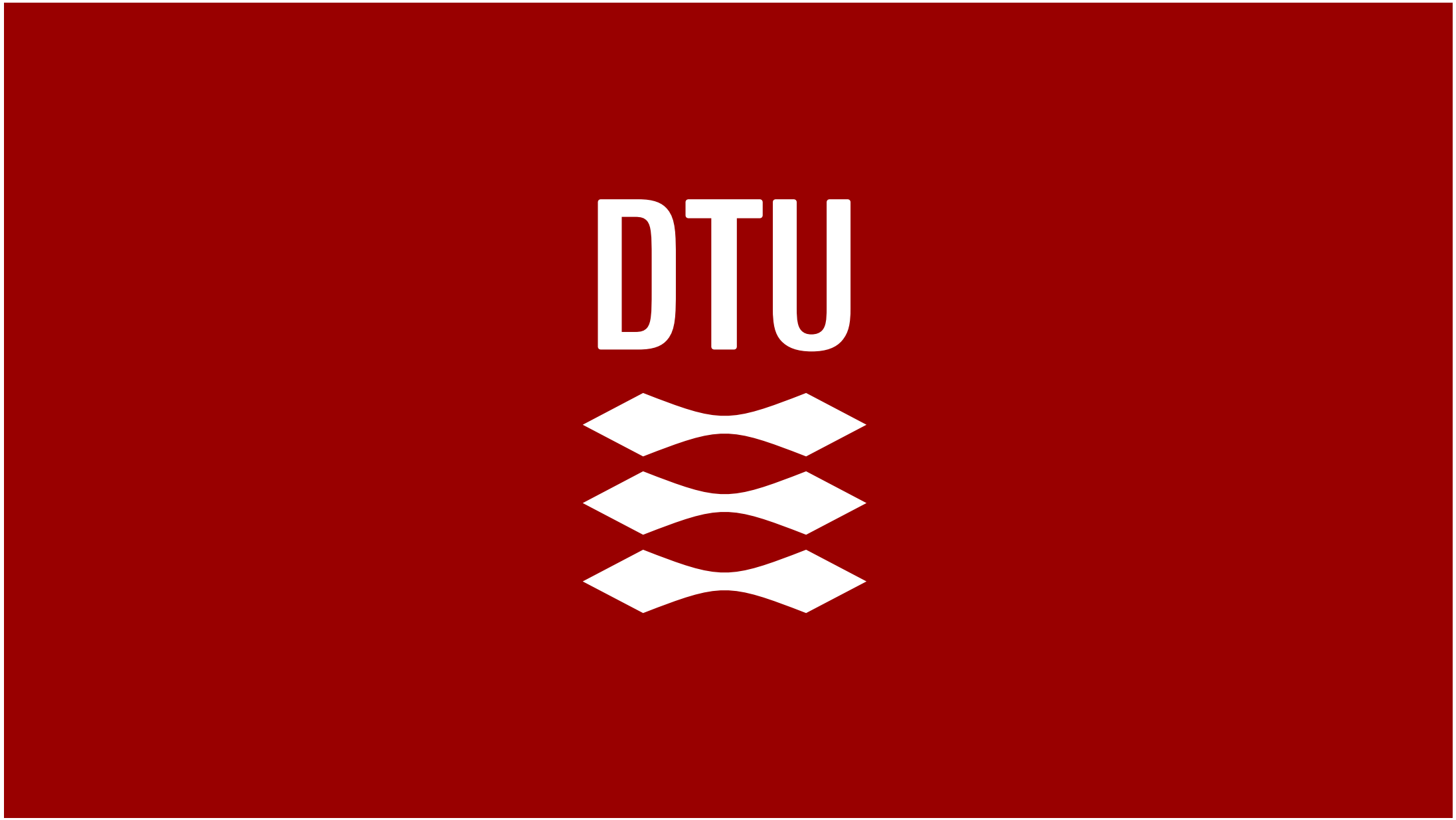

DTU

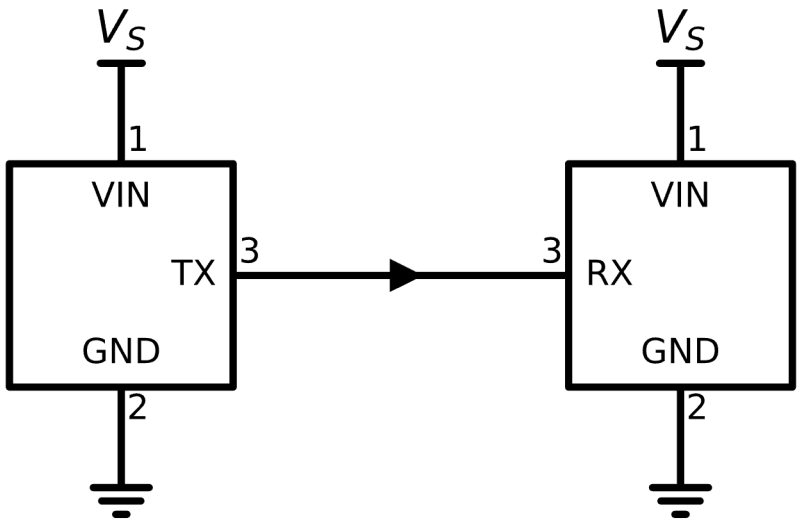
Networked Embedded Systems

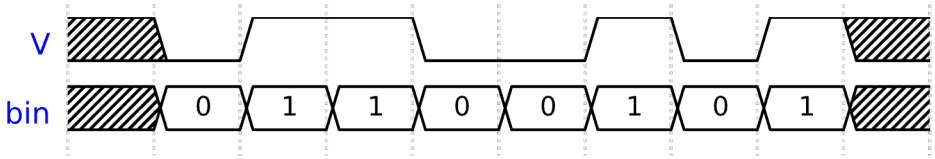
Week 5: Serial Communication

Xenofon (Fontas) Fafoutis
Professor
xefa@dtu.dk
www.compute.dtu.dk/~xefa

Serial Communication

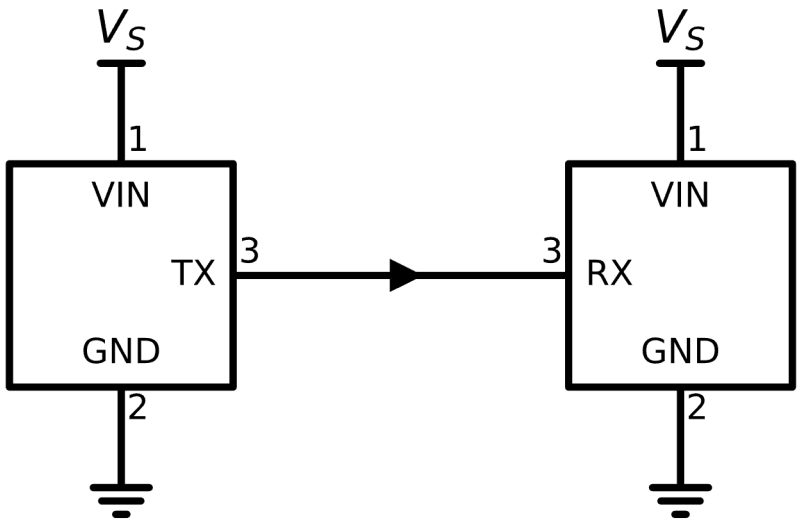
- Bits are transmitted over a wire **in sequence**
  - One after the other
- Bits are encoded in the voltage level of the wire
  - E.g. high voltage is '1', low voltage is '0'

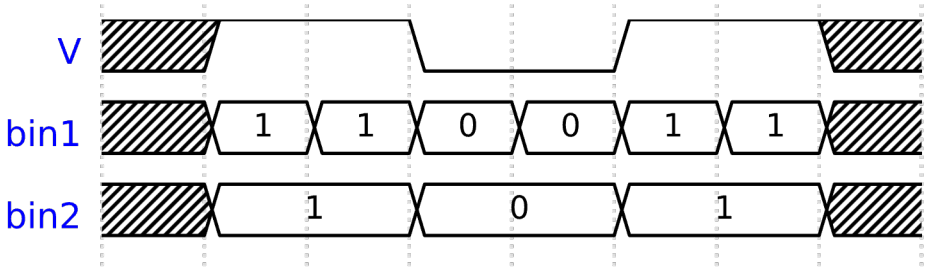




Timing the Receiver

- The receiver can detect changes in the input
- How often shall the receiver read the input?
- What is the correct binary interpretation of the voltage V? bin1 or bin2?

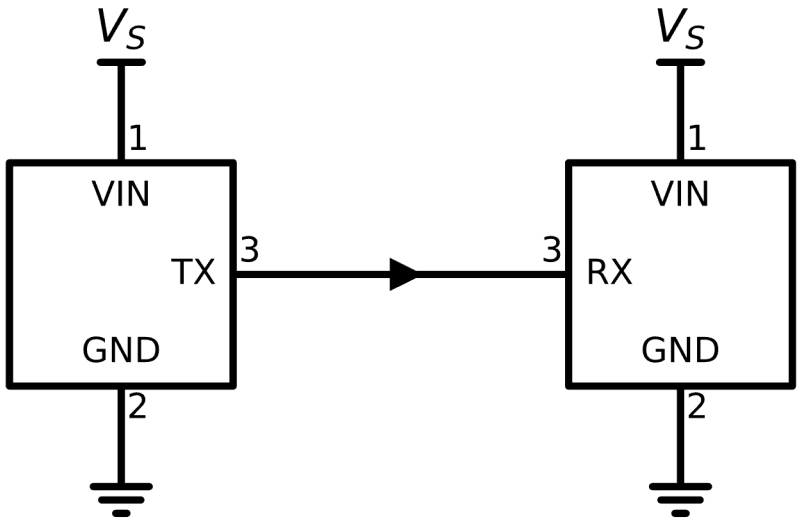


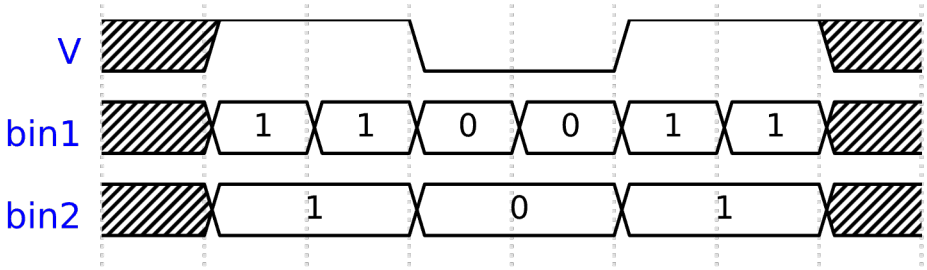


Timing the Receiver

- The receiver can detect changes in the input
- How often shall the receiver read the input?
- What is the correct binary interpretation of the voltage V? bin1 or bin2?

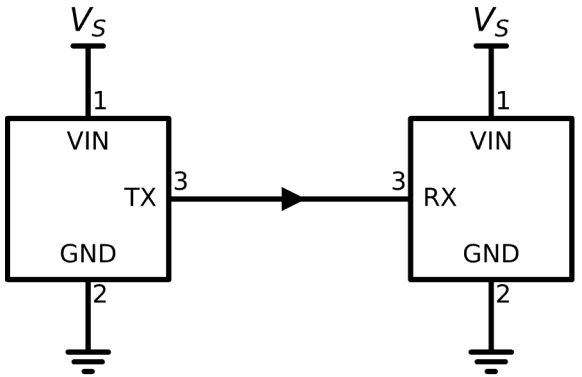
We can’t know for sure! We need a way to synchronise the transmitter and the receiver!

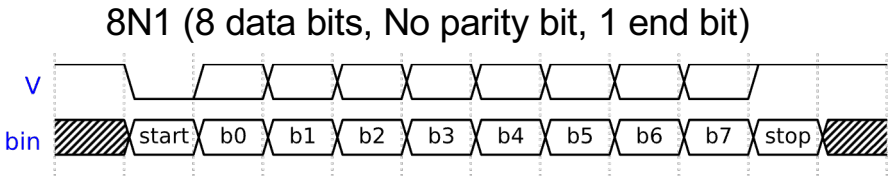


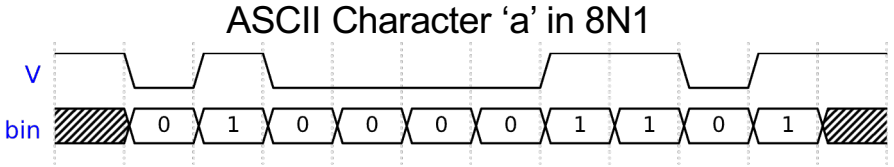


Asynchronous Serial Communication

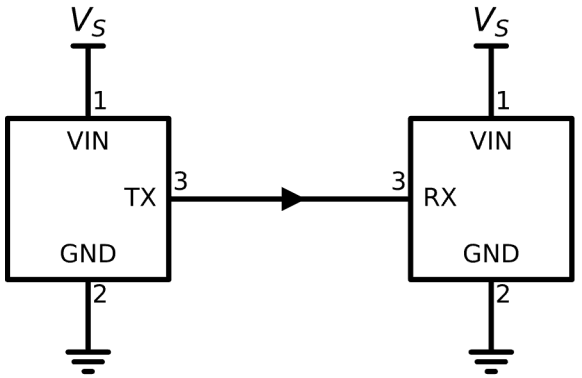
- Transmitter and receiver do not share the same clock
- Synchronisation
  - When idle the line is at a high state, '1'
  - The beginning of each transmission is signalled by setting the line to '0' (start bit)
  - Bits transferred at a **pre-agreed baud rate** (~KHz)
  - The end of the transmission is signalled by setting the line to '1' and holding it (end bit)
  - There is one or two end bits
- Data are transmitted in a **pre-agreed format**
  - Data size (5-9 bits, typically 8 bits)
  - Bit order (typically, least significant bit first)
  - Parity (sum of all bits for error checking, optional)



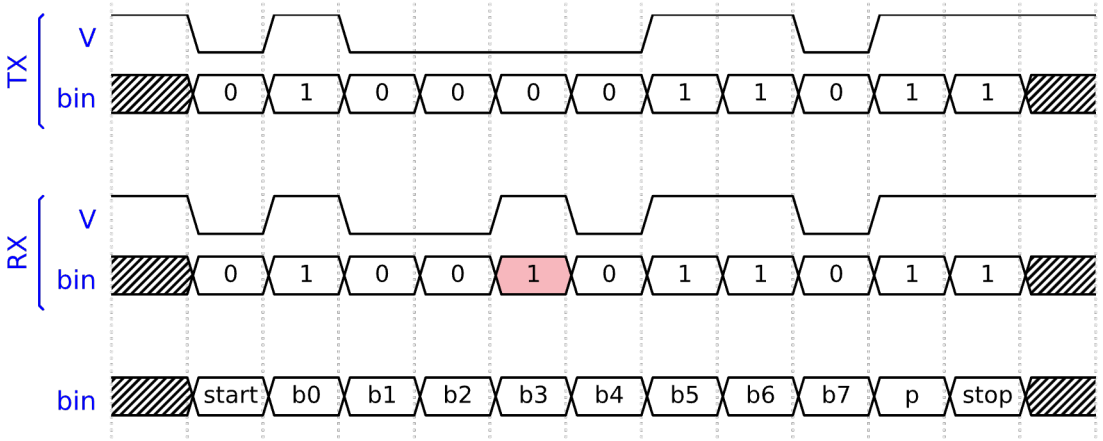




Error Detection: Parity Check



- Even Parity: The parity bit is set so that the total number of '1' transmitted is even
- Odd Parity: The parity bit is set so that the total number of '1' transmitted is odd
- If received total number of '1' is not as expected, an error occurred
  - One error (bit flip) can be detected
- Parity is optional
  - Helps in noisy mediums
  - Slows down data transfers



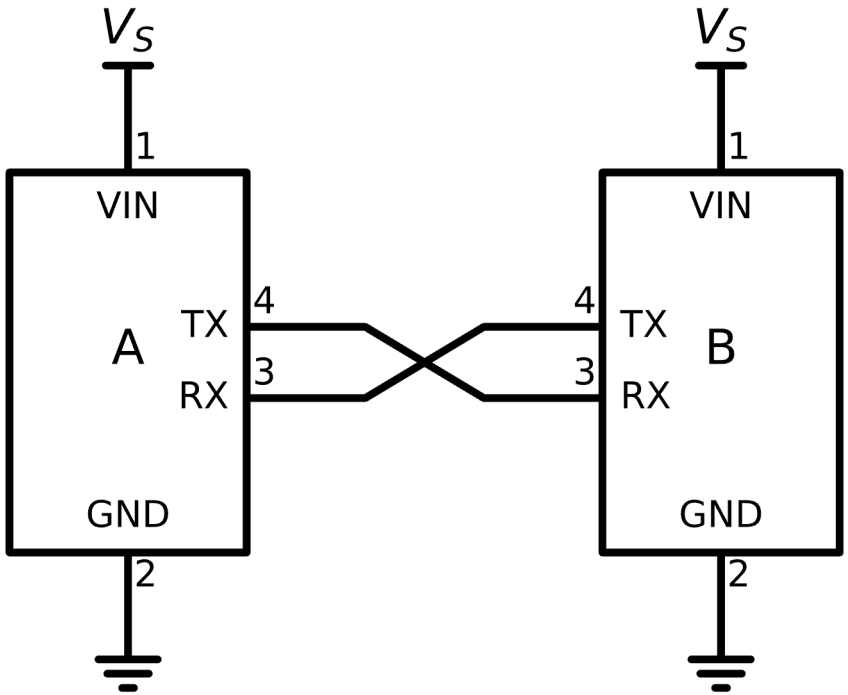
Bidirectional Asynchronous Serial

- For bidirectional communication we need 2 lines
  - The TX of A is connected to the RX of B
  - The RX of A is connected to the TX of B

- **Full-duplex**: both can transmit and receive simultaneously

- **Half-duplex**: devices take turns in transmitting and receiving

- **Simplex**: unidirectional only



Implementation of Asynchronous Serial

- **TTL Serial** (Transistor-Transistor Logic)
  - '1' is encoded as positive voltage, typically 5V or 3.3V
  - '0' is encoded as zero voltage (ground)

- **RS-232 Serial**
  - '1' is encoded as negative voltage, typically -12V
  - '0' is encoded as positive voltage, typically +12V

- **UART** (Universal Asynchronous Receiver-Transmitter)
  - Hardware implementation of asynchronous serial communication
  - Creates the frames and controls the physical lines

- **Software UART** (bit-banging)
  - Software implementation directly controlled by the processor
  - Inefficient, but an option if UART not available in MCU

Example of a UART Peripheral

- **CozIR-A:** A CO2 Sensor
- Supports a 9600 baud rate, 8N1 serial interface
- Provides a ASCII-based command/control interface over serial

CONTROL INTERFACE TIMING - UART MODE

PARAMETER	SYMBOL	MIN	TYP	MAX	UNIT
Baud Rate			9600		Bits/s
Data Bits			8		
Parity			None		
Stop Bits			1		
Hardware Flow Control			None		

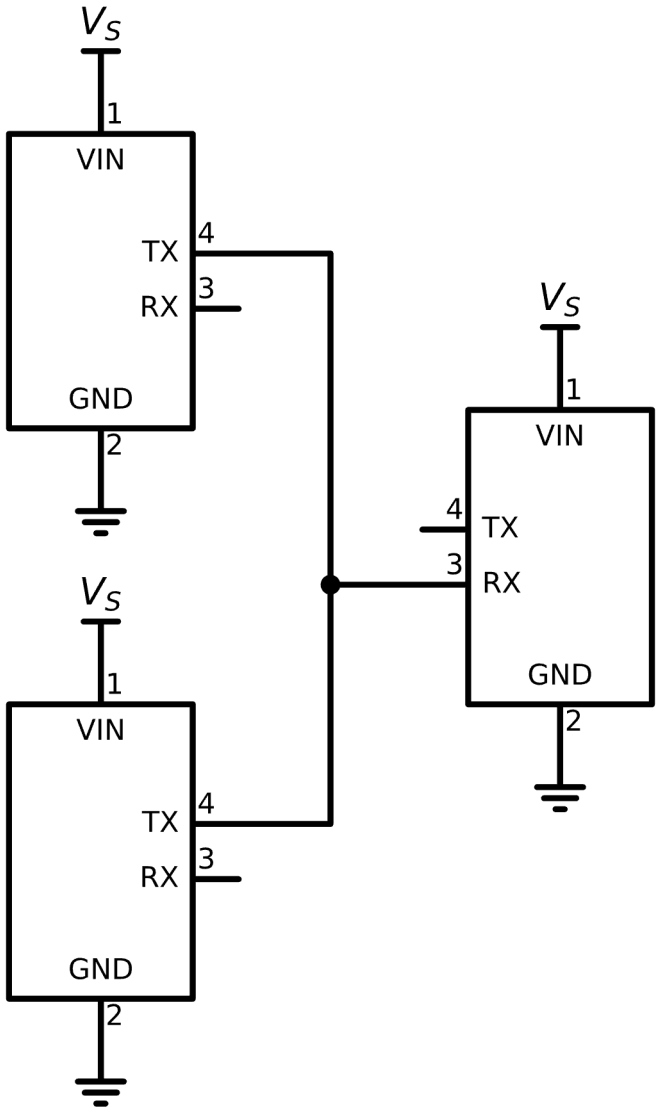
Syntax	Use	Example	Response
Z\r\n	Return the most recent filtered CO_2 2 measurement in ppm	Z\r\n	Z 01521\r\n
z\r\n	Return the most recent unfiltered CO_2 2 measurement in ppm	z\r\n	Z 01521\r\n

PIN-OUT DESCRIPTION: CozIR®-A (Both Types)

PIN	NAME	TYPE	DESCRIPTION
1	GND	Supply	Sensor ground
2	NC	Unused	Do Not Connect
3	VDD	Supply	Sensor supply voltage
4	GND	Supply	Sensor ground
5	Rx_In	Digital Input	UART Receive Input
6	GND	Supply	Sensor ground
7	Tx_Out	Digital Output	UART Transmit Output
8	NITROGEN_ZERO	Digital Input	Set low to initiate a Zero in Nitrogen Calibration Cycle
9	ANALOGUE_OUTPUT	Analogue Output	CO_2 Level (Optional)
10	FRESH_AIR_ZERO	Digital Input	Set low to initiate a Zero in Fresh Air Calibration Cycle

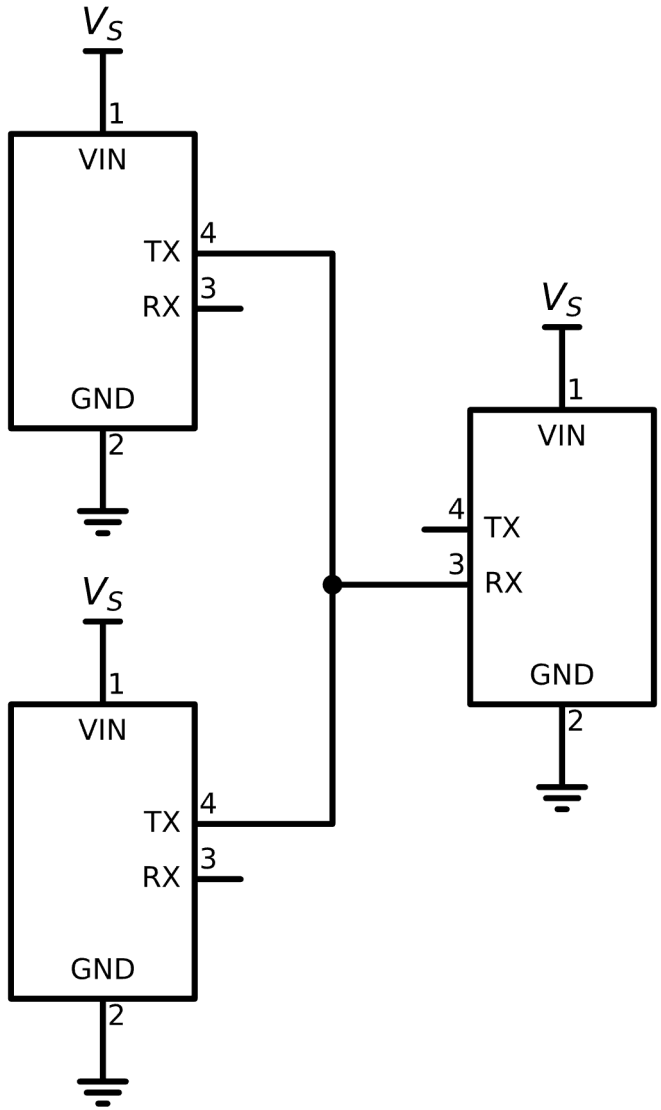
Image source: CoZIR Datasheet by GSS

Can UART be shared?



UART cannot be shared

- Asynchronous Serial (UART) does not support multiple devices
- It is designed for **two devices** to communicate
- Multiple UART transmitters must not share the same line
  - Output is push-pull
  - Shorts and hardware damage are possible
  - Communication will fail

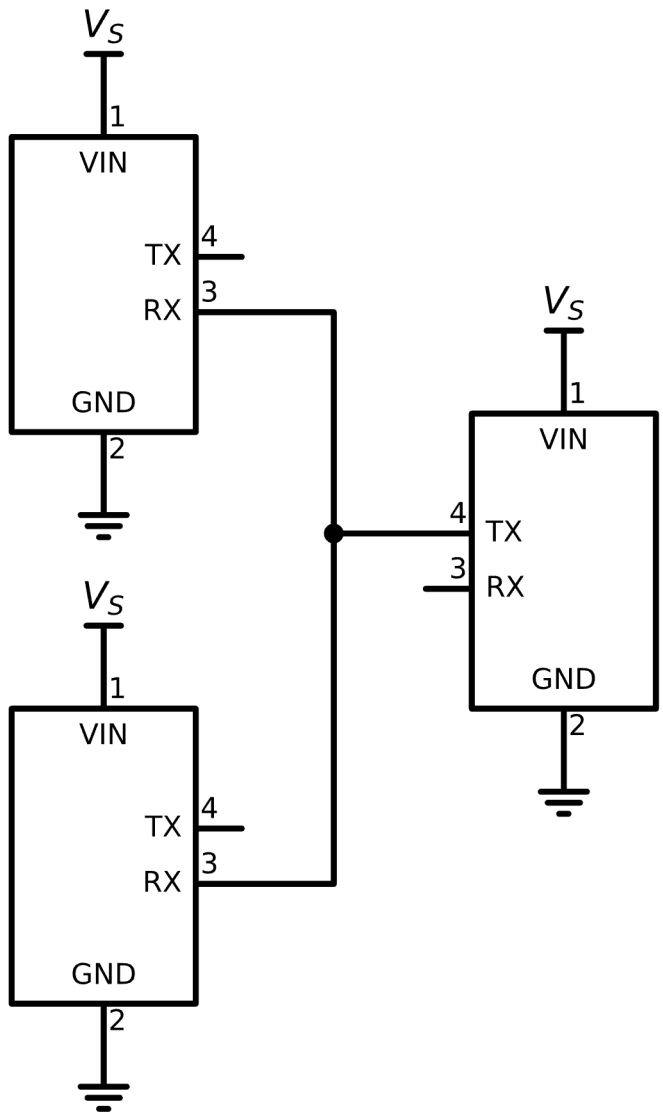


R: Reads

T: Transmit (sends)

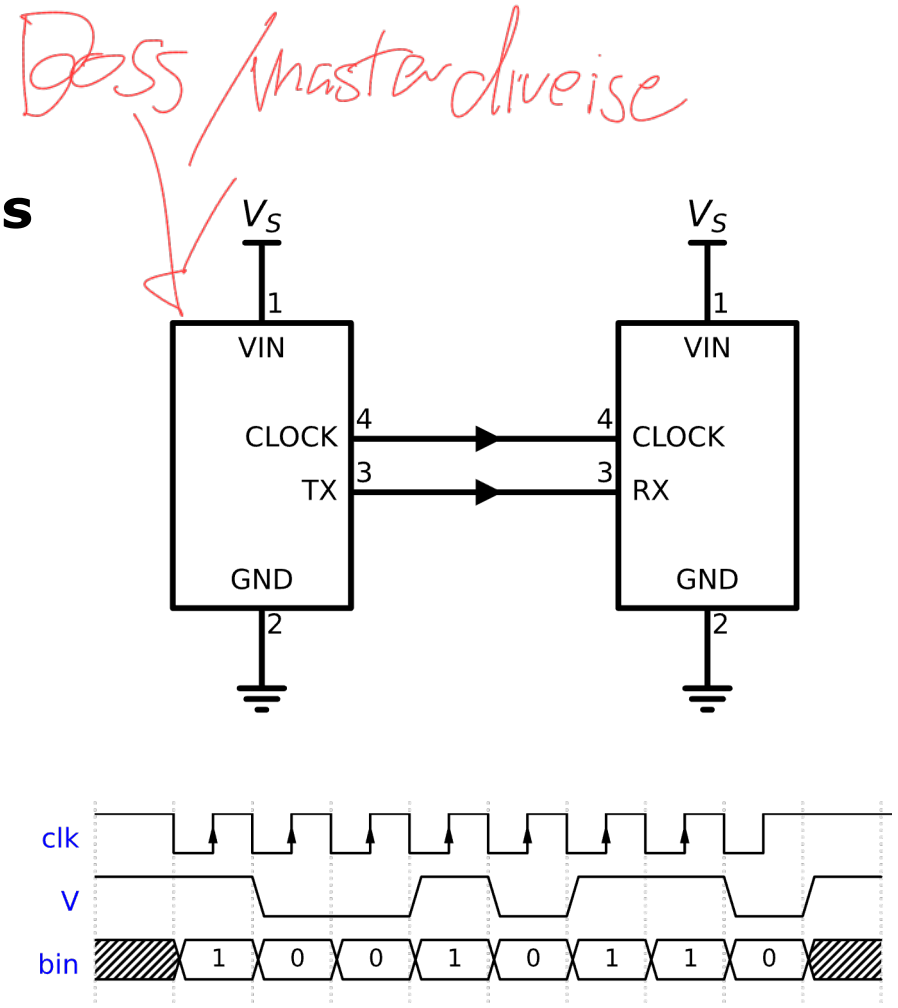
UART cannot be shared

- Asynchronous Serial (UART) does not support multiple devices
- It is designed for **two devices** to communicate
- Multiple UART receivers is safe but…
  - It is impossible for the transmitter to select a receiver
  - All receivers will receive everything
  - Not proper use of the standard
  - It may work on some applications



Synchronous Serial Communications

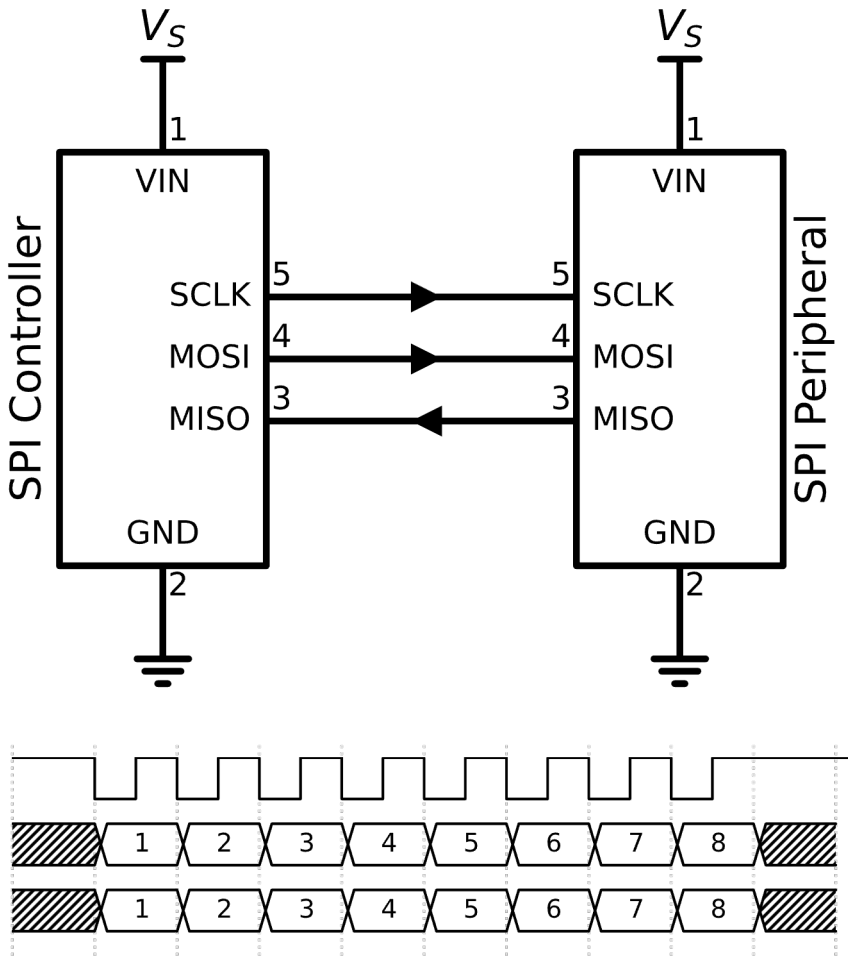
- One device, the controller, generates a clock signal
- All devices use this clock signal to synchronise their transmitters and receivers with the controller
  - Transmissions (i.e. driving the line) occur on the falling edge of the clock (high to low)
  - Receptions (i.e. reading the line) occur on the rising edge of the clock (low to high)
  - Or vice versa
- Advantages
  - No need to pre-agree the baud rate
  - No overhead of synchronisation bits
  - Requires cheaper hardware than UART



Master output Slave Input = MOSI

SPI (Serial Peripheral Interface)

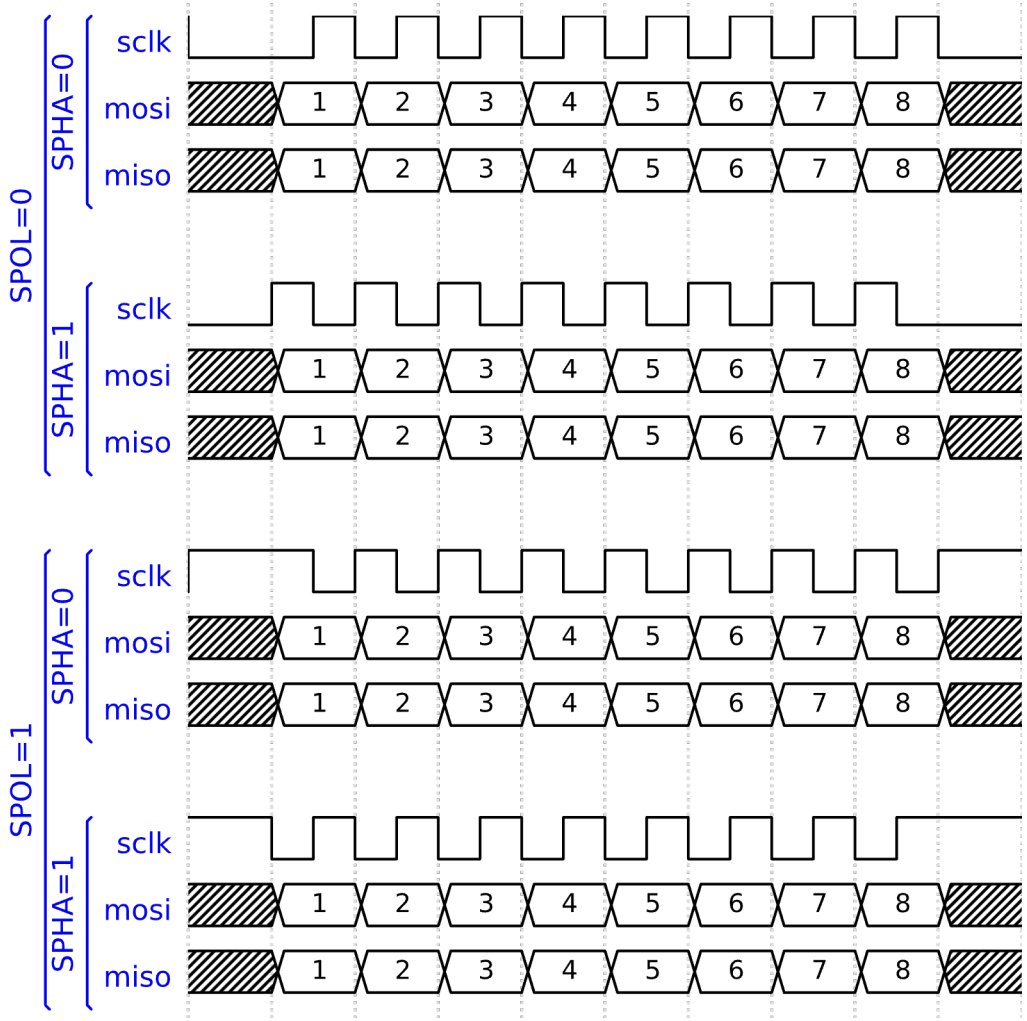
- SCLK (Serial Clock)
  - The SPI controller (or master) drives the clock
  - The controller selects the frequency of the clock dictating the rate of communication (~MHz)
  - Must be supported by SPI peripheral(s)
- MOSI (Master Out, Slave In)
  - Data from SPI controller to SPI peripheral(s)
- MISO (Master In, Slave Out)
  - Data from SPI peripheral(s) to SPI controller
- Full-duplex: On each clock cycle a bit is written/read on MOSI and a bit is written/read on MISO
  - This is maintained even when only one-directional data transfer is intended



SPI Modes

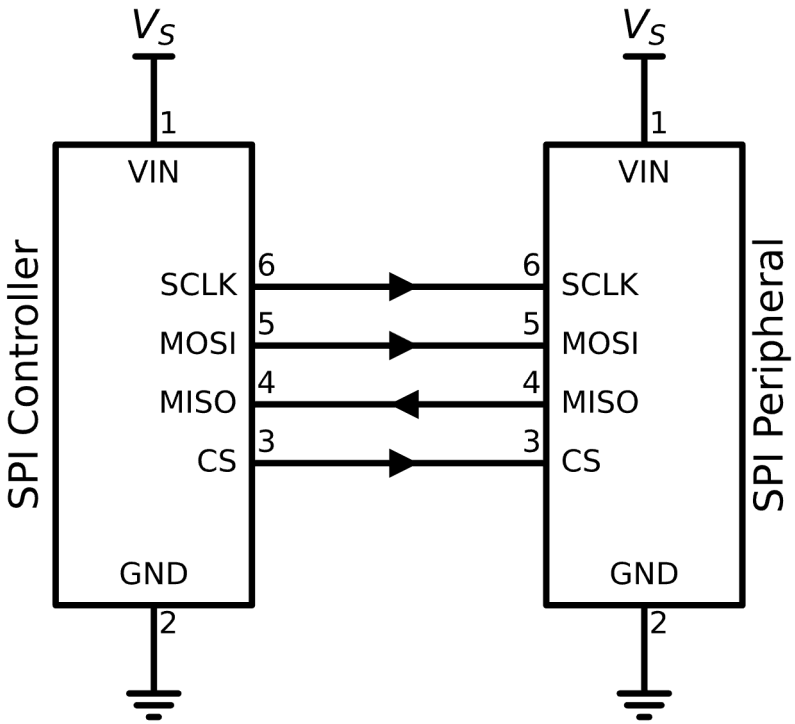
- CPOL=0: clock idles at 0, the leading edge is a rising edge, and the trailing edge is a falling edge
- CPOL=1: clock idles at 1, the leading edge is a falling edge, and the trailing edge is a rising edge
- CPHA=0: the data changes on the trailing edge, the data is read on the leading edge
- CPHA=1: the data changes on the leading edge, the data is read on the trailing edge

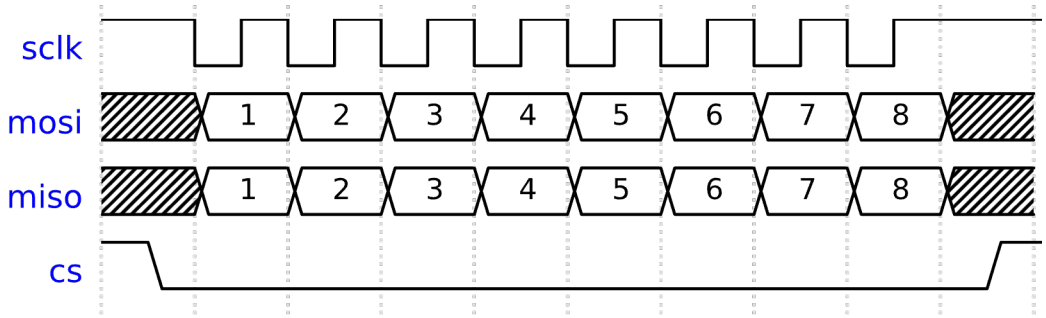
SPI Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1



SPI: Chip Select

- Chip Select (CS): The controller uses it to select the peripheral to communicate with
- When the peripheral detects a CS transition from high to low, it activates itself and gets ready to transmit/receive
- When the peripheral detects a CS transition from low to high, it puts MISO in Hi-Z (deactivates the output)
- When deactivated, the peripheral ignores the SCLK and MOSI
- Recommended to have a pull-up resistor on CS for MCUs that boot with their GPIOs in Hi-Z

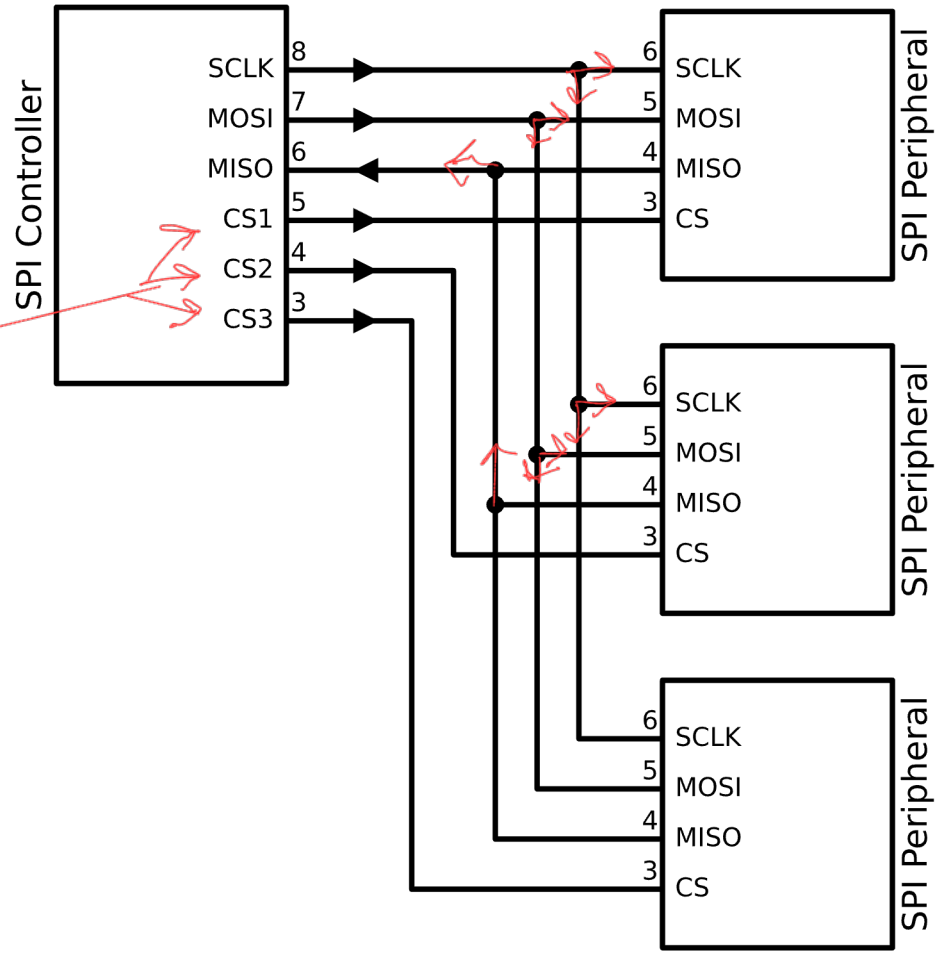




All clocks are connected!

SPI: Multiple Peripherals

- Multiple SPI peripherals are supported
- All are controlled by the SPI controller by selecting them via the respective CS output
- The SPI controller must make sure to have only one SPI peripheral enabled at a time
- Data transfers are only possible between the SPI controller and an SPI peripheral
- Requires 3 GPIOs + 1 GPIO per SPI peripheral
- Only one device can be the SPI controller

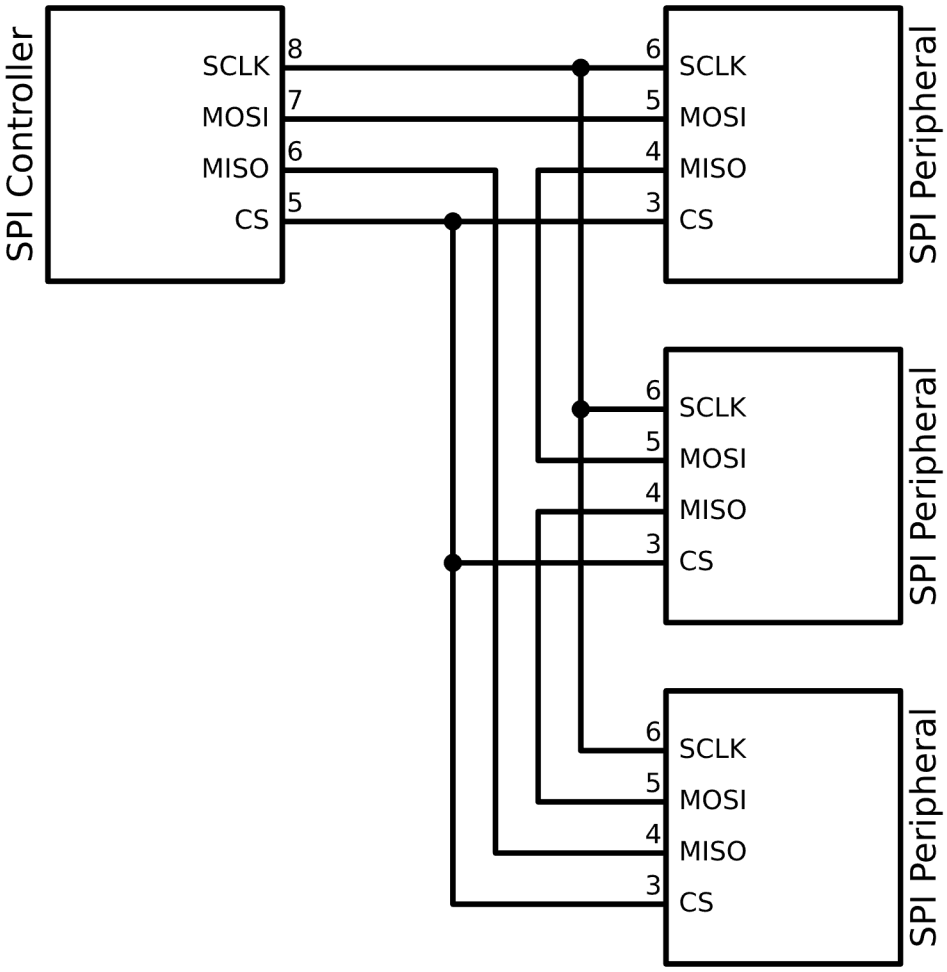


Not as common

de er linket sammen, de repeater signalet.

SPI: Daisy Chain

- SPI controller and peripherals form a circle
  - The out of the first peripheral goes at the input of the second, etc
  - The output of the last peripheral goes to the input of the controller
- Peripherals forward data to the next
- All peripherals are enabled together for a data transfer
- Less pins required but all peripherals need to be active, even if the controller needs to talk to one



delay med scale

Example of an SPI Peripheral

- **ADXL362:** A 3-axes accelerometer
- Provides an SPI interface
- Provides a command/control interface over SPI

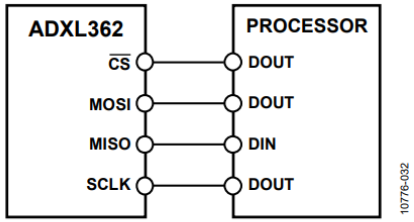


Figure 35. 4-Wire SPI Connection Diagram

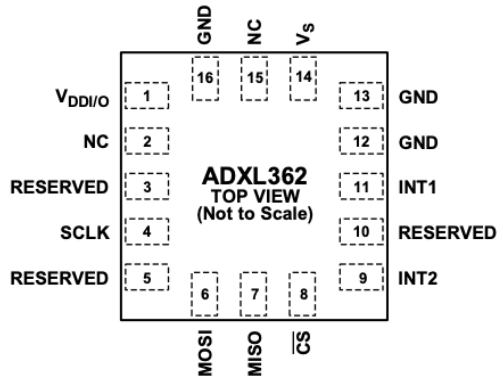
SPI COMMANDS

The SPI port uses a multibyte structure wherein the first byte is a command. The ADXL362 command set is

- 0x0A: write register
- 0x0B: read register
- 0x0D: read FIFO

Table 11. Register Summary

Reg	Name	Bits	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset	RW
0x00	DEVID_AD	[7:0]	DEVID_AD[7:0]								0xAD	R
0x01	DEVID_MST	[7:0]	DEVID_MST[7:0]								0x1D	R
0x02	PARTID	[7:0]	PARTID[7:0]								0xF2	R
0x03	REVID	[7:0]	REVID[7:0]								0x01	R
0x08	XDATA	[7:0]	XDATA[7:0]								0x00	R
0x09	YDATA	[7:0]	YDATA[7:0]								0x00	R
0x0A	ZDATA	[7:0]	ZDATA[7:0]								0x00	R



The SPI timing scheme follows CPHA = CPOL = 0.

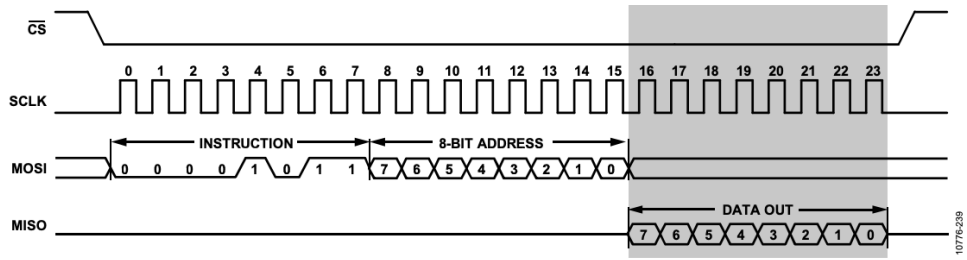


Figure 36. Register Read

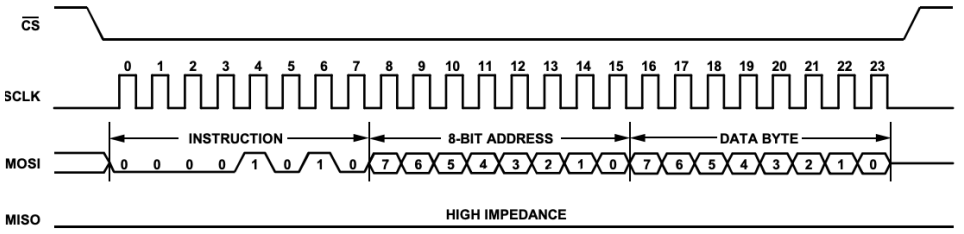


Figure 37. Register Write (Receive Instruction Only)

Image source: ADXL362 Datasheet by Analog Devices

Synchronous Serial with less wires

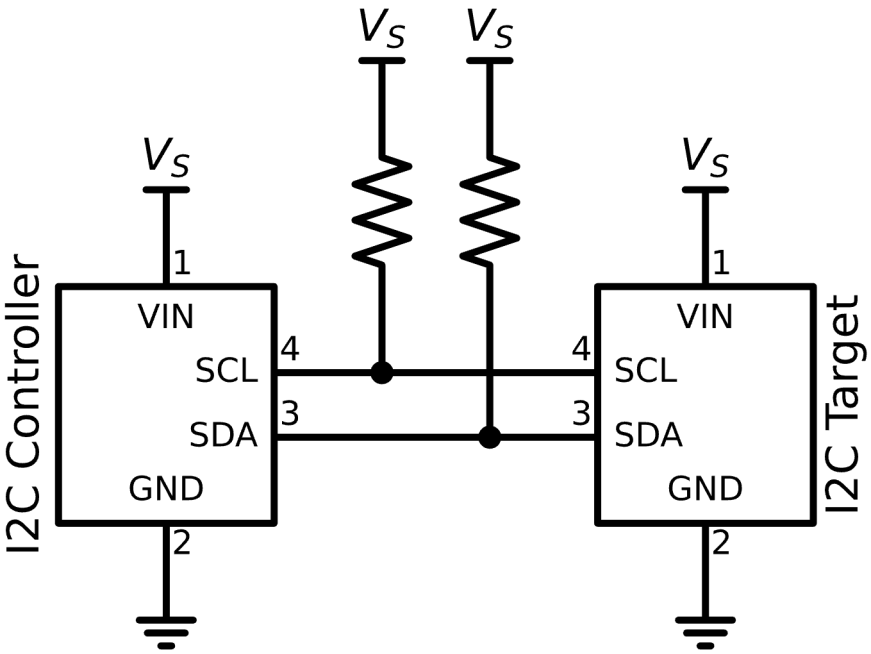
- SPI is faster and cheaper than UART, supports multiple peripherals
  - But requires a lot more GPIO pins, practical limit on the number of peripherals
- How can we reduce the number of required GPIOs?

Synchronous Serial with less wires

- SPI is faster and cheaper than UART, supports multiple peripherals
  - But requires a lot more GPIO pins, practical limit on the number of peripherals
- How can we reduce the number of required GPIOs?
- Eliminate the need for Chip Select (CS)
- Bi-directional data bus

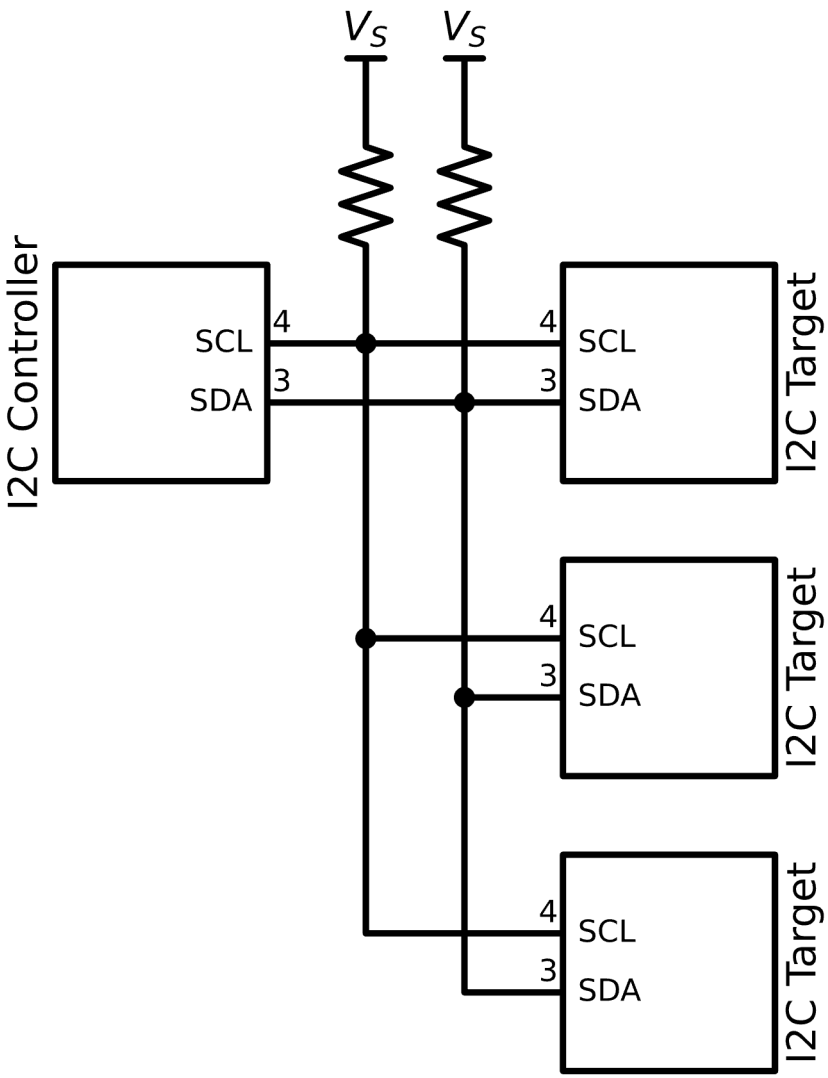
I2C (Inter-Integrated Circuit)

- Only 2 wires: a clock (SCL) and data bus (SDA)
- Outputs are configured as **open-drain**
  - Actively transmit '0' by pulling line to GND
  - Transmit '1' by disabling output (pullup)
  - Different to SPI and UART (push-pull)
- Multiple transmitters can safely share the same bus
- Clock is generated by the controller (100-400 KHz)
  - Faster than UART, slower than SPI
  - Open-drain is slower than push-pull
- Consumes energy when transmitting '0'
  - If V_S=5V, each pullup resistor (5K) consumes 5mW when transmitting '0'



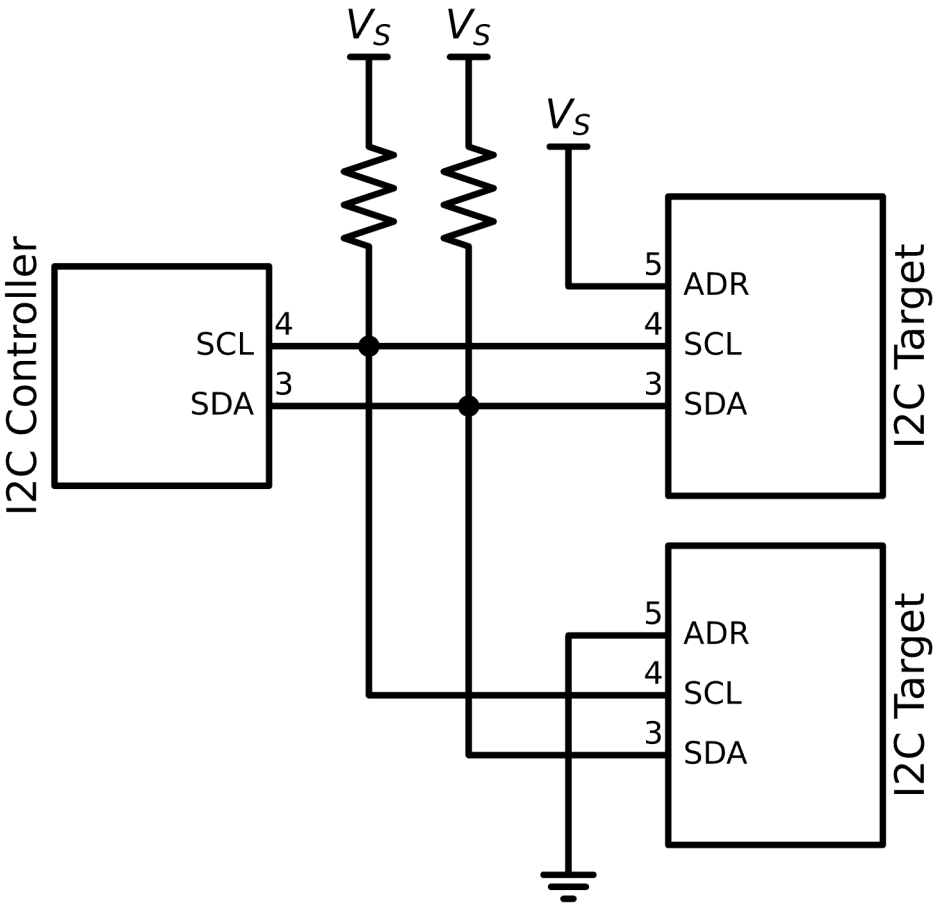
I2C: Multiple Peripherals

- Peripherals, known as I2C targets, simply connect to the SCL and SDA buses
- The CS functionality is replaced by **addresses**
- Each I2C target has a 7-bit address
  - Theoretically, supporting up to 107 peripherals (some addresses are reserved)
- Addresses of peripherals are often fixed
  - Sometimes they are configurable
- It is vital not to have two peripherals with the same address on the same I2C bus
- A variant of I2C with 10-bit addresses also exists



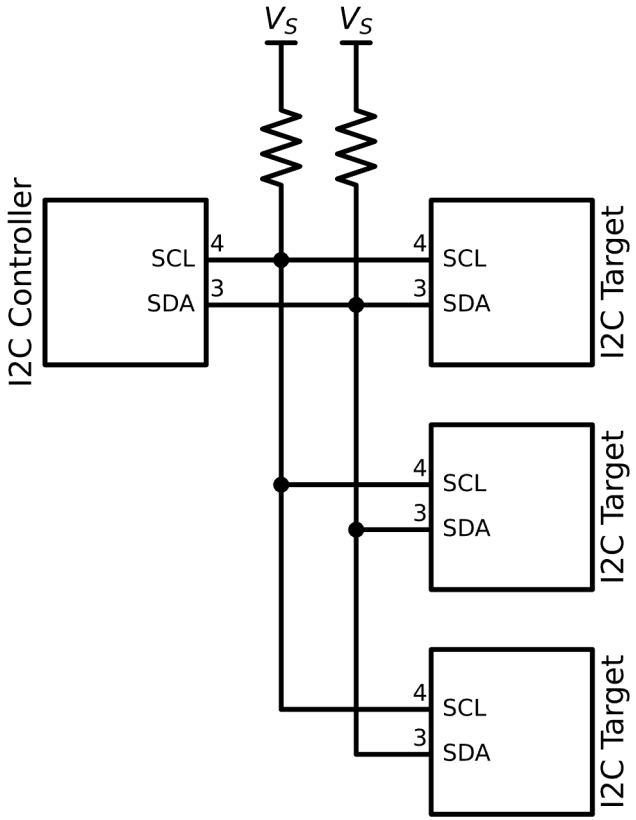
I2C: Configurable Addresses

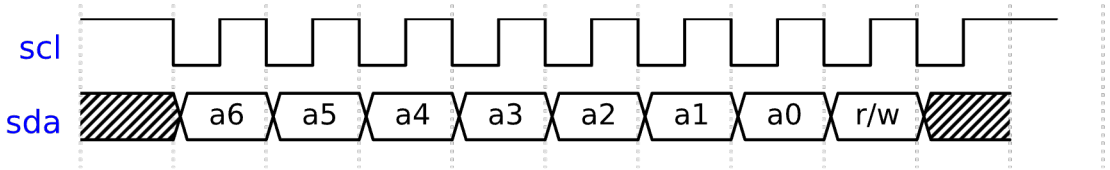
- Consider that you want to connect two chips of the same sensor on the I2C bus
- The address of the sensor has 6 fixed bits but the 7th is read from an input pin
- For example:
  - A sensor has I2C address: 0b011010x
  - 'x' is read from the ADR pin
  - The top target has address 0b0110101
  - The bottom target has address 0b0110100
- A target may have zero, one, or more configurable address bits



I2C: Address Frame

- All I2C data transfers are initiated by the controller
- The first byte that the controller puts on the line is composed of the **7-bit address** and a bit that indicates if the controller wants to **read (1)** from or **write (0)** to the peripheral
- Peripherals whose I2C address does not match will ignore it
- Example:
  - Peripheral 1 has I2C address: 0b0110101
  - To write to it the controller writes: 0b01101010 (0x6A)
  - To read from it the controller writes: 0b01101011 (0x6B)

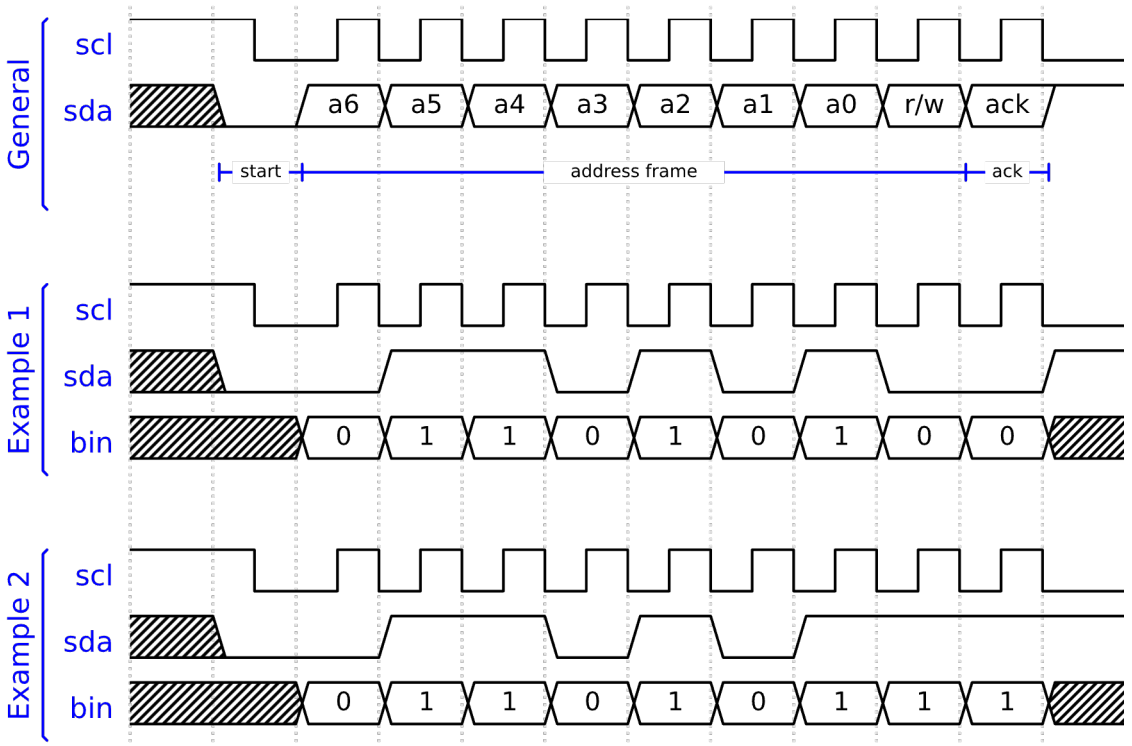




I2C Protocol: Start Condition and Acknowledgement

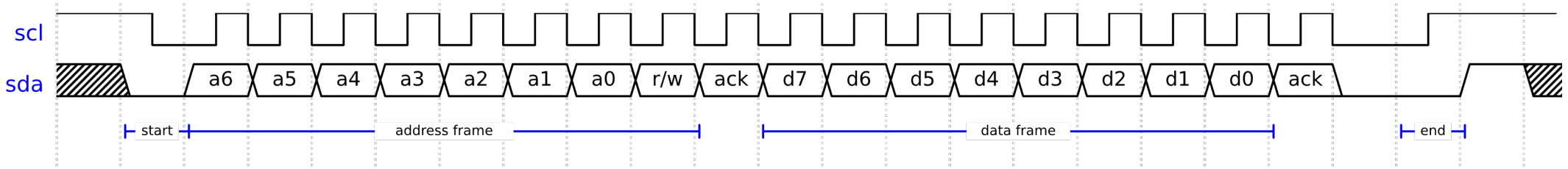
- **Start Condition**: to initiate a transaction, the controller device leaves SCL high and pulls SDA low
- All peripherals get ready to receive
- The controller writes the **address frame** byte indicating the target address and its intention to read or write
- The controller disables its output and gives control of SDA to the target
- **ACK Bit**: The target pulls SDA low to indicate a successful reception
- If error, the pullup will keep line high

- Example 1: target address 0x0110101, write operation, target acknowledges
- Example 2: target address 0x0110101, read operation, error



I2C Protocol: Data Frame and Stop Condition

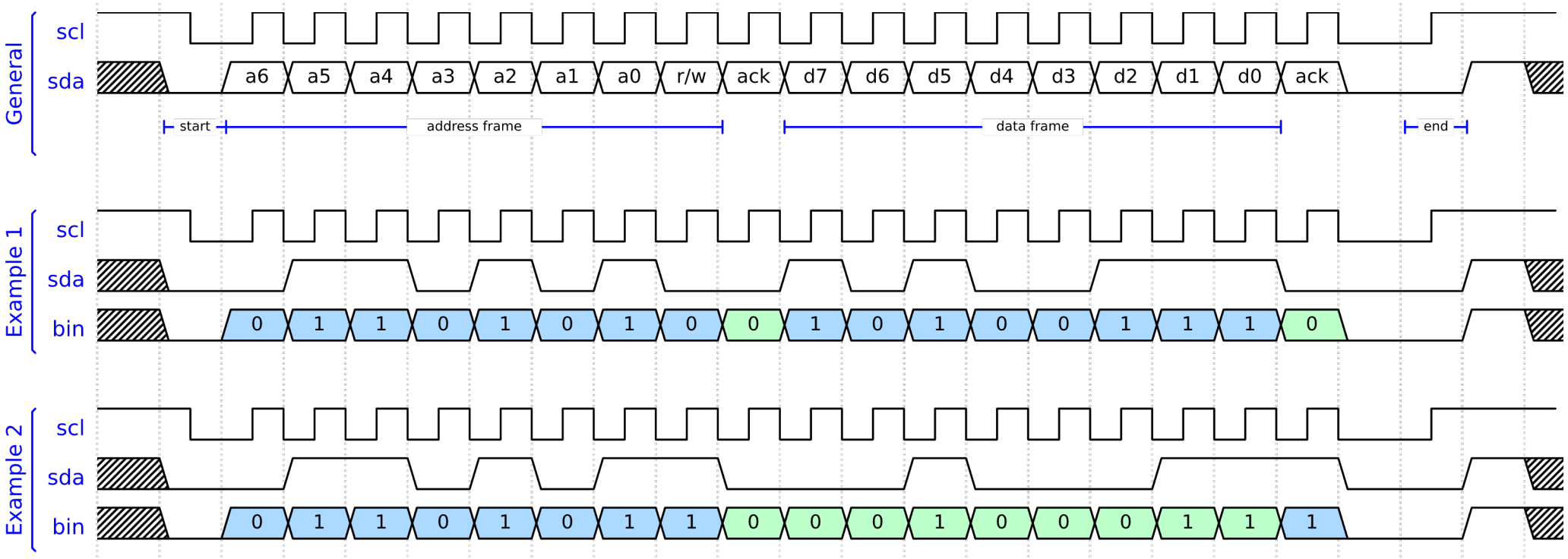
- After the successful transmission of the address frame, the controller continues generating clock pulses and the data transmission begins
- The SDA is controlled by either the controller or the target, according to the W/R bit
- Each **data frame** is a byte (8 bits) followed by an acknowledgement bit from the receiver
  - One or more data frames can be transmitted
- If the last data frame is a read, the controller sends a negative acknowledgement (NACK) to indicate it intends to terminate the transfer
- **Stop Condition**: The controller ends the transaction by low to high transition on the SDA line after a low to high transition on the SCL line



I2C: Transaction Examples

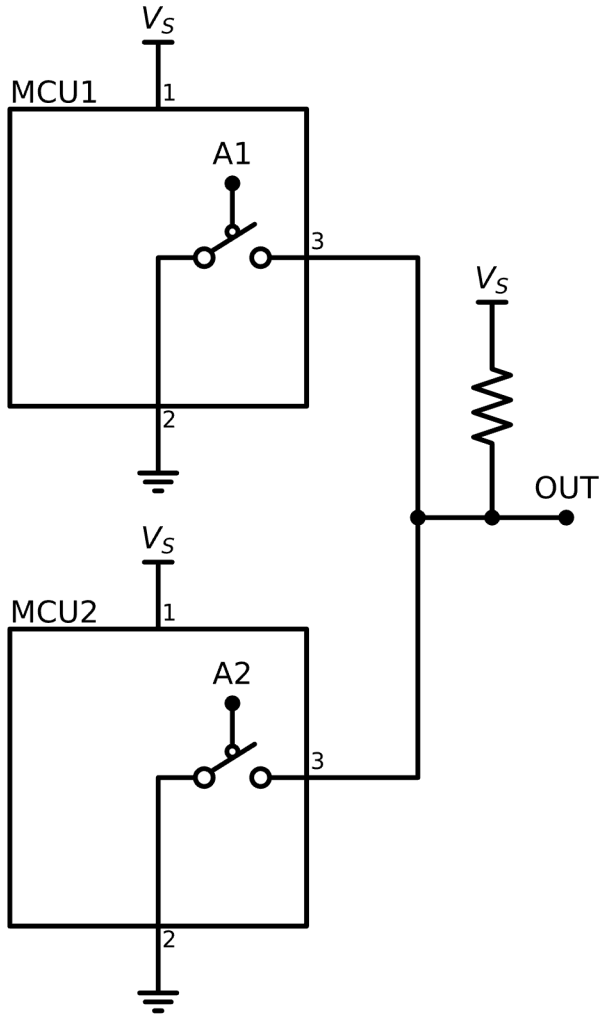
- Example 1: Controller writes a byte to target
- Example 2: Controller reads a byte from target

Blue: Controller controls the SDA line
Green: Target controls the SDA line



Open-Drain Output

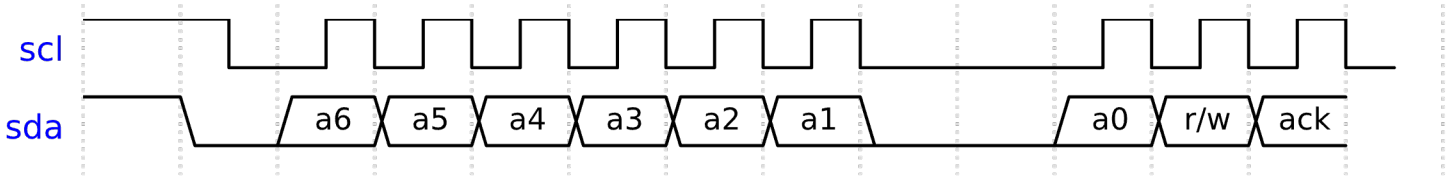
- Open-Drain mode has two states:
  - Logical ‘0’ (i.e., low voltage)
  - Hi-Z (i.e., disconnected)
- Sets the line to ‘0’ **actively**
- Sets the line to ‘1’ **indirectly** by disconnecting the output and letting the pullup resistor raise it to ‘1’
- The line is ‘1’ only if nobody transmits ‘0’



A1	A2	OUT
0	0	0 (both)
0	1	0 (MCU1)
1	0	0 (MCU2)
1	1	1 (pullup)

I2C: Clock Stretching

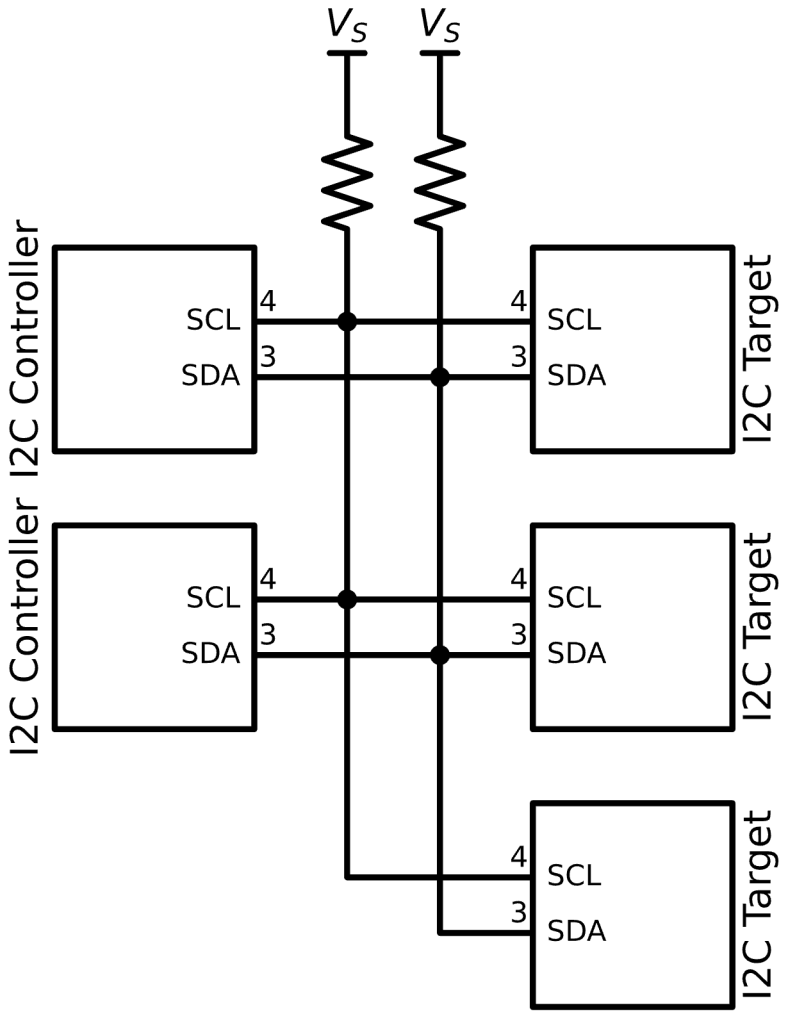
- In an open-drain bus, like I2C, if two devices control the line at the same time, ‘0’ always wins
- A target that needs more time to process data, can pull the SCL line to ‘0’
- The controller cannot transmit ‘1’ to generate a new clock pulse until SCL is released
- Essentially, the target temporarily stops time



I2C: Multiple Controllers

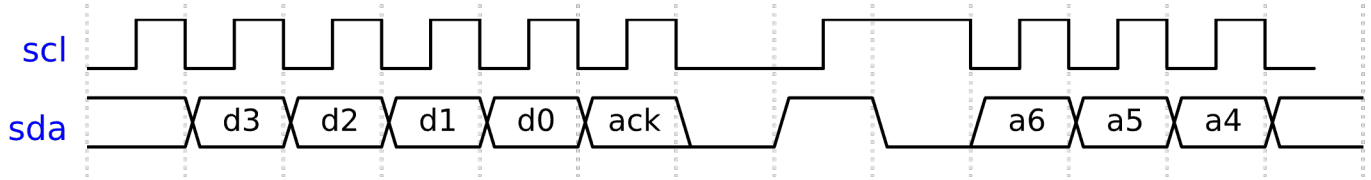
- I2C supports multiple controllers due to the open-drain bus
  - Controllers monitor the bus for start/stop conditions and do not initiate a start condition
  - However, two controllers may start a transmission at the same time

- **Arbitration**
  - If two devices control the line at the same time, '0' always wins
  - When a device transmits '1' (indirectly by disabling its output), it can read the state of the line in parallel
  - If the line remains '0', it means somebody else is active at the same time
  - The first controller that detects an inconsistency backs off and lets the other finish
  - The last to transmit '1' wins



I2C: Repeated Start Condition

- The controller can make transactions back-to-back by generating a repeated start condition
- First, the controller raises both SDA and SCL high avoiding generating a stop condition
- Then, it generates a new start condition, by pulling SDA down before the SCL
- Other I2C controllers do not interfere waiting for a stop condition

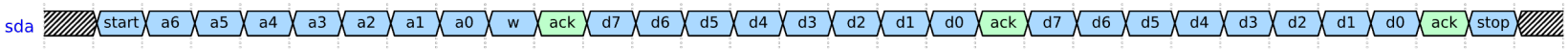


I2C: Write/Read Target Register

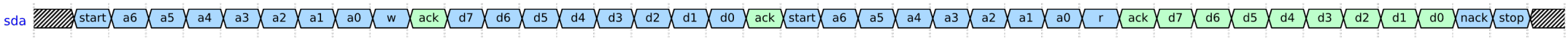
Blue: Controller controls the SDA line
Green: Target controls the SDA line

- Peripherals typically organise their internal memory in registers
- Each register has an address and a value
- To read/write a register, the controller needs to write the register address it wishes to read/write

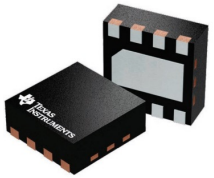
- Write to target register (3 bytes):
  - Start -> Target Address (W) -> Register Address -> Register Value -> Stop



- Read register from target (4 bytes):
  - Start -> Target Address (W) -> Register Address -> Re-Start -> Target Address (R) -> Register Value -> Stop



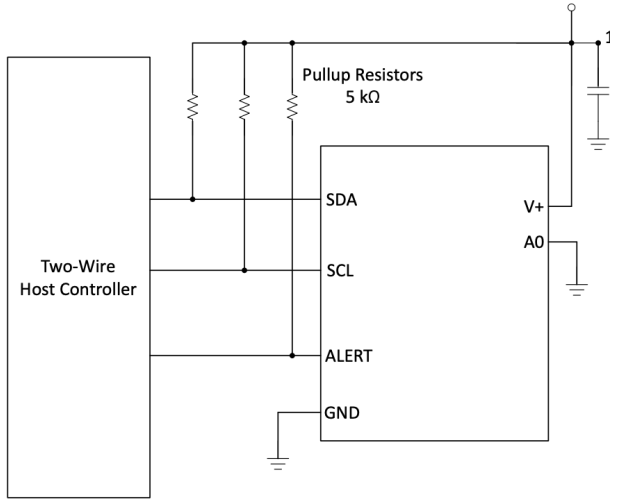
Example of an I2C Peripheral



- TMP1075: A temperature sensor
- Provides an I2C interface
- Configurable I2C address: 10010xx
- Provides a command/control interface over I2C

Table 9-5. TMP1075 Register Map

ADDRESS	TYPE	RESET	ACRONYM	REGISTER NAME
00h	R	0000h	TEMP	Temperature result register
01h	R/W	00FFh	CFGR	Configuration register
02h	R/W	4B00h	LLIM	Low limit register
03h	R/W	5000h	HLIM	High limit register
0Fh[(1)]	R	7500h	DIEID	Device ID register



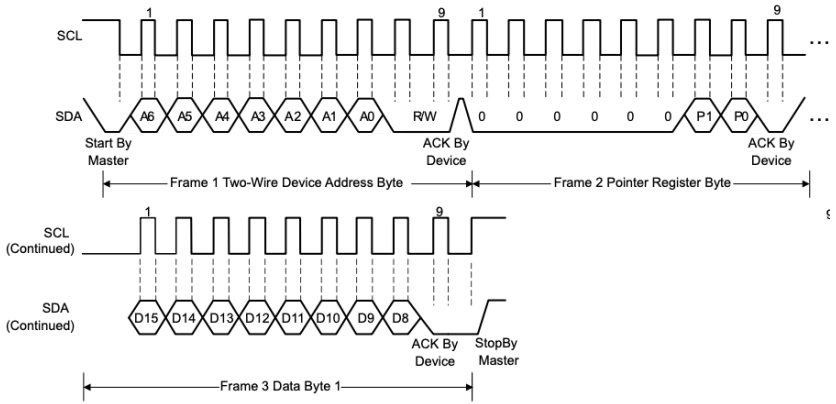


Figure 9-4. Two-Wire Timing Diagram for Write Single Byte Format

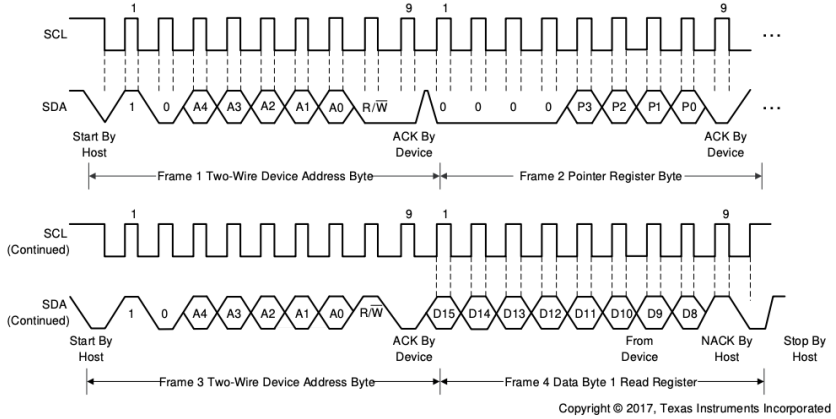


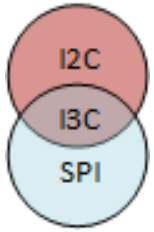
Figure 9-6. Two-Wire Timing Diagram for Read Single Byte Format

Image source: TMP1075N Datasheet by Texas Instruments

Summary

Property	UART	SPI	I2C
No. Peripherals	Point-to-Point	Many (GPIO)	Many (address)
Multi-Controller	No	No	Yes
Rate (typical)	10-100 Kbps	1-20 Mbps	100-400 Kbps
GPIOs Required	2	3 + 1 per peripheral	2
Simultaneous Tx/Rx	Depends on hardware	Full-duplex	Half-duplex
Overhead	2-4 bits per byte	0	11 + 1 per byte
Energy Requirements	++	+	+++

I3C



- Aim to be the best of both worlds
- Primary controller controls the clock
  - Secondary controller may exist
- Switches from open-drain to push-pull on the SDA whenever possible
  - After start condition controller switches the SDA to push-pull, allowing the clock to be increased up to 12.5 MHz
- Pullup resistors are provided by the I3C controller
- Clock stretching and 10-bit addresses not supported

Image source: Wikipedia