Digital systems and basics of electronics

Adam Szmigielski

aszmigie@pjwstk.edu.pl

materials: $ftp(public)://aszmigie/SYC/ENG$

Communication interfaces - lecture 14

Communication with external devices

- *Ports*
- *Parallel links*
- *Serial links*

Information exchange - processor, memory and input / output devices

- Most microcontrollers (Intel, AVR, PIC) use one *octet* (8 bits) to sending or receiving data in one order cycle,
- For input-output operations, similar information transfer mechanisms are used as for memory (for some microprocessors, the same mnemonics are used to exchange information with both memory and In-Out devices),
- In *Intel, AVR* microprocessors, to emphasize the difference between data exchange between memory and In-Out devices, other mnemonics are used (*in*, *out*, *input*, *output* - for In-Out and *ld*, *ldi*, *mov* - for memory).

Information exchange - processor, memory and input / output devices

- Some processors (e.g. Z80) distinguish between memory access request *Memory Request* and input devices - outputs *Input-Output Request* - separate outputs (pins) of the processor - included in the construction of the address decoder.
- The distinction between memory and input-output devices is the basis for extracting the *input-output port* or *machine port*

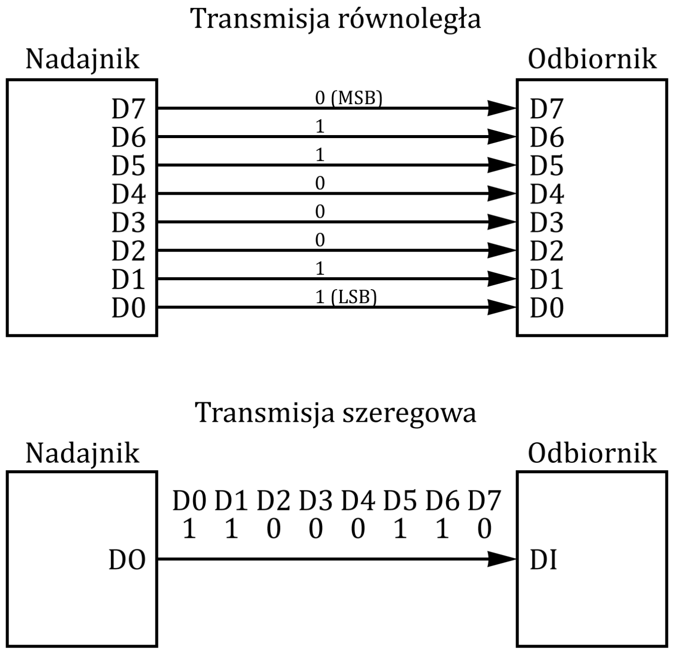
Ports

- Generally, *port* is the interface between a computer and another computer or peripheral devices,
- *hardware ports* - is a piece of hardware that allows you to connect devices external and exchange of information between them and the controller,
- *software ports* - is a virtual (logical) connection that can be used by programmers to directly exchange data (bypassing swap files or other structures temporarily storing data) e.g. TCP, UDP ports and other.

Direct Input Outputs

- Data transmission via I / O port lines - digital signals compliant with CMOS / TTL standard,
- I / O ports - byte organization with the option of setting read / write individual lines (bit addressing),
- After the reset, the level of the port line is in a high impedance state,
- Port lines can perform alternative functions,

Serial and parallel communication



Due to the method of data transfer:

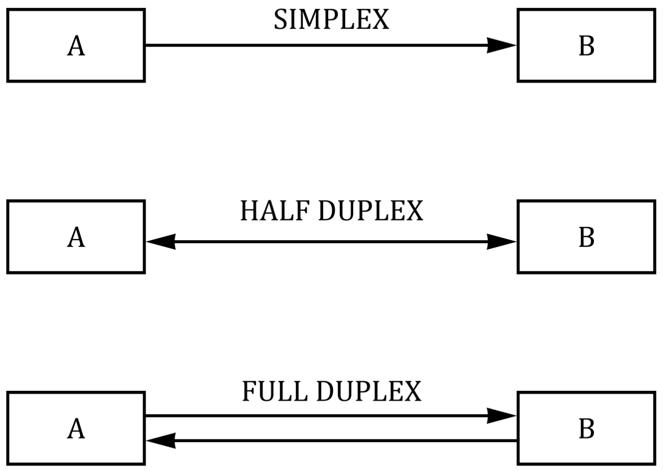
- *serial communication* - the bits are sent in succession,
- *parallel communication* - several bits, usually the byte multiplicity, is sent simultaneously.

Synchronous and asynchronous communication

Due to the type of synchronization:

- *Synchronous transmission* - data transmission takes place at certain times, as determined via synchronization signal (clock),
- *Asynchronous transmission* - data transmission can be started at any time.

Simplex, Duplex and Full Duplex

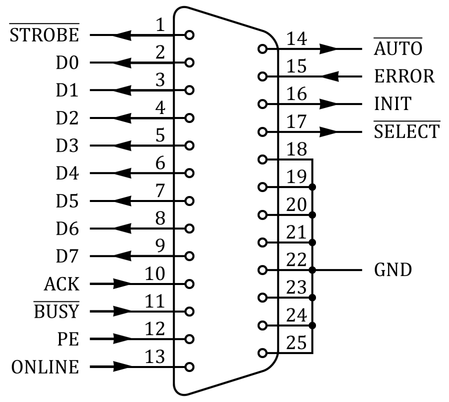


- *Simplex* - The information is sent only from the transmitter to the receiver,
- *Duplex* - Information can be sent in both directions. Only one direction of transmission is possible at a time,
- *Full Duplex* - Information can be sent in both directions at the same time.

Parallel communication

1. *Output / direct output* - usually a port can perform many different functions. The choice of function to perform the port is made programmatically,
2. *Parallel Port* - a port where data is sent simultaneously through several cables, each of which carries one bit of information.

Port LPT



$\overline{\text{STROBE}}$ 1
D0 2
D1 3
D2 4
D3 5
D4 6
D5 7
D6 8
D7 9
ACK 10
$\overline{\text{BUSY}}$ 11
PE 12
ONLINE 13
14 $\overline{\text{AUTO}}$
15 ERROR
16 INIT
17 $\overline{\text{SELECT}}$
18
19
20
21
22 GND
23
24
25

- IEEE 1284 interface - the name of the 25-pin connector on personal computers in the TTL standard.
- IEEE 1284 is a parallel port that is mainly used to connect devices peripherals: printers, scanners, plotters.
- It was developed in 1984.
- The port supports the 8255 system, consisting of *data register* 00H, *input register* (Status) 01H, *control register* (Control) 02H

Serial communication

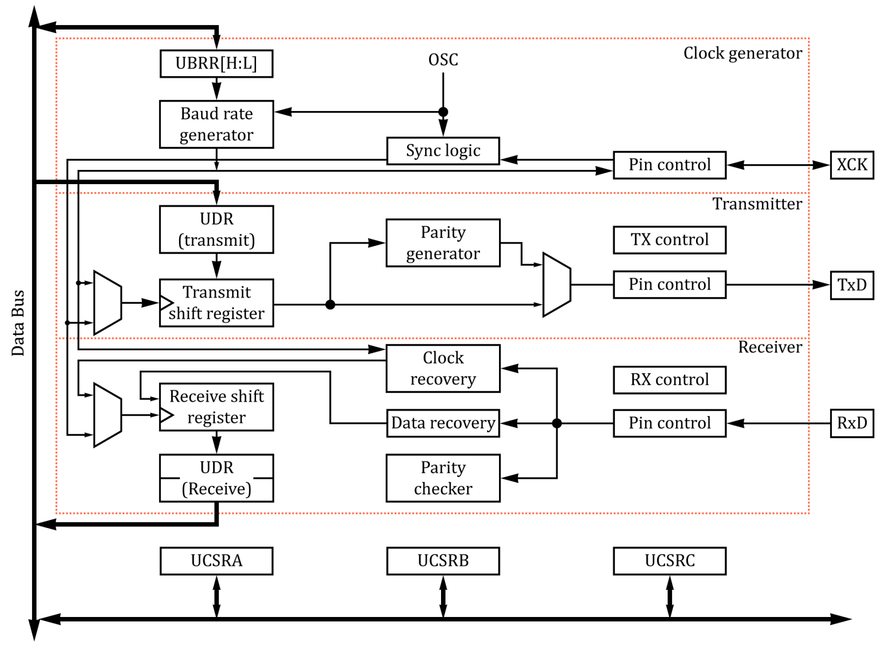
- *Serial transmission* - data is sent over one cable (or one pair), and individual bits of information are sent sequentially.
- Usually, this name is understood as the transmission developed for communication between a computer and an external modem (RS-232 standard).
- Serial transmission can be *synchronous* or *asynchronous*

Types of serial interfaces

- *RS-232* – standard serial interface, the standard has been designed for communication with the modem. Voltage levels must be converted in this standard.
- I^2C - The *Inter-Intergrated Circuit* interface means "intermediate between integrated circuits". The I^2C standard defines the two lowest layers of the OSI reference model: the physical layer and the data link layer.
- *SPI* - interface with capabilities and properties similar to I2C, with a larger one data transfer rates of up to several MB / s. This interface is built into many microcontrollers.
- D^2BUS - A Digital Data Bus interface developed by Philips, created to connect a small number of devices in a small area, enabling data transmission at the speed of $100\frac{kbit}{s}$ between devices separated by $150ms$ from each other. Allows you to address 4096 units.

- *CAN* - Controller Area Network (CAN) is a serial communication bus created in the 1980s at *Bosch GmbH* for applications in the automotive industry (ABS, engine control).
- *IEEE 1394* - FireWire is a serial link standard for synchronous communication. Developed in 1995 (by Apple Inc.) for personal computers and digital optical devices.
- *USB* (Universal Serial Bus) - An advanced serial interface, developed mainly for use in PCs, implementing the *plug and play* concept in relation to external devices. It can support up to 127 peripherals using a high data rate - $1.5 \frac{Mbit}{s}$ for reduced speed and $12 \frac{Mbit}{s}$ at full speed.

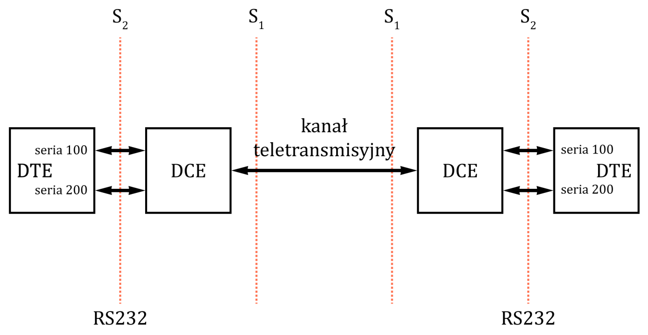
Serial interface USART



- Full Duplex Operation, synchronous and asynchronous transmission,
- 5, 6, 7, 8 or 9 bit data transmission with 1 or 2 stop bits, hardware parity bit, error detection transmission,
- Interrupts: TX Complete, TX Data Register Empty, RX Complete.

Interface *RS*232

- Originally, the RS232 standard was developed to unify the serial interface between the terminal and the *modem* in remote access installations,
- The RS232 interface is now also often used to directly connect various types of devices (computers with measuring devices, controllers, gsm phones, GPS receivers, etc.).
- The accepted terminology distinguishes between two types of cooperating devices:



  - DTE - Data Terminal Equipment,
  - DCE - Data Communication Equipment.

Main interface signals $RS232$

The $RS232$ connector can be found as DB9 and DB25.

Main signals *of the RS232 interface*:

- *TxD – Transmitted Data*
- *RxD – Received Data*
- *SG – Signal Ground* **- sets common reference potential for all lines signal,**

Other interface signals $RS232$

- *DSR - Data Set Ready* - DCE readiness - confirms the fact of establishing a connection,
- *DTR - Data Terminal Ready* - DTE ready - DTE ready to work with DCE,
- *RTS - Request To Send* - a request to send,
- *CTS - Clear To Send* - ready to send,
- *DCD - Data Carrier Detected* - the presence of a carrier wave in the transmission channel,
- *RI - Ring Indicator* - a signal issued by DCE (often unused),
- *PG - Protective Ground* - reduces the level of interference, in the DB-9 connector the role of the mass is played by a metal shield.

Interface voltage levels $RS232$

- On data lines (RxD, TxD), *negative logic* applies, i.e. logical "1" corresponds to low

logic	voltage
0	$+3V \leq U_{Rx,Tx} \leq +15V$
1	$-15V \leq U_{Rx,Tx} \leq -3V$

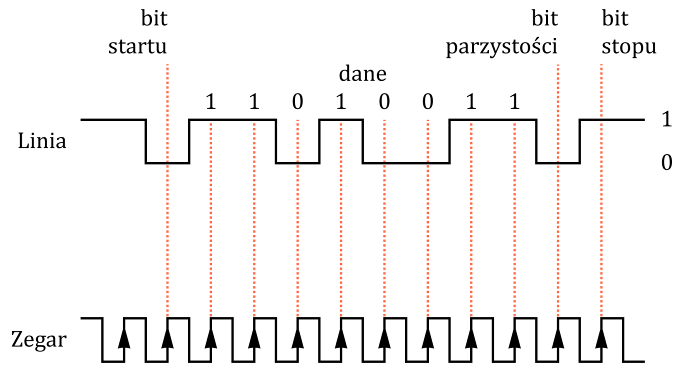
- The other lines use it positive logic

logic	voltage
0	$-15V \leq U_{Rx,Tx} \leq -3V$
1	$+3V \leq U_{Rx,Tx} \leq +15V$

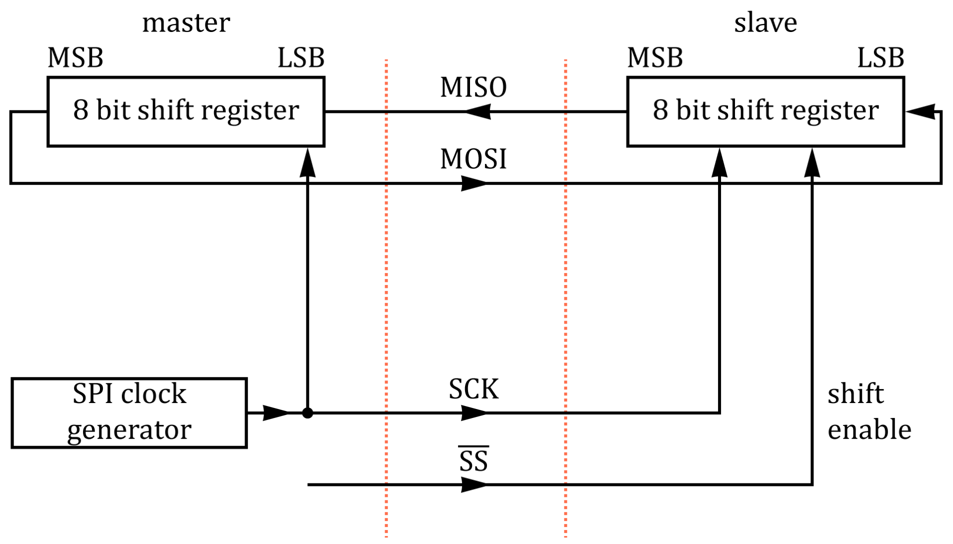
- The $< -3V, +3V >$ range does not specify the circuit status,
- **Do not connect the $RS232$ interface lines directly to TTL lines.** Specialized circuits - MAX232 adjusts voltage levels.

$RS232$ - synchronous and asynchronous transmission

- In *synchronous mode* the transmitter and receiver are clocked from the same clock signal source. Signal clock between DTE and DCE is sent directly on separate lines,
- In *asynchronous*, the transmitter and receiver use separate, independent signal generators clock. START and STOP are added to each data word to indicate when it starts and ends transmission.

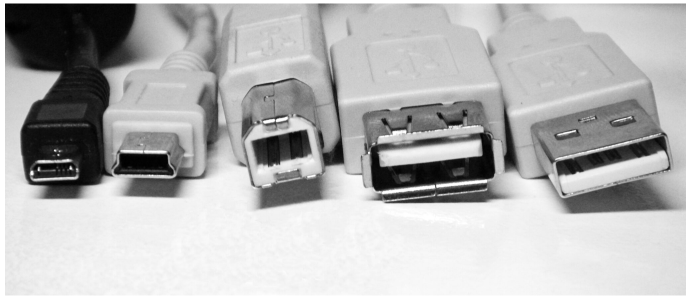


SPI - Serial Peripheral Interface



- Full-duplex, synchronous data transfer, 7 programmable transmission speeds,
- The ability to work in Master or Slave mode,
- End of transmission identified by an interrupt flag.

Universal Serial Bus USB 2.0



- The signal is sent using two lines $D-$ and $D+$,
- The $D+$ signal is negated on the $D-$ line - this doubles the signal amplitude,
- Data is sent in half-duplex mode,
- Derived power source $+5V$.

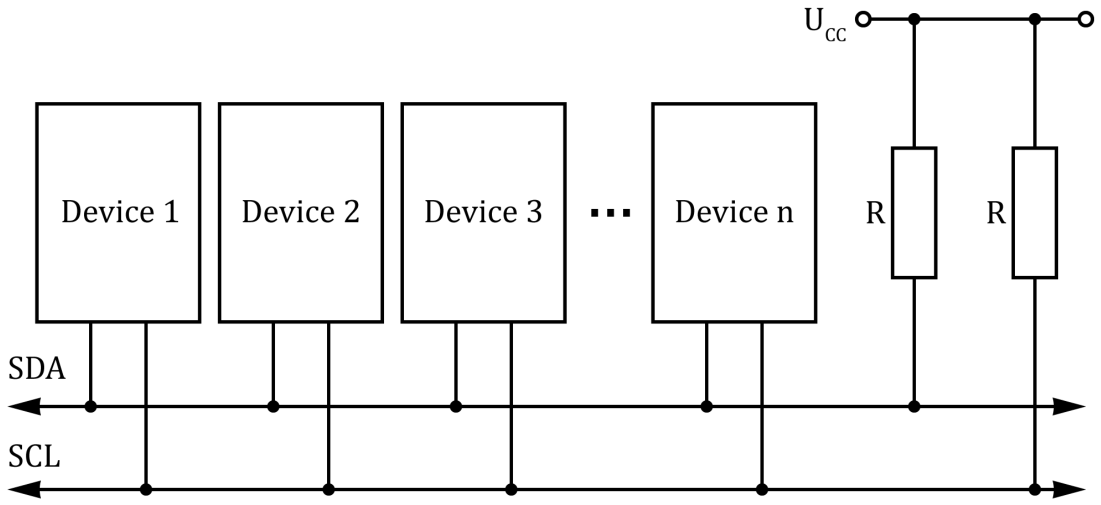
cable	color	description
V_{CC}	red	$+5V$
$D-$	white	data-
$D+$	green	data+
GRD	black	ground $0V$

USB packet transmission

There are five basic types of packets that are used for communication in USB transmission:

- *Handshake packets* - The answer consists only of an 8-bit PID field,
- *Start of Frame SOF* - Indicates the beginning of a transmission frame,
- *Token packets* - The announcement is sent by the computer to initiate the exchange,
- *Data packets* - Data can be sent by both the computer and the receiving device,
- *Special packets* - the *Start-split (SSPLIT) Token* and *Complete-split (CSPLIT) Transaction stand out Token.*

Serial interface - Two-wire

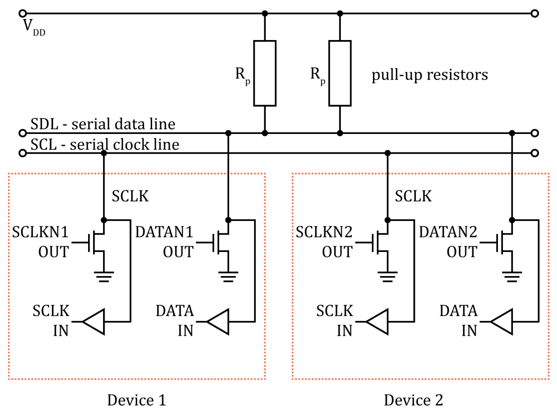


- Support for Master and Slave modes, Multi-master arbitration,
- 7-bit address (128 Slave addresses), fully programmable Slave address under General Call Support
- Transmission speed up to $400kHz$

Interface I^2C

- The standard was developed at the beginning of the 1980s (currently referred to as the standard operating mode) and was characterized by: 100 kbps transmission speed 7-bit address space
- In 1992, version 1.0 was developed: adding a mode with a transmission speed of 400 kbps (Fast Mode), extending the standard with the option of 10-bit addressing
- In 1998, version 2.0 was developed: adding High Speed Mode, allowing transmission speed of 3.4 Mbps. Increasing the voltage tolerance range in high condition: 2.3 - 5.5 V
- Standard I^2C specifies the two lowest layers of the OSI reference model: the physical layer and the data link layer.

Interface I^2C - physical layer



- I^2C uses two bidirectional lines for transmission:
  - *SDA* - Serial Data Line
  - *SCL* - Serial Clock Line.

  Both lines are permanently pulled up to the power source via pull-up resistors,
- I^2C uses positive logic, so the low state on the bus corresponds to “0” logic, while the high state of “1” logical.
- All transmitters are of the open collector type.

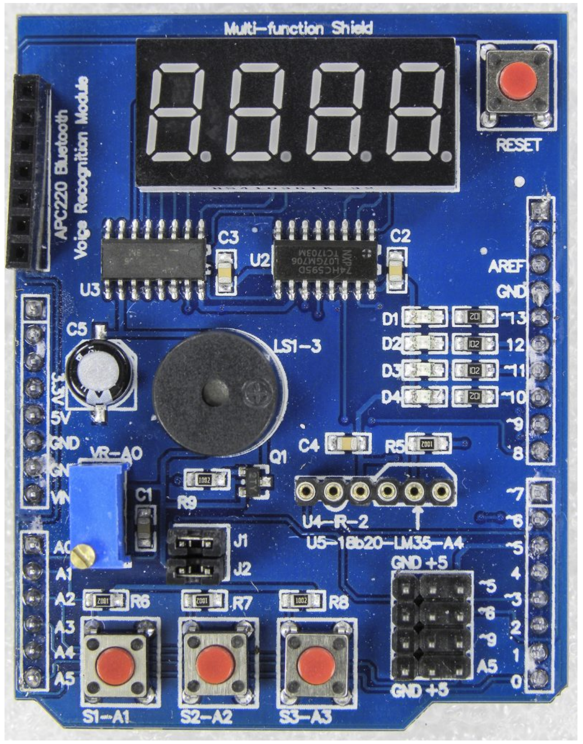
- The basic version of I^2C assumes that there is only one device that it can initialize transmission (master),
- *Collision detection mechanism*, it is possible to work in multi-master mode. Because the data they are transmitted in order from the oldest bit to the youngest, in the case of simultaneous sending, the device transmitting the address with the higher number will withdraw first.
- *Arbitration* with a constant allocation of priorities. Devices with lower addresses have higher priority.
- The change on the data line during transmission can only occur when the clock line is found low.
- Start bit occurs when the data line changes its state from “1” to “0”.
- After the transmission, a stop bit is generated, i.e. the data line goes into state high at high clock lines.
- The length of the line is limited only by its maximum capacity, which is 400 pF.

Interface I^2C - data link layer

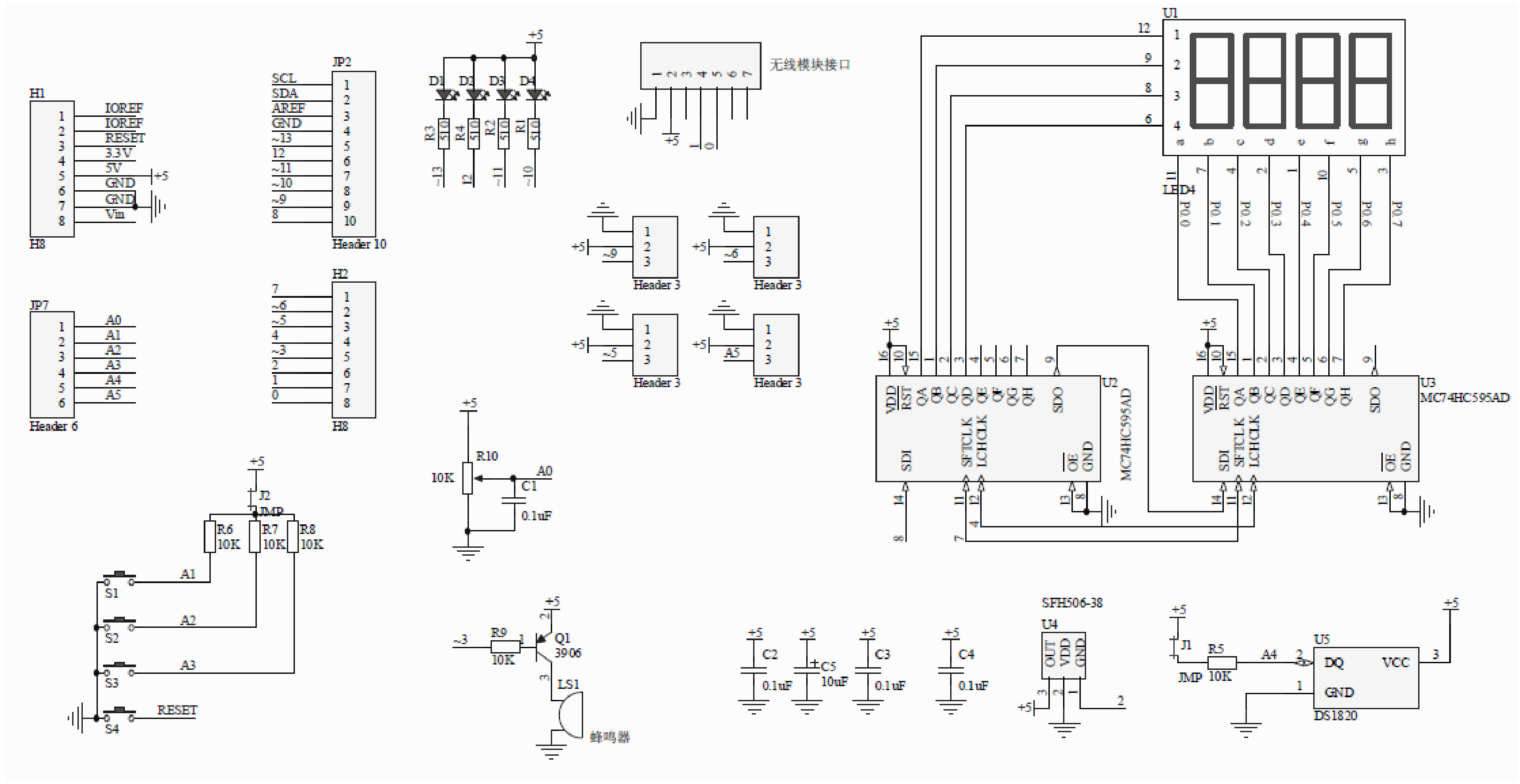
- I^2C is a byte oriented bus (bits grouped at 8),
- After sending 8 bits, an additional ACK confirmation confirmation bit is sent,
- The first byte is always the slave device address assigned by the master device, which in addition to the 7 bits of the correct address contains the transmission direction bit (at the youngest position),
- The value “0” of this bit means transmission from master to slave (write), while the value “1” opposite direction (read). After the first byte, data is sent,
- Standard assumed a 7-bit address space, i.e. the ability to address up to 128 devices (in practice 112 devices),
- One of the addresses reserved is the so-called General call (address 0), which sends data to all devices,

Multifunctional training module

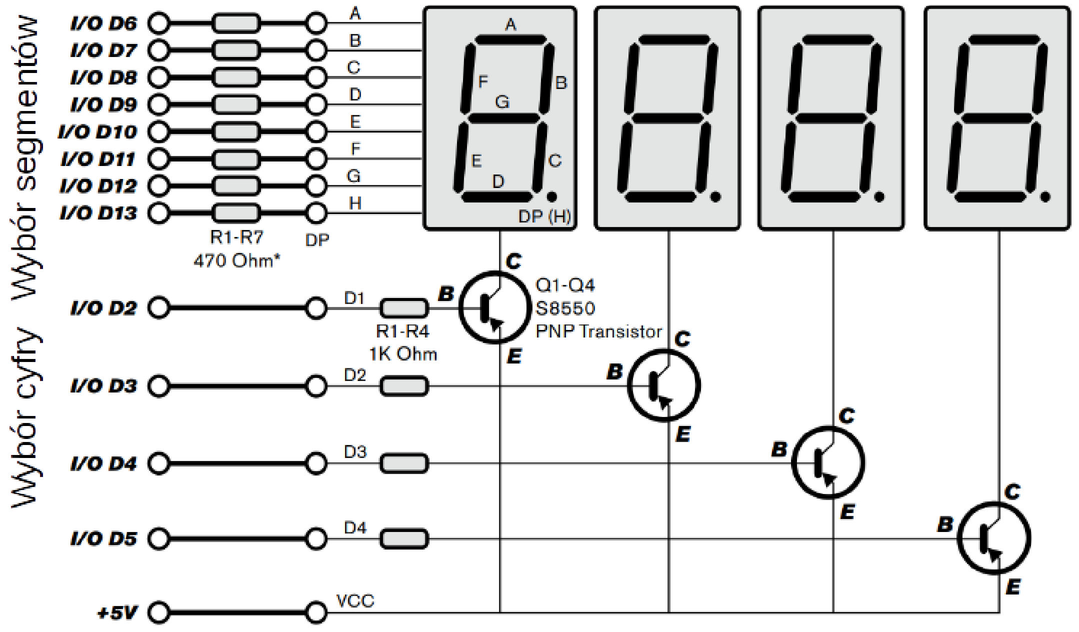
- A0 - analog input from the potentiometer,
- A1, A2, A3 - button inputs (with pull-up resistors)
- 10, 11, 12, 13 diodes
- A4, A5 - i2c input
- 5, 6, 8, A5 pins (VCC + GND)
- 2 - sensor button (int0)
- 3 buzzer
- 4 LATCH (register)
- 7 CLK (register)
- 8 DIO (register)



Multifunctional training module - diagram



LED display



- first register - selection of the displayed digit,
- second register - type of displayed digit

SPI - Writing the digit to the display

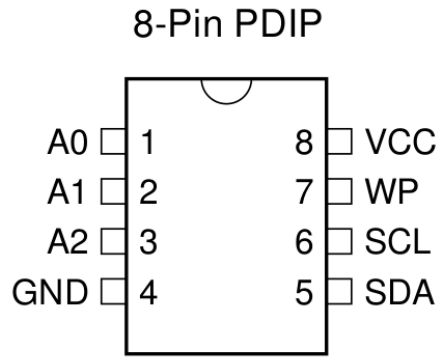
```
#define LATCH_DIO 4
#define CLK_DIO 7
#define DATA_DIO 8
 const byte SEGMENT_MAP[] = {0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0X80,0X90};
 const byte SEGMENT_SELECT[] = {0xF1,0xF2,0xF4,0xF8};


 digitalWrite(LATCH_DIO,LOW);
 shiftOut(DATA_DIO, CLK_DIO, MSBFIRST, SEGMENT_MAP[Value]);
 shiftOut(DATA_DIO, CLK_DIO, MSBFIRST, SEGMENT_SELECT[Segment] );
 digitalWrite(LATCH_DIO,HIGH); ;
```

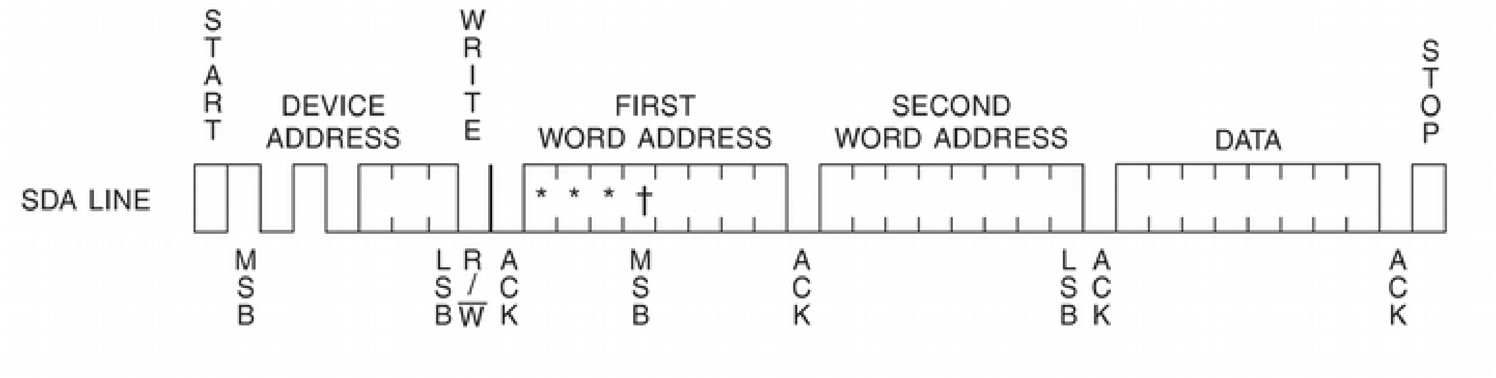
I2C memory chip

1	0	1	0	A_2	A_1	A_0	R/W
MSB							LSB

Pin Name	Function
A0 - A2	Address Inputs
SDA	Serial Data
SCL	Serial Clock Input
WP	Write Protect

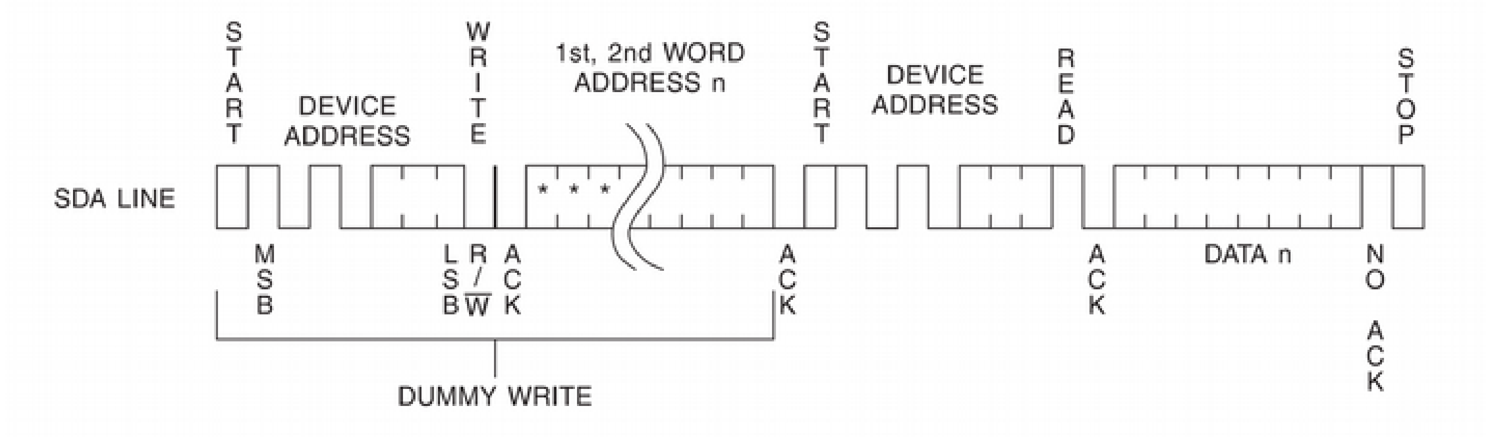


I2C - Writing byte to memory



```
Wire.beginTransmission(deviceaddress);
Wire.write(eeaddress >> 8);    // MSB
Wire.write(eeaddress & 0xFF);  // LSB
Wire.write(data);
Wire.endTransmission();
```

I2C - Reading byte from memory (Random Read)



```
Wire.beginTransmission(deviceaddress); //dummy write
Wire.write(eeaddress >> 8);    // MSB
Wire.write(eeaddress & 0xFF); // LSB
Wire.endTransmission();

Wire.requestFrom(deviceaddress,1);
if (Wire.available()) byte rdata = Wire.read();
```

I2C - Byte reading from a thermometer DS3231

```
byte displayTemperature()
{
    byte data = 0xFF;

   Wire.beginTransmission(0x68);
   Wire.write(0x11); // the integer portion
   Wire.endTransmission();

   Wire.requestFrom(0x68, 1);
    if (Wire.available()) { data = Wire.read();}

   return data;
}
```

Task for labs

1. SPI (shift register) - Write a program that displays "HELPón the display
2. I2C - Write a program that reads from the thermometer (device address 0x68, register address 0x11) the numerical value of temperature. Display the result on the display.
3. I2C - Write a program reading the temperature from the thermometer with fractional values (device address 0x68, register address 0x11, register address of fractional part 0x12). Display the result with a decimal point on the display.