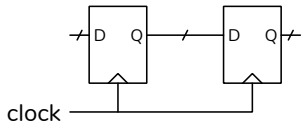
Digital Interfaces

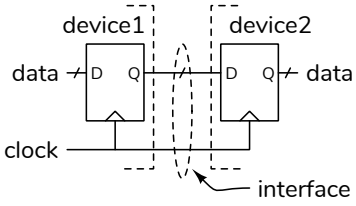
Digital circuits are used to transfer data between modules and devices. This lecture describes the operation and design of some common interfaces.
After this lecture you should be able to: classify an interface as serial or parallel and uni- or bi-directional and explain the advantages of each; ; determine when data is transferred over a ready/valid interface; draw the schematic or write the Verilog for an SPI transmitter or receiver; convert data transmitted over an SPI interface to the interface waveform(s) and 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:



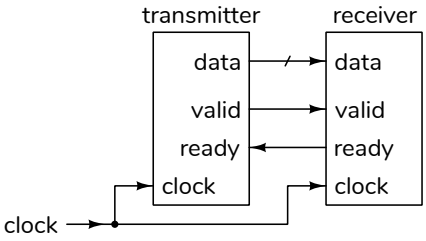
If the two flip-flops are on different devices – whether two IC packages or two pieces of equipment – we can connect them this way to transfer data between them:



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

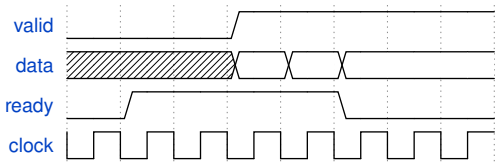
Ready/Valid Interfaces

There may not be new (valid) data to transfer on every clock edge and the receiving device may not be ready to accept data on every clock edge. For example, some devices may need multiple clock cycles to process the data being transmitted or received. Two "handshaking" signals, often called valid and ready, can be used to control the transfer of data[1]:



- The transmitting device asserts a **`valid`** output when its data output is valid.
- The receiving device asserts a **`ready`** output when it is ready to accept data on the next clock edge.
- Data is only transferred on clock edges where *both* **`valid`** and **`ready`** are asserted.

Note that both devices use the same clock signal. Thus the states of both devices change at the same time.



Exercise 1: Mark the clock edges where data is transferred.

The **`valid`** and **`ready`** outputs are generated by the state machines that control the transmitter and receiver respectively.

Note that it is possible to transfer data on every clock edge. This happens when valid and ready are asserted on every clock edge.

Exercise 2: Draw the state transition diagrams for the transmitter and receiver if the transmitter has a data word ready two clock cycles after the previous one is read and the receiver requires three clock cycles to process each received word. Draw the timing diagram assuming valid and ready are asserted to start with.

[1] This is sometimes called a FIFO (First-In First-Out) interface and can also be used between modules.

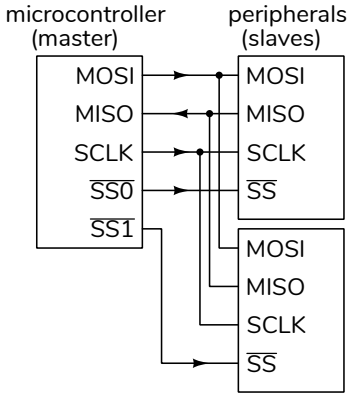
Serial Interfaces

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

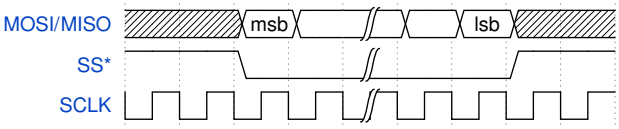
Example: SPI

The Serial Peripheral Interface (SPI, pronounced "ehs-pea-eye" or "spy")) 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 ($\overline{\text{SS}}$) signal.



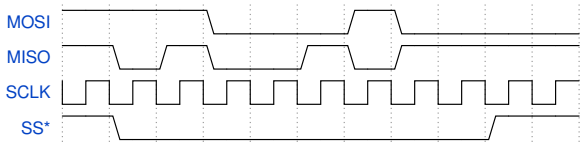
The following timing diagram shows the operation of the bus:



The data transfer begins when the master asserts $\overline{\text{SS}}$. On the following clock edges[2] one bit is transferred. $\overline{\text{SS}}$ is de-asserted when the transfer is done. Note that the same number of bits are always transferred in each direction. Transfers are typically in multiples of 8 bits, most-significant bit (m.s.b.) first.

The SPI interface only specifies how bits are transferred. Their meaning depends on the specific slave device and will be defined in the datasheet for that device.

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

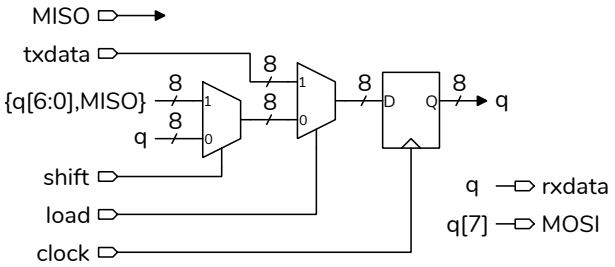


Exercise 3: The diagram above shows a transfer over an SPI bus. How many bits of data are transferred? What is the value, in decimal, of the data transferred from the master to the slave? From the slave to the master?

Implementing an SPI Interface

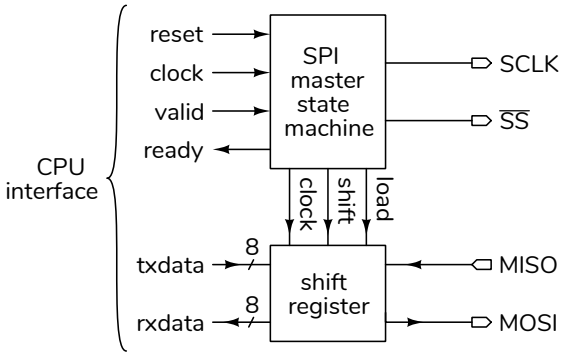
An SPI interface is typically connected to a CPU over a parallel interface with ready/valid handshaking. We can divide the design of such an interface into two parts: a "datapath" that contains data registers and logic, and a "controller" state machine that controls the operation of the datapath.

The datapath of an SPI master interface can be implemented with a shift register with a parallel input (**txdata**) and a parallel data output (**rxdata**) as well as serial input (**MISO**) and output (**MOSI**) as shown below:



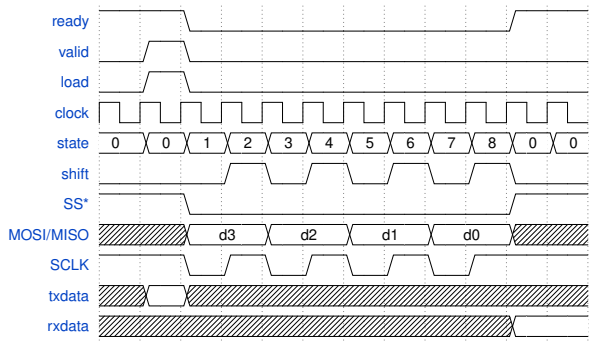
When **load** is asserted, the register loads eight bits from **txdata**. When **shift** is asserted, a **MISO** bit shifts in on the right and the **MOSI** bit shifts out on the left. After 8 clock cycles the transfer is done and the eight received **MISO** bits can be read in parallel from the shift register on **rxdata**. The parallel data is typically read and written by a microcontroller over a parallel interface.

An example of an 8-bit SPI interface is shown below. It has two 8-bit parallel data signals (**rxdata** and **txdata**) and two control signals: a **ready** output that is set true when the interface can accept another byte and a **valid** input that is set true when **txdata** holds the next byte to be transmitted.



The controller for the SPI interface is a sequence generator state machine with 16 states for each byte. It has two states for each bit (one for **SCLK** high and one for **SCLK** low).

The diagram below shows the waveforms for a 4-bit SPI transmit interface:



Exercise 4: The diagram above shows the signals and state variable for a 4-bit SPI interface. The controller sequence generator states use a binary encoding. Write a state transition table for this state machine. Write Verilog expressions for the controller outputs.

Note that **SCLK** is generated by the controller state machine; it is not the clock signal used by the controller or datapath.

The slave SPI interface will also be implemented with a state machine that synchronises to the transmitter using $\overline{\text{SS}}$. 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 serial interfaces 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, including "RS-232", Manchester, and Differential coding.

Bi-Directional Interfaces

We can also use one conductor to transmit data in both directions. This requires the use of tri-state or open-collector outputs and a protocol to control which device may transmit.