

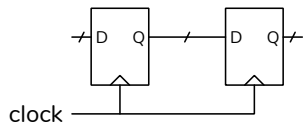
Interfaces

Digital circuits are used to transfer data between devices. This lecture describes the operation and design of some common interfaces.

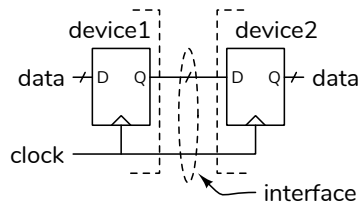
After this lecture you should be able to: classify an interface as serial or parallel, synchronous or asynchronous, uni- or bi-directional and explain the advantages of each; draw the schematic or write the Verilog for a synchronous serial transmitter or receiver; convert data transmitted over an asynchronous serial or SPI interface to the interface waveform(s); extract the data from these waveforms.

Parallel Interfaces

We've seen how data can be transferred between two flip-flops by connecting the Q output of one flip-flop to the D input of another and using a common clock:



If the two flip-flops are on different devices – whether two IC packages or two pieces of equipment – we can connect them this way to transfer data between them:



This is the simplest type of interface between two devices and can transfer any number of bits on the same clock edge.

An example is the 8-bit **parallel printer port** that was used by early personal computers. This interface used the falling edge of a **STROBE** signal as the clock and an active-high **BUSY** signal that indicated the printer was unable to accept another character.

Parallel interfaces to memory IC's are divided into a “data bus” for the bits to be transferred and an “address bus” for the memory location where the data is to be read from or written to. These interfaces will be covered in more detail later.

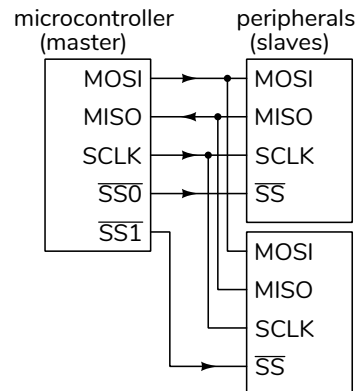
Serial Interfaces

The bits of a word can also be transferred over an interface sequentially (serially), typically one bit at a time. Although serial interfaces are more complex, this is often more than offset by lower costs due to fewer IC pins, smaller connectors, less PCB area, and lower cost cables.

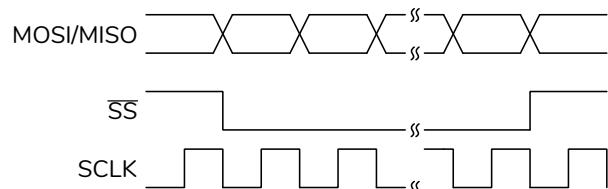
Example: SPI

The **Serial Peripheral Interface** (SPI) is a common serial interface between a “master” (typically a microcontroller) and a “slave” (typically a peripheral IC). Applications include LCD controllers and SD cards.

The SPI interface has one data signal in each direction (named **MOSI** and **MISO**), a clock signal (**SCLK**) and a (typically active-low) slave-select (**\overline{SS}**) signal.

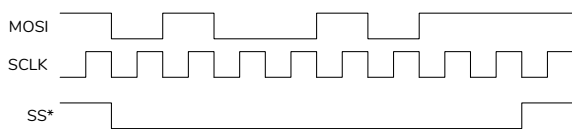


The following timing diagram shows the operation of the bus:

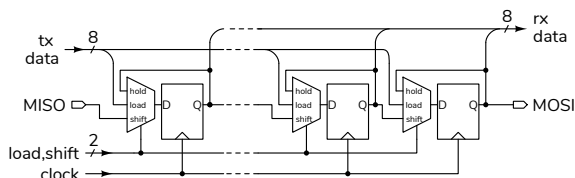


The data transfer begins when the master asserts \overline{SS} . On the following clock edges¹ one bit is transferred in each direction. Typically, multiples of 8 bits are transferred, most-significant bit first. \overline{SS} is de-asserted when the transfer is done.

Exercise 1: The diagram below shows a transfer over an SPI bus. How many bits of data are transferred and what is the decimal value of this data?



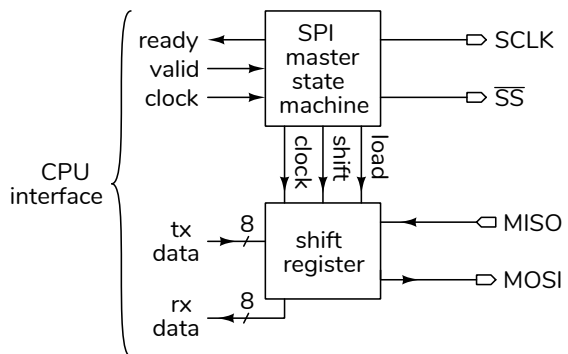
An SPI master interface can be implemented with a shift register that has parallel inputs and outputs as shown below:



When **load** is asserted the shift register loads the eight **MOSI** bits to be transmitted. When **shift** is asserted a **MISO** bits shift in on the left and a **MOSI** bit shifts out on the right. At the end of the transfer the eight received **MISO** bits can be read in parallel from the shift register.

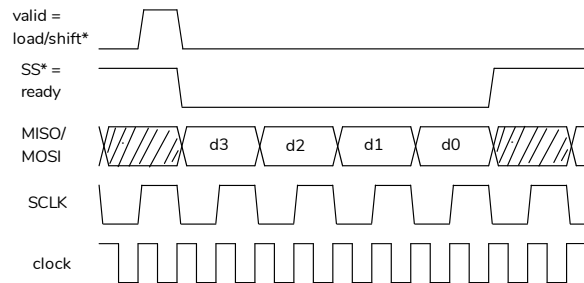
The interface to the processor has two 8-bit parallel data signals (input and output) and two control signals: **ready**, set true when the interface can accept another byte and **valid**, set true when another byte can be transmitted.

The controller is a state machine that sequences through 16 states to transmit a byte, two states for each bit (one for **SCLK** high and one for **SCLK** low).



¹SPI interfaces can be configured so that the data is sampled on either the rising or falling edge of **SCLK**.

The diagram below shows the waveforms for an abbreviated (4-bit) transfer over an SPI interface:



Exercise 2: Based on the diagram above, write a state transition table for an SPI interface controller that transfers four bits at a time. Include an idle state. In which states are **SCLK** and \overline{SS} asserted?

Note that **SCLK** is the clock signal for the interface, not for the interface's logic circuits.

The slave SPI interface will also be implemented with a state machine that synchronises to the transmitter using \overline{SS} . Note that both master and slave must be configured for the same bit order and for whether **MOSI/MISO** and **SS*** change on the rising or falling edge of **SCLK**.

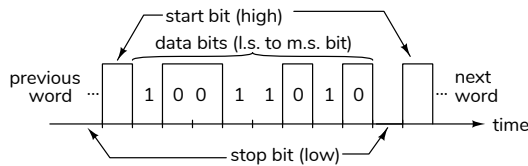
Asynchronous Interfaces

We can simplify the interface further by omitting the clock. This requires that the clock signal be regenerated at the receiver so that the bits can be sampled and shifted in at the correct time.

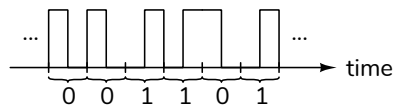
The receiver uses an internal clock running at approximately the same frequency as the transmitter. But it must periodically re-synchronize its clock with the transmitter clock to ensure the two clock's edges remain aligned. To do this the receiver looks for changes in the input data signal in-between its clock edges.

Accurate synchronization thus requires periodic changes in data signal level, even if the data itself does not contain transitions (e.g. it's always low or always high). There are various ways of ensuring this:

- Sending a pair of bits of opposite level in-between words. An example is the "stop" and "start" bits used in "RS-232" (and "RS-485") asynchronous serial interface. In this interface a low level is used for a 1 and a high level for a 0:

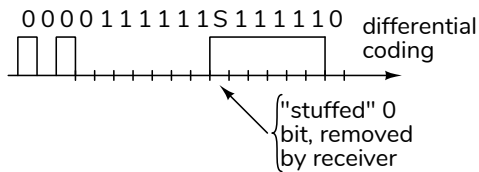


- Encoding each bit as either a pair of H-L or L-H bits, called “Manchester” coding.



There are two conventions. That shown above, where the rising edge in the middle of the bit represents a ‘1’, is used by the 10 Mbps **10BASE-T Ethernet** standard.

- **Differential coding** in which a 0 is transmitted as a change in level, and a 1 as no change, combined with “bit stuffing” in which a zero (a transition) is inserted by the transmitter and removed by the receiver after 6 consecutive one’s (no transitions). An example is the **USB 2** interface.



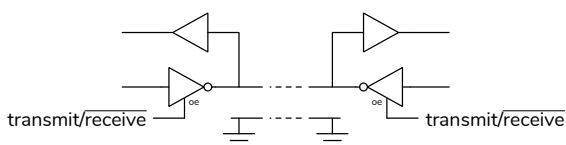
Most asynchronous serial interfaces, including the three mentioned above, transmit the data bits least-significant bit (l.s.b.) first.

Exercise 3: Draw the RS-232, Manchester and USB 2 waveforms that would be transmitted for the byte value 8’h3f.

Bi-Directional Interfaces

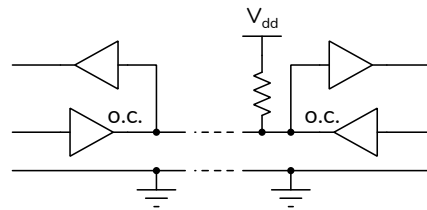
We can further reduce the number of conductors required by using the same ones to transmit data in both directions.

One way is by using tri-state outputs that are alternately enabled so that only one side of the interface is configured as an output at any time:



this is the approach used by **USB**.

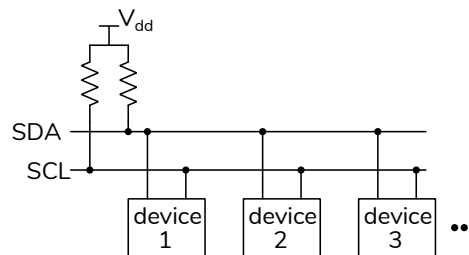
Another way is by using open-collector outputs so that any device can pull the bus low in a “wired-OR” configuration:



Addressable Multipoint Interfaces

Instead of using enable signals (e.g. \overline{SS}) to enable specific devices, each device can be assigned its own address. A master can enable a specific device by sending the slave’s address followed by the data. This also requires a way to indicate the start and end of the data (framing).

The Inter-IC Communications (I^2C) protocol is an example. This bus connects IC’s in parallel with two signals: **SDA** (data) and **SCL** (clock), both of which use open-collector bidirectional buses. A 7-bit address is transmitted before the data to enable a specific device. I^2C is often used for low-speed (100 kbps) peripherals such as temperature sensors and configuration memories.



Often, one device is a master and “polls” the slave. After each poll the master turns off its driver and the slave turns on its driver for the duration of the response. However, there are also interfaces (including I^2C) where multiple devices can contend to become the bus master using an “arbitration” protocol.

Some interfaces, such as **1-wire** and **CAN bus** eliminate both the clock and the select lines by using asynchronous signalling and device addressing.

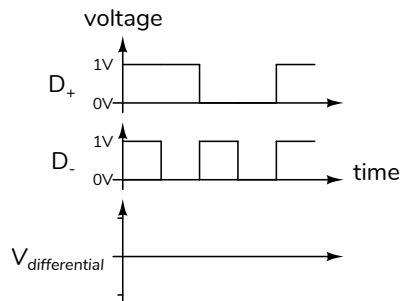
Device Descriptors

Some interfaces support retrieval of “descriptors” – blocks of data that identify a device. For example, each USB interface contains an 18-byte descriptor that identifies the device class (e.g. a human interface device), a subclass (e.g. a keyboard), and vendor and product ID numbers. These allow an operating system to configure the interface and load the appropriate device drivers.

High-Speed Interfaces

Different electrical standards are needed at higher speeds or longer distances (>100’s of MHz and >10’s of cm). Data is often transmitted as the voltage difference between two signals – “[differential signalling](#)”. The lengths of the two conductors needs to be the same. At high speeds the impedance of the transmitter and receiver need to be matched to the impedance of the transmission line to avoid reflections.

Lower voltages are often used to reduce power consumption.



Exercise 4: The diagram above shows the voltages on a differential pair. The differential voltage is $V_{\text{differential}} = D_+ - D_-$. Draw the differential voltage waveform.

Common examples of high-speed differential interfaces used in computers include [LVDS](#) (Low Voltage Differential Signalling) for internal LCD displays, HDMI and DisplayPort for external displays, PCIe for internal peripherals, and SATA for storage devices.

A [SerDes](#) (Serializer/De-Serializer) is the circuit within an IC that converts between serial and parallel formats. A SerDes often provides clock recovery, framing and error detection as well.

Example: USB

Universal Serial Bus (USB) is a popular peripheral interface. It’s an asynchronous bidirectional serial interface. Earlier versions (“USB 2.0”) of the interface used four conductors: two for a differential, bidirectional data signal, one for ground, and one for power (+5 V). Each USB bus has a master that controls the bus by polling (although devices are not connected in parallel) and each device has a descriptor.