

# 3.30 Synchronous Serial Communications

Inter Integrated Circuit (I<sup>2</sup>C)

The Serial Peripheral Interface (SPI)

EE302 – Real time and embedded systems

1

## Overview

2

### □ 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<sup>2</sup>C
- **Describe** detailed operation of SPI and I<sup>2</sup>C, including line coding and handling of single and multi-byte communications
- **Interpret** a synchronous serial communications specification (SPI or I<sup>2</sup>C) 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<sup>2</sup>C 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/I<sup>2</sup>C exchanges
- **Write/show** the line coding of a sequence of SPI/I<sup>2</sup>C exchanges

17 November 2020

2

## Synchronous Communication

3

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

17 November 2020

3

## Synchronous serial vs. asynchronous serial communications

4

*Synchronous serial communications differs from async serial communications in several respects, namely...*

- Uses Controller-Peripheral architecture
- Controller and peripheral are connected via a communications bus (not a point to point line)
- Transmits both data and an explicit clock signal
  - Controller 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 asynchronous communications

17 November 2020

4

## Inter Integrated Circuit (I<sup>2</sup>C)

5

17 November 2020

5

## I<sup>2</sup>C (Inter Integrated Circuit)



6


- 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 Controller/Peripheral architecture
  - Controller and Peripherals are connected by a bus
  - A dedicated clock signal (SCL) is transmitted independently of the data
  - Data is transmitted MSb (most significant bit) first

17 November 2020

6

## I<sup>2</sup>C unique features (different to SPI) - 1

7

- Bus consists of just two wires: serial data (SDA) and serial clock (SCL)
  - Saves wires/pins
  - Because there is just one data line, all communication is **half-duplex**, i.e. Controller and Peripheral 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
  - Controller usually controls SCL, but Peripheral can pull SCL low to pause a transfer (this is called **clock stretching**) 

17 November 2020

7

## I<sup>2</sup>C unique features (different to SPI) - 2

8

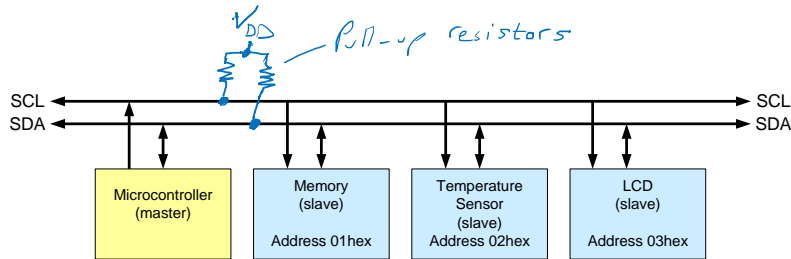
- I<sup>2</sup>C protocol
  - Divides all communication into two operations: read (Controller asks Peripheral for data) and write (Controller sends data to Peripheral)
  - Receiver must acknowledge all data received (ACK or NAK) which helps make communication robust
  - The Controller transmits an I<sup>2</sup>C 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)

17 November 2020

8

## I<sup>2</sup>C Connectivity and addressing

9



I<sup>2</sup>C transmits a 7 or 10-bit addressing scheme to select devices.

(Contrast this with explicit chip select lines in SPI – see later)

(Note: We will focus on 7-bit addressing from here on.)

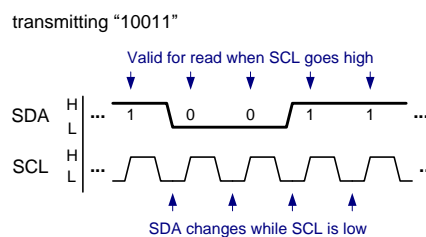
17 November 2020

9

## I<sup>2</sup>C Line coding

10

- The transmitter (Controller or Peripheral) 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 (Peripheral or Controller) assumes that the SDA level is stable and valid and reads the corresponding bit value



Q. How does the raw bit rate compare to the SCL frequency? E.g. If SCL frequency is 100KHz, what is the raw bit rate (based on when SDA levels can change)?

17 November 2020

10

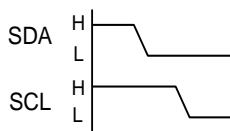
## I<sup>2</sup>C Line coding: start and stop conditions

11

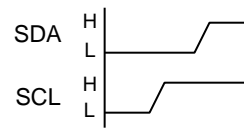
- I<sup>2</sup>C can support multiple Controller devices on a single bus, so it uses explicit start and stop “conditions” (like start and stop bits in async comms) to delimit communications.
- In the idle state (between communications) SCL and SDA are high (remember pull up resistor so it floats high)
- **Start** (SDA low followed by SCL low) reserves the I<sup>2</sup>C 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



17 November 2020

11

## I<sup>2</sup>C Protocol overview


12

*I<sup>2</sup>C defines a specific protocol, with the following protocol elements which must be followed for each communication...*

- **START condition** [Controller]
  - All transmissions begin with a START condition
- **I<sup>2</sup>C address and Read/Write bit** [Controller], **ACK** [Peripheral]
  - First is the 7 bit or 10 bit I<sup>2</sup>C address which identifies the Peripheral device for this communication is transmitted
  - Next is the Read/~Write bit is transmitted
    - 1 means Read, 0 means Write
  - The selected Peripheral 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).
    - 0 means ACK, 1 means NACK

17 November 2020

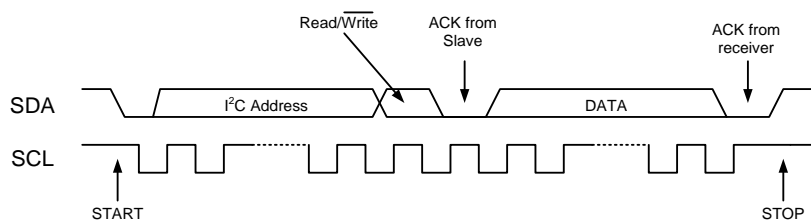
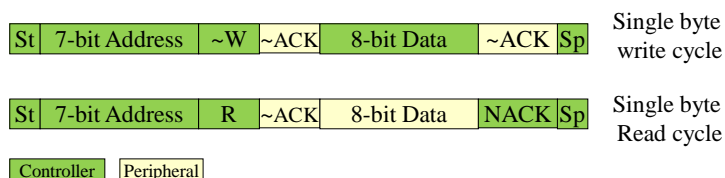
12

- **Data** [Transmitter = Controller or Peripheral] and **ACK/NACK** [Receiver = Peripheral or Controller]
  - The transmitter (Controller for write operation, Peripheral for read) transmits the data one byte at a time and wait for ACK/NACK before continuing
  - The receiver (Peripheral for write operation, Controller for read) must acknowledge each byte by setting the SDA to  $\sim$ ACK/NACK for one bit period immediately after each byte has been received
    - ACK means OK, and can receive more data
    - NACK (leaving SDA float high) can be used by a peripheral to indicate a problem, but it is also used by the **Controller in a read operation to indicate no more data wanted/needed**
- **STOP condition** [Controller] 
  - Communication ends with a stop condition

17 November 2020

13

## I<sup>2</sup>C Protocol summary



Read/Write Cycles

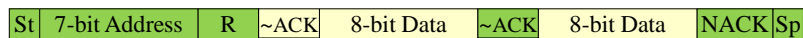
17 November 2020

14

## 2 Byte Write Example



## 2 Byte Read Example



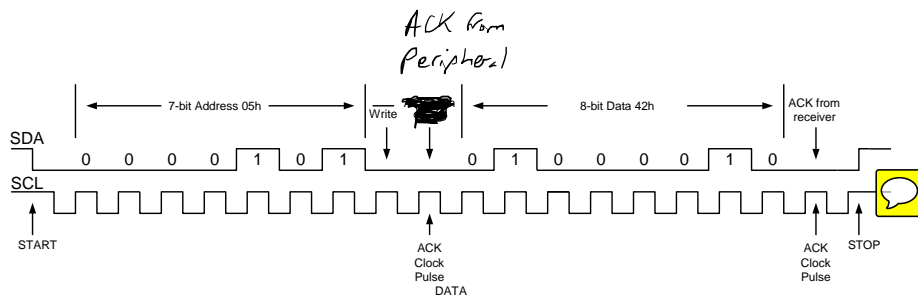
17 November 2020

15

## Detailed Example: 1 byte write

In this example we write the byte 0x42 to the peripheral device at I<sup>2</sup>C address 0x05.

Note that data is sent MSb first.



17 November 2020

16



## Self test questions on I<sup>2</sup>C (basic)

17

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

17 November 2020

17

## Self test questions on I2C (intermediate)

18

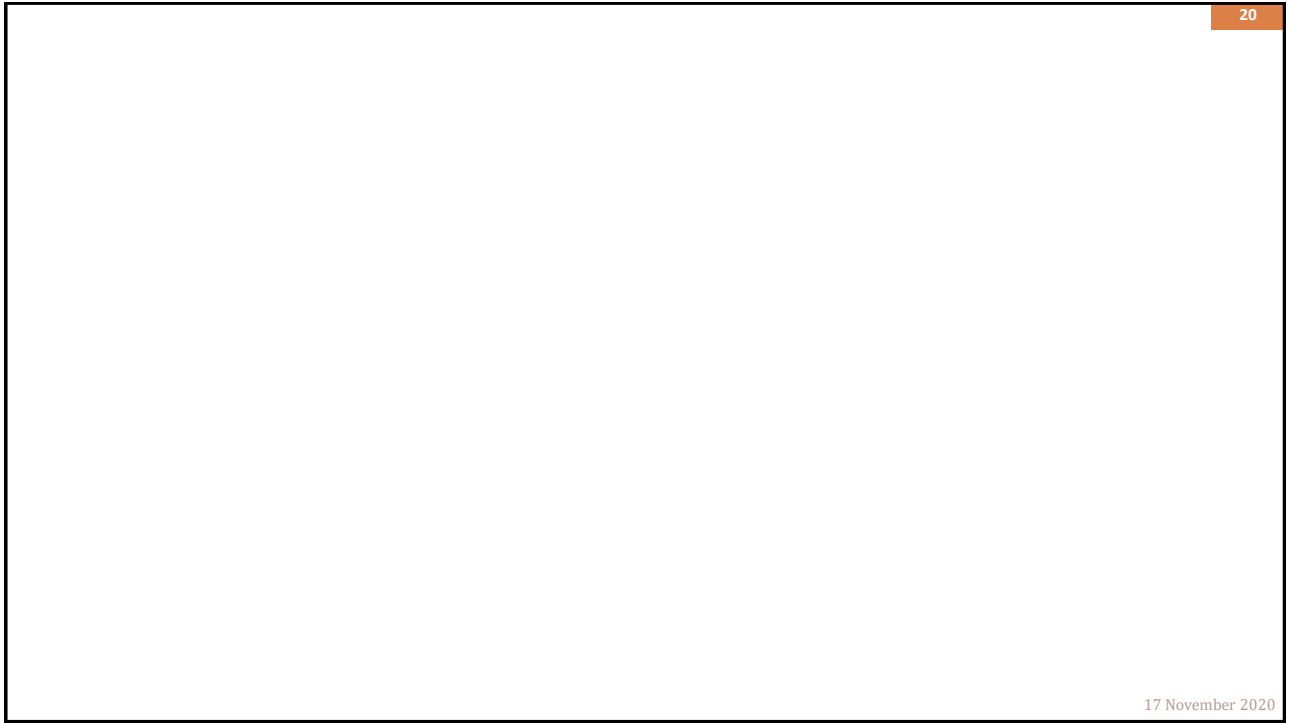
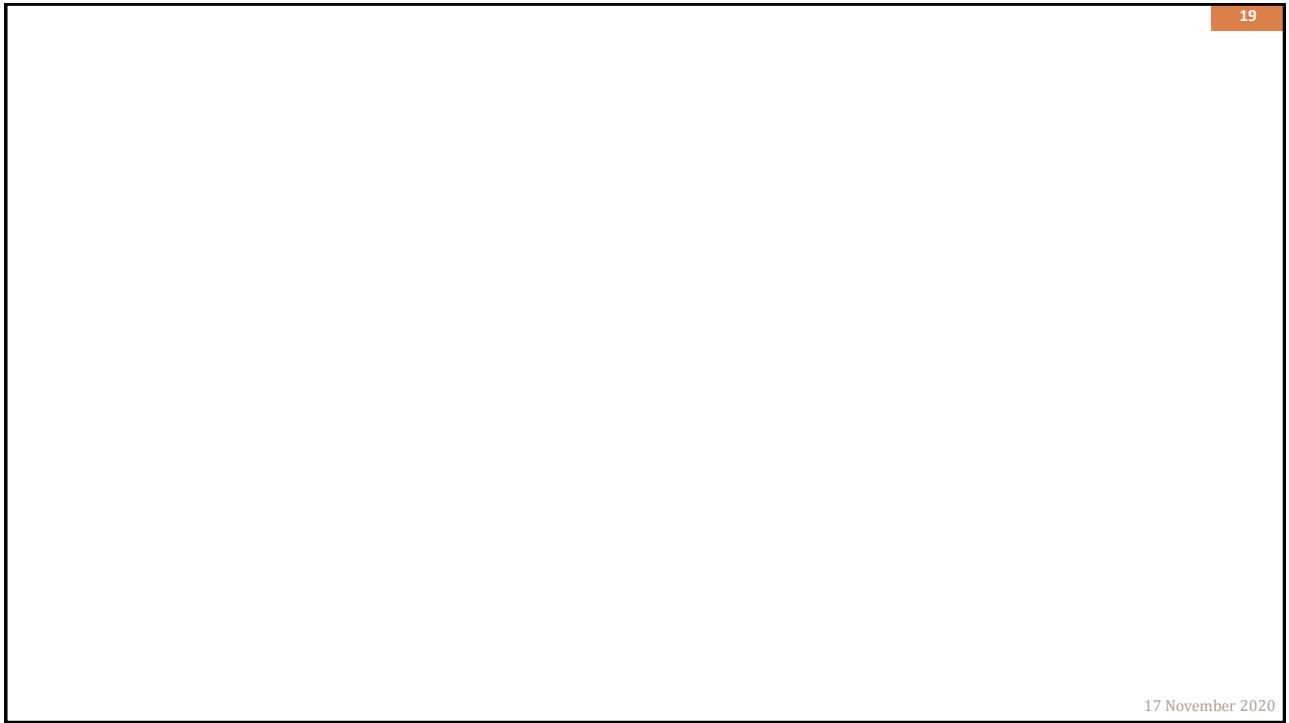
Q1. A Controller reads the 2 byte value 0x0142 from a I2C peripheral (Peripheral). Draw and carefully label the complete timing diagram explaining any assumptions made.

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

- ☐ 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

17 November 2020

18



## Exam type I<sup>2</sup>C question

25

*Background: A PIC microcontroller is connected to an EEPROM (address 0x40) via I2C*

- *The EEPROM keeps a single memory address pointer which points to the current data location to be read or written. It is automatically incremented after each data byte is written or read.*
- *To write to the EEPROM: Use a write operation and send the memory address pointer value (MSByte first) that you want to access, then send the data bytes to be written to the EEPROM starting at that location*
- *To read from the EEPROM*
  - *If the memory address pointer needs to be changed, first do a write operation to set it (as per EEPROM write, but without sending any subsequent data bytes for writing)*
  - *Then do an I2C read, and ACK each byte received, sending a NACK after the last desired byte*

17 November 2020

25

## contd

26

### Questions

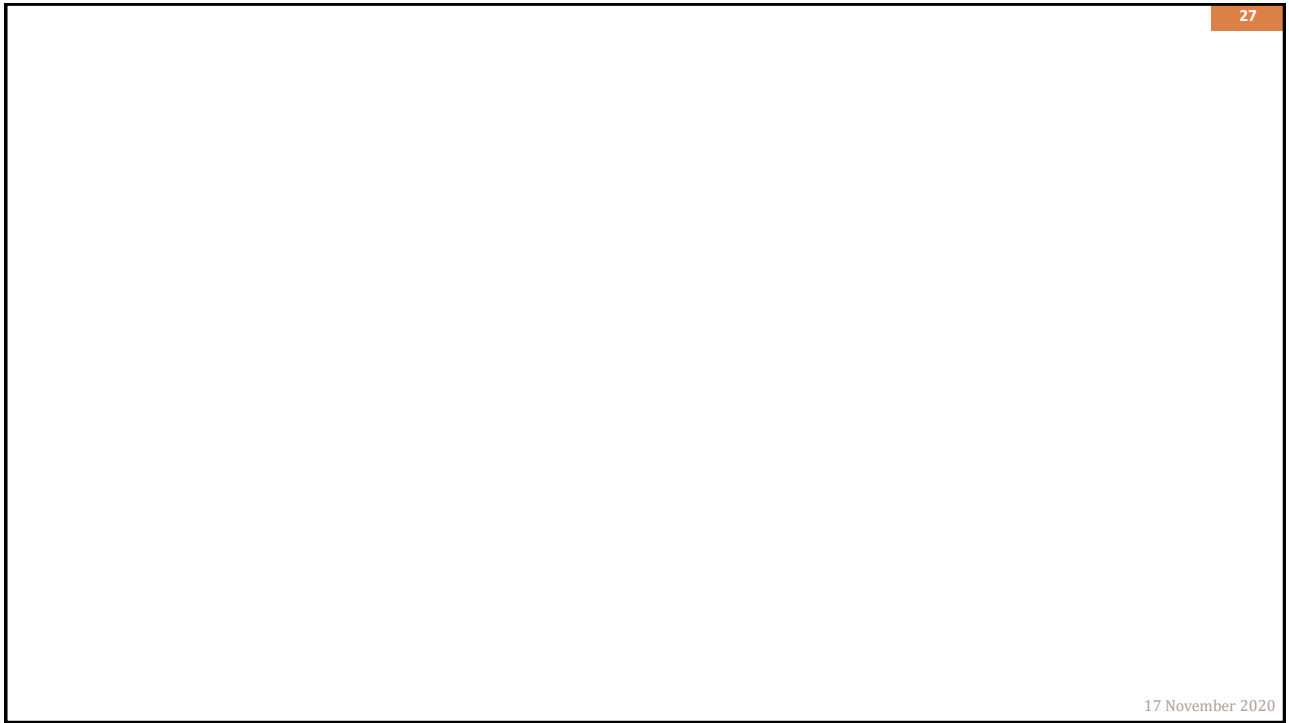
- *Draw the high level connectivity of the PIC and EEPROM*
- *Write the high level code required to write the values 10, 11, 12 starting at location EEPROM location 0x1230*
- *Write the high level code required to read one value from location 0x2340*

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

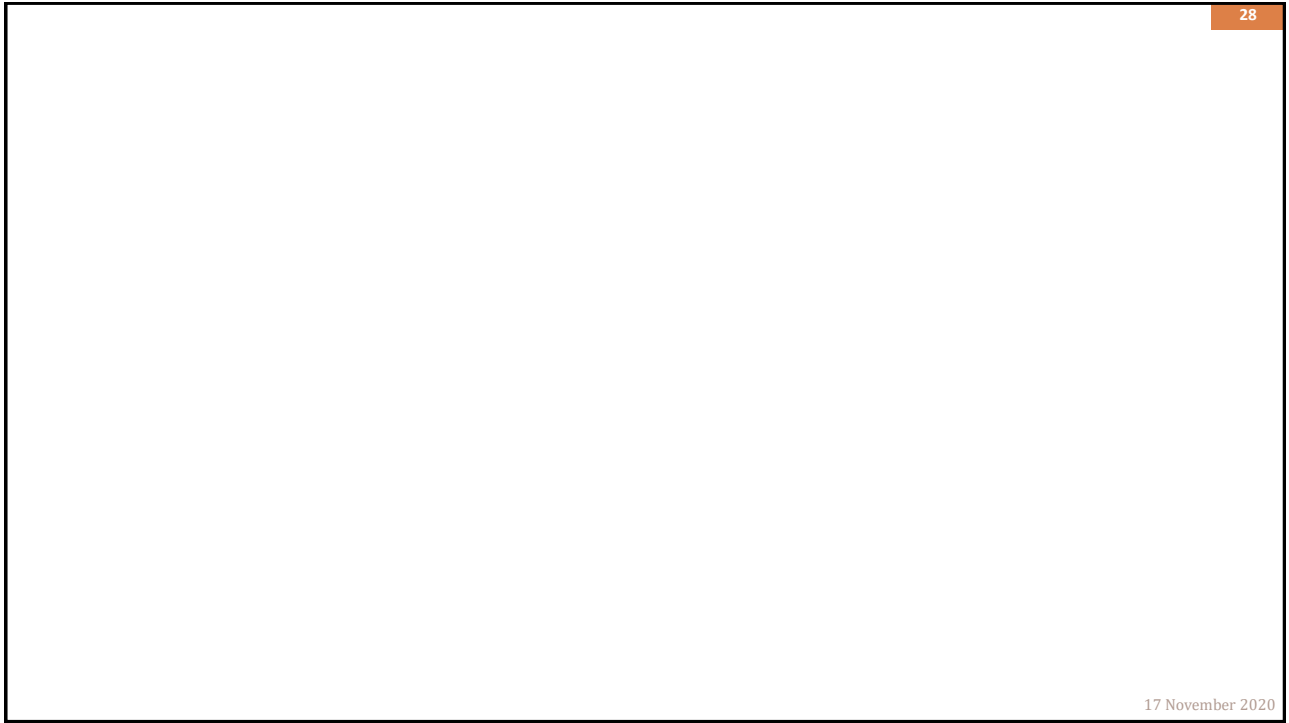
```
ack = i2c_start(i2cAddress, readOrWrite)
ack = i2c_write_byte(byteValue)
byteValue = i2c_read_byte(ackOrNackResponse)
i2c_stop()
```

17 November 2020

26



27



28

## Serial Peripheral Interface (SPI)

30

17 November 2020

30

## Serial Peripheral Interface (SPI)

31

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

17 November 2020

31

## SPI unique features (compared to I2C) - 1

32

- 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 Controller controls the clock – Peripherals may not manipulate the clock (compare to I<sup>2</sup>C)
  - Data is exchanged on clock edge transitions
  - Clock polarity and edge/phase is configurable (details on later slide)

17 November 2020

32

## SPI unique features (compared to I2C) - 2

33

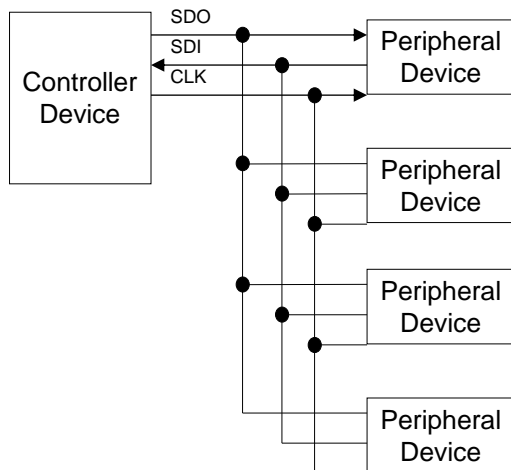
- To communicate with a Peripheral device, the Peripheral must first be selected
  - Peripheral Select lines are unidirectional point-to-point and independent of the bus (see next slide)
- How does SPI compare to I<sup>2</sup>C?
  - Full duplex comms and no start/stop conditions, no protocol (address, read/write, ACK/NACK, etc)
  - Hence, SPI has higher effective data rate for same clock speed

17 November 2020

33

## Controller/Peripheral architecture

34



### COPI

- ☐ Controller Out, Peripheral In
- ☐ Controller SDO connects to Peripheral SDI

### CIPO

- ☐ Controller In, Peripheral Out
- ☐ Controller SDI connects to Peripheral SDO

### Question:

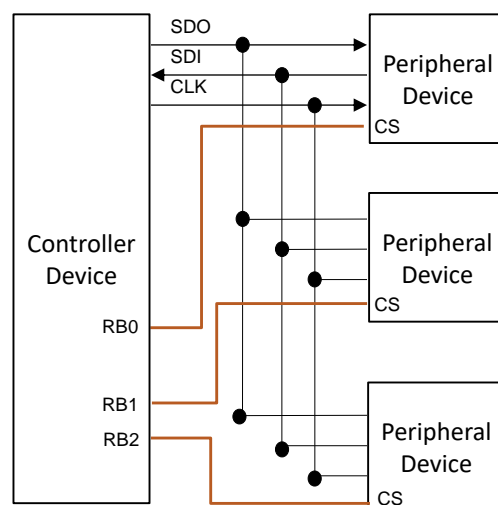
Each device has a **Chip select (CS)** line.  
So how might a particular Peripheral device be selected???

17 November 2020

34

## Peripheral selection/activation

35



Example showing chip selects being driven directly by digital output port on microcontroller

17 November 2020

35

## SPI Clock

37

- Clock polarity
  - specifies the idle state of clock line between data exchanges: it can be 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 (both edge triggered – compare to I2C)
    - Set on rising edge, read on falling edge
    - Set on falling edge, read on rising edge

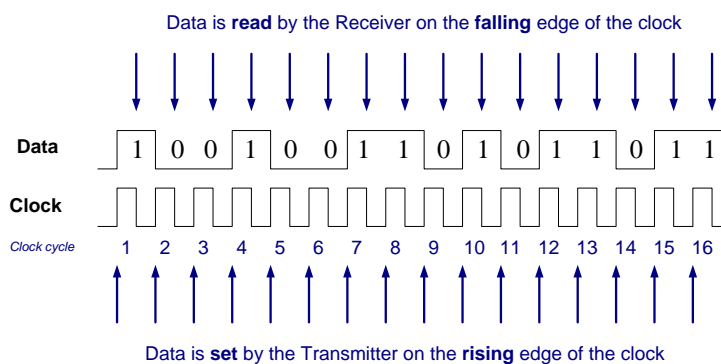


17 November 2020

37

## SPI timing (each byte)

38



In this example:

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

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

17 November 2020

38

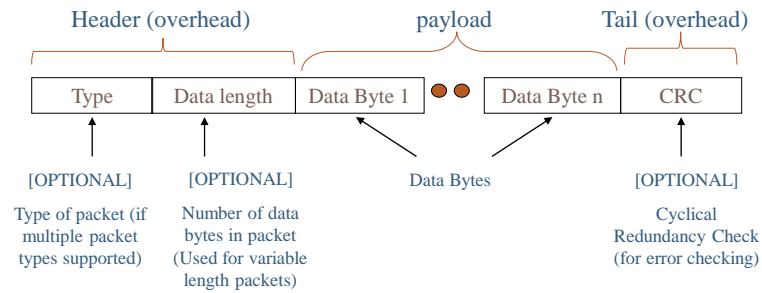


## SPI multi-byte packets

39

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)*

17 November 2020

39

## SPI self test questions (basic)

40

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)?



17 November 2020

40

## SPI self test questions (intermediate)

41

*A Controller transmits 0x53 to a Peripheral device using the SPI protocol. Draw the timing diagram of the Controller'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 Peripheral must be deselected for 100 microsecs between packets*
- c) Packets are 32 bytes long (variable) but include a single byte header; the Peripheral must be deselected for 100 microsecs between packets*

17 November 2020

41

42

17 November 2020

42

## SPI question (more advanced)

45

A **PIC microcontroller** is connected to an EEPROM using SPI and selected (active low) via RC7

- To read from EEPROM: transmit read code (0x03), followed by 2 byte memory address (MSByte 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 (MSByte 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. In pseudocode, you use it as follows (substituting rxChar and txChar as necessary).

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



Questions:

- i. Draw the high level connection diagram
- ii. write the high level pseudocode required to read 4 bytes starting at EEPROM address 0x1000
- iii. write the high level pseudocode required to copy the 4 bytes just read out in (ii) back to the EEPROM starting at address 0x2000

17 November 2020

45

46

17 November 2020

46

