INTRODUCTION TO FPGA PROGRAMMING

Lesson 13: External Interfaces

Dr. Davide Cieri[1]

[1]Max-Planck-Institut für Physik, Munich

September 2024

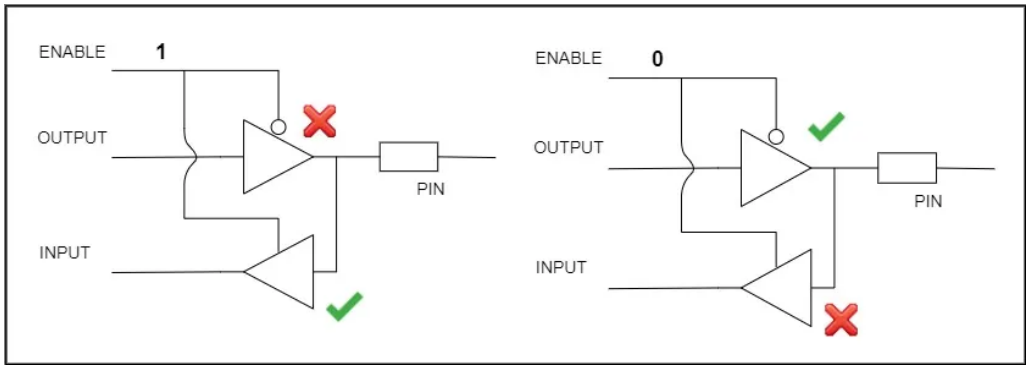
MAX-PLANCK-INSTITUT FÜR PHYSIK

GETTING DATA IN AND OUT

- So far, we mainly discussed about what is inside an FPGA
- Typically your FPGA will have to interface with some peripherals
- It requires the understanding of the electrical interface signals to configure the FPGA pins correctly (Voltage, single/differential ended)
- On the FPGAs typically you have General Purpose Input/Output (GPIO) ports and high-speed Gigabit Transceivers
  - In our projects, we already used GPIO pins to connect to buttons and LEDs
  - GPIO pins can be configured using physical constrains

GPIO PINS

- GPIO pins can be configured as input/output or bidirectional
- Behaviour is driven by port declaration in VHDL entity

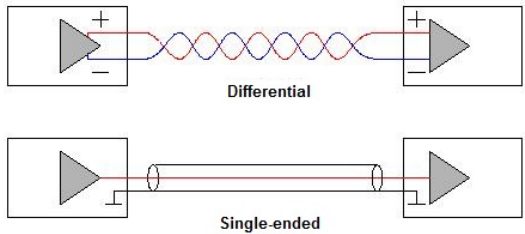


ELECTRICAL CHARACTERISTICS OF GPIOS

- Many electrical characteristics of GPIO pins can be configured with physical constrains
- *Operating Voltage or IOSTANDARD*: this tells the FPGA how to represent a particular voltage on a pin
  - For example `LVCMOS33` means that the 0 and 3.3 V voltages on the pin represent the 0 a 1 logic inputs.
- *Drive Strength or DRIVE*: Tells the FPGA what is the maximum current can be driven to the pin in mA. Typically can be kept as default.
- *Slew rate or SLEW*: The rate of transition (0-1) on the pin. Unless you expect a very fast rate, you can keep default also here.

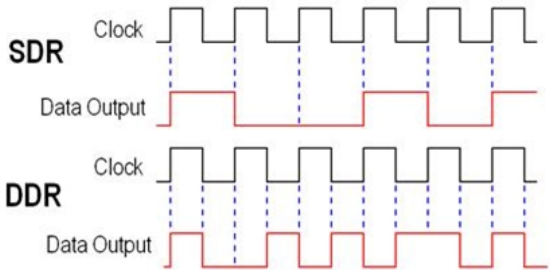
SINGLE-ENDED VS. DIFFERENTIAL SIGNALING

- To correctly decode the 0-1 logic values, FPGAs need to know what is the *ground* state.
- Signals can be sent from one device to the other in two ways
  - *Single-Ended signaling*: One wire is dedicated to the transmission of the ground path. Data is sent as a voltage on the data wire. Same ground can be used for multiple data paths.
  - *Differential signaling*: No ground reference. Voltage is the difference between the two terminals.
- Single-ended signaling requires less pins for the same amount of data
- Differential signaling more immune to noise and doesn't require a common ground.



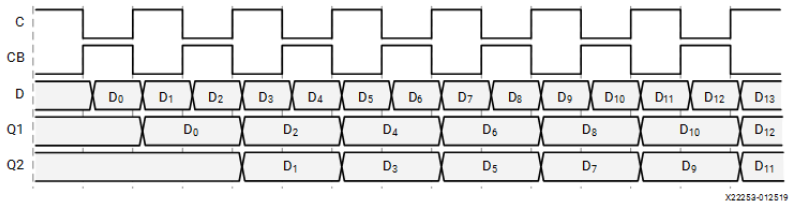
DOUBLE-DATA RATE

- So far, we said that signals inside an FPGA must be synchronised to the rising edge of a clock
- It is possible to configure pins to be sensitive to both rising and falling edge of the clock (*double data rate* or DDR)



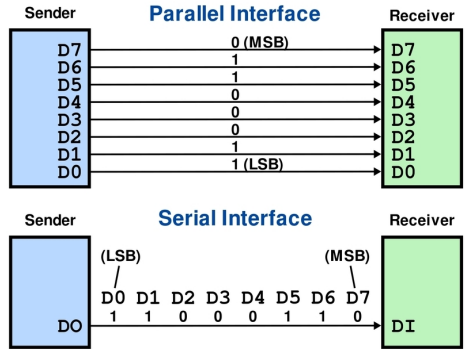
DOUBLE-DATA RATE

- So far, we said that signals inside an FPGA must be synchronised to the rising edge of a clock
- It is possible to configure pins to be sensitive to both rising and falling edge of the clock (*double data rate* or DDR)
- You need to instantiate an IDDR or ODDR buffer IP to use DDR data in your FPGA



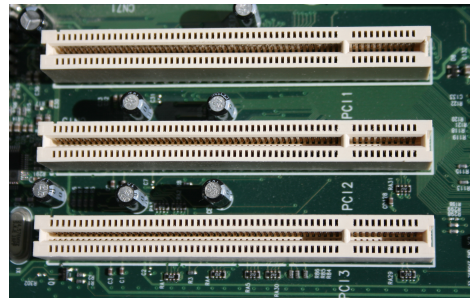
PARALLEL VS SERIAL COMMUNICATION

- There are two possibilities to send data
  - Serial: Data is sent on a single channel
  - Parallel: Multiple communication channels are used
- On top of that data can be synchronous or asynchronous to a clock signal common to both devices
- Naively we could think that parallel communication would be the best choice to sent large amounts of data (WRONG!)



LIMITS OF PARALLEL COMMUNICATIONS

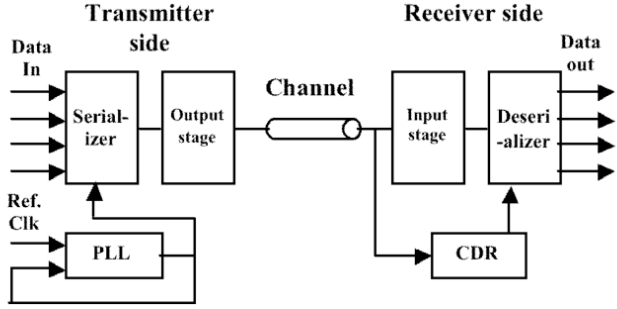
- Sending data parallel requires multiple wire connections
- This will eventually needs actual space on the PCB
  - Example: The 32-bit PCI slot is a few centimeter wide
- All data lines must be synchronous to the same clock
  - High risk of clock skew (same clock arrives at different times to each flip-flop)
- Clock speed becomes faster and faster allowing different approaches



A photo of three 32-bit PCI slots

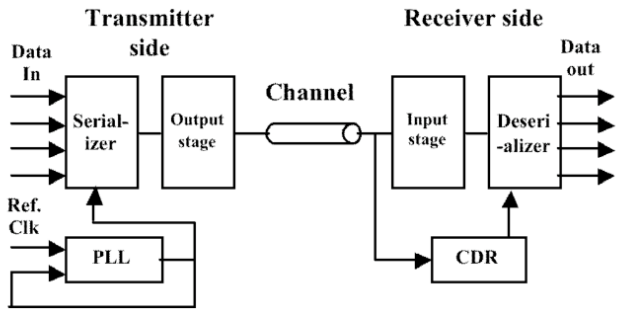
SERDES

- Serializer/Deserializer or Transceivers are powerful components of (most) FPGAs
- They allow us to receive or transmit data at very high speeds (Gb/s)
- As they name suggests, they (de)serialise the data before(after)transmitting(receiving) it to(from) a single line
- They can achieve data rates that are impossible with standard pins



HOW A TRANSCEIVER WORKS

- On the transmitter side, data is serialised and transmitted to the output channel at a chosen clock speed
- On the receiver side, data is analysed by a Clock Data Recovery (CDR) module, that recovers the clock signal, which is finally used by a deserializer module to decode the data

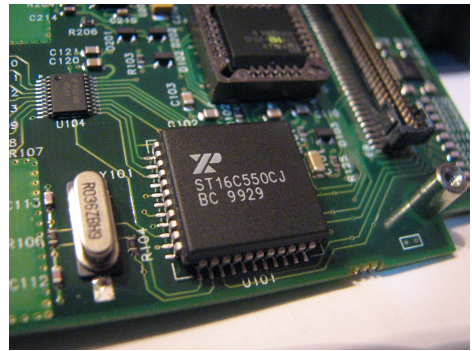


TYPES OF SERIAL COMMUNICATION

- **Synchronous Communication:**
  - Sender and receiver share a common clock signal.
  - Example: SPI (Serial Peripheral Interface), I2C.
- **Asynchronous Communication:**
  - No shared clock; sender and receiver synchronize based on start and stop bits.
  - Example: UART (Universal Asynchronous Receiver-Transmitter).

UART

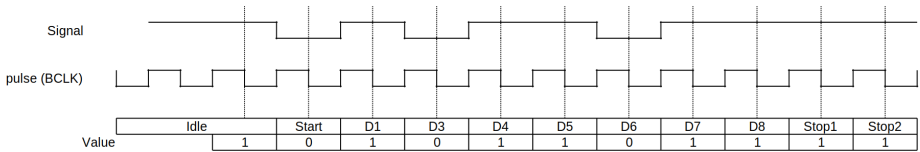
- UART (Universal Asynchronous Receiver-Transmitter) is a hardware communication protocol used for asynchronous serial communication between devices
  - Asynchronous: No shared clock between sender and receiver.
  - Full-duplex communication: Simultaneous transmission and reception of data.
  - Commonly used in embedded systems for simple serial data transfer.



The EXAR 16550 IC, implementing UART interface

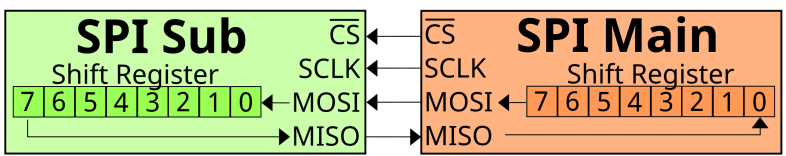
HOW UART WORKS

- **Data Frame Structure:**
  1. **Idle State:** The line is high when idle.
  2. **Start Bit:** Signals the beginning of a data frame.
  3. **Data Bits:** Typically 5 to 9 bits, representing the actual data.
  4. **Parity Bit (optional):** Used for error detection.
  5. **Stop Bit(s):** Indicates the end of the data frame (1 or 2 bits).
- **Baud Rate:**
  - The speed of communication, measured in bits per second (bps).
  - Both transmitter and receiver must agree on the same baud rate.



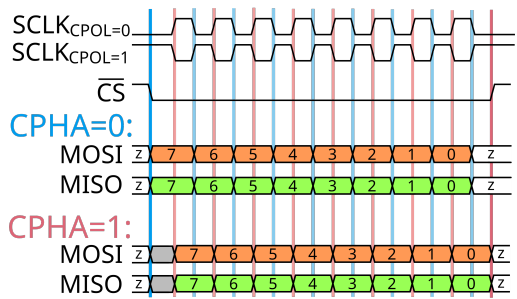
SPI

- Serial Peripheral Interface (SPI) is a standard for synchronous serial communication
  - Widely used for short-distance communication between ICs
- Uses a Main(Master)/Sub(Slave) architecture
  - Main can be connected to multiple slaves
- Four serial lines between Main and Sub(s):
  - Active low chip select signal $\overline{\texttt{CS}}$
  - Clock signal from Main `SCLK`
  - Serial Data from Main to Sub `MOSI` (Main Out, Sub In)
  - Serial Data from Sub to Main `MISO` (Main In, Sub Out)



SPI OPERATATION

1. SPI Main first selects a sub device by pulling its $\overline{CS}$ low
2. SPI Main generates the clock signal (SCLK) to synchronize data transfer.
3. Data is simultaneously transmitted on MISO and MOSI lines.
4. When complete, the main stops toggling the clock signal, and typically deselects the sub.



LINE CODE

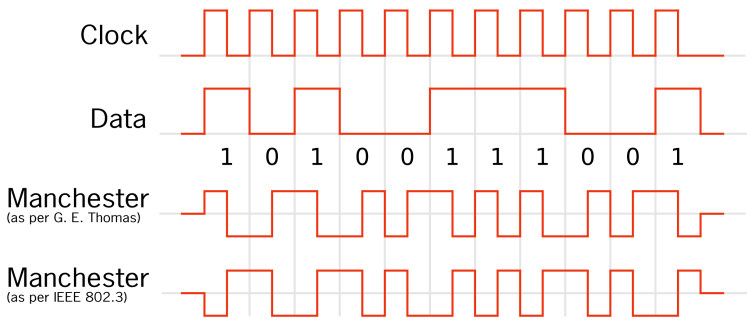
- So far, we saw the simple case where the data is transmitted without being encoded
- This is the most simple line code, called non-return-to-zero (NRZ)
  - Ones are represented by a positive voltage, zero by a negative (or ground).
- It is unsuitable for long communication channels and transceiver interface
- There is no guarantee on the number of 0-1 transitions in the transmission
  - Difficult to extract the clock information from the data
  - If words with many consecutive 1s or 0s are sent, there is a risk to overcharge the coupling capacitor, resulting in a bit error (Direct Current or DC bias)

TYPES OF LINE CODES

- To avoid DC bias, you can use a DC-balanced line code, that eliminates the bias.
- Three types:
  - *Constant-weight code*: each transmitted word that contains positive or negative levels, is designed to contain enough of the opposite levels, such that the average level is zero. E.g. Manchester code
  - *Paired disparity code*: Code words that averages to a certain level are paired with words that averages to the opposite level. Receiver decodes in such a way, that both words are translated in the same way. E.g. 8b/10b encoding
  - *Scrambling code*: Data is manipulated before transmitting. Manipulation is reversed at the receiver side by a descrambler module. E.g. 64b/66b encoding.

MANCHESTER CODE

- A logical ’1’ is represented by a transition from low to high (0 to 1) in the middle of the bit period.
- A logical ’0’ is represented by a transition from high to low (1 to 0) in the middle of the bit period.
- The existence of guaranteed transitions allows the signal to be self-clocking



LAB 20: DESIGN AN UART TRANSMITTER

LAB 21: CONNECTING TO A VGA DISPLAY

The figures in these slides are taken from:

- Digital Design: Principles and Practices, Fourth Edition, John F. Wakerly, ISBN 0-13- 186389-4.

- allaboutfpga.com
- nandland.com
- docs.amd.com
- https://www.symmetryelectronics.com/
- https://www.edn.com/
- Stephen A. Edwards, Columbia University, Fundamentals of Computer Systems, Spring 2012
- adafruit.com
- www.icdesigntips.com
- techdocs.altium.com
- anysilicon.com
- Yngve Hafting 2021, University of Oslo
- www.myomron.com