EEE3096S: Embedded Systems II

LECTURE 11:

EMBEDDED COMMUNICATIONS (PART 2)

Presented by:

Dr Yaaseen Martin

Electrical Engineering
University of Cape Town

ES COMMUNICATIONS TRAINING PLAN

- Intro: Serial vs Parallel, Synch vs Asynch
- I2C & SPI
- UART
- RS232 & RS485
- Timers (briefly)

THE NEXT DAY...

SPI:
Serial Peripheral Interface

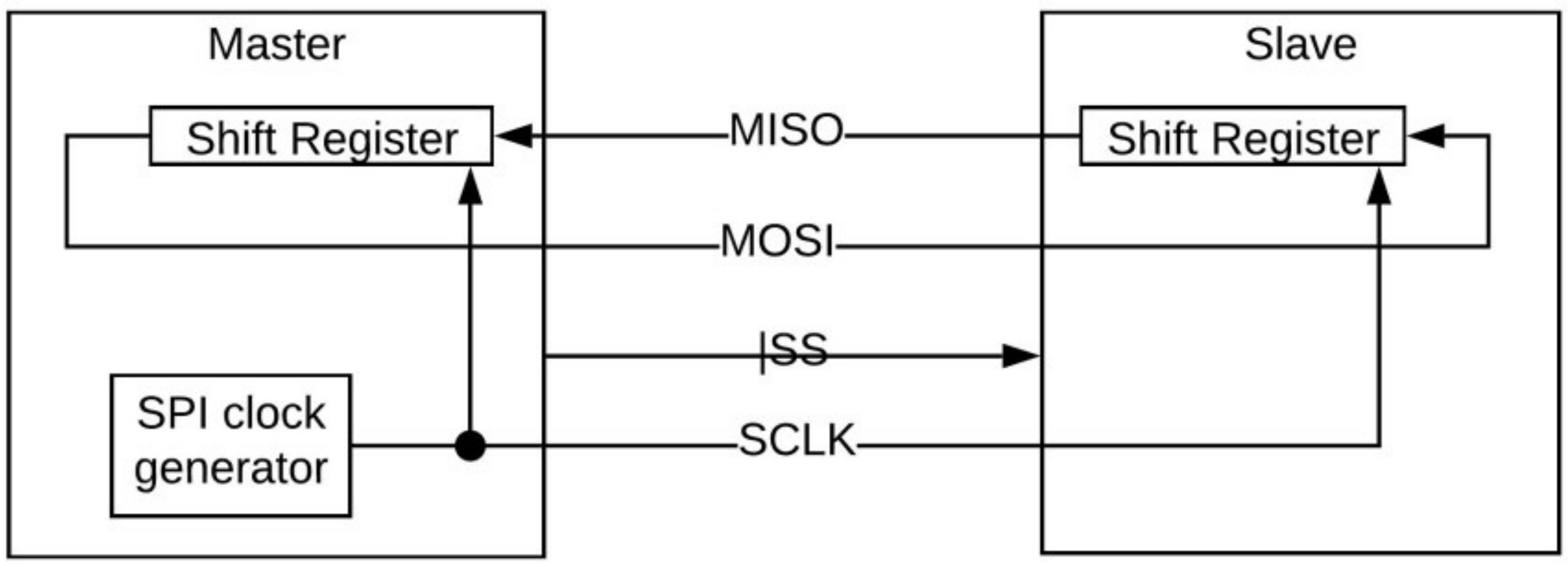
SPI OVERVIEW

SPI = Serial Peripheral Interface

- Communication protocol developed by Motorola (Freescale) in 1979
- Full duplex (both ends can transmit simultaneously)
- Serial (bits sent sequentially),
- Synchronous (uses a shared clock)
- Faster than I2C, but slightly more complex and requires more lines.
- No Start/Stop commands, or addressing; instead, a dedicated slave select line is used
- Supports multiple slave devices on the same bus with a single master
- Also used on PCBs and between nearby chips (only works over short distances).
  e.g. ADC, Temperature sensor, RTC, SD card, EEPROM

SPI OVERVIEW

SPI Interface Lines



Lines required:

- SCLK: Clock generated by master
- MOSI: Master Out Slave In
- MISO: Master In Slave Out
- !SS: Slave Select, active low (hence !)
  Note: SS also called CS = Chip Select

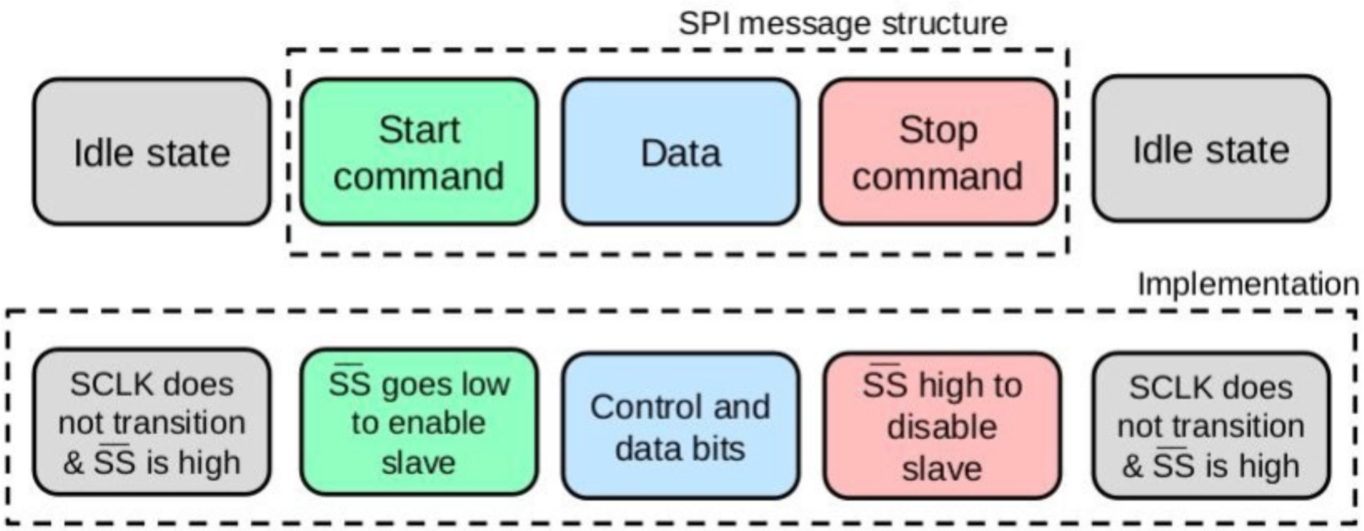
Can use | or ! to indicate SS active low

<u>Operation:</u>
Master-generated CLK moves data onto the data transfer lines (MOSI, MISO) and from/into the end point shift registers simultaneously.

SPI PROTOCOL

State of operation

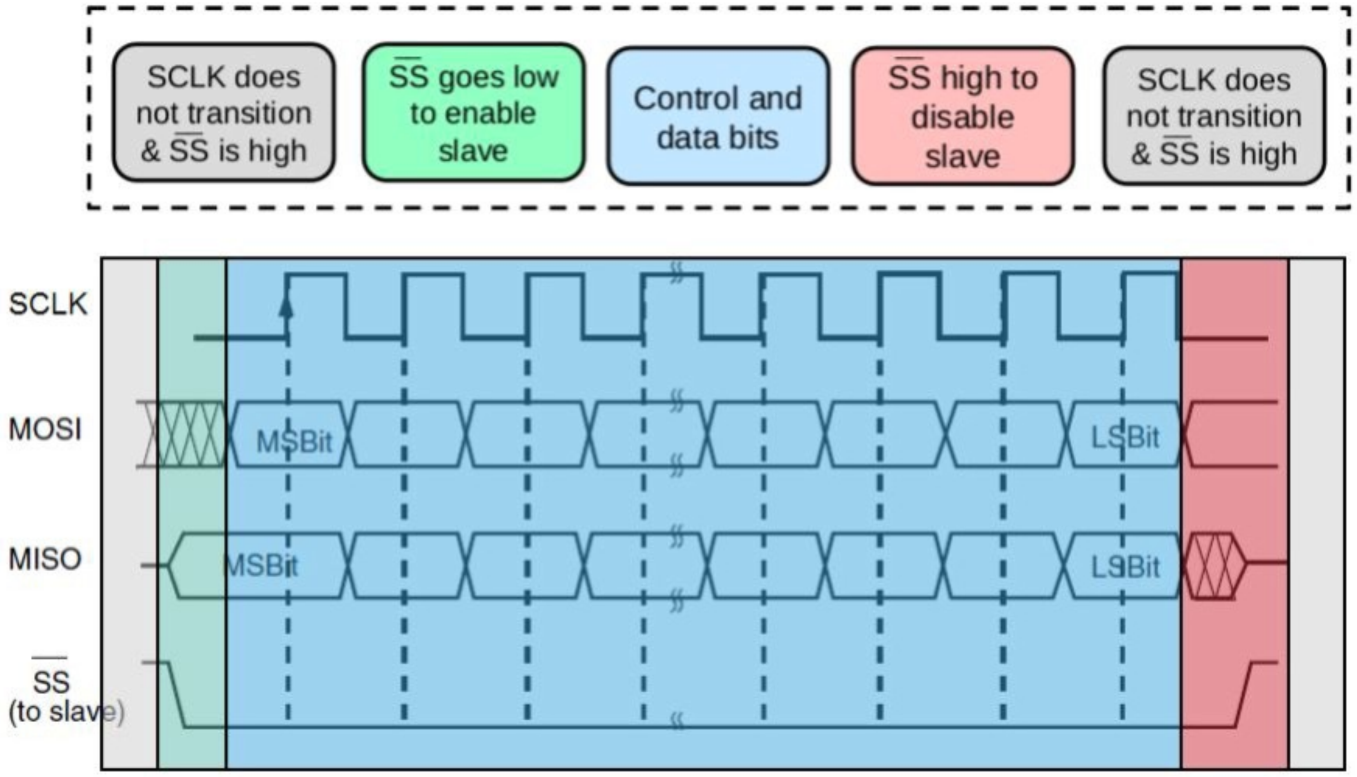
What happens on signal lines


SPI message structure
Idle state
Start command
Data
Stop command
Idle state
Implementation
SCLK does not transition & $\overline{SS}$ is high
$\overline{SS}$ goes low to enable slave
Control and data bits
$\overline{SS}$ high to disable slave
SCLK does not transition & $\overline{SS}$ is high

Three pretty simple parts…

1. Idle state: SCLK not switching, !SS HIGH
2. !SS pulled LOW: Starts data transfer from the 1st clk edge (rising or falling depending on mode)
3. !SS going HIGH: Disables any slaves and stops data movement

SPI PROTOCOL


SCLK does not transition & $\overline{SS}$ is high
$\overline{SS}$ goes low to enable slave
Control and data bits
$\overline{SS}$ high to disable slave
SCLK does not transition & $\overline{SS}$ is high
SCLK
MOSI
MSBit
LSBit
MISO
MSBit
LSBit
$\overline{SS}$ (to slave)

Three pretty simple parts…

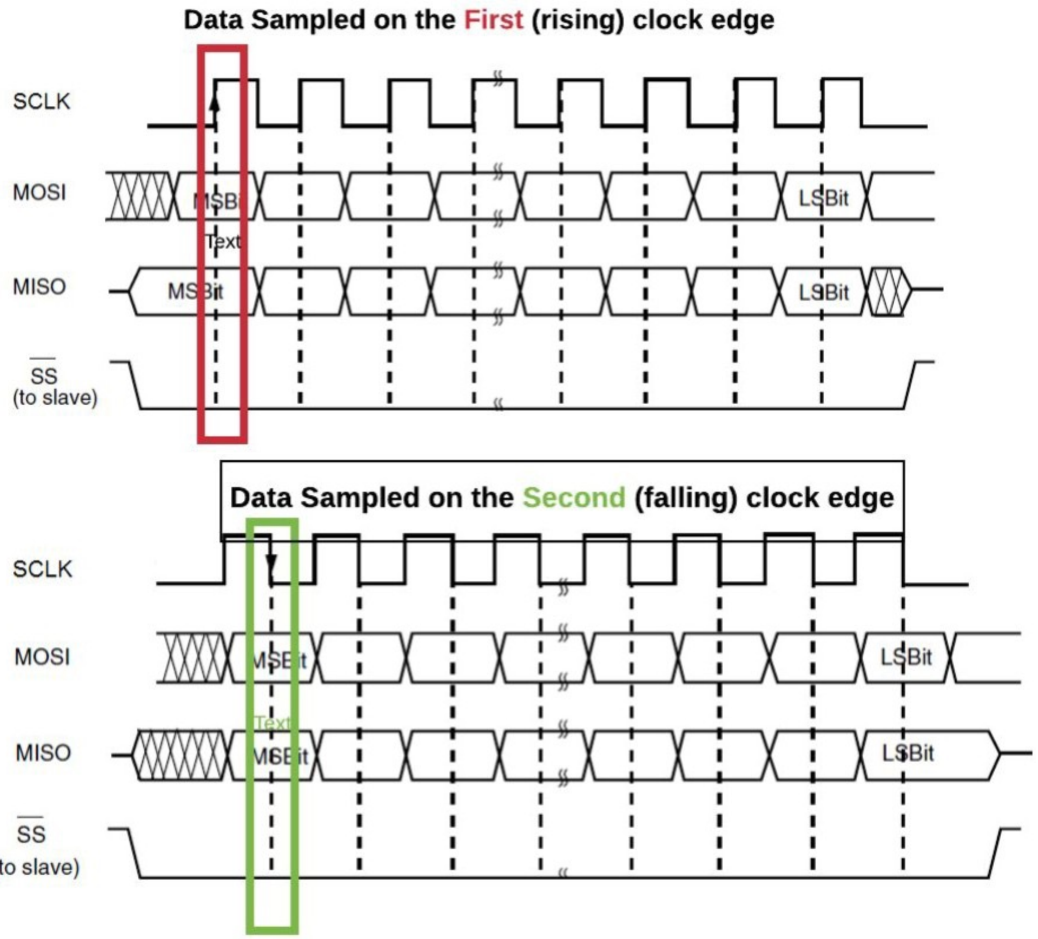
1. Idle state: SCLK not switching, !SS HIGH
2. !SS pulled LOW: Starts data transfer from the 1st clk edge (rising or falling depending on mode)
3. !SS going HIGH: Disables any slaves and stops data movement

SPI PROTOCOL: CLOCK PHASE (CPHA) SETTINGS

SPI Settings …

Clock Phase (CPHA) = 0
→
Data sampled on the first clock edge (rising, in this case)

Clock Phase (CPHA) = 1
→
Data sampled on the second clock edge (falling, in this case)

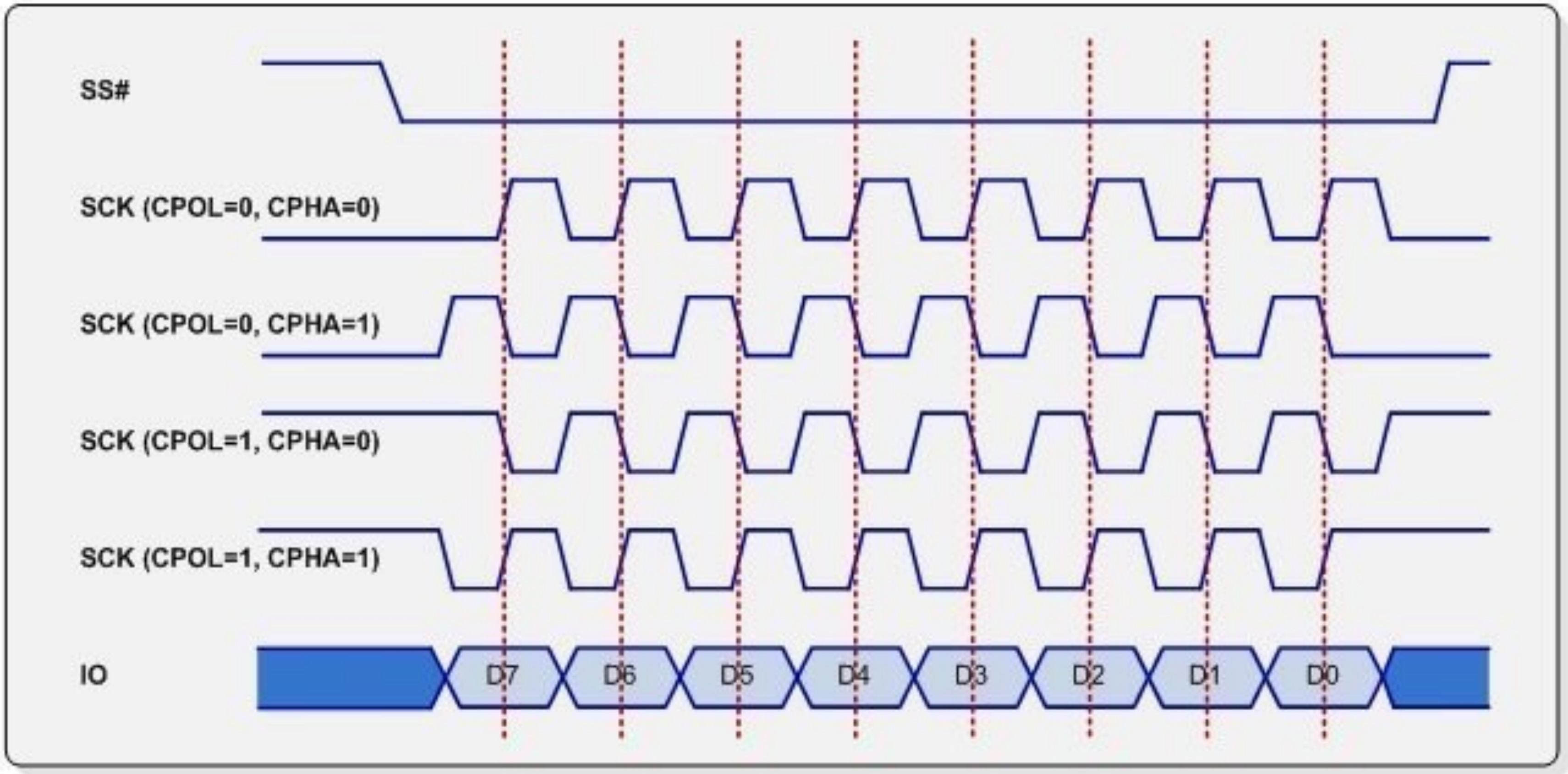


SPI MODES

- These “SPI Modes” are used for some interface module settings and datasheets:

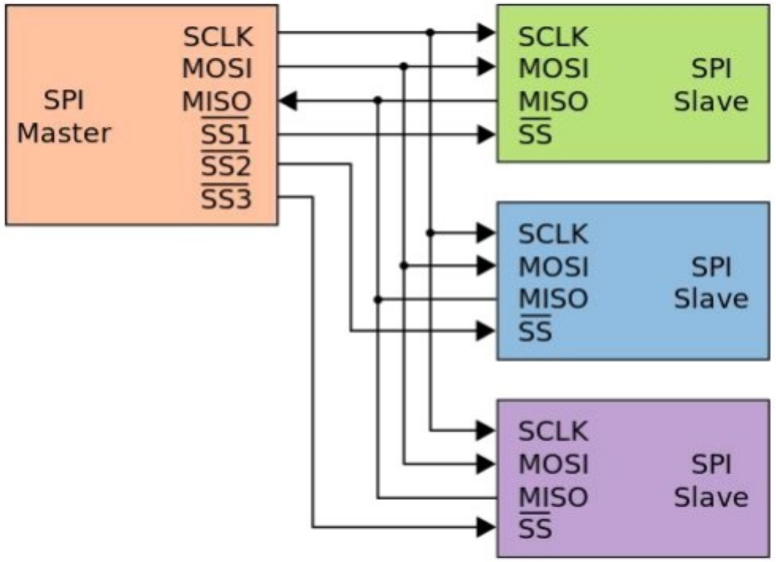
SPI Mode	Clock Polarity [CPOL] Clk level when idle	Clock Phase [CPHA] Sample on first or second edge
0	0	0
1	0	1
2	1	0
3	1	1

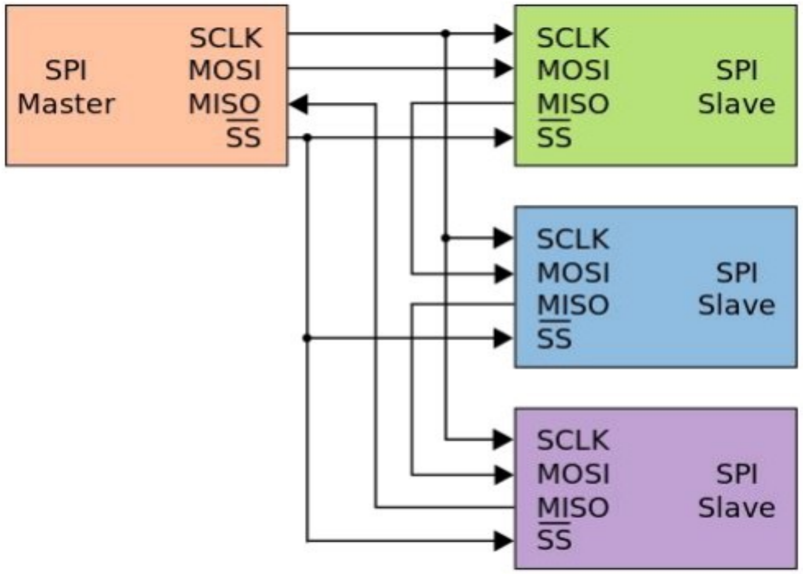
SPI MODES


SS#
SCK (CPOL=0, CPHA=0)
SCK (CPOL=0, CPHA=1)
SCK (CPOL=1, CPHA=0)
SCK (CPOL=1, CPHA=1)
IO
D7
D6
D5
D4
D3
D2
D1
D0

SPI: MULTIPLE SLAVES

Approaches to set up SPI connections for **one** Master and **multiple** Slave devices:





<u>A typical SPI bus configuration with dedicated slave select lines:</u> e.g. Master and three independent slaves. In this configuration each new slaves requires an additional dedicated !SS line for individual control

<u>Daisy-chained SPI bus:</u> master and cooperative slaves. Each slave forwards its MOSI input to its MISO output on the next SCLK cycle
(can allow for more devices connected up with shorter cables, but results in **higher latency**)

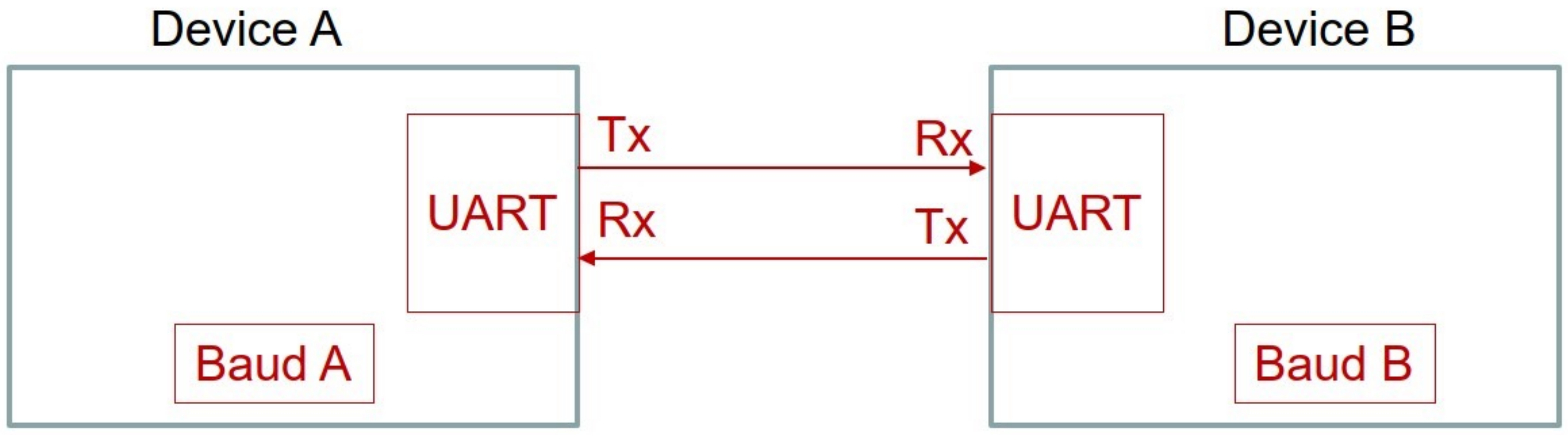
SPI IN CONTEXT

- **Key Disadvantages**:
  - Only 1 master possible
  - If independent access to many slaves is required, the number of !SS lines quickly grows
- **Key Advantages**:
  - No overhead of addressing and start/stop/ACK
  - Full duplex

UARTː
Universal Asynchronous Receiver Transmitter

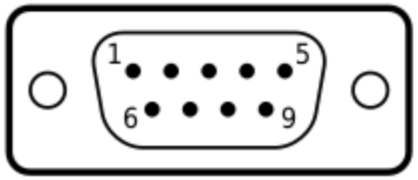
WHAT IS MEANT BY UART?

- The Universal Asynchronous Receiver Transmitter (UART) refers to a generic hardware interface Tx/Rx control module, **not a protocol**! A UART could be an underlying driver that implements a protocol.
- **Baud rate**: rate at which information is transferred in a communication channel; typically a multiple of 150 bps
- Both devices must agree upon the Baud rate since there is **no clock signal** to indicate such transitions



UART & RS-232

- **Recommended Standard 232** (RS-232) is an **asynchronous serial** communication signaling standard
- Defines **the signals** between two devices, particularly their voltage levels, connectors, and pinouts for interoperability between two pieces of hardware
- Can be configured as Full Duplex, Half Duplex, or Simplex
- It formally defines signals connecting between a DTE (data terminal equipment) e.g. computer, and a DCE (data circuit-terminating equipment), e.g. modem
- Mainly used in industrial and networking equipment nowadays



Reference source & further info: https://en.wikipedia.org/wiki/RS-232

UART & RS-232

Common approach is to control RS-232 lines via a UART (commonly built into a microcontroller or an on-board component) and add some "voltage conditioning" circuitry, e.g. MAX232 chip

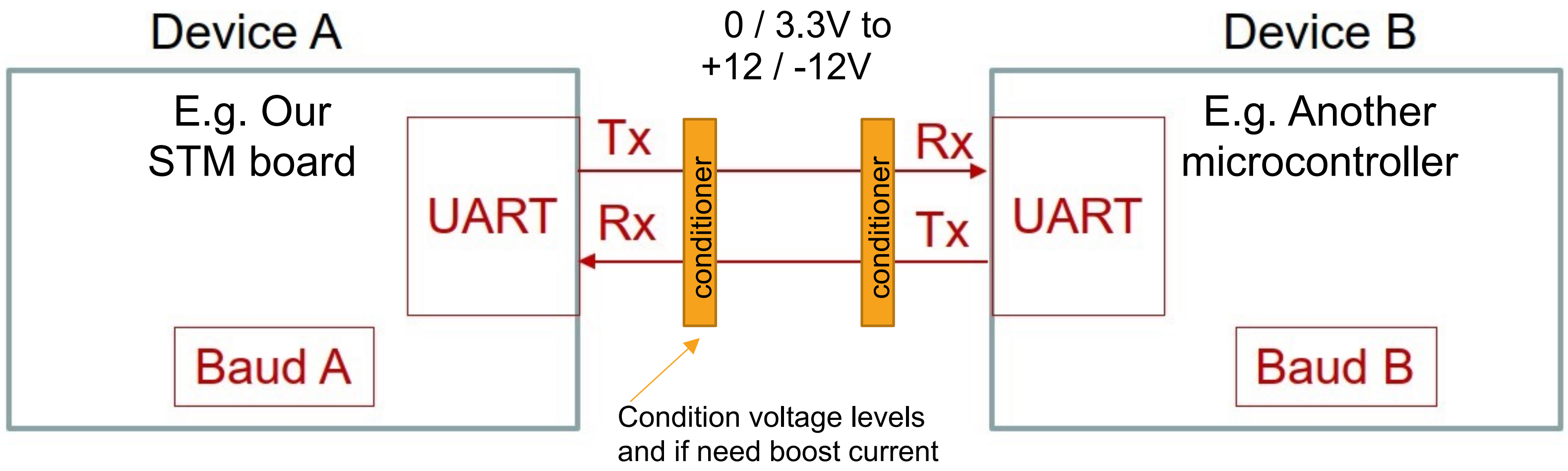
How do UART and RS-232 relate?

RS-232 voltage levels:

- +3V to +15V = logic 0
- -3V to -15V = logic 1

Use a **MAX232** to convert:

- STM 0V to +12V = logic 0
- STM 3.3V to -12V = logic 1

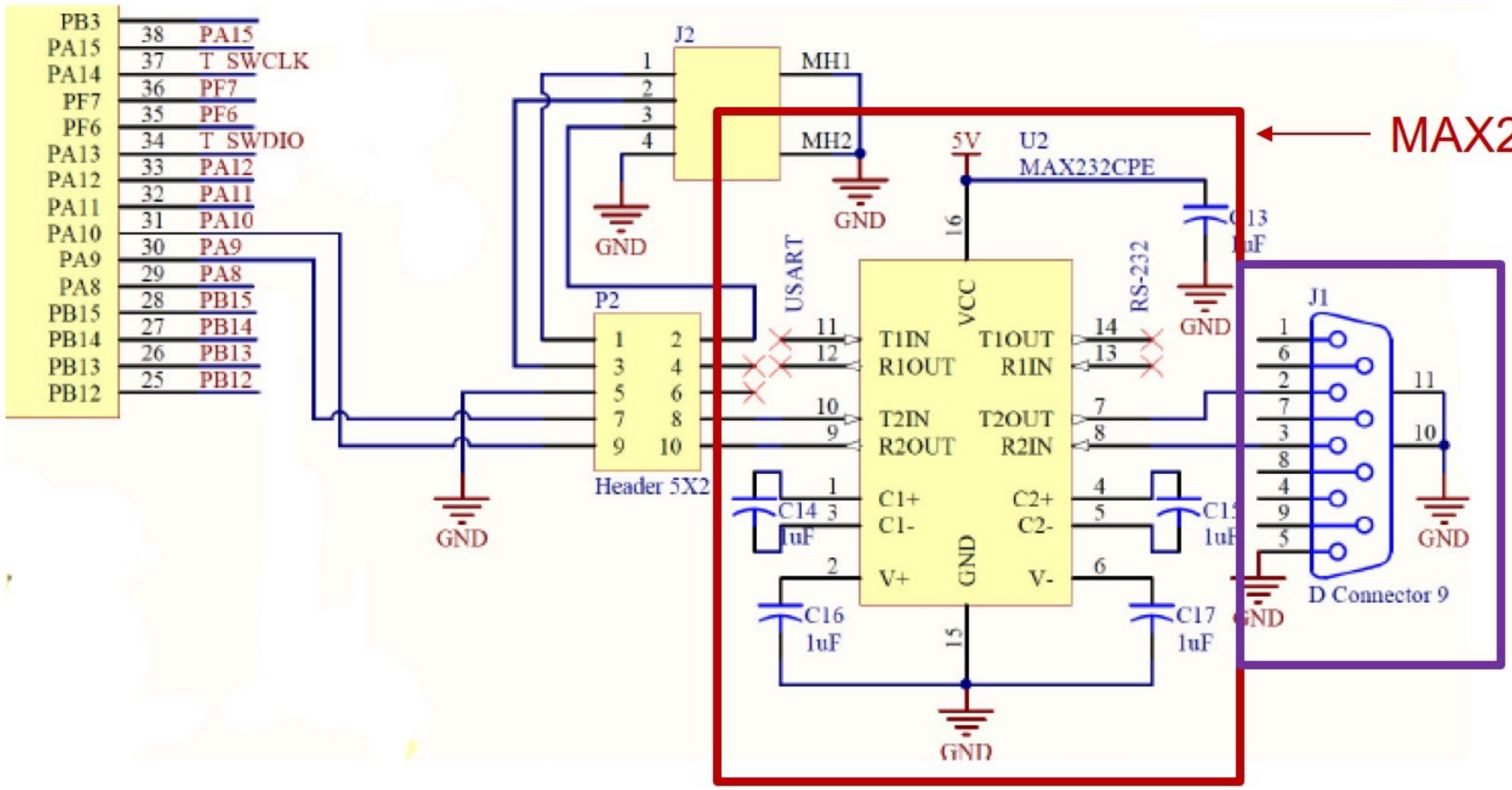


RS-232 STANDARD CONFIGURATION

- **Baud Rate** = 150 x 2^n bits per second (n >=0) e.g.
  - 1.2 kbps, 2.4 kbps, 4.8 kbps, 9.6 kbps, 14.4 kbps, 19.2 kbps, 115.2 kbps
- RS-232 was originally intended more for bit rates lower than 20 kbps, but was kept going and still useable way beyond that
- The max RS-232 speed is around **1 Mbps**
- Max limit for UART? No explicit limit, left to developer to decide

RS-232 ON STM32 UART → MAX232 → DB9

Illustration of connections needed to provide a RS-232 port on the STM32F0:



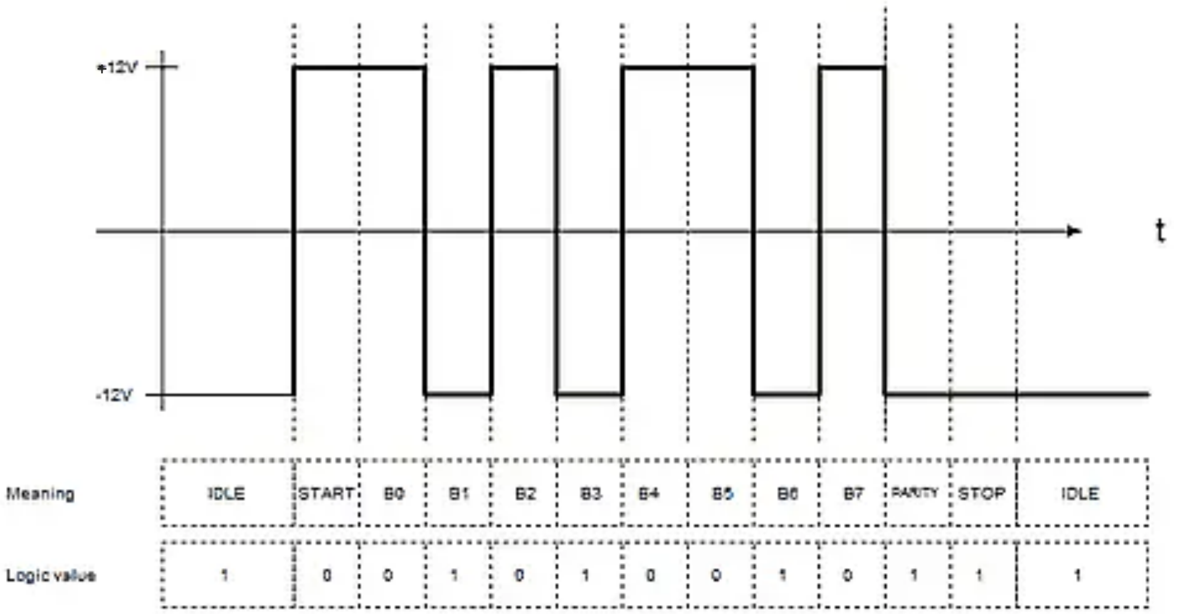
Portion of schematic for STM32F0 development board.
Source: Embedded Systems I course notes

RS-232 TIMING

There are some further requirements and slight complications to using RS-232:

- The bits timing and packaging needs to follow a specific approach
- Timing for start bit, ‘text’ or data bits, parity bit and stop bit:

RS232 Transmission of the letter 'J' - 0x4A



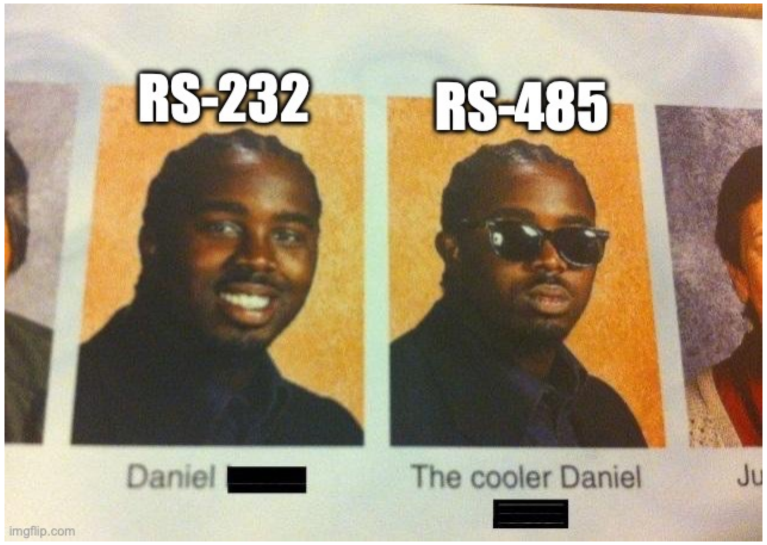
Meaning	IDLE	START	B0	B1	B2	B3	B4	B5	B6	B7	PARITY	STOP	IDLE
Logic value	1	0	0	1	0	1	0	0	1	0	1	1	1

RS-232 LIMITATIONS

- Supports only **one** transmitter and **one** receiver on a communication bus
- Allows data transfer for distances *less than* 15m. Signal lines (Tx and Rx) are referenced w.r.t. ground and performance degrades quickly when there is noise present.
- Allows rather limited speed of data transfer (especially for lengths beyond about a ~1m)

RS-485

- **What is RS-485?**
  - Essentially an **enhanced** version of RS-232
  - Can operate over a maximum of around 1200 m
  - **Twisted cables** used to pick up **less noise**
  - Used in commercial aircraft cabins, building automation, industrial control systems, etc.
  - Greater power consumption than RS-232



PRAC 2 PREP: TIMERS AND SPI

- Practical 2 will focus on Timers, Interrupts, and using SPI to read/write data to/from EEPROM on your STM board
- Quick recap on timers, including ARR and PSC calculations: https://embedded-lab.com/blog/stm32-timers/3/
- Use these slides for your SPI info :)