Integrated and external peripherals in microprocessor systems, part 3
Lecture 8

Semester 20L – Summer 2020

© Maciej Urbanski, MSc

email: M.Urbanski@elka.pw.edu.pl

Important remark

This material is intended to be used by the students during the Microprocessor Systems course for educational purposes only. The course is conducted in the Faculty of Electronics, Warsaw University of Technology.

The use of this material in any other purpose than education is prohibited.

This material has been prepared based on many sources, considered by the author as valueable, however it is possible that the material contains errors and misstatements.

The author takes no responsibility for the usage of this material and any potential losses this usage can lead to. Furthermore the author will be very grateful for pointing out any errors found and also for any other useful remarks on the course material and potential upgrades.

Test 1 will be on 29th April.

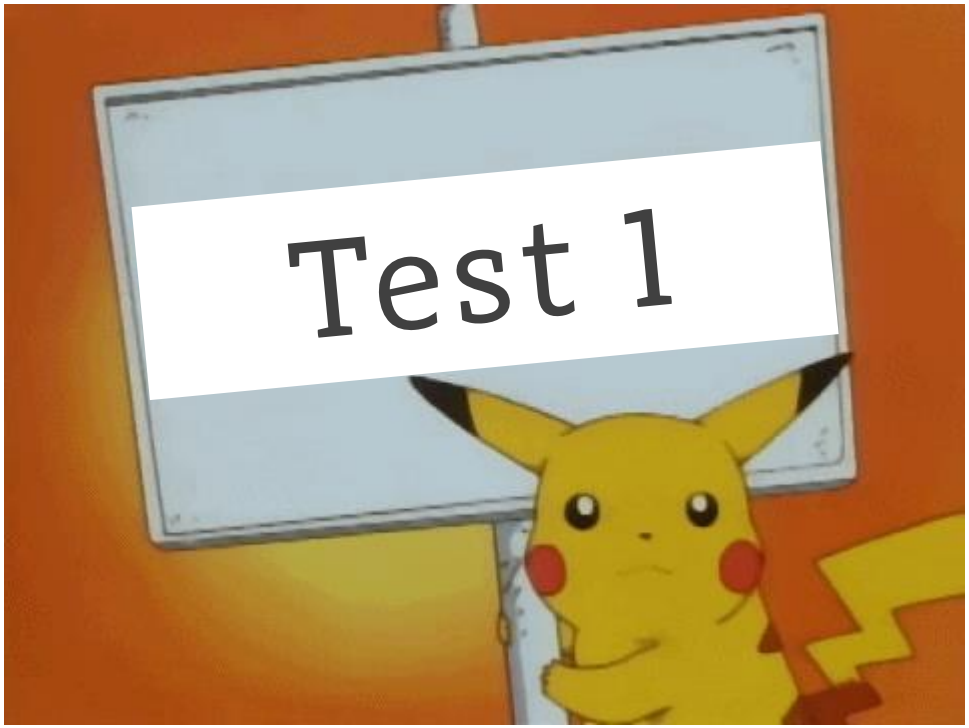
It will be done remotely – the final technical form has not yet been decided.

It will cover lecture material only, up to (and including) Lecture 6.

Sample test is provided in Studia server.

The test will consist of two parts – theoretical questions (5 questions for 10 points total) and one design task for 15 points total.

Maximum score is therefore 25 points.



Sequential operation of microcontroller

- It has already been mentioned several times – microcontroller is a sequential device
- It performs operations according to internal/external clock signal
- Operations are executed one by one
- But what if we want to react on some unexpected situation or event?

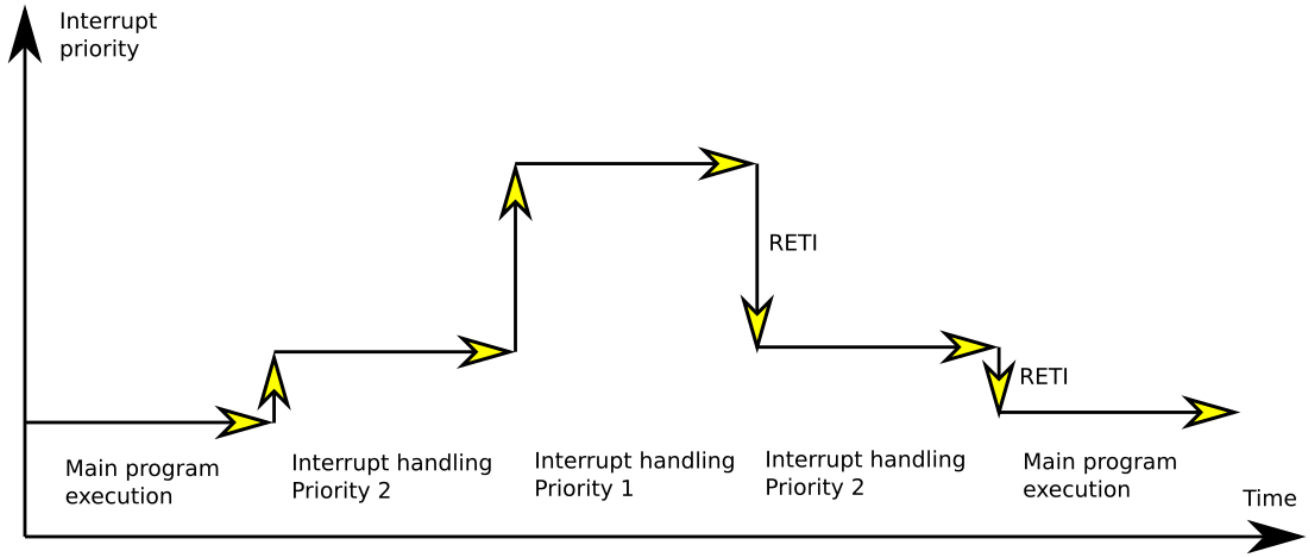
- The answer is – we have to configure and use interrupts and interrupt controller
- In response to certain internal or external signals microcontroller can temporarily stop the execution of current operation and starts so called interrupt procedure.
- After completion of interrupt procedure the interrupted operation is restored and continued.
- Each interrupt has its vector address – it stores the address of interrupt procedure
- Interrupts are handles by the interrupt controller

Interrupts – why? Interrupts priority

- Interrupts were introduced to allow hardware to control events.
- Event can be timer overflow, pressing a switch, resetting the microcontroller, etc.

- Software handling of events (polling) is very inefective:
  - It consumes lots of microprocessor resources
  - It blocks the execution of other operations
  - It is slow

- Interrupts are not equal and can be sorted using their priorities.
- The higher the priority the more important the interrupt
- Higher priority interrupt are executed first
- The highest priority interrupt (in 8051) is RESET, with vector address of 0x00



Interrupts in 8051

- 8051 architecture handles 5 interrupt sources:
  - 2 internal interrupts for timers T0 and T1 (not mentioning T2 in 8052 variant)
  - 2 external interrupts INT0 and INT1
    - They are handled when level or edge is provided on P3.2 or P3.3 pins
  - 1 serial interrupt for serial communication controller

- Priority table for 8051 interrputs is given below:

Interrupt	Priority level
RESET	0
INT0	1
INT1	3
TF0	2
TF1	4
TF2	6
SERIAL RI/TI	5

Interrupts in 8051 - configuration

- To enable and control interrupts special function register IE is used
- Interrupt priority can be configured using SFR IP register.
- To control external interrupts the four lower bits of TCON register are used.

IE register

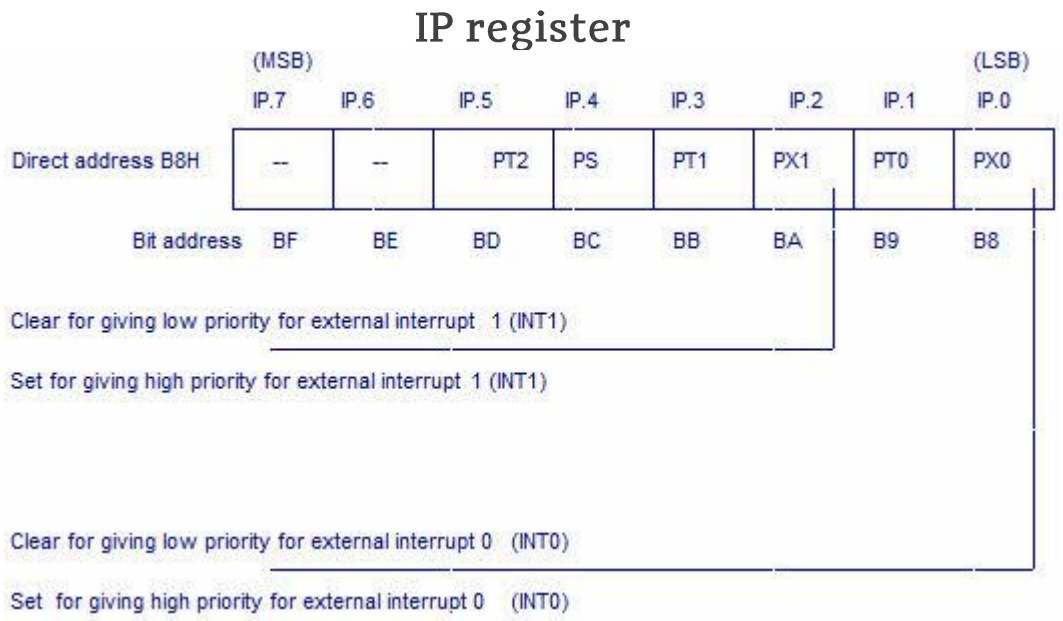
EA	–	–	ES	ET1	EX1	ET0	EX0

EA	IE.7	Disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, interrupt source is individually enable or disabled by setting or clearing its enable bit.
-	IE.6	Not implemented, reserved for future use*.
-	IE.5	Not implemented, reserved for future use*.
ES	IE.4	Enable or disable the Serial port interrupt.
ET1	IE.3	Enable or disable the Timer 1 overflow interrupt.
EX1	IE.2	Enable or disable External interrupt 1.
ET0	IE.1	Enable or disable the Timer 0 overflow interrupt.
EX0	IE.0	Enable or disable External Interrupt 0.

Image source: https://www.elprocus.com/types-of-interrupts-in-8051-microcontroller-and-interrupt-programming/

Interrupts in 8051 - configuration

- To enable and control interrupts special function register IE is used
- Interrupt priority can be configured using SFR IP register.
- To control external interrupts the four lower bits of TCON register are used.



- PX0 – external interrupt, PT0 – Timer0, PX1 – external interrupt, PT1 – Timer 1,
- PS – serial port interrupt, PT2 – Timer2, IP6 and IP7 reserved for future use

Image source: https://www.elprocus.com/types-of-interrupts-in-8051-microcontroller-and-interrupt-programming/

Interrupts in 8051 - configuration

- To enable and control interrupts special function register IE is used
- Interrupt priority can be configured using SFR IP register.
- To control external interrupts the four lower bits of TCON register are used.

TCON register

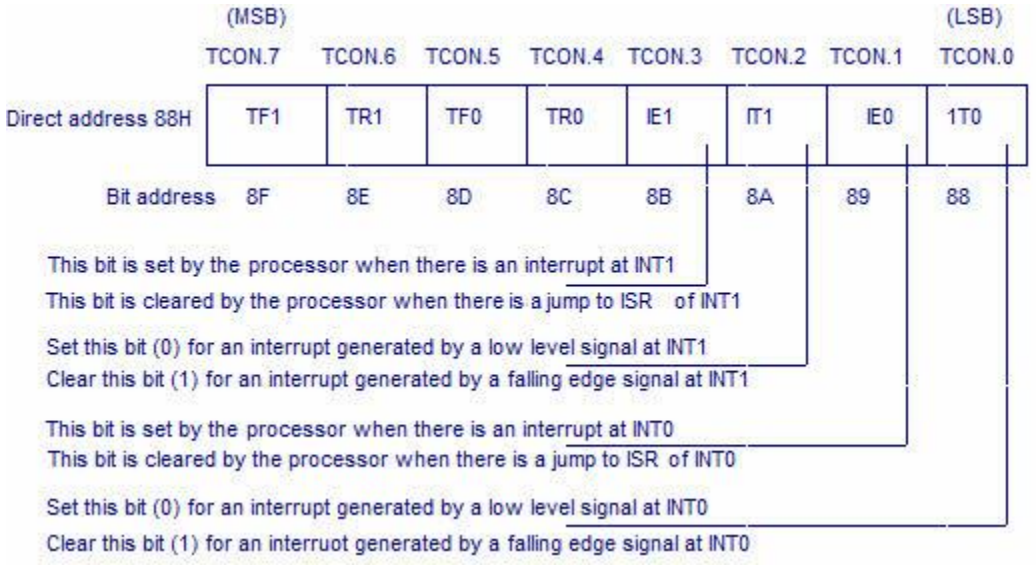


Image source: https://www.elprocus.com/types-of-interrupts-in-8051-microcontroller-and-interrupt-programming/

Interrupt controller in 8051

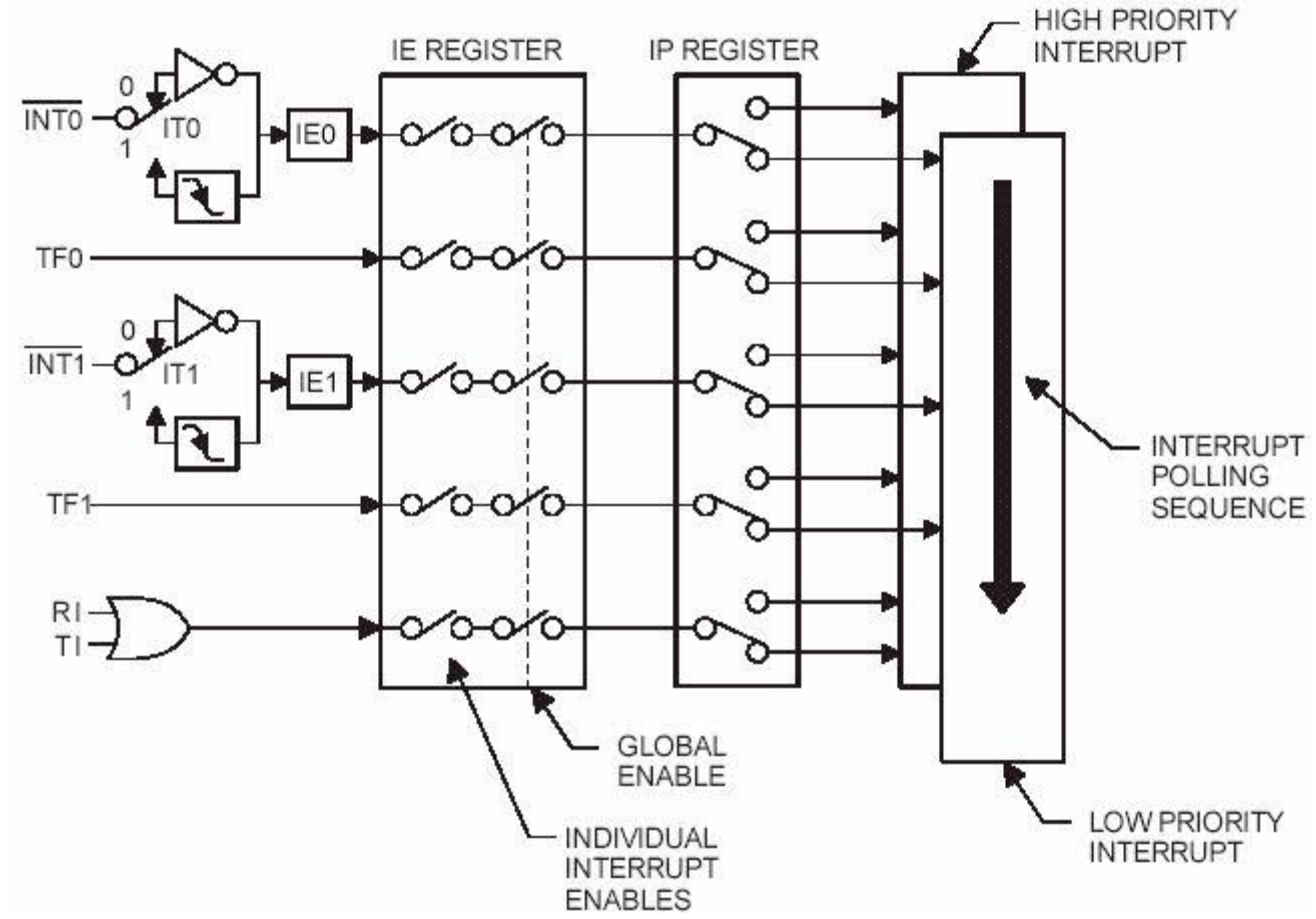


Image source: https://www.elprocus.com/types-of-interrupts-in-8051-microcontroller-and-interrupt-programming/

Interrupts in 8051 – example of T2 timer interrupt configuration (Lab 2)

```
;Displaying one letter with 7 segment LED display. M. Urbanski 2019
ORG 8000h               //start memory write at 8000H
JMP START               //go to initialization procedure
ORG 802BH               //start the memory write at 802BH (include JMP START command)
T2_HANDLER:             //handler for T2 interrupt
            MOV P1, #04H            //turn off displays anodes
            JMP DISPLAY1            //start DISPLAY1 subroutine
            CLR 0C8H+7              //clear TF2 interrupt flag - overload T2
            POP DPL                 //read the DPL value from the stack and store it in DPTR
            POP DPH                 //read the DPH value from the stack and store it in DPTR
            POP Acc                 //read the ACC value from the stack and store it in ACC
            RETI                    //quit the subroutine and go back (return address in the stack)
DISPLAY1:                           //subroutine for displaying something in display 1 (first to the left)
            MOV DPTR, #0F001H       //store the data for the displays in 0F001H register (anodes).
                                    //This way the CS3 will be strobed automatically
            MOV A, #01H             //turn on display 1
            MOVX @DPTR, A           //copy to the address stored in DPTR value stored in ACC
                                    //write to 0F001H value 01H
            MOV DPTR, #0F000H       //send data to the cathodes register
            MOV A, #00011100B       //send letter code
            MOVX @DPTR, A           //send it to the register
            MOV P1, #00H            //turn on all the displays
            RETI                    //quit the subroutine
START:                              //initialization subroutine
            MOV P1, #04H            //turn off all the anodes
            MOV DPTR,#0E000H        //use the 0E000H register - CS3 config register
            MOV A,#08H  //prepare CS3 command - update CS3 with strobing RD/WR
                                    //according to lab instruction
            MOVX @DPTR, A           //send the command to the register
            MOV 0C8H, #0            //write zero to TF2 register of T2
            MOV 0CBH, #0FDH         //write value to RCAP2H - set the switching frequency
                                    //to around 1.8 kHz
            SETB 0A8H+5             //turn on T2 interrupts
            SETB EA                 //turn on interrupts
            SETB 0C8H+2             //start T2 counter
LOOP:
            JMP LOOP                //infinite loop
END
```

Watchdog

- Watchdog is a special timer (internal or external) that can detect if something went wrong with the microprocessor system.
- It is a counter that causes a system reset when overflow occurs. Therefore during normal operation it has to be zeroed repeateadly
- If the counter is not zeroed then it is able to count up to overflow and then cause the system reset – this is useful when the microcontroller code execution hangs – then it is not possible to zero watchdog
- Watchdog circuits are implemented in most modern microcontrollers, but not in 8051

This is not a watchdog...

But it's still cute...

Image source: http://www.hellokids.com/c_28240/drawing-for-kids/drawing-tutorials-step-by-step/animals/how-to-draw-a-simple-puppy

Communication buses and standards in microprocessor systems

- Microprocessor system consists of several components, including microcontroller/microprocessor and its peripheral circuits.
- There is a need to communicate with peripheral circuits
  - Microcontroller internal peripherals are routed inside the chip and connected using internal buses
  - External peripheral circuits, in separate chips, have to be connected using a standarized way that will ensure proper transmission parameters
- There is a need to communicate with other microprocessor systems
- There is a need to communicate with external devices – PCs, terminals, etc.
- For these purpose various communication buses and standards have been developed.
- Some of them (most commonly used in digital systems) will be presented in general
  - For more detail refer to linked websites or other literature

Communication buses and standards in microprocessor systems

- Communication buses can be divided into several groups, based on parameters:
  - Parallel buses
    - Few bits transmitted in the same time
    - Fast
    - Require sophisticated hardware – drivers, multi track buses, etc.
    - Examples:
      - Communication using LPT port in older PCs
      - Connection between microcontroller and HD44780 LCD display driver
  - Serial buses
    - One bit at a time
    - Theoretically slower than parallel buses, nowadays they can be also fast
      - But they require much higher clock frequency to maintain baudrate
    - Simpler to develop in hardware
      - Simple drivers and buses
    - Examples:
      - RS232 communication
      - I2C communication
      - SPI communication
  - Synchronous transmission – clock is sent with data, usually with separate wire
  - Asynchronous transmission – clock is not sent, data has to start and end with predefined sequence of bits

I2C

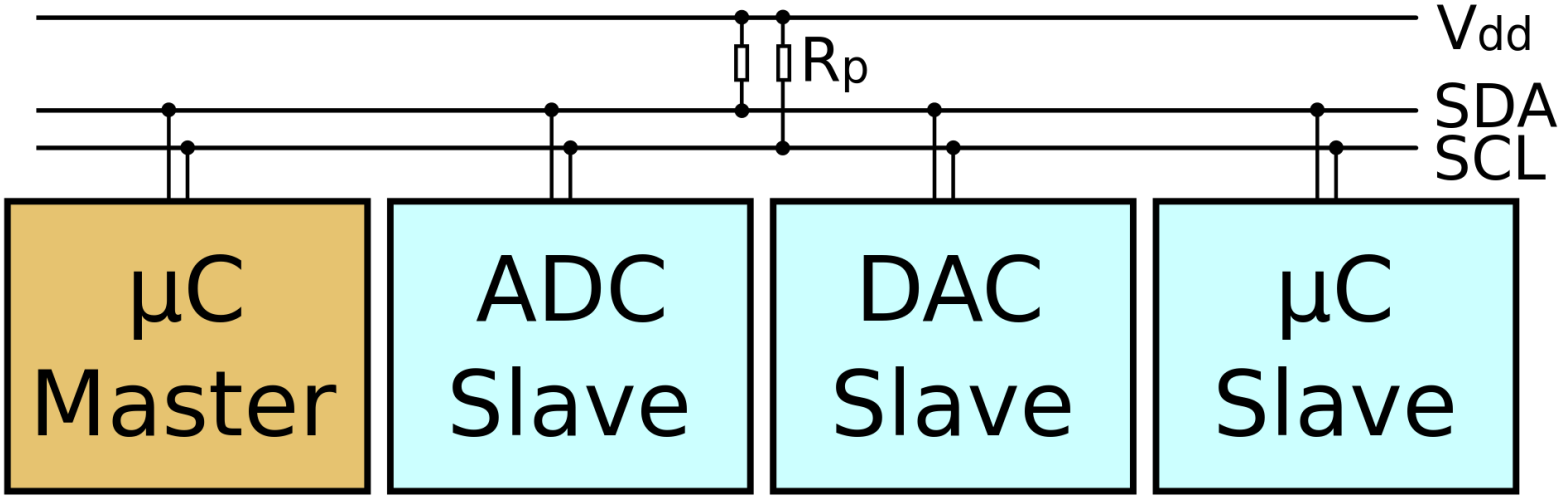
- I2C – Inter-Integrated Circuit – it is a synchronous multi-master multi-slave serial bus invented in 80s by Phillips.

- Used for low-speed communication between peripheral circuits in microprocessor systems, like
  - EEPROM memories
  - Thermal sensors
  - GPIO expand ports
  - Etc.

- Additional devices can be connected to the bus without affecting already present modules

- Only two (three, including GND) connections are needed:
  - SCL – clock signal
  - SDA – data signal

I2C – configuration and pullup resistors



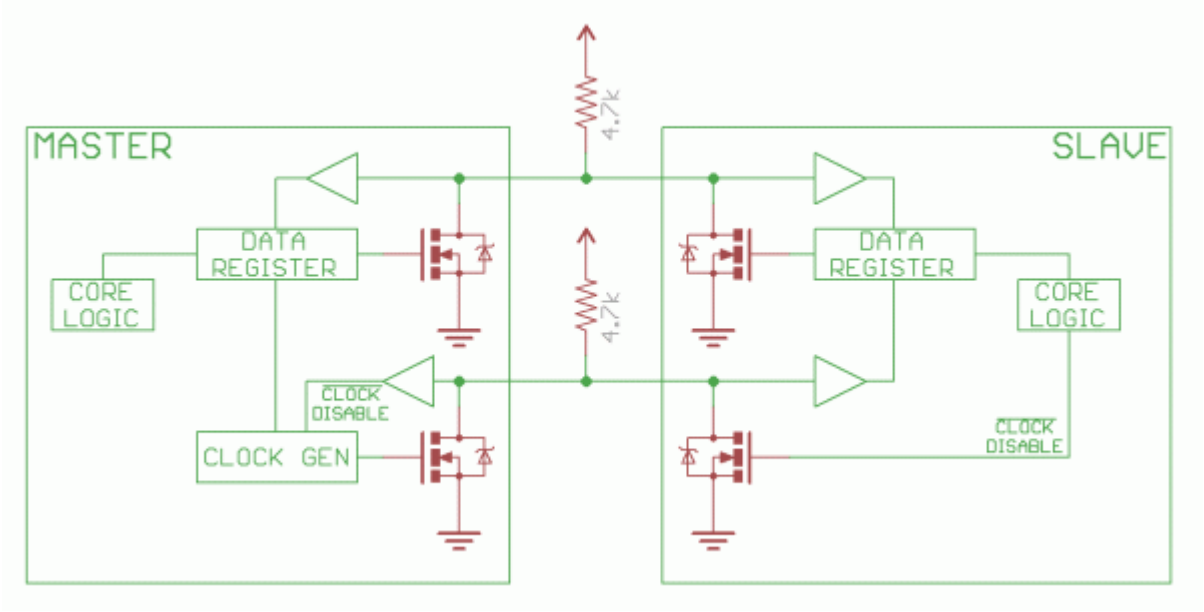
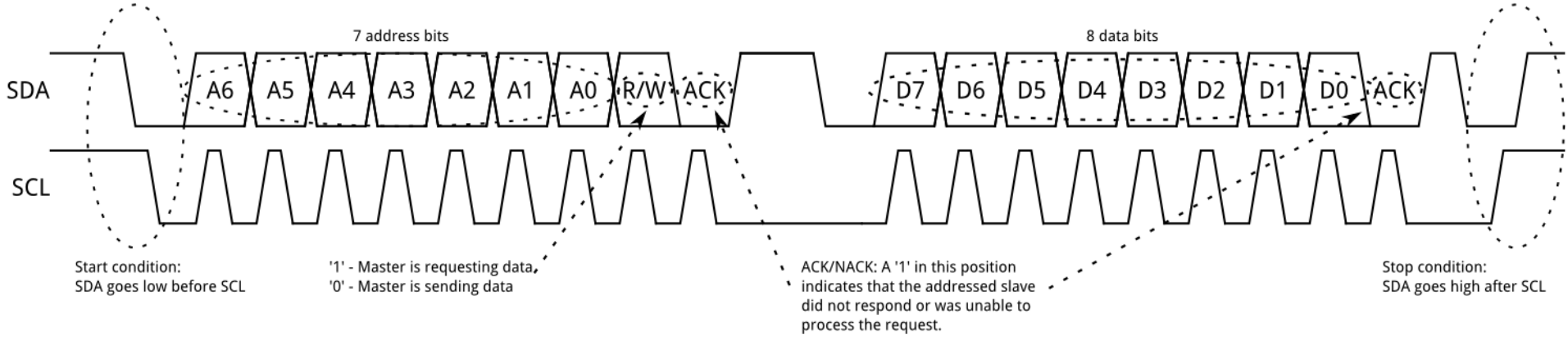


Image source: https://learn.sparkfun.com/tutorials/i2c/all http://en.wikipedia.org

I2C – timing and transmission

- 7 bit addressing



- 10 bit addressing

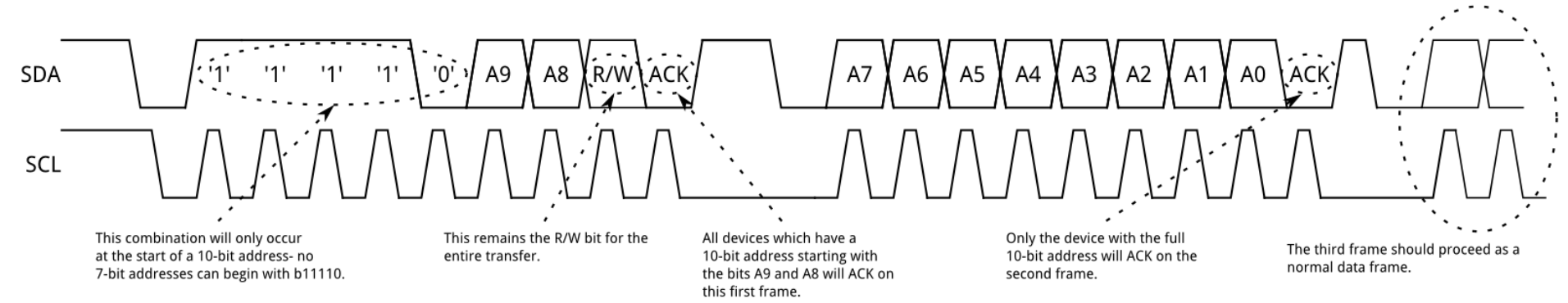
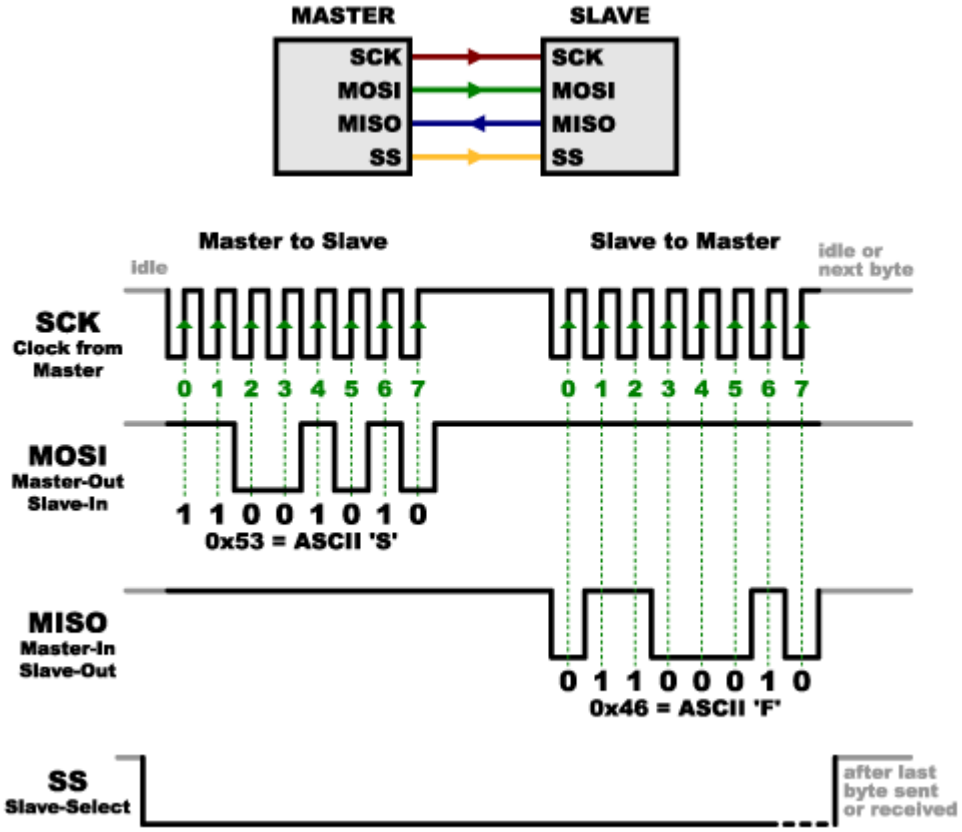


Image source: https://learn.sparkfun.com/tutorials/i2c/all

I2C – timing

- Each transmission is started with **START indicator – pulling data low, when clock is high**
- Each transmission is stopped with **STOP indicator – pulling data high, when clock is high**
- The amount of bytes between the start and stop indicators is not limited, but is defined by the transmitter or receiver – each peripheral device has it's own way of communicating via I2C
- Data is sent from MSB to LSB
- Typical transmission starts with:
  - START indicator
  - Slave address (7 bit), ended with 0 or 1 as eight bit, depending whether we want to read from slave or write to slave
  - Data bytes followed by ACK (after each byte)
  - END indicator

SPI – Serial Peripheral Interface

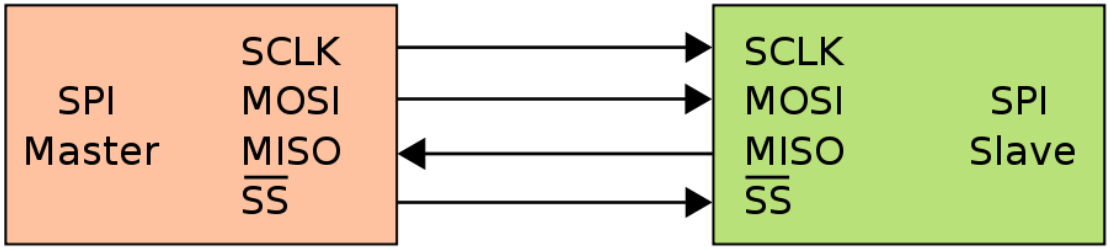


- SPI is a synchronous serial communication interface specification to be used in embedded systems, like microprocessor systems
- Communication is established using 4 wire (5 with ground) bus:
  - SCK – clock for the bus
  - MOSI – Master Out Slave In
  - MISO – Master In Slave Out
  - SS, or CS – chip select
    - Used to turn on/off SPI device
- Master is a device that generates clock signal
- Unlike I2C it can be quite fast

- SPI can operate in Full Duplex mode – master device simultaneously transmits data to a slave and receives data from a slave.
- In Half Duplex mode SPI master device only receives data from a slave or sends to slave

SPI – configuration

- SPI driver should allow to configure:
- Clock polarity – CPOL – clock that idles at logic 0
- Clock phase – CPHA – determines the timing of the data bits relative to clock pulses
- SPI receivers may have the bus configured in different ways and it's always required to check the datasheet for proper configuration



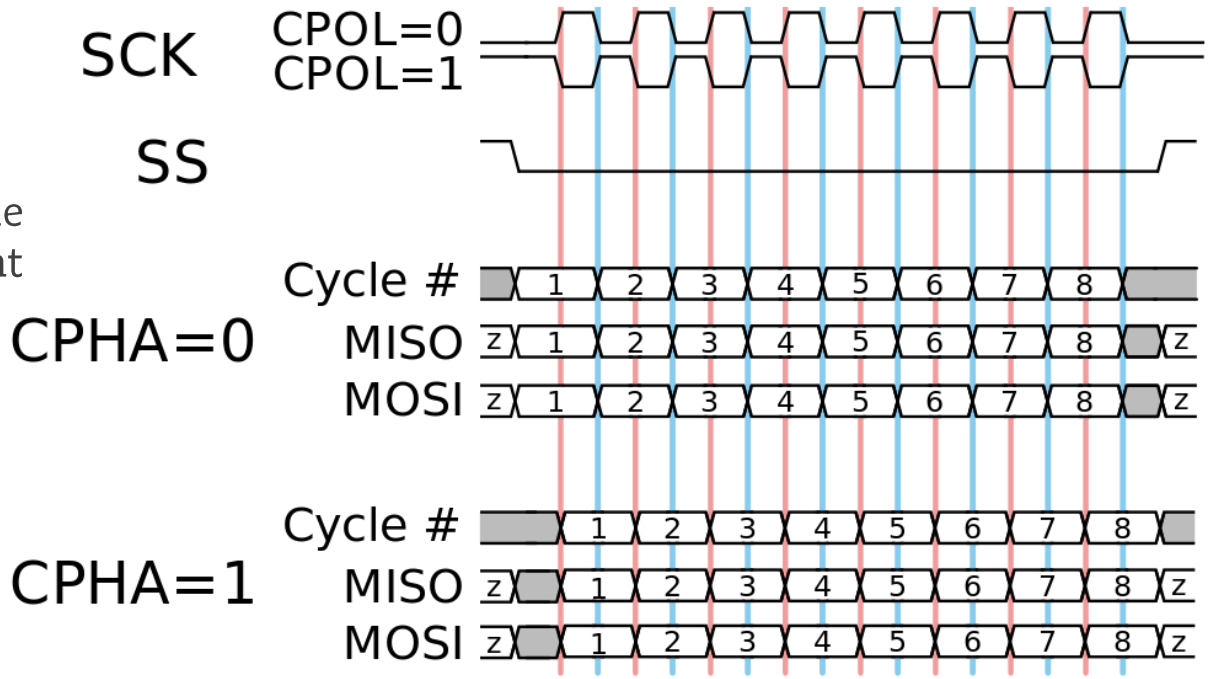


 Image source: en.wikipedia.org

SPI – multiple slave configuration

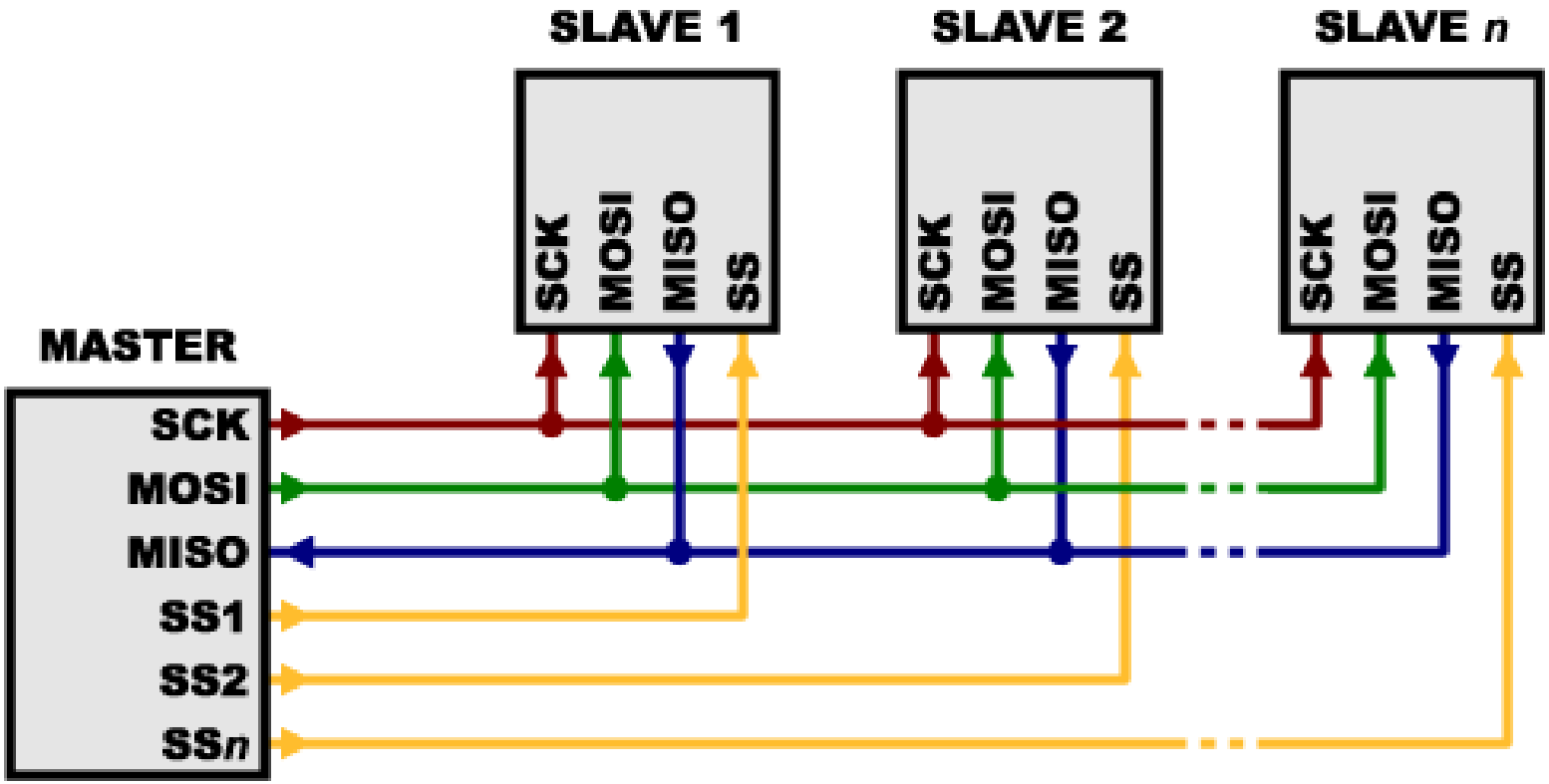


Image source: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all

SPI – transmission in Full Duplex mode

- When SS goes low the master starts generating SCK signal. When CPOL=CPHA=0 data is valid on rising edge of SCK and they are changing on falling edge.

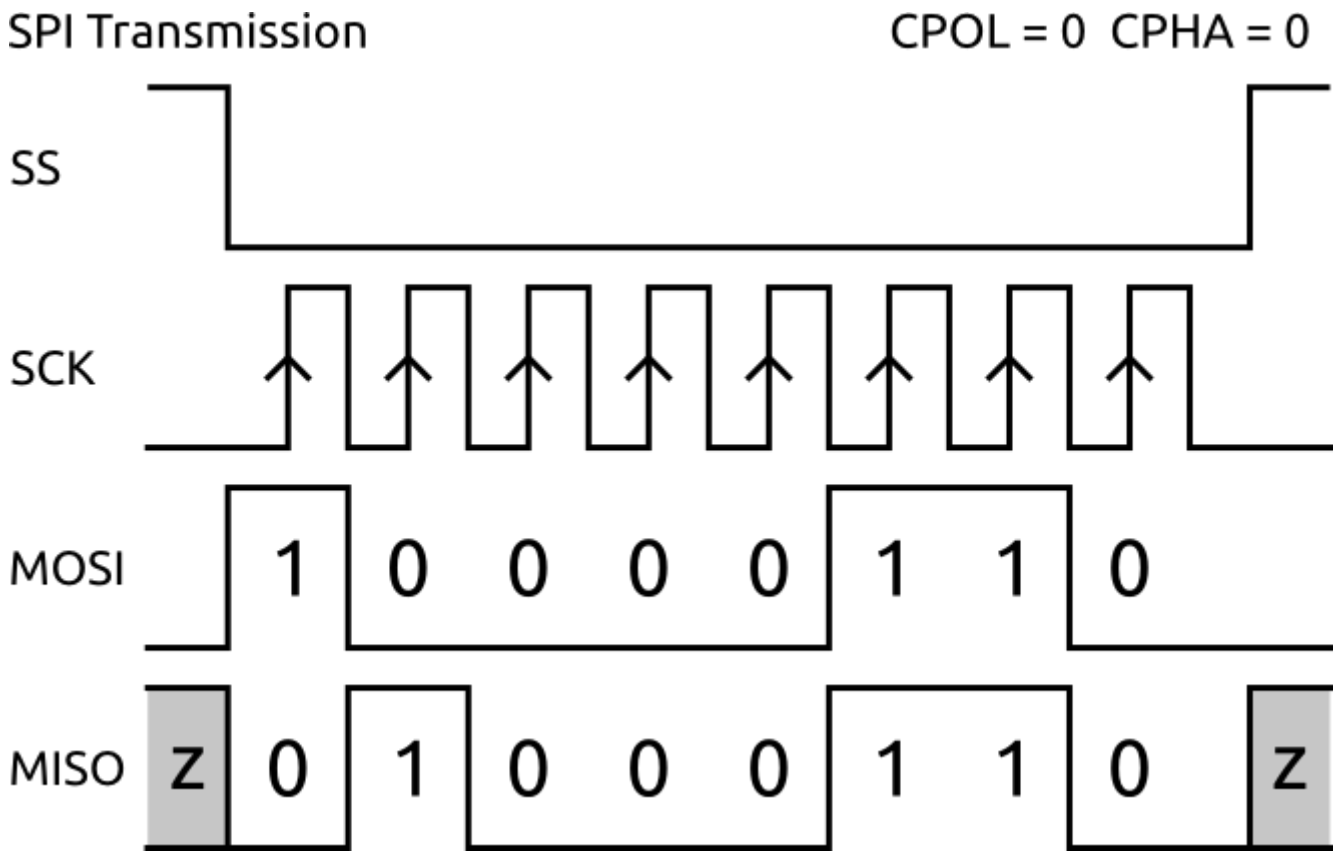
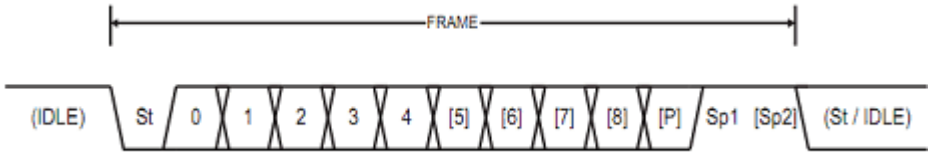


Image source: https://alchitry.com/blogs/tutorials/serial-peripheral-interface-spi

UART and RS232

- UART – Universal Asynchronous Receiver – Transmitter – is a computer hardware device for asynchronous serial communication in which the data format and transmission speed are configurable.
- USART device allows also for synchronous operation
- As there is no clock sent in UART there is a need to send synchronization information in data – it has to start and stop with predefined sequence that can be detected and used for synchronization



- RS-232 is a standard for serial communication and data transmission. Despite the fact it is old and obsolete it is still used in industrial machines, networking equipment and scientific instruments.
  - Logic one is voltage in range from -15V to -3V
  - Logic zero is voltage in range from +3V to +15V

Image source: forbot.pl

UART in 8051

- Serial communication in 8051 is configured with:
  - PCON register
  - SCON – serial control register
  - TCON – timer control register for baudrate generator
  - TMOD – timer mode control for baudrate generator
  - SBUFF – serial buffer that holds the data to be transmitted and data received

- To connect two devices and make them use UART only two wires are needed.

- If UART-RS232 converter is needed it can be connected like shown. MAX232 is classic converter chip from Maxim that allows to convert between UART (0-5V) to RS-232 voltage range.

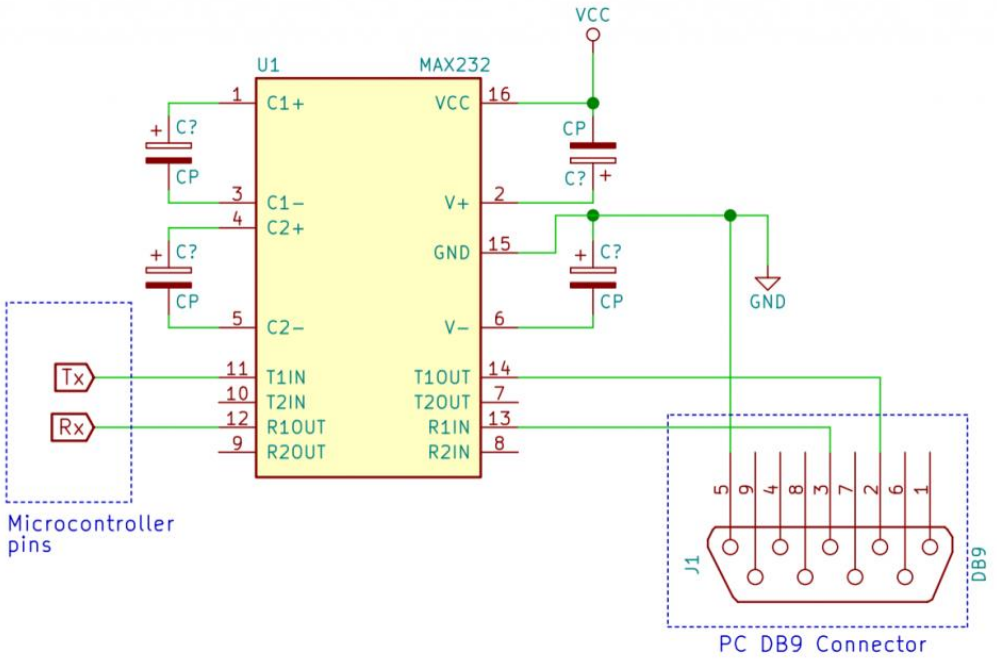


Image source: https://openlabpro.com/guide/uart-8051/

USB (overview)

- USB – Universal Serial Bus – most popular standard for serial communication and power supply.
- There are three generations of USB: USB 1.x, USB 2.0 and USB 3.x
- Originally it was designed to standarize connection of peripherals – video cameras, external disks, keyboards, etc.
- Full description of USB standard is not a purpose of this lecture and course (too much complicated)
- During this course we will focus only on the aspect of using USB-UART converters that will establish USB connection with microprocessor system.

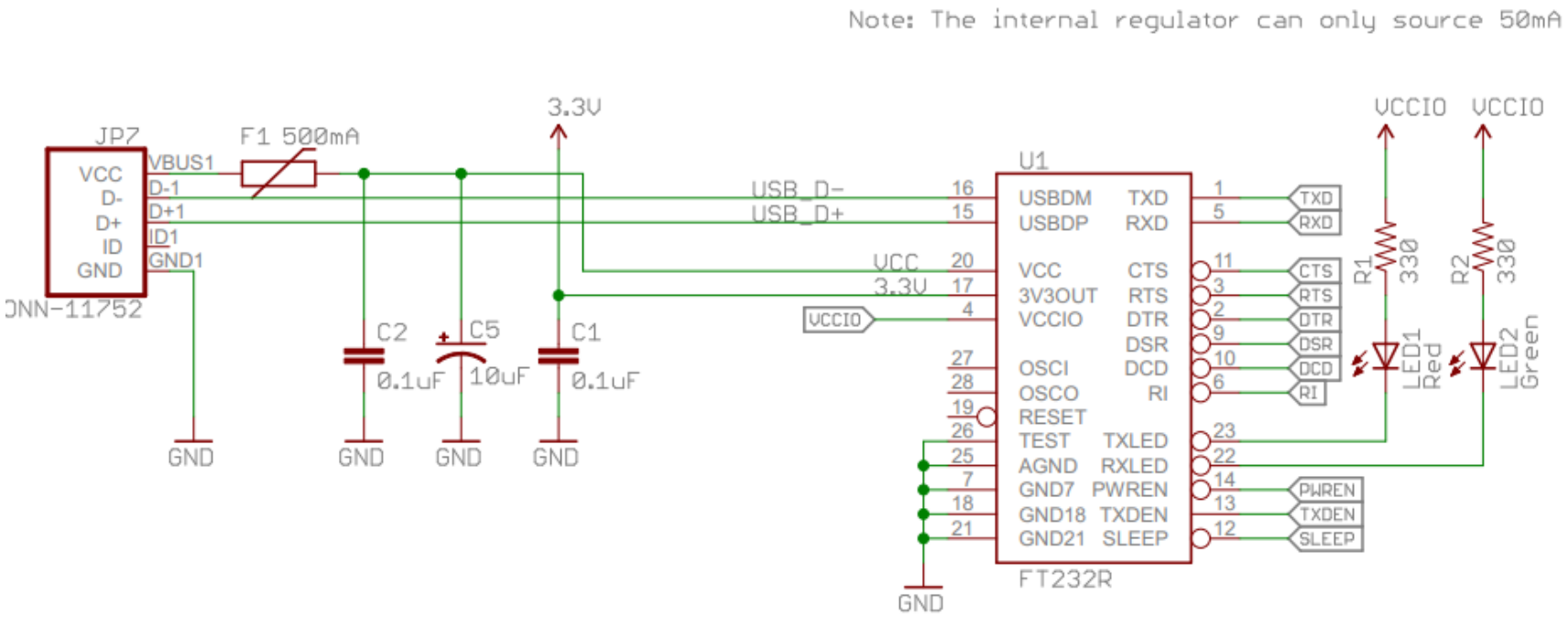
Image source: botland.com.pl

USB – FT232RL, MC2200, etc.

- In most cases we do not have to use all the features provided by the USB standard.
- In microprocessor systems we can establish USB communication using special USB-UART chip (or internal converter, like STM32 USB-VCP)
- In this mode USB driver acts as a converter between USB and UART, therefore allowing to be interfaced with microcontroller via standard UART interface (available even in 8051)
- The most popular USB-UART chip is FT232 by FTDI, but there are lots of other chips doing the same thing.

USB – FT232RL, MC2200, etc.

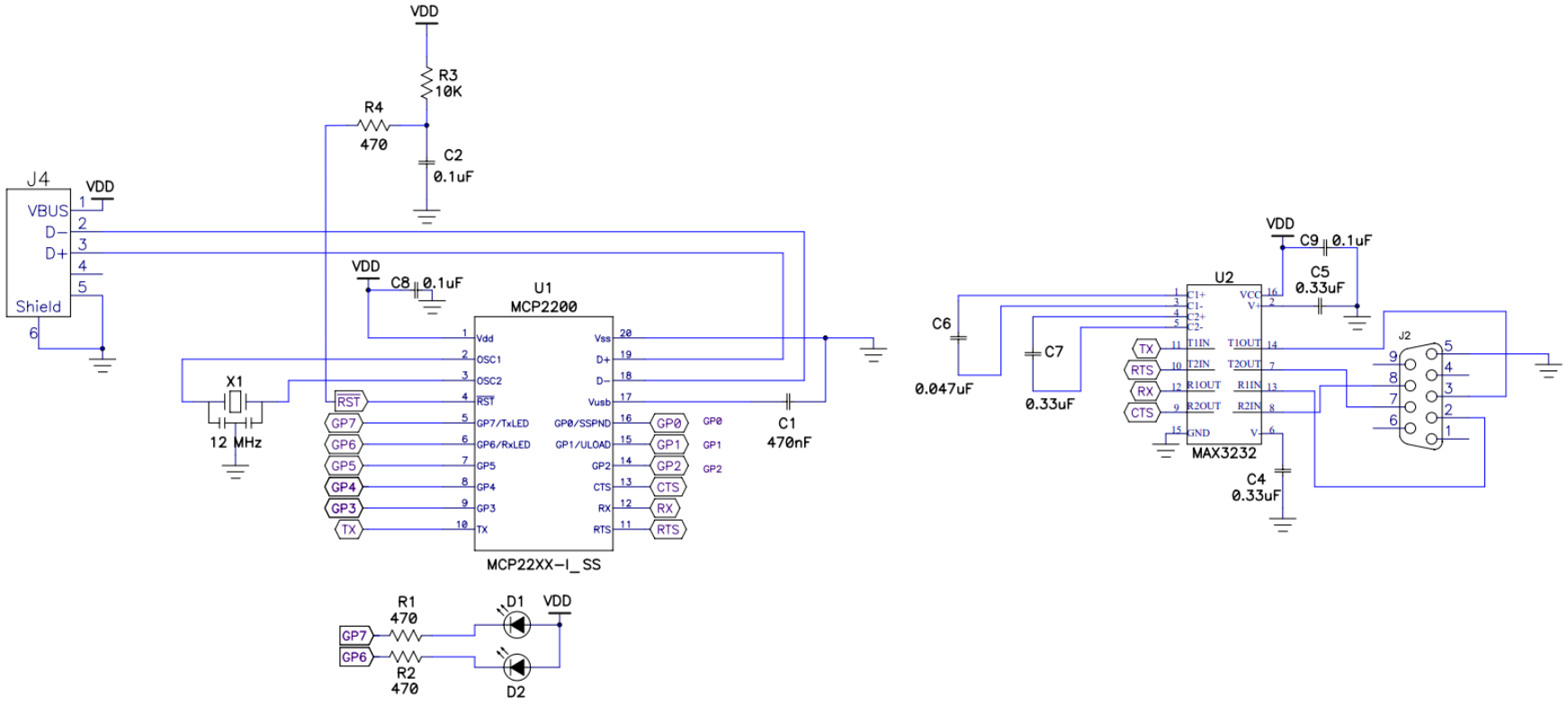
- FT232 typical application



- MCU should be connected to pins TxD and RxD (USART). For full RS232 compatibility pins CTS, RTS, DTR, DSR, DCD may be used
- LEDs are optional

USB – FT232RL, MC2200, etc.

- MCP2200 typical application



Ethernet (overview)

- Ethernet is a computer networking technology, most commonly used in local area network, but also metropolitan and wide area networks.

- Implementing Ethernet connectivity in microprocessor system can greatly improve its flexibility and range of applications. For this purpose an Ethernet controller is needed

- This controller can be implemented in modern microcontroller or it can be a separate peripheral device
- There are several Ethernet drivers for microcontrollers available, like for instance ENC28J60 from Microchip. It's 10BASE-T controller with SPI interface and onboard PHY (recommended for project applications)

The Ethernet controller is the primary component of an embedded-Ethernet system. The controller can be a separate device or integrated with the host MCU.

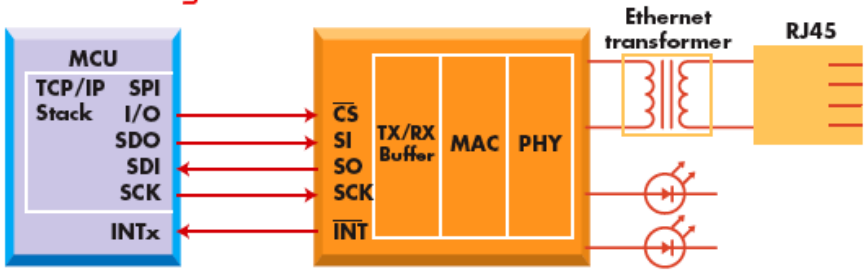


Figure 1

Image source:https://www.embedded.com/design/real-time-and-performance/4006800/Implement-reliable-embedded-Ethernet-connectivity

Ethernet – ENC28J60, etc.

- ENC28J60 typical application