Lecture 6

Embedded system design

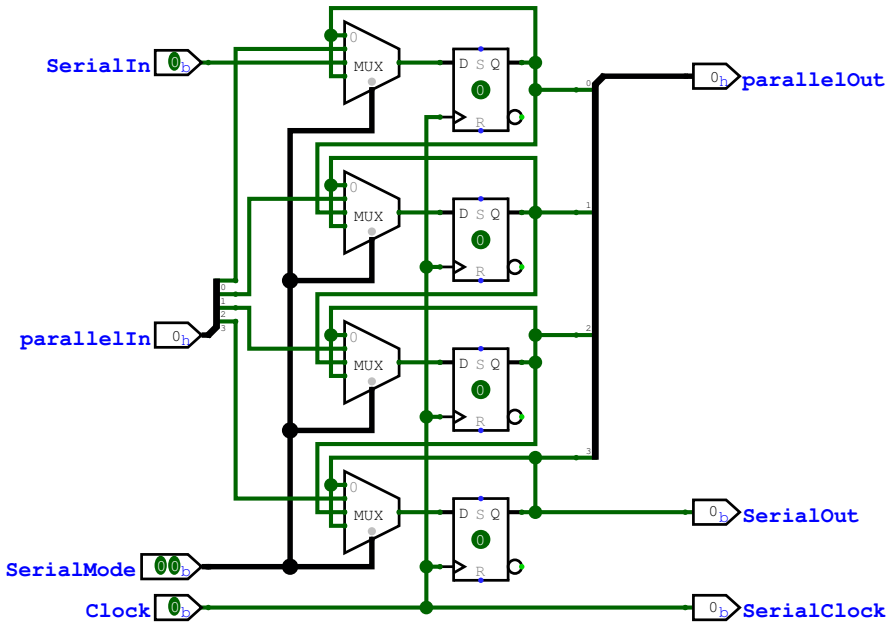
Serial protocols

CS476 - ESD
April 16, 2024

Dr. Theo Kluter
EPFL

Introduction

- Now that we have seen most of the interior of an embedded system we are going to look into the peripherals.
- Many peripherals are based on serial protocols, like I^2C, I2S, RS232, CAN, ...
- To be able to convert parallel data to serial and vice versa, a shift register is used.
- Note that shifting to the left or shifting to the right is basically the same circuit!
- In this lecture we are going to visit some of these protocols.



SerialMode:	Function:
00	Keep current value
01	Load parallel data
10	Shift Data

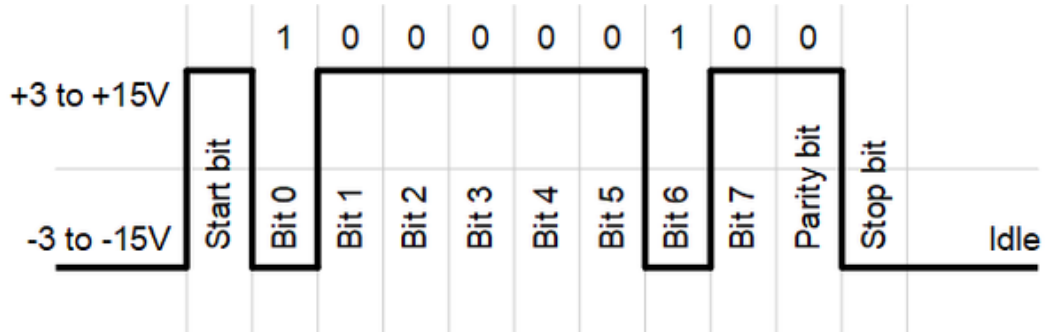
History: RS-232 still a protocol to be found everywhere

- Arguably the “oldest” serial protocol is the RS-232 introduced by the *Electronic Industries Association (EIA)* in 1960.
- It is an asynchronous point-to-point protocol that still is very “active” today. Note that it was the “enabler” for the internet as we know it today.
- RS-232 is also know as *Universal Asynchronous Receiver/Transmitter (UART)*.
- Although the protocol dates from 1960, the latest “update” dates from 2012.
- It is a “proven protocol” that can be even found in current server systems as “backup” interface in case something went wrong.
- And you are using it each time for uploading your program to the VP. So how does it work....
- The original interface was:

Signal:	**Function:**
TxD	Transmit Data
RxD	Receive Data
DTR	The slave is ready to receive, initiate, or continue a call
DCD	The slave is detecting a carrier from the remote device
DSR	The slave is ready to receive and send data
RI	The slave detected an incoming call
RTS	The slave is ready to receive data
CTS	The master is ready to send data

Today: RS-232 still a protocol to be found everywhere

- Although most of the signals are “obsolete”, even most interfaces found nowadays provide them for back-ward compatibility.
- However, most of the time we only use the `TxD` and `RxD` signals. These are cross-connected between the two devices allowing for full-duplex communication.
- The RS-232 sends the information by frames:

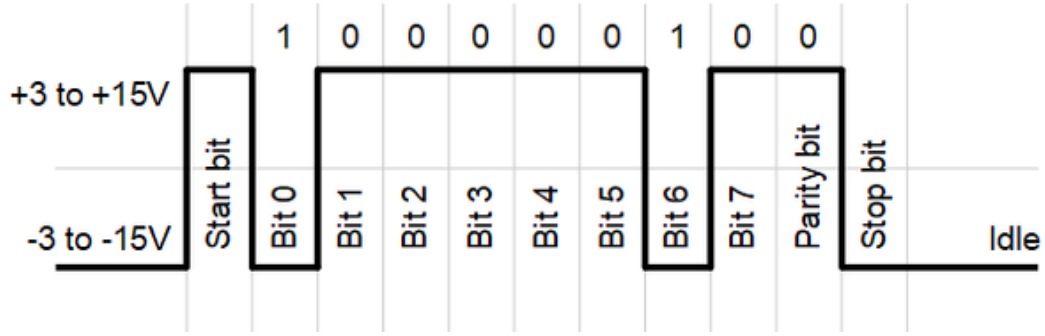


source: Opencircuits

  - There are 8 data-bits shown above, however, the protocol allows for 5 to 9 data bits.
  - The parity-bit is optional and can be odd, even, mark(`1`), or space(`0`).
  - The frame is ended by one or two stop bit(s).
- The speed of the communication is defined by the *baudrate*. Basically this measures the bits transferred per second.
- Note that the voltage levels are nowadays also be allowed to be `GND` and `VCC`.

RS-232 is robust but slow

- Although the RS-232 is so wide spread, it is relatively slow.
- Let's take a baudrate of 115200 baud and a frame as depicted below:



source: Opencircuits

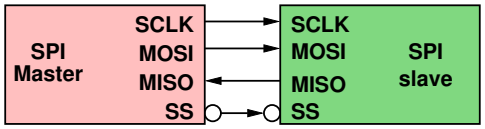
As the frame consist of 11 bits (containing one data byte), we can transfer a maximum of:

$$\frac{115200}{11} \approx 10472.7 \frac{\text{bytes}}{\text{s}} \approx 10.5 \frac{\text{kBytes}}{\text{s}}$$

- Hence we require more performing protocols.

Serial Peripheral Interface (SPI)

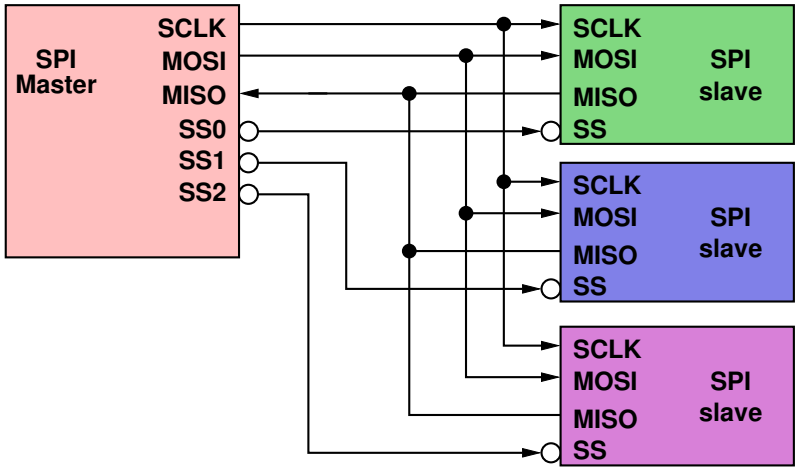
- The arguably “simplest” serial interface to implement is the *Serial Peripheral Interface (SPI)*.
- The SPI is a synchronous serial communication interface specification.
- Developed by Motorola in the mid 1980s and nowadays a de facto standard.
- The SPI has following signals:



  - `SCLK`: Serial CLocK (output from the master).
  - `MOSI`: Master Out Slave In (data output from the master).
  - `MISO`: Master In Slave Out (data input to the master).
  - `SS`: Slave Select (often active-low). Output(s) from the master to select the slave to communicate with.
- Signal/pin names as well as their timing constraints vary among manufacturers. Always check the device data sheet!
- Most slave devices have tri-state outputs, i.e., their `MISO` output becomes high impedance if their `SS` input is not active. This allows that in a multiple-slave system all `MISO` signals can be connected together.

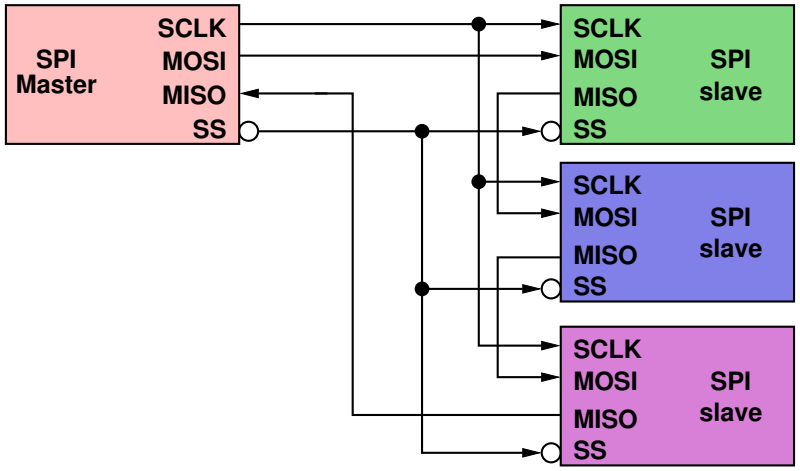
SPI configurations with multiple slaves

- A SPI configuration with independent slaves:



- This configuration requires one dedicated `SS` line per slave.

- Daisy-chained SPI configuration with cooperative slaves:

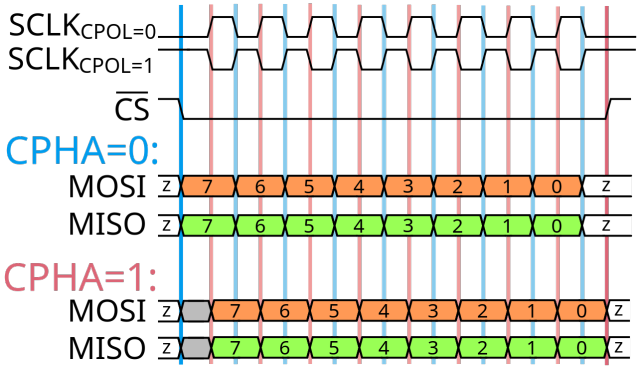


- All slaves need to use the same SPI mode, need to use the same data word length, and need to send out during second/third group of clock pulses an exact copy of the data received during first/second group of clock pulses.

SPI data transmission

- The SPI-protocol defines four modes of operation. The slave device(s) define which mode to use (see the datasheets of the slave device(s)). Hence it is well possible that a master device requires in a multiple-slave configuration to “talk” in different modi to the different slave.
- Furthermore, the slave device(s) determine the maximum `SCLK`-frequency that can be used during the communication (see the datasheets of the slave device(s)). In case of a daisy-chain, the slowest device restricts the communication speed.
- The four SPI-modi are defined by the polarity and phase of the clock (note: a communication is started/ended by the `SS` signal (in the timing diagram below the low-active $\overline{\texttt{CS}}$-signal):

SPI Mode:	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1



source: wikipedia

- `CPOL`: Clock POLarity
  - `CPOL=0`: clock idles at `0`, each cycle is a pulse of `1` with a leading rising and a trailing falling edge.
  - `CPOL=1`: clock idles at `1`, each cycle is a pulse of `0` with a leading falling and a trailing rising edge.
- `CPHA`: Clock PHAse
  - `CPHA=0`: Data on the `MOSI/MISO` is clocked out on the second clock-edge, and data on the `MISO/MOSI` is clocked in on the first clock-edge.
  - `CPHA=1`: Data on the `MOSI/MISO` is clocked out on the first clock-edge, and data on the `MISO/MOSI` is clocked in on the second clock-edge.

SPI: Applications

- Very simple and efficient for single master/single slave applications (e.g., digital audio, DSP, telecommunication channels) due to its full- duplex capability and high achievable clock speeds compared to UART.
- Widely used in embedded systems to interconnect components on PCBs and inside FPGAs due to considerable savings in board estate / routing resources.
- Microcontrollers and Systems on Chips (SoCs) typically contain SPI controller(s) to communicate with attached peripherals such as:
  - ADCs, DACs, audio codecs.
  - Sensors: temperature, pressure, distance, sound, touch.
  - Transceivers for other communication standards (Ethernet, USB, CAN, etc.).
  - Memories: EEPROM, Flash, SD cards
  - Displays/cameras: for configuration and sometimes even pixel data.

SPI: Extensions

- Although the SPI-interface is relatively “fast”, it only transports 1-bit each clock cycle. Especially for devices as Flash and SD-cards, this might be limiting as most of the time we only "read" their contents. This has lead to some extension to the SPI-protocol (note: not all slaves support these modes, you always have to consult the datasheets!).
- Note that these extensions need to be activated in the slave device; all slave devices start out with the “standard” SPI-configuration!
- These extensions are:
  - *Dual SPI*: In this case the `MOSI` and `MISO` wires are transformed to a bi-directional communication channel. This allows to read or write 2 bits each clock cycle, doubling the data-throughput.
  - *Quad SPI*: In this case we require two more connection (often the reset and write-protect signal). Here the `MOSI`, `MISO`, and the two extra signals are used as bi-directional communication channel. This allows to read or write 4 bits each clock cycle. This is used for example with SD-cards and the SPI-Flash that is on your GECKO4.
  - *Octal SPI*: I think that you get the idea. It requires six more connections that are not used in “normal” SPI-mode.