

# Serial Communication

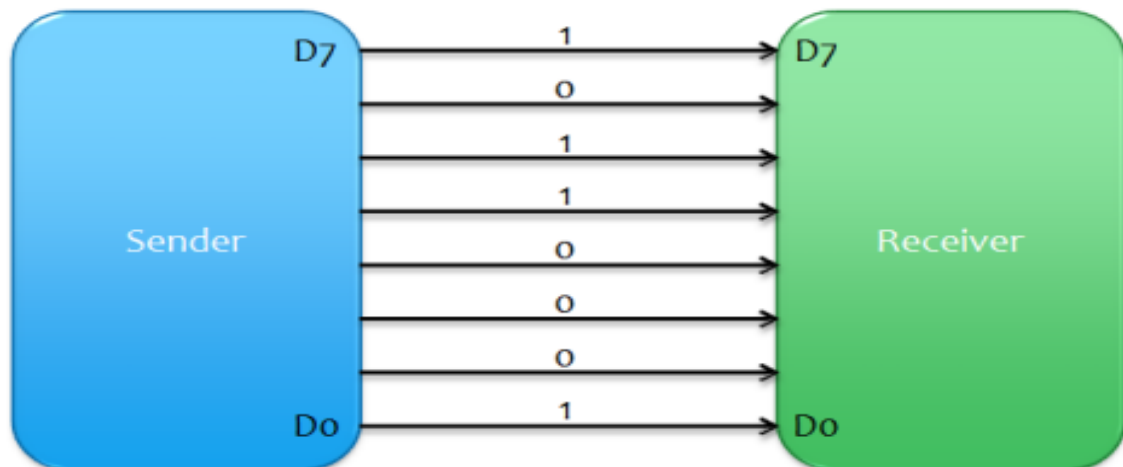


# Serial Communication

- > Data transmission
- > USART fundamentals
- > SPI
- > I2C
- > PS/2 Keyboard

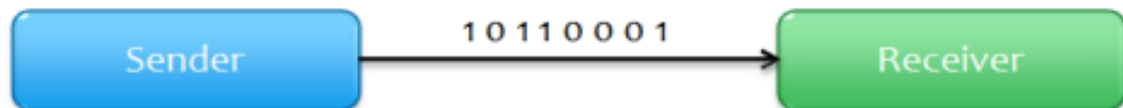
[https://en.wikipedia.org/wiki/Universal\\_asynchronous\\_receiver-transmitter](https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter)  
<http://maxembedded.com/2013/09/serial-communication-introduction/>  
[https://upload.wikimedia.org/wikipedia/commons/1/1f/Serial\\_Programming.pdf](https://upload.wikimedia.org/wikipedia/commons/1/1f/Serial_Programming.pdf)

## Parallel Transfer



(c) Copyright, Mayank Prasad, 2012  
maxEmbedded.wordpress.com

## Serial Transfer



(c) Copyright, Mayank Prasad, 2012  
maxEmbedded.wordpress.com

# Data Transmission

Inside the CPU or Microcontroller data is transmitted in parallel using buses (8/16/32 Bit).

For long distances we use a single wire and serial transmission.



(c) Copyright, Mayank Prasad, 2012  
maxEmbedded.wordpress.com

Data Transfer in  
Serial Communication

# Data Receive

## How long does a Bit last?

-> Transmission speed, Bitrate, Baudrate (Bit/sec)

## When does the transmission begin / end?

### Synchronous Transmission

-> Clock signal -> additional wire necessary

### Asynchronous Transmission

-> Start-Bit & Stop-Bit

-> fixed data rate (known by sender and receiver)

# USART - Universal Synchronus / Asynchronus Receiver Transmitter

RxD – Receive Data

TxD – Transmit Data

GND

Point-to-Point Connection / Transmission

The Transmit-Pin (Tx) from the Sender has to be connected with the Receive-Pin (Rx) from the Receiver.

Asynchronus works without a clock signal that shows when the data is valid. As a result both microcontrollers need to use the same data transmission rate well known as the baudrate !  
(Typical values are 9600 // 19200 // 57600 // 115200)

Additional Control Lines -> RS232

# Universal Synchronus / Asynchronus Receiver Transmitter

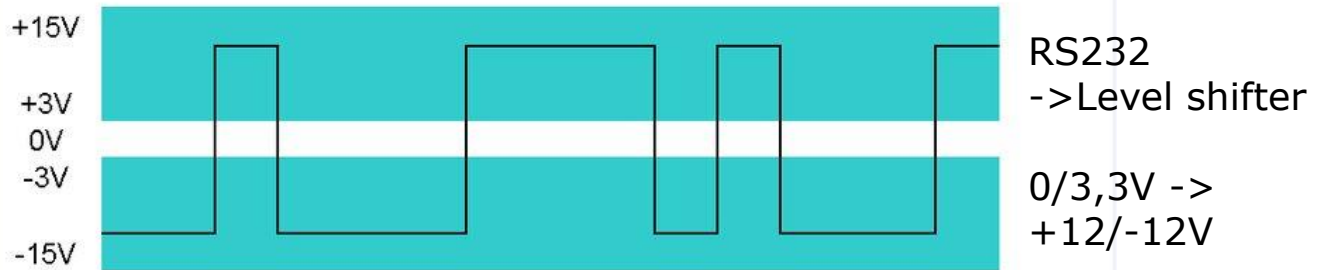
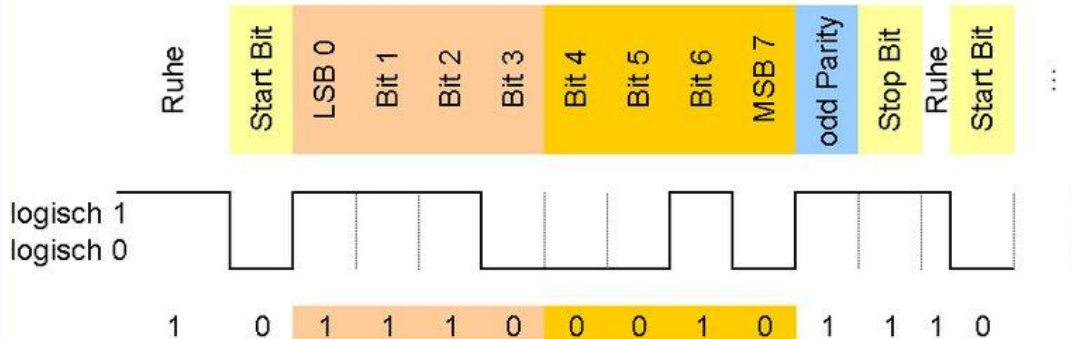
Synchronisation

Daten low & high

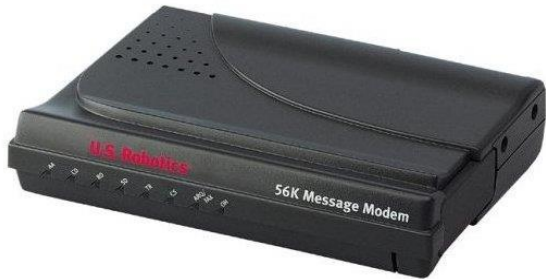
Check

9600 8O1 = 9600 Baud; 8 Datenbits; odd Parity; 1 Stopbit

ASCII "G" = \$47 = 0100 0111



# RS232 – PC - Modem

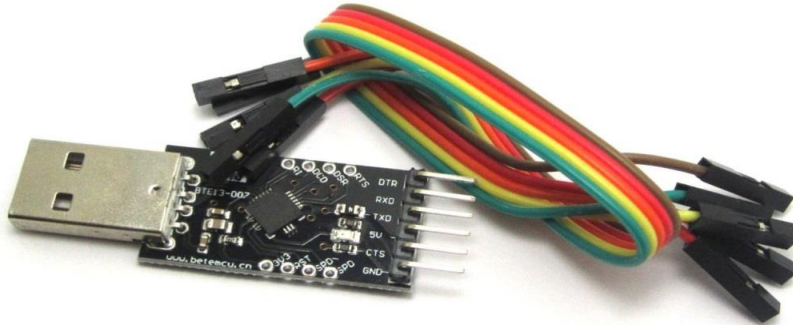


classic 2 wire  
baudrate 56kBit/sec



# USB – Universal Serial Bus

Due to the widespread use of the Arduino platform and the constant growth of the maker community, adapters are available that convert from USB to USART with a voltage level of 5V or 3.3V. (FTDI chip)



<http://www.netzmafia.de/skripten/hardware/PC-Schnittstellen/usb.html>

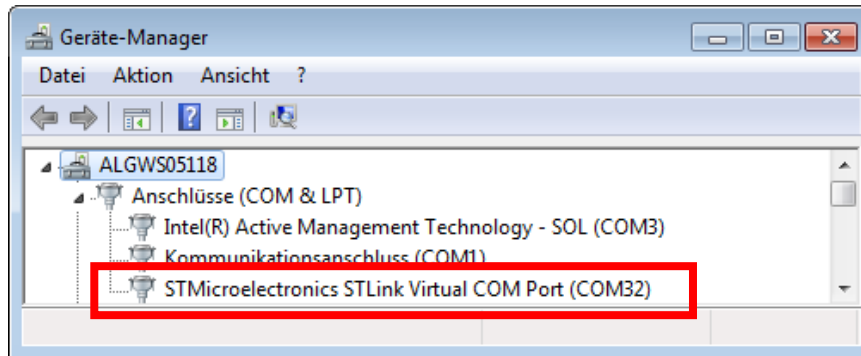
<https://en.wikipedia.org/wiki/USB>

# USART - STM32 - STLink

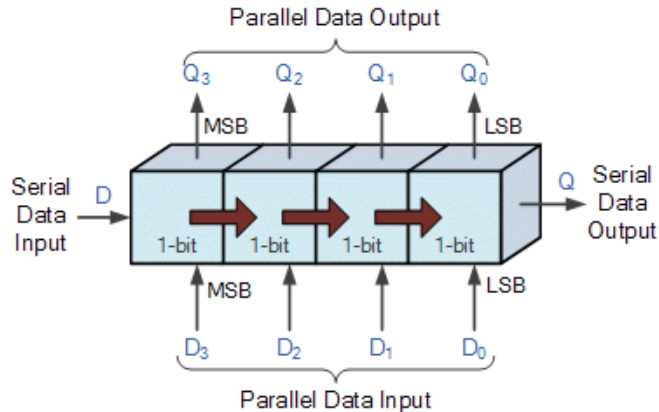
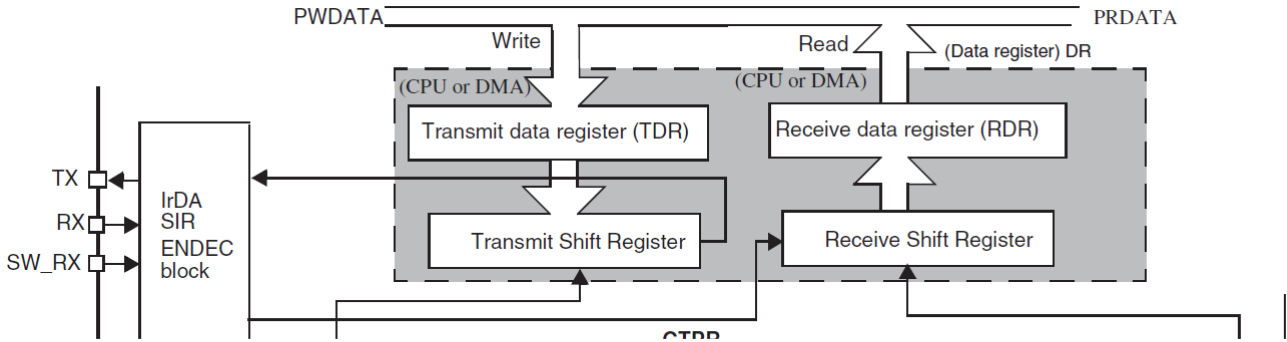
STM32 with Software (Firmware):

-> **Virtual Com Port** (USART to USB Tunnel)

CP210X and others



# USART - STM32 - Shift Register



# USART - ASCII

The image shows a browser window displaying an ASCII table. The table is organized into two columns. The first column lists characters from 0 to 27, and the second column lists characters from 96 to 127. Each row includes the decimal, hexadecimal, binary, and octal representations of the character, followed by the character itself. The characters are color-coded: digits are red, letters are green, and symbols are purple.

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char		
48	30	110000	60	0	96	60	1100000	140	\		
49	31	110001	61	1	97	61	1100001	141	a		
50	32	110010	62	2	98	62	1100010	142	b		
51	33	110011	63	3	99	63	1100011	143	c		
52	34	110100	64	4	100	64	1100100	144	d		
53	35	110101	65	5	101	65	1100101	145	e		
54	36	110110	66	6	102	66	1100110	146	f		
15	[CARRIAGE RETURN]	61	3D	111101	75	=	109	6D	1101101	155	m
16	[SHIFT OUT]	62	3E	111110	76	>	110	6E	1101110	156	n
17	[SHIFT IN]	63	3F	111111	77	?	111	6F	1101111	157	o
20	[DATA LINK ESCAPE]	64	40	1000000	100	@	112	70	1100000	160	p
21	[DEVICE CONTROL 1]	65	41	1000001	101	A	113	71	1100001	161	q
22	[DEVICE CONTROL 2]	66	42	1000010	102	B	114	72	1100010	162	r
23	[DEVICE CONTROL 3]	67	43	1000011	103	C	115	73	1100011	163	s
24	[DEVICE CONTROL 4]	68	44	1000100	104	D	116	74	1101000	164	t
25	[NEGATIVE ACKNOWLEDGE]	69	45	1000101	105	E	117	75	1101001	165	u
26	[SYNCHRONOUS IDLE]	70	46	1000110	106	F	118	76	1101010	166	v
27	[END OF TRANS. BLOCK]	71	47	1000111	107	G	119	77	1101011	167	w
				1001000	110	H	120	78	1110000	170	x
				1001001	111	I	121	79	1110001	171	y
				1001010	112	J	122	7A	1110100	172	z
				1001011	113	K	123	7B	1110101	173	{
				1001100	114	L	124	7C	1110100	174	
				1001101	115	M	125	7D	1110101	175	}
				1001110	116	N	126	7E	1111100	176	~
				1001111	117	O	127	7F	1111101	177	[DEL]

# PC Terminal Program - TeraTerm

Tera Term: New connection

TCP/IP    Host: myhost.example.com

History    TCP port#: 22

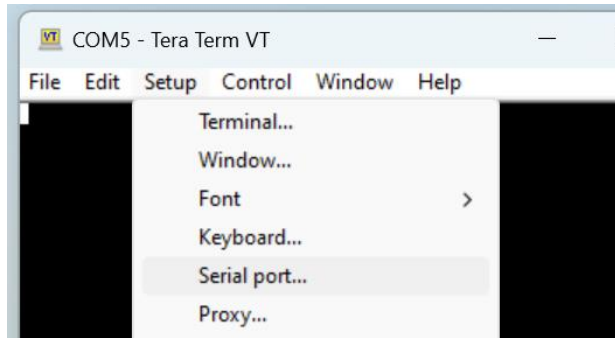
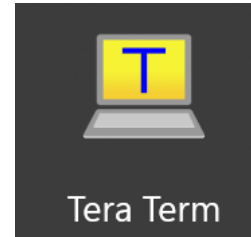
Service:  Telnet    SSH version: SSH2

SSH    IP version: AUTO

Other

Serial    Port: COM5: STMicroelectronics STLink Virtu

OK    Cancel    Help



Tera Term: Serial port setup and connection

Port: COM5    New setting

Speed: 9600

Data: 8 bit    Cancel

Parity: none

Stop bits: 1 bit    Help

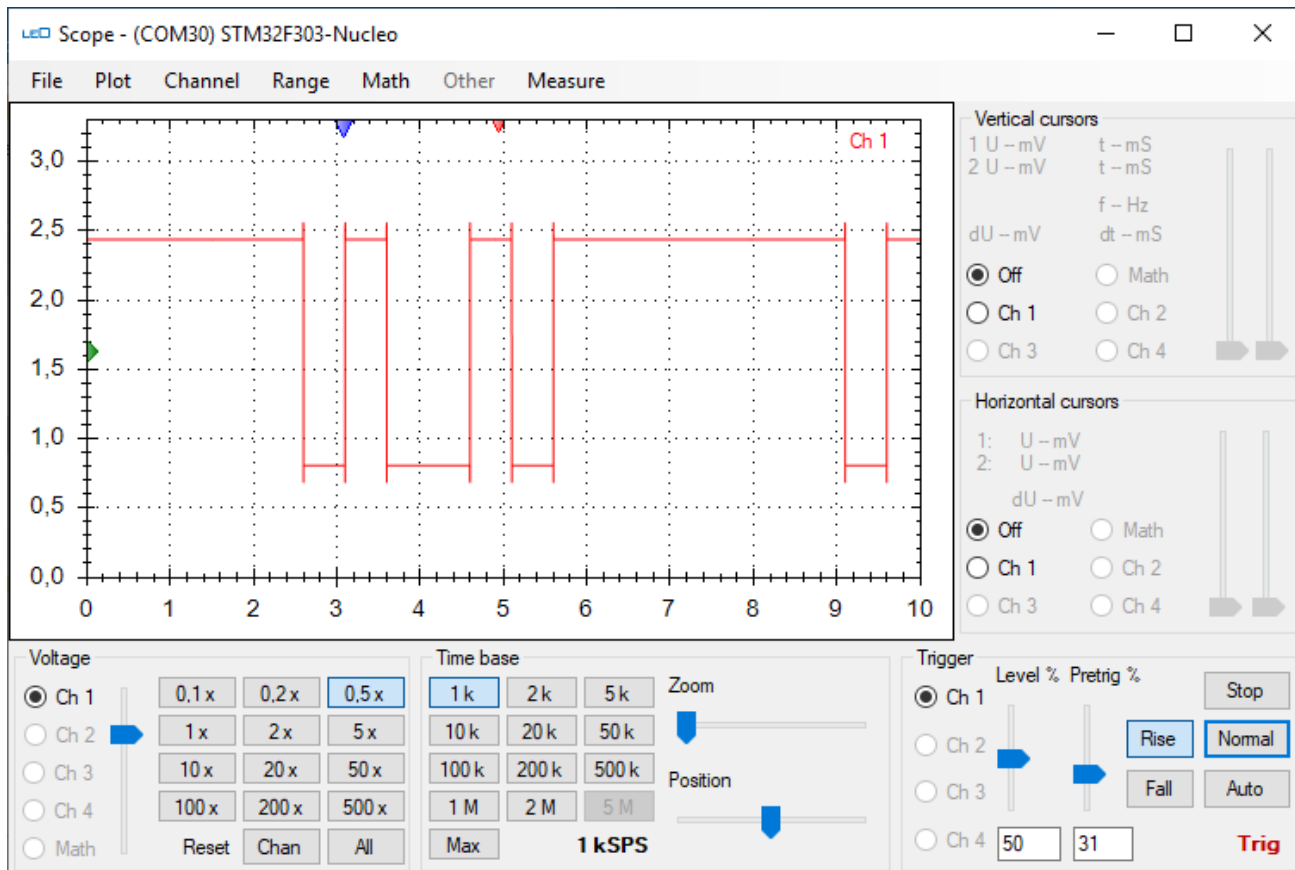
Flow control: none

Transmit delay

0 msec/char    0 msec/line

Device Friendly Name: STMicroelectronics STLink Virtual COM P  
Device Instance ID: USB\VID\_0483&PID\_374B&MI\_0217&1E99D5A  
Device Manufacturer: STMicroelectronics

# Data length 10k samples – 10 sek



HOME

ESP32

ESP8266

ESP32-CAM

MICROPYTHON

ARDUINO

REVIEWS

PROJECTS

## Learn ESP32

ESP32 Introduction

ESP32 Arduino IDE

ESP32 Arduino IDE 2.0

VS Code and PlatformIO

ESP32 Pinout

ESP32 Inputs Outputs

ESP32 PWM

ESP32 Analog Inputs

ESP32 Interrupts Timers

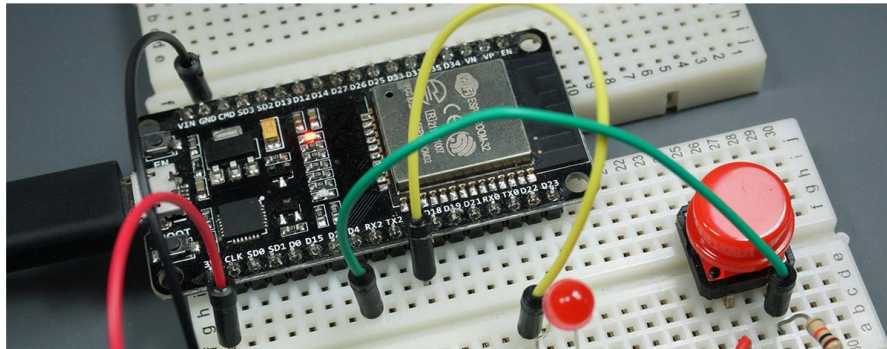
ESP32 Deep Sleep

## Protocols

ESP32 Web Server

# ESP32 Digital Inputs and Digital Outputs (Arduino IDE)

In this getting started guide you'll learn how to read digital inputs like a button switch and control digital outputs like an LED using the ESP32 with Arduino IDE.



The screenshot shows the Arduino Docs website interface. At the top, there is a teal navigation bar with 'ARDUINO.CC' and 'STORE' on the left, and 'HARDWARE', 'SOFTWARE', 'CLOUD', 'PROGRAMMING', 'TUTORIALS', and 'LEARN' on the right. Below this is a secondary teal bar with the 'DOCS' logo and a search icon. The main content area has a light gray background. On the left, there is a sidebar with a 'Physical Pixel' header and a list of menu items: 'Hardware Required', 'Software Required', 'Circuit', 'Schematic', 'Code', 'Processing Code', 'Max patch', and 'Learn more'. The 'Hardware Required' item is highlighted. The main content area features a breadcrumb trail 'Built-in Examples > Physical Pixel' and a large heading 'Physical Pixel'. Below the heading, there is a paragraph: 'Turn a LED on and off by sending data to your Arduino from Processing or Max/MSP.' A revision date 'LAST REVISION: 20.02.2023, 10:07' is shown with a document icon. The next paragraph states: 'This example example uses the Arduino board to receive data from the computer. The board turns on an LED when it receives the character 'H', and turns off the LED when it receives the character 'L'.' The final paragraph reads: 'The data can be sent from the Arduino Software (IDE) serial monitor, or another program like Processing (see code below), Flash (via a serial-net proxy), PD, or Max/MSP.' A section heading 'Hardware Required' is visible at the bottom of the main content area.



```
int incomingByte = 0; // for incoming serial data
const int ledPin = 33;
void setup() {
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  //Serial.println("Hello");
  //delay(1000);
  if (Serial.available() > 0) {
    // read the oldest byte in the serial buffer:
    incomingByte = Serial.read();
    // if it's a capital H (ASCII 72), turn on the LED:
    if (incomingByte == 'H') {
      digitalWrite(ledPin, HIGH);
    }
    // if it's an L (ASCII 76) turn off the LED:
    if (incomingByte == 'L') {
      digitalWrite(ledPin, LOW);
    }
  }
}
```

```
1  int incomingByte = 0; // for incoming serial data
2  const int ledPin = 33;
3  void setup() {
4  Serial.begin(115200);
5  pinMode(ledPin, OUTPUT);
6  }
7
8  void loop() {
9  //Serial.println("Hello");
10 //delay(1000);
11  if (Serial.available() > 0) {
12      // read the oldest byte in the serial buffer:
13      incomingByte = Serial.read();
14  // if it's a capital H (ASCII 72), turn on the LED:
15      if (incomingByte == 'H') {
16          digitalWrite(ledPin, HIGH);
17      }
18      // if it's an L (ASCII 76) turn off the LED:
19      if (incomingByte == 'L') {
20          digitalWrite(ledPin, LOW);
21      }
22  }
23 }
```

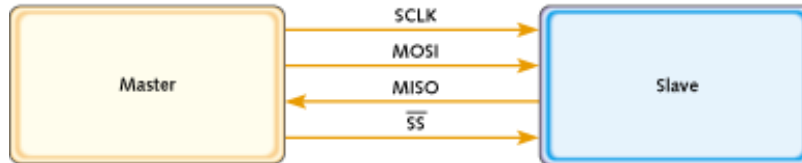
# Anhang

> SPI

> I2C

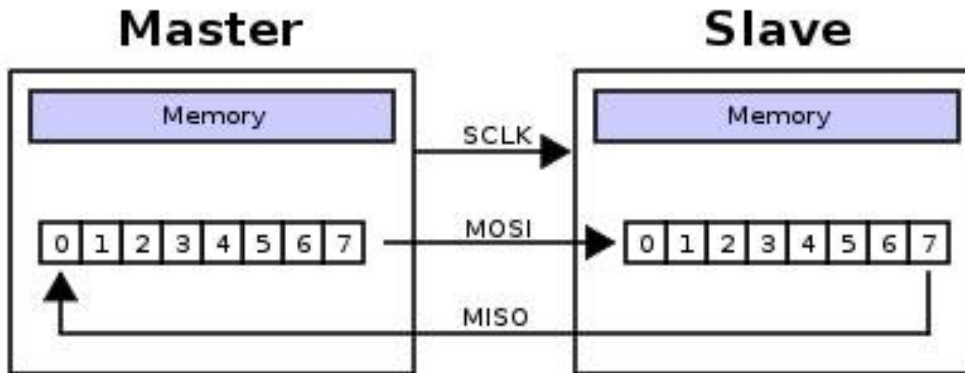
> PS/2 Keyboard

## SPI Bus



- > Synchronous serial data link operating at full duplex
- > Master/slave relationship
- > 2 data signals:
  - » MOSI – master data output, slave data input
  - » MISO – master data input, slave data output
- > 2 control signals:
  - » SCLK – clock
  - »  $\overline{SS}$  – slave select (no addressing)

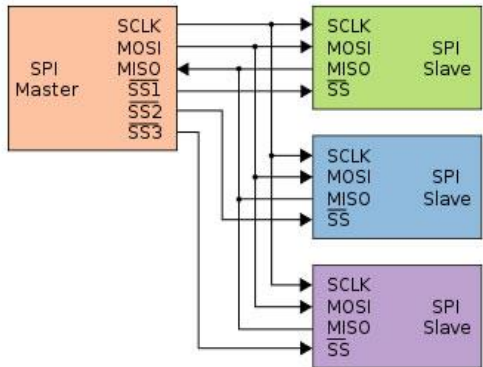
# SPI uses a "shift register" model of communications



Master shifts out data to Slave, and shifts in data from Slave

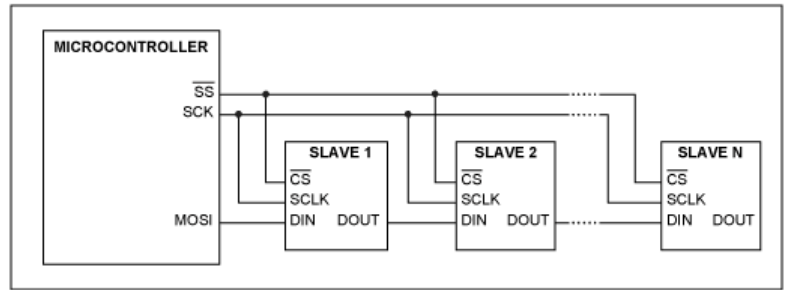
[http://upload.wikimedia.org/wikipedia/commons/thumb/b/bb/SPI\\_8-bit\\_circular\\_transfer.svg/400px-SPI\\_8-bit\\_circular\\_transfer.svg.png](http://upload.wikimedia.org/wikipedia/commons/thumb/b/bb/SPI_8-bit_circular_transfer.svg/400px-SPI_8-bit_circular_transfer.svg.png)

# Two bus configuration models



## Master and multiple independent slaves

[http://upload.wikimedia.org/wikipedia/commons/thumb/f/fc/SPI\\_three\\_slaves.svg/350px-SPI\\_three\\_slaves.svg.png](http://upload.wikimedia.org/wikipedia/commons/thumb/f/fc/SPI_three_slaves.svg/350px-SPI_three_slaves.svg.png)

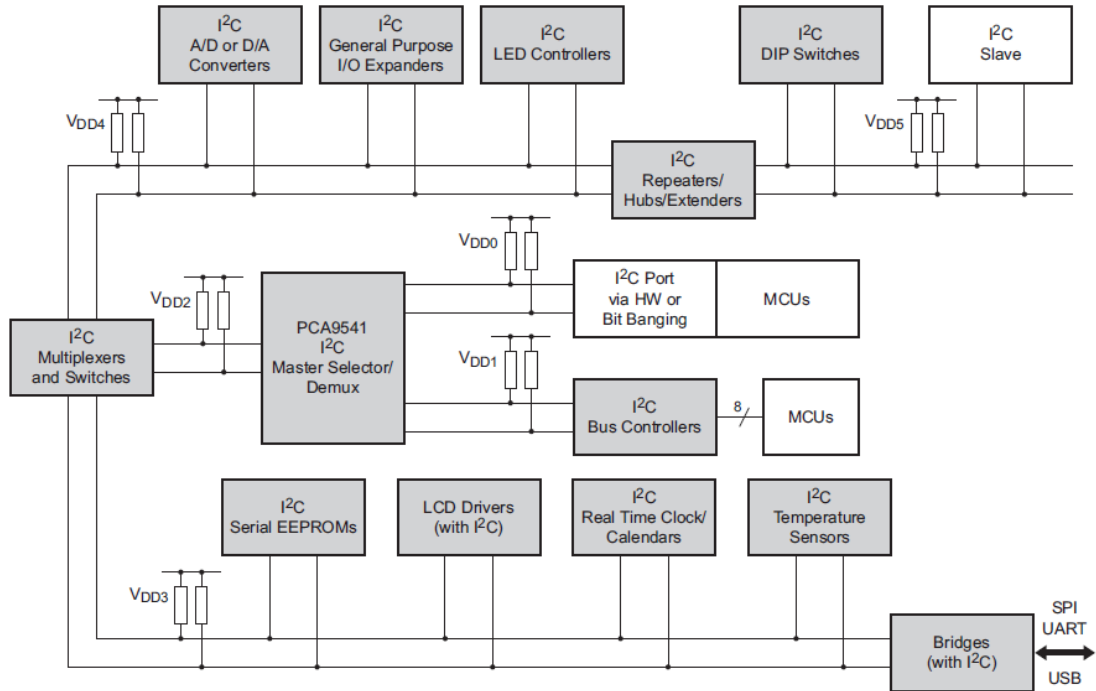


Some wires have been renamed

## Master and multiple daisy-chained slaves

[http://www.maxim-ic.com/appnotes.cfm/an\\_pk/3947](http://www.maxim-ic.com/appnotes.cfm/an_pk/3947)

# I2C / NXP UM10204

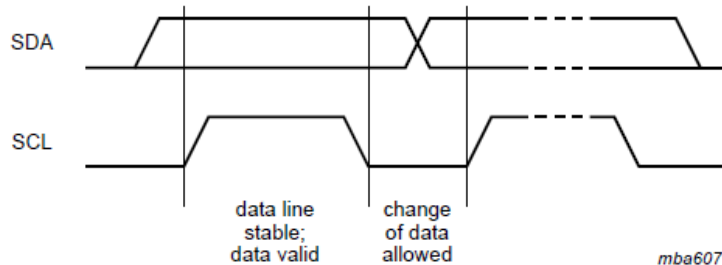


002aac858

# I2C NXP UM10204

## Data validity

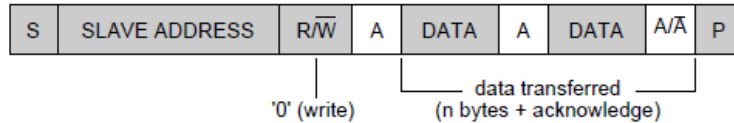
The data on the SDA line must be stable during the HIGH period of the clock. The HIGH or LOW state of the data line can only change when the clock signal on the SCL line is LOW (see [Figure 4](#)). One clock pulse is generated for each data bit transferred.



**Fig 4. Bit transfer on the I<sup>2</sup>C-bus**



# I2C NXP UM10204



from master to slave

from slave to master

A = acknowledge (SDA LOW)

$\bar{A}$  = not acknowledge (SDA HIGH)

S = START condition

P = STOP condition

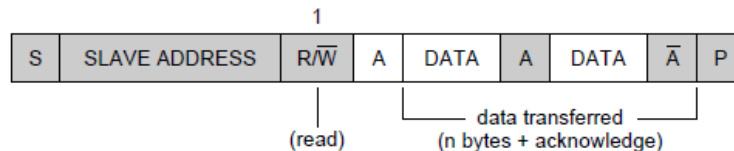
*mbc605*

**A master-transmitter addressing a slave receiver with a 7-bit address (the transfer direction is not changed)**

---



---

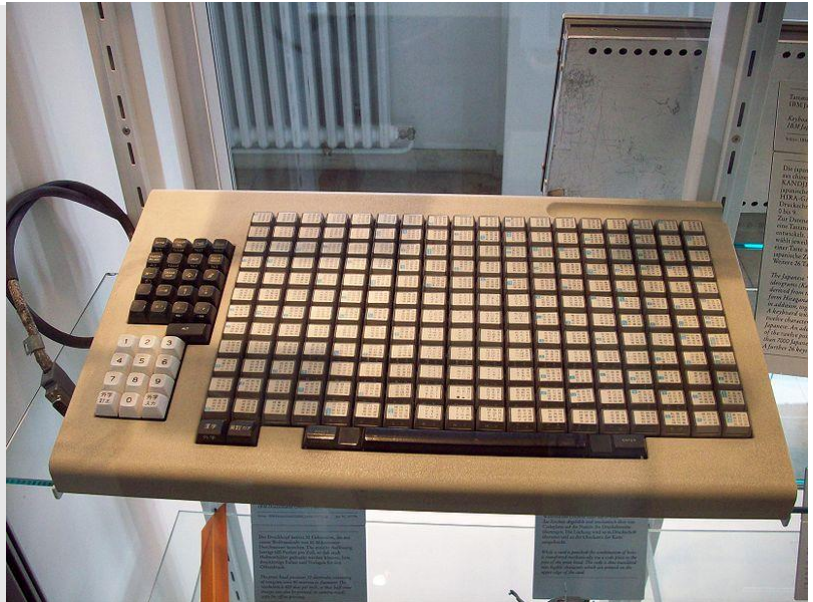


*mbc606*

# Microcontroller

## PS/2 – Keyboard

Thomas Fischer



<http://www.marjorie.de/ps2/start.htm>

<http://www.computer-engineering.org/>

<http://www.schatenseite.de/mamecontrol.html>

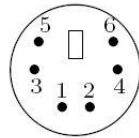
<http://de.wikipedia.org/wiki/Tastatur>

## PS/2 Keyboard

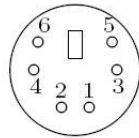
- > If every key would be connected to one pin you would need a controller with 100pins. Within an infinite loop you could poll every pin. -> not the best solution!
- > Better solution is to use the keys as connectors between rows and columns ([matrix](#)), 10 each. If a key is pressed down there will be a connection between one row and one column. Within an infinite loop you set one row to zero and ask all columns if there level is forced to zero. Now you need only 20 pins!
- > A microcontroller (XT-keyboards an 8042) is sending this information to the PC using a [Scancode](#).

## PS/2 Keyboard

- > Clock is zero when data is valid
- > Data line – transmit data bit by bit (serial transmission)



Stecker



Kupplung

## 6-pin Mini-DIN (PS/2):

- 1 - Data
- 2 - nicht belegt
- 3 - Ground
- 4 -  $V_{CC}$  (+5 V)
- 5 - Clock
- 6 - nicht belegt



START DATA0 DATA1 DATA2 DATA3 DATA4 DATA5 DATA6 DATA7 PARITY STOP



# **Serial Communication**