SLAVE ADDRESS + WRITE		REGISTER ADDRESS		START[1]	SLAVE ADDRESS + READ			ACK		NACK	STOP
SLAVE			ACK		ACK			ACK	DATA		DATA		

CLOCK STRETCHING

If active slave holds the SCL line low after the master releases it, master must refrain from clock pulses or data transfer until slave releases the SCL line:



EXAMPLE



Note: in this example, the address C0 refers to the 7-bit address followed by the R/W bit: 1100 0000 (i.e., 1100 000 is the 7-bit address and the final 0 is the R/W bit.)

REFERENCES

- Making Embedded Systems: Design Patterns for Great Software, Chapter 6, Elecia White.
- Sparkfun.com tutorials:
  - Serial Communication:
  - https://learn.sparkfun.com/tutorials/serial-communication
  - Serial Peripheral Interface: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi
  - I2C: https://learn.sparkfun.com/tutorials/i2c