3.30 Synchronous Serial Communications

The Serial Peripheral Interface (SPI)

Inter Integrated Circuit (I^2C)

EE302 – Real time and embedded systems

Overview

- Aims
  - Introduce synchronous serial communications protocols commonly used to interface to peripheral devices or distributed devices in embedded systems
- Learning outcomes – you should be able to…
  - **Compare** advantages/disadvantages and **differentiate** between asynchronous and synchronous serial communications
  - **Compare** advantages/disadvantages and **differentiate** between SPI and I^2C
  - **Describe** detailed operation of SPI and I^2C, including line coding and handling of single and multi-byte communications
  - **Interpret** a synchronous serial communications specification (SPI or I^2C) and **calculate** the effective data rate, packet rate, etc.
  - **Interpret** a synchronous serial device specification (e.g. for a serial EEPROM) to identify the sequence of SPI or I^2C exchanges necessary to achieve some goal (e.g. store some data to the serial EEPROM).
  - **Write/show** the code/pseudocode necessary to implement a sequence of SPI/I2C exchanges
  - **Write/show** the line coding of a sequence of SPI/I2C exchanges

Synchronous Communication

- We'll look at two synchronous communications protocols
  - Serial Peripheral Interface
  - I^2C
- Typical use
  - connect microcontroller to periperipherals
  - Example: PIC microcontroller to ADC, or DAC, or EEPROM, or USB interface, or Bluetooth Radio, etc.

Synchronous serial communications differs from async serial communications in several respects. Synchronous…

- Uses Master/Slave architecture
- Master and slaves are connected via a communications bus (not a point to point line)
- Transmits both data and an explicit clock signal
  - Master provides clock
  - Reduces/eliminates synchronization error
  - Better bandwidth efficiency (fewer framing/overhead bits)
  - Clock frequency and duty cycle can vary without problems
- Is often faster than asychronous communications

Inter Integrated Circuit (I^2C)

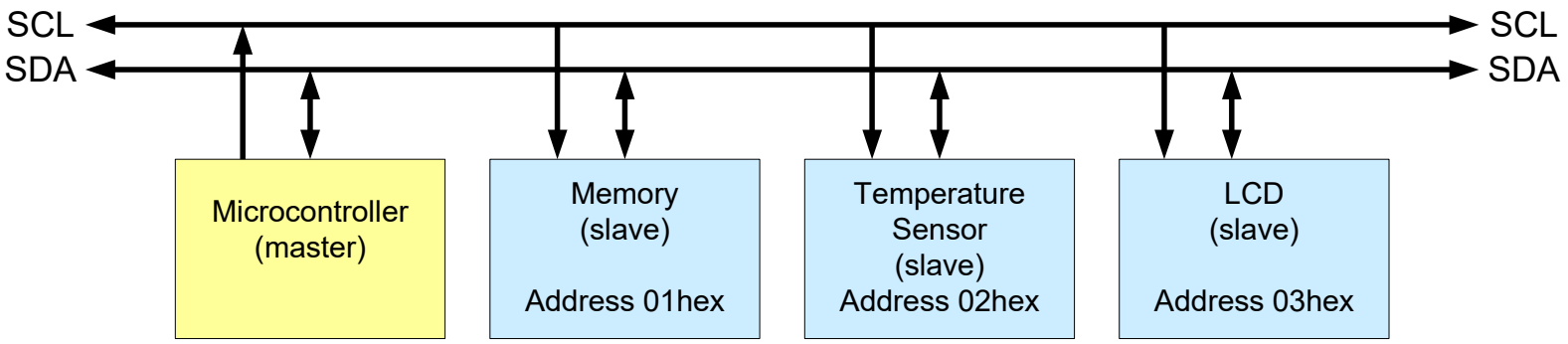
I^2C (Inter Integrated Circuit)

- Commonly used synchronous serial protocol, often used to connect microcontroller to peripherals
  - e.g. ADC, DAC, EEPROM, USB interface, etc.
- In common with SPI (see later):
  - It uses a master/slave architecture
  - Master and slaves are connected by a bus
  - A dedicated clock signal (SCL) is transmitted independently of the data
  - Data is transmitted MSb (most significant bit) first

I^2C unique features

- Bus consists of just two wires: serial data (SDA) and serial clock (SCL)
  - Saves wires/pins
  - Because there is just one data line, communication is **half-duplex**, i.e. master and slave cannot transmit simultaneously (speed implication?)
- SDA and SCL lines use pull up resistors and "float high"
  - Devices pull the SDA and SCL lines low as needed
  - Master usually controls SCL, but slave can pull SCL low to pause a transfer (this is called **clock stretching**)
- I2C protocol
  - Divides all communication into two operations: read (master asks slave for data) and write (master sends data to slave)
  - Receiver must acknowledge all data received (ACK or NAK) which helps make communication robust
  - The master transmits an I^2C address as the first part of any read/write operation to select which bus connected device (of the possibly many) to communicate with
  - Protocol is more complex than SPI (see later)

I²C Connectivity and addressing

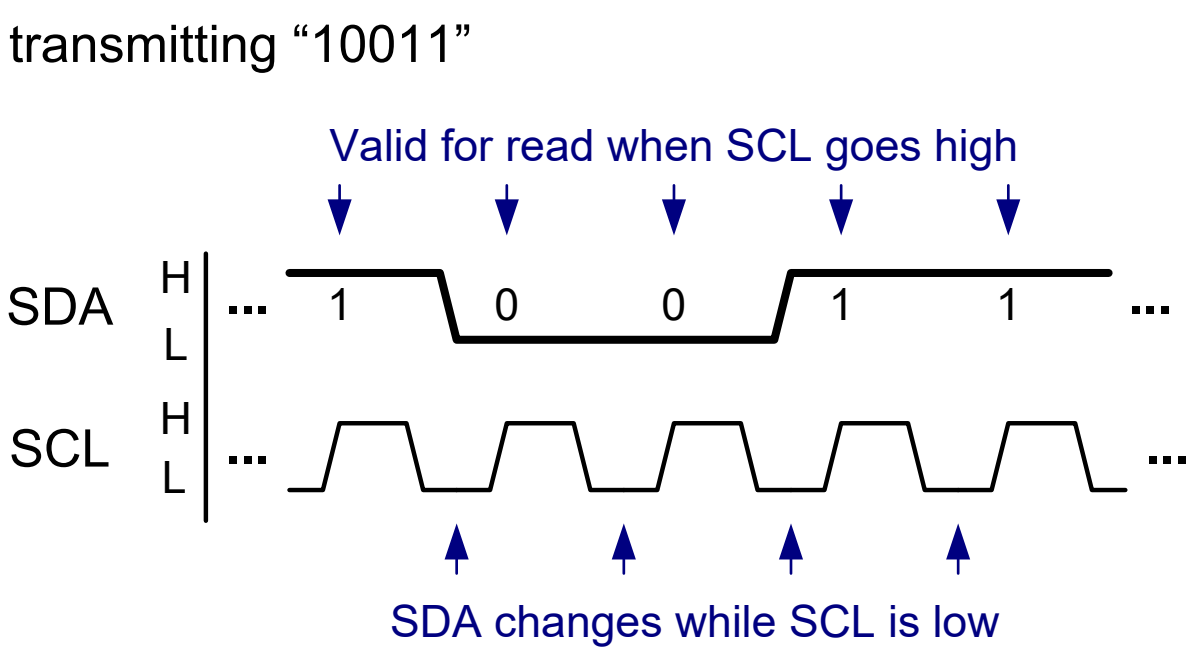


I²C uses a 7 or 10-bit addressing scheme to select devices.

(Contrast with explicit slave select line in SPI – see later)
(We will only consider 7-bit addressing from here on.)

I²C Line coding

- The transmitter (master or slave) changes the SDA level (to specify a 1 or 0 bit) only when SCL is Low
  (compare with SPI later)
- When SCL transitions from low to high the receiver (slave or master) assumes that the SDA level is stable and valid and reads the corresponding bit value

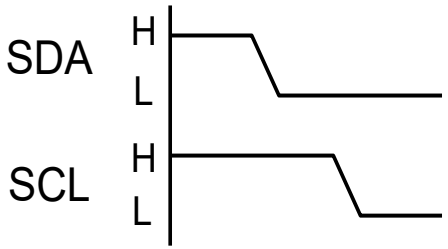


I²C Start and stop conditions

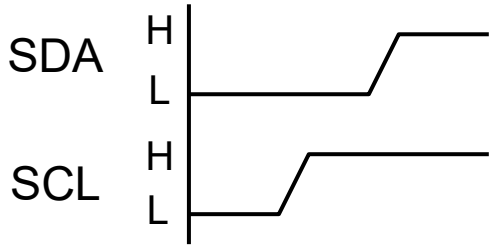
- I2C can support multiple master devices on a single bus, so it uses explicit start and stop "conditions" (like start and stop bits in async comms) to delimit communications.
- In the idle state (between communications) SCL and SDA are high (remember pull up resistor so it floats high)
- **Start** (SDA low followed by SCL low) reserves the I2C bus and signals the start of data transfer
- **Stop** (SCL high followed by SDA high) signals the end of data transfer and releases the bus

(Compare the start and stop condition with "normal" line coding)

A START Condition



A STOP Condition



I^2C Protocol overview

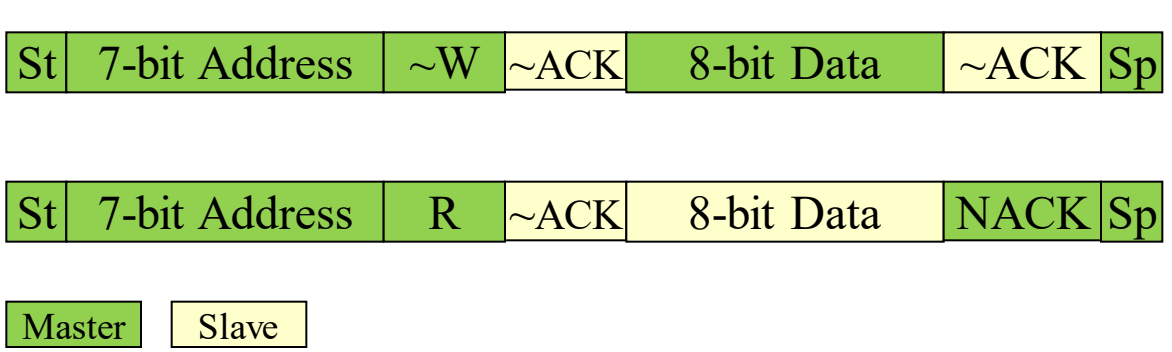
I^2C defines a specific protocol which must be followed for each communication…

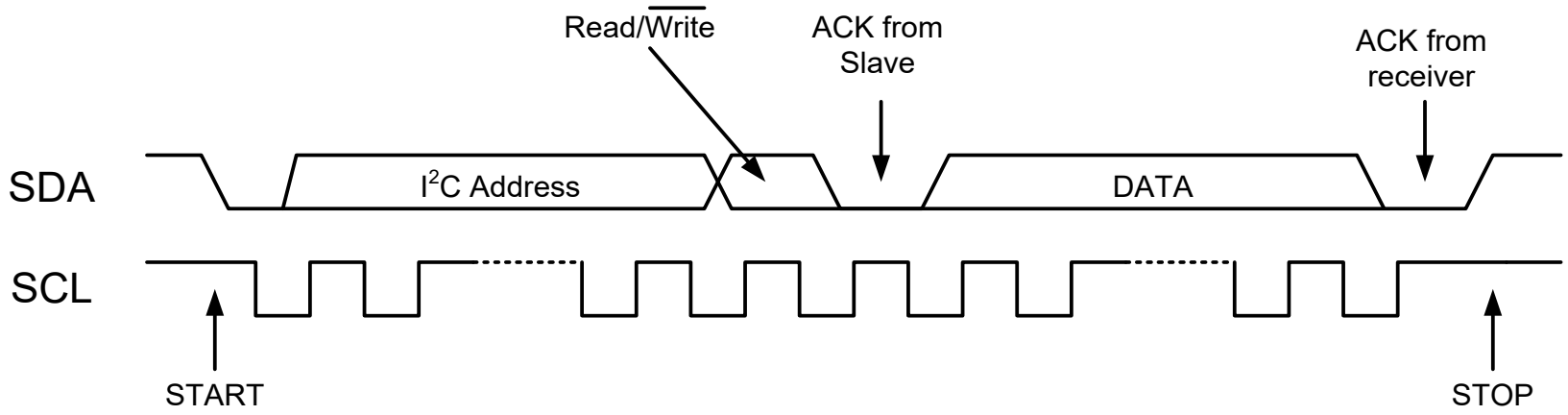
- START [master]
  - All transmissions begin with a START condition
- I^2C address and Read/Write bit [master], ACK [slave]
  - First is the 7 bit or 10 bit I^2C address which identifies the slave device for this communication is transmitted
  - Next is the Read/~Write bit is transmitted
    - Read/~Write = 1 ➔ Read
    - Read/~Write = 0 ➔ Write
  - The selected slave device must respond during the next clock period by taking control of SDA to acknowledge the operation start by setting ~ACK /NACK = 0 (pull SDA low)

Contd.

- Data [master or slave] and ACK/NACK [slave or master]
  - The transmitter (master for write operation, slave for read) transmits the data one byte at a time
  - The receiver (slave for write operation, master for read) must acknowledge each byte by setting the SDA to ~ACK/NACK for one bit period immediately after each byte has been received
    - NACK (leaving SDA float high) can indicate a problem
    - NACK can also be used by master to indicate no more data wanted/needed in a read operation
- STOP [master]
  - Communication ends with a stop condition

I²C Protocol summary



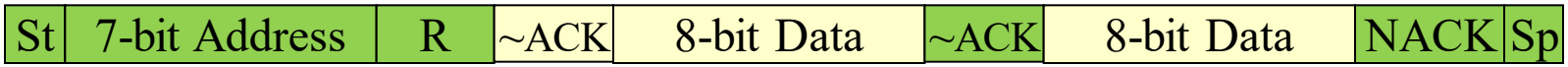


Read/Write Cycles

I²C Multi-byte streaming

2 Byte Write Example

2 Byte Read Example

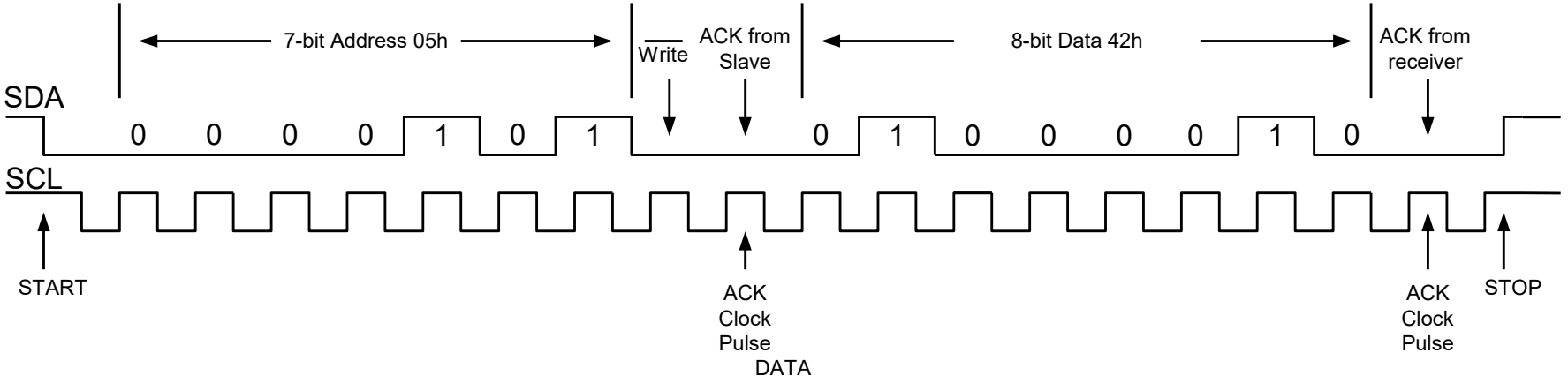


MASTER

SLAVE

Detailed Example: 1 byte write

In this example we write the byte 0x42 to device at I2C address 0x05.



Example questions on I^2C (basic)

Q1. Explain how the ACK and NACK signals are used in the I2C protocol?

Q2. Explain the I2C protocol?

Q3. With the aid of sketches show the Start and Stop condition in the I2C synchronous protocol?

Q4. Show how two microcontrollers could be connected using I2C?

Q5. Discuss the main differences between I2C and SPI

Example questions on I2C (intermediate)

Q1. A master reads the 2 byte value 0x0142 from a I2C peripheral (slave). Draw and carefully label the complete timing diagram explaining any assumptions made.

Q2. Assume the serial clock is 100 kHz. How fast (bps) can data be received from slave…

- If up to 32 data bytes may be read per operation?
- As above except that there is an additional 100 microsec delay before a subsequent read can be started

$$T_{bit} = \frac{1}{100000} = 10\,\mu s$$

ST + I²C Addr + R + ACK + 32 (8 bit Data + ACK/NAK) + SP

$$\frac{1}{2} + 7 + 1 + 1 + 32(8 + 1) + \frac{1}{2} = 298 \text{ bits}$$

$$T_{op} = 298 \times 10\,\mu s = 2.98\,ms$$

$$\text{Op rate} = \frac{1}{T_{op}} = \frac{1000}{2.98} \approx 335.6 \text{ ops/sec}$$

$$\text{Total data bits/op} = 32 \times 8 = 256 \text{ bits/op}$$

$$\text{op rate} \left(335.6 \frac{\text{ops}}{\text{sec}}\right)\left(256 \frac{\text{bits}}{\text{op}}\right) = 85913.6 \text{ bps}$$

effective data rate

More I²C questions

- How would you use bit banging (in the master) to perform an I2C read operation for a single byte of data
  - First do it using English and bullet points
  - Then attempt it in pseudocode

Another I²C question

- A PIC microcontroller is connected to an EEPROM via I2C
  - I2C address of the EEPROM is 0x40
  - The EEPROM keeps a single memory address pointer which is automatically incremented after each data byte is written or read
  - To write to the EEPROM
    - First write the memory address (MSByte first), then the data bytes to be written.
  - To read from the EEPROM
    - If the auto-incremented memory address needs to be changed, do as for EEPROM write, but don't write any subsequent data
    - Initiate an I2C read, and ACK each byte received until the last desired byte has been read
- Then
  - Draw the high level connectivity of the PIC and EEPROM
  - Write the high level code required to write the values 10, 11, 12 starting at location EEPROM location 0x1230
  - Write the high level code required to read one value from location 0x2340

2 byte internal address

contd

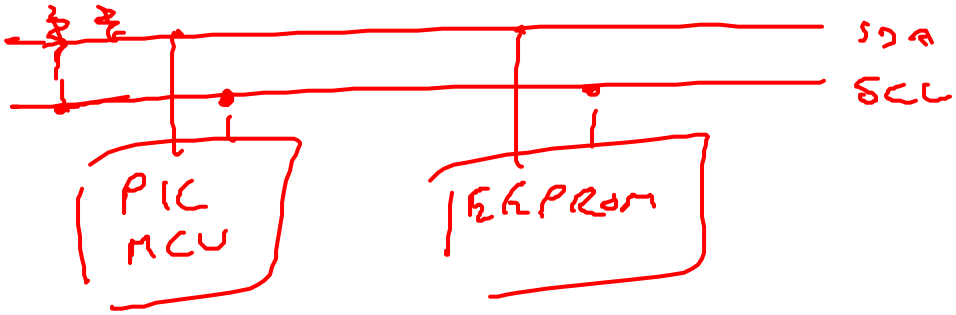
- Assume the existence of the following I/O function (pseudocode)

```
ack = i2c_start(i2cAddress, readOrWrite)

ack = i2c_write_byte(byteValue)

byteValue = i2c_read_byte(ackOrNackResponse)

i2c_stop()
```



WRITE

```
i2c_start (0x40, W)
i2c_write_byte (0x02)   } EEPROM internal addr
i2c_write_byte (0x30)   }
i2c_write_byte (12)     } writing the actual data
..             (49)     }
..             (12)     }
i2c_stop ()
```

READ

```
i2c_start (0x40, W)
i2c_write_byte (0x23)
..             (0x40)
i2c_stop ()
// read the value
i2c_start (0x40, R)
val =
   i2c_read_byte (NAK)
i2c_stop ()
```

Serial Peripheral Interface (SPI)

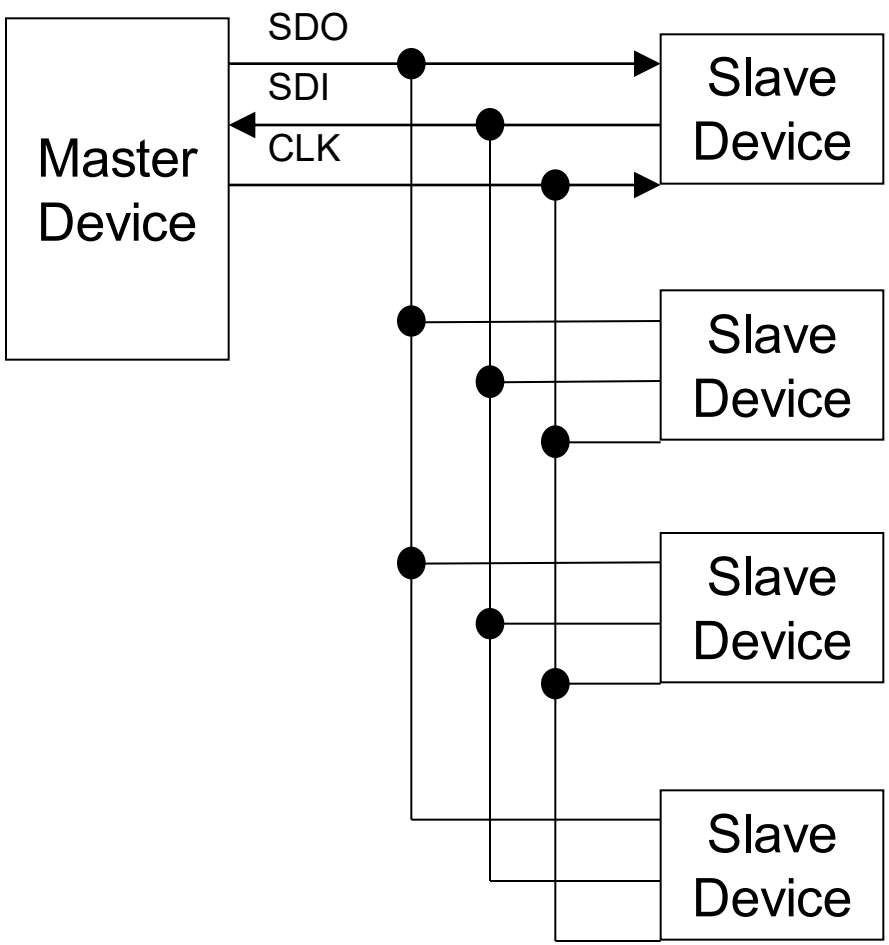
Serial Peripheral Interface (SPI)

- Another commonly used synchronous protocol
  - E.g. used to communicate with MMC/SD cards in digicams
- In common with I^2C:
  - It uses a master/slave architecture
  - Master and slaves are connected by a bus
  - A dedicated clock signal (SCL) is transmitted independently of the data
  - Data is transmitted MSb (most significant bit) first

SPI unique features

- The bus has 3 lanes
  - At a given device, these are called serial data in (SDI), serial data out (SDO), and serial clock (SCK)
  - Supports full duplex communications
- Every data exchange must be full duplex
  - Data always transmitted in both directions simultaneously
  - If one side has nothing useful to transmit, then it transmits dummy data (e.g. zeros) which receiving end then ignores
- Clock controls the data exchange
  - Only the master controls the clock – slaves may not manipulate the clock (compare to I^2C)
  - Data is exchanged on clock edge transitions
  - Clock polarity and edge/phase is configurable (details on later slide)
- To communicate with a slave device, the slave must first be selected
  - Slave Select lines are unidirectional point-to-point and independent of the bus (see next slide)
- How does SPI compare to I^2C?
  - Full duplex comms and no start/stop conditions, no protocol (address, read/write, ACK/NACK, etc)
  - Hence, SPI has higher effective data rate for same clock speed

Master/Slave architecture

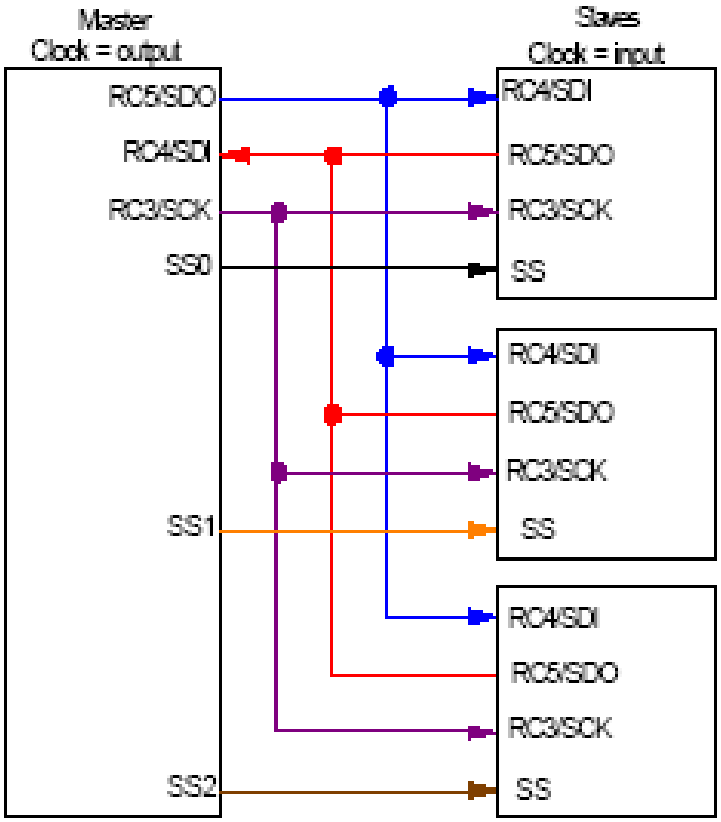


- MOSI
  - Master Out, Slave In
  - Master SDO connects to slave SDI
- MISO
  - Master In, Slave Out
  - Master SDI connects to slave SDO

Question:
Each device has a **slave select** (SS) line.
So how might a particular slave device be selected???

Example: PIC to PIC communications with one master multiple slaves

PIC Master

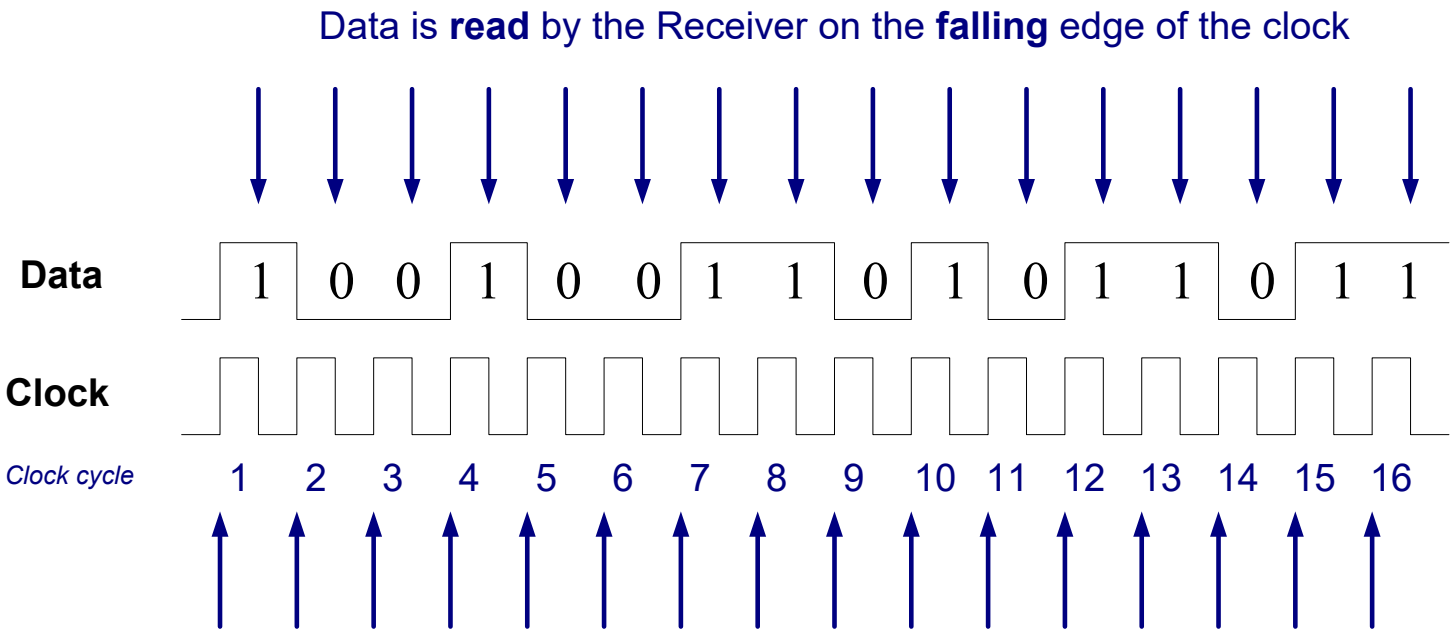


PIC Slaves

SPI Clock

- Clock polarity
  - specifies the idle state of clock line between data exchanges
    - Idle low or idle high
- Clock edge (sometimes called clock phase)
  - Specifies when levels on SDO/SDI lines should be set (changing from their previous values if necessary) and when they are read
  - Two possibilities:
    - Set on rising edge, read on falling edge
    - Set on falling edge, read on rising edge

SPI timing (each byte)



In this example:

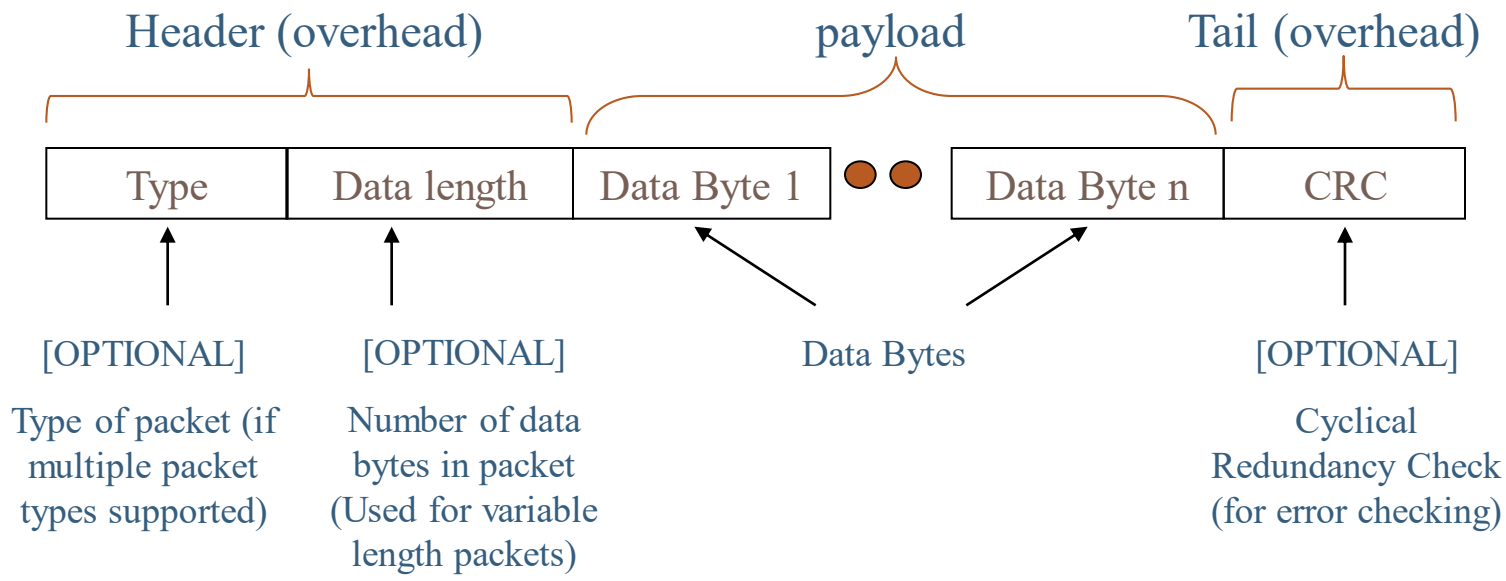
Clock polarity = idle low, clock edge = set on rising edge (read on falling edge)

Q. *How does max bit rate relate to clock freq?*

SPI multi-byte packets

In SPI, multi-byte packets may introduce additional overhead, but this depends on which features are required.

Example: no overhead required if just one type of packet supported, all packets of identical fixed length, and no error check support



Note: Header fields not necessarily 1 byte each (could be smaller/bigger)

SPI self test questions (basic)

- Explain the SPI protocol?
- Explain how the SS (or CS) signal is used in SPI?
- Explain and show how two microcontrollers could be connected for SPI communications?
- Explain Clocked Serial Data (synchronous communication)?

SPI self test questions (intermediate)

- A master transmits 0x55 to a slave device using the SPI protocol. Draw the timing diagram of the master's SCL and SDO lines for the transfer? State any assumptions made.
- Assume serial clock is 100 kHz. How fast (bps) can data be transmitted if…
  a) packets are 32 bytes long (fixed) and there is no gap between packets
  b) packets are 32 bytes long (fixed) and the slave must be deselected for 100 microsecs between packets
  c) Packets are 32 bytes long (variable) but include a single byte header; the slave must be deselected for 100 microsecs between packets

SPI question (more advanced)

- A PIC microcontroller is connected to an EEPROM using SPI and selected (active low) via RC7
- First, draw the high level connection diagram
- Now, assume the following simplified details…
  - To read from EEPROM: transmit read code (0x03) , followed by 2 byte memory address (high byte first). Data bytes, starting at the specified address, are received on subsequent exchanges (transmitted values ignored). End of read when the EEPROM is deselected.
  - To write to EEPROM: transmit write code (0x02), followed by 2 byte memory address (high byte first), followed by data bytes. End of write is indicated when EEPROM is deselected.
  - There is a synchronous function to transmit and receive a single byte over SPI with pseudocode usage as follows:

```
rxChar = spi_exChar(txChar);
```

- Then write the high level pseudocode required to
  - read 4 bytes starting at EEPROM address 0x1000
  - copy the bytes just read back out to EEPROM address 0x2000