# IDEs, programmers, programming techniques and other programming languages for microprocessor systems
# Lecture 12

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.

# Low-level and high-level programming languages for microprocessor systems

- There are many different programming languages that can be used to describe how a microprocessor system should work
- In general these languages can be sorted into two categories:
  - Low-level programming languages – they provide little or no abstraction from a hardware instruction set and architecture. Assembler is a typical example. In low-level programming languages code is usually not easily portable (if at all), but there is no need to use a compiler or interpreter.
  - High-level programming languages – these languages are usually not directly related to hardware architecture and instruction set. This means that programs can be usually easily converted and ported between different microprocessor systems or computers. They require compilation – translation to machine code. For microprocessor systems typical high-level programming languages are:
    - C, C++
    - BASIC and its versions
    - Interpreted languages, like Java or Python

# IDE - definition

- There are many ways to create a program for a microcontroller-based circuit
  - Small programs can be written in assembler language and calculated by hand – it is possible, but it's rather a geek way of having fun, than a practical approach
  - Programs can be written in assembler language, or C/C++ and then assembled/compiled by external application (assembler/compiler). For this a text editor and an assembler/compiler are required.
    - More useful approach, but still not recommended – it does not allow for instance for debbuging the code

  - In almost all cases programs are designed using IDEs

- IDE (Integrated Development Environment) is a program or set of programs that are used to create, modify, test and debug applications, including microprocessor systems' firmware

# The most popular IDEs for microcontrollers

- IDEs are rather big and complicated programs or set of programs. This clearly leads to a conclusion that good and robust IDEs are paid software.
- There are however some very good and very useful IDEs that are freeware
- The most popular IDEs are (listed according to microcontroller types that they are used with, not all IDEs are listed):

  - 8051
    - KEIL uVision
    - Raisonance RIDE
    - Eclipse
    - BASCOM-51
    - Simplicity Studio
  - AVR
    - AVR Studio
    - Eclipse
    - BASCOM-AVR
  - STM32
    - Atollic TrueStudio
    - Eclipse
    - STM32CubeIDE
    - KEIL uVision

# Microcontroller programming – why?


- In other words how to write your code in microcontroller's memory?
- There are few different ways and each should be described:
  - External parallel programming – this method requires an external programmer with proper socket. Microcontroller, or EPROM/FLASH memory is placed in programmer's socket (outside of the microprocessor system) and programmed using GPIO pins.
    - Very fast
    - Allows for bringing „bricked" microcontrollers back to life (for instance it allows to reset AVR fault fuse bits)
    - The microcontroller must be taken out of the circuit – it is inconvenient.
    - In some cases it may use significantly higher voltage
      - For instance 8051 is programmed with RESET pin tied to +12V

# Microcontroller programming – why?

- There are few different ways and each should be described:
  - ISP – In System Programming – usually serial transmission (SPI) based way of programming microcontrollers, supplied by for instance some 8051 microcontrollers and almost all AVRs and PIC
    - Not as fast as parallel programming
    - It allows to program microcontrollers **in system** – there is no need to remove any chips for programming – very convenient
    - It does not allow to unstuck „bricked” microcontrollers
    - It does not always allow for debugging
    - Currently, due to low cost of programmers, it is the most popular way to program microcontrollers in smaller applications and in amateur applications.

# Bootloaders

- It is often required to provide a possibility for firmware upgrade in designed applications.
- In most cases end users do not have programmers, nor any idea how the microcontroller works (they would fail this course).
- There is a need to provide a simple way to update the firmware without any external hardware. The best would be to allow microcontroller for self-update.
- Such feature is possible thanks to bootloaders. These are small programs placed in special part of microcontroller memory and allow for reprogramming of firmware (updates) using UART – it is then possible to uptade the firmware using RS232, or USB (thanks to chips like FT232RL).

- JTAG – Joint Test Action Group – IEEE 1149.1 name given for a protocol used to test connections on PCB modules.
- It is also used to debug, run, configure and program integrated circuits, including CPLDs, FPGAs and microcontrollers
- It allows ISP programming

- Full JTAG consists of 5 lines – Test Data In (TDI), Test Data Out (TDO), Test Clock (TCK), Test Mode Select (TMS) and Test Reset (TRST)

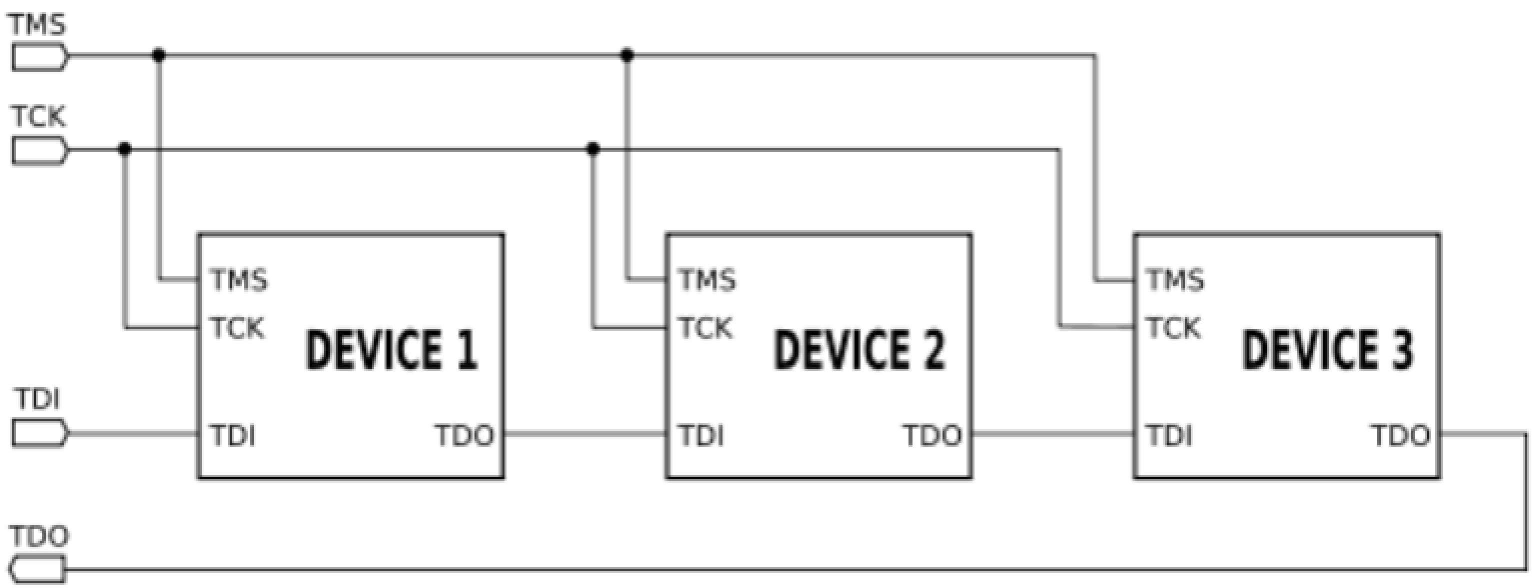




Image source: pl.wikipedia.org

# SWD – Serial Wire Debug interface

- SWD (Serial Wire Debug) interface is very similar to JTAG, but it uses less pins. SWD uses ARM standard for bi-directional wire protocol. In SWD JTAG TMS and TCK are named as SWDIO and SWCLK pins

- It is most commonly used with ARM microcontrollers, like STM32

- It is particularly useful in pin limited applications and small microcontrollers.

- SWD is able to make use of the full clock cycle, unlike JTAG, where data is driven only on the falling Edge.

- SWD provides some protection against errors, including simple parity checking, overrun checking, et.c

- SWD provides full access to the debug and trace functionality


Image source: pl.wikipedia.org, further reading : https://www.arm.com/files/pdf/Serial_Wire_Debug.pdf

# Parallel programming example for AT89S4051

**Figure 21-1.** Flash Parallel Programming Device Connections

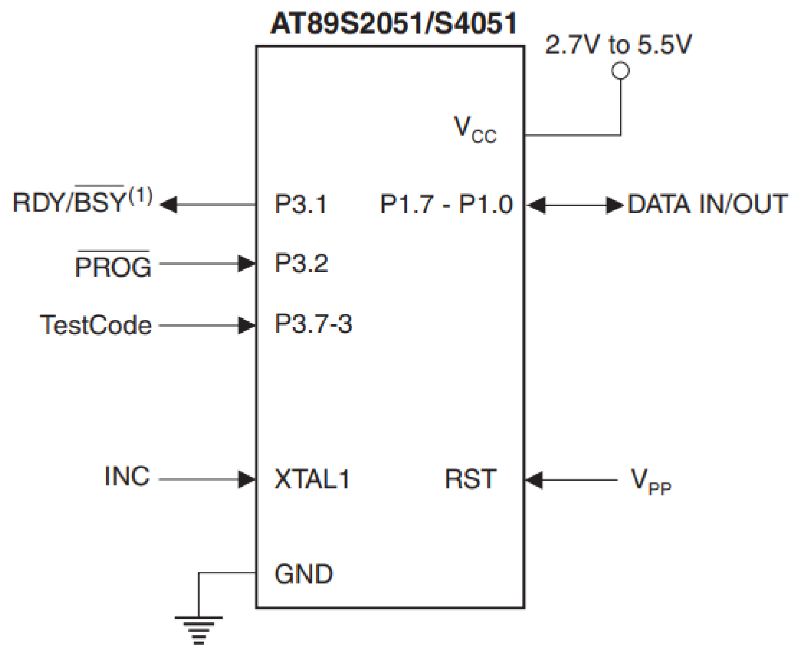



**Table 21-2.** Parallel Programming Mode Command Summary

| Mode | | Test Control | | | Test Selects | | | | Data I/O |
|---|---|---|---|---|---|---|---|---|---|
| | | P3.2 | RST[1] | INC | P3.3 | P3.4 | P3.5 | P3.7 | P1.7-0 |
| Chip Erase[5] | | 1.0 µs | 12V | L | H | L | L | L | XX |
| Load X-Address[2] | | H | 12V | 0.1 µs | H | L | H | H | $D_{IN}$ |
| Page Write[3][4][6] | **Code Memory** | 1.0 µs | 12V | 0.1 µs | L | H | H | H | $D_{IN}$ |
| Page Read[3] | **Code Memory** | H | 12V | 0.1 µs | L | L | H | H | $D_{OUT}$ |
| Page Write[3][4][6][7] | Sig. Row | 1.0 µs | 12V | 0.1 µs | L | L | L | L | $D_{IN}$ |
| Page Read[3][8][10] | Sig. Row | H | 12V | 0.1 µs | L | L | L | H | $D_{OUT}$ |
| Write Fuse/Lock Bit[5][9] | | 1.0 µs | 12V | L | H | H | H | H | $D_{IN}$ |
| Read Fuse/Lock Bit[9] | | H | 12V | L | H | H | L | L | $D_{OUT}$ |

Image source: AT89S4051 datasheet

# Parallel programming example for AT89S4051

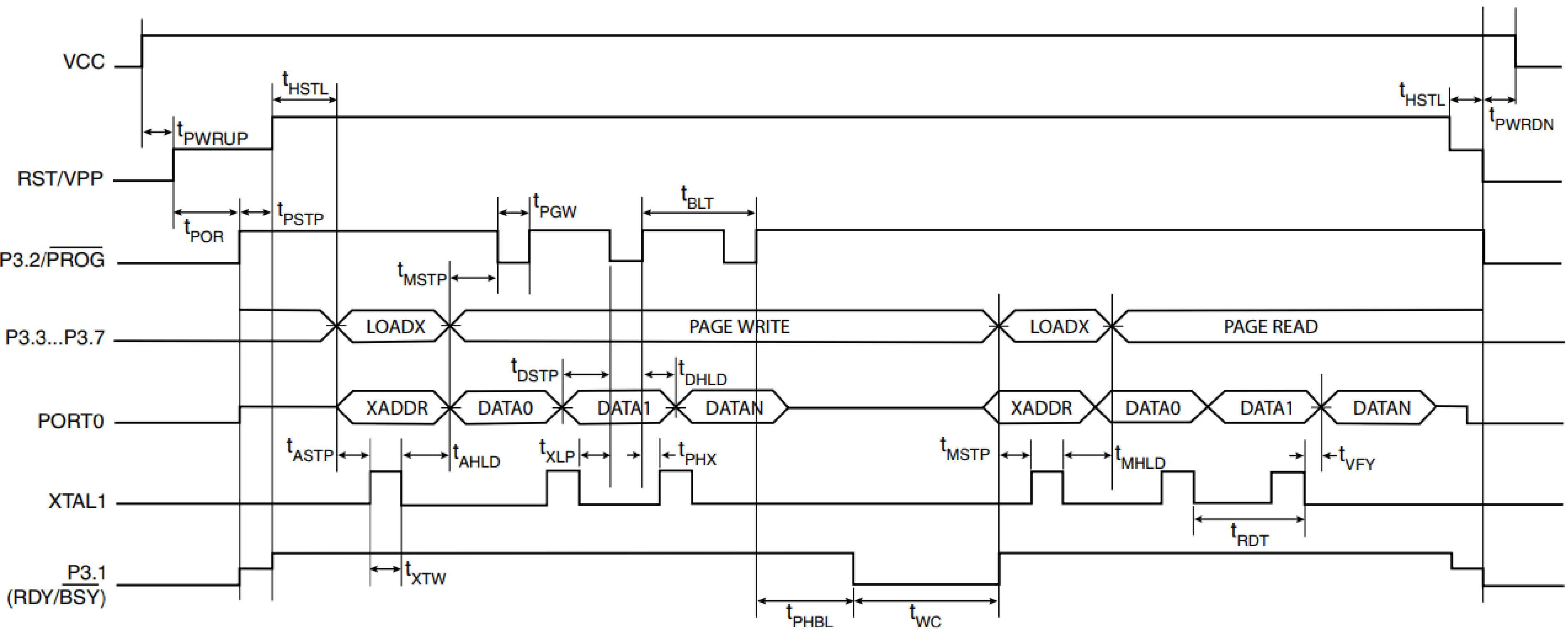



Image source: AT89S4051 datasheet

# Serial (ISP) programming example for AT89S4051

**Figure 33-1.** ISP Programming Device Connections

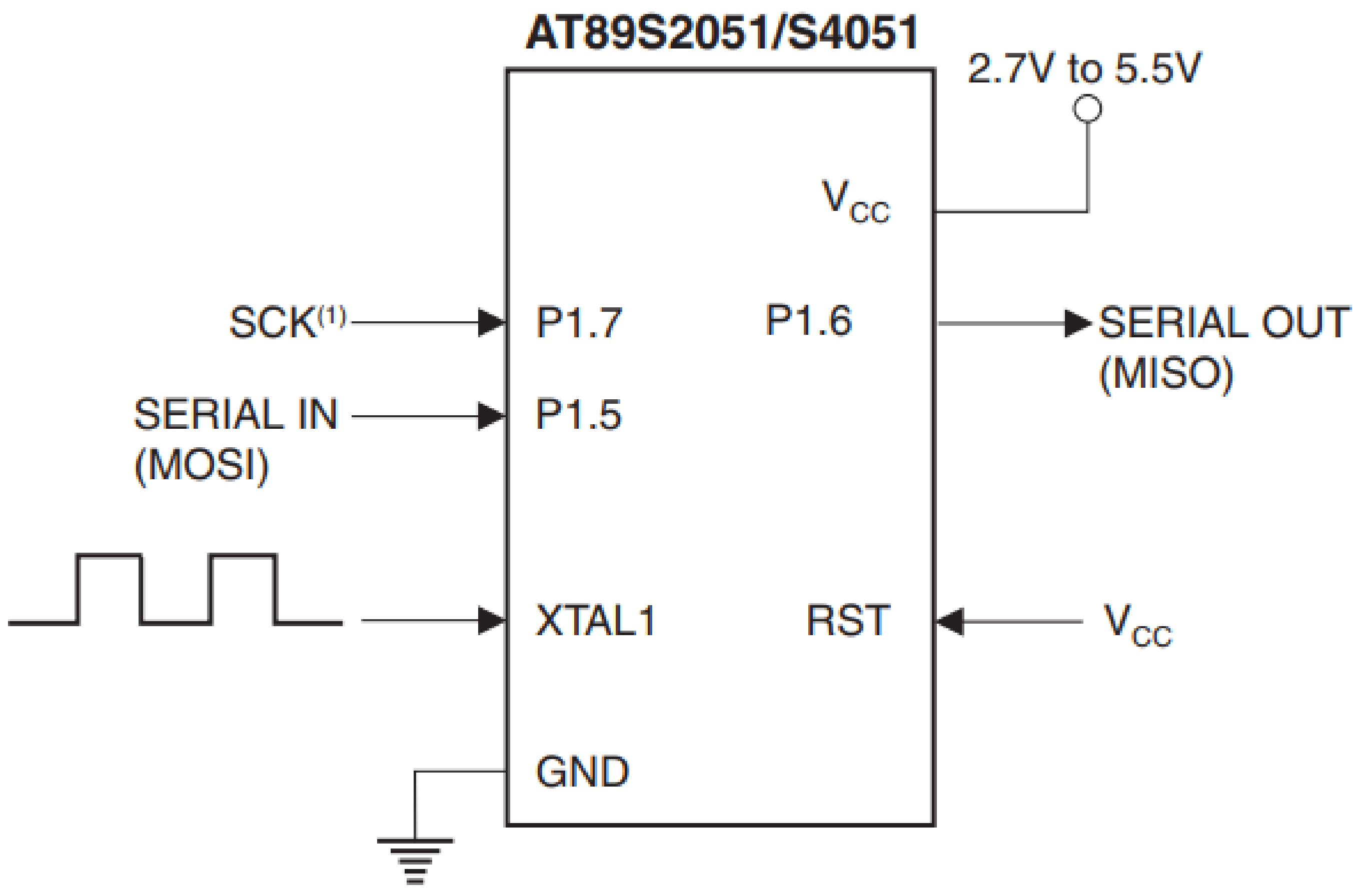





Image source: AT89S4051 datasheet

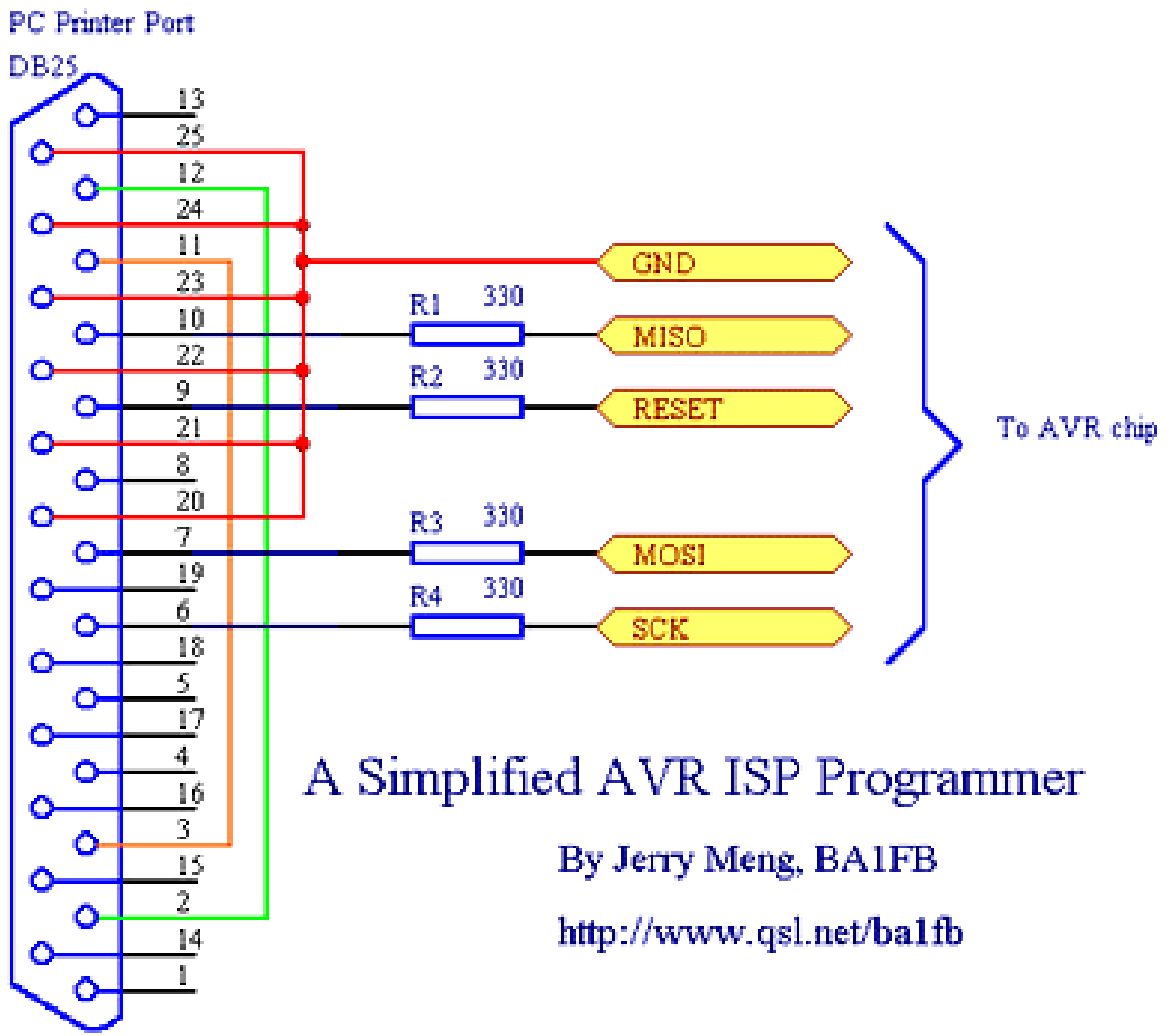
# Serial (ISP) programming example for AT89S4051

## 34. Serial Programming Command Summary

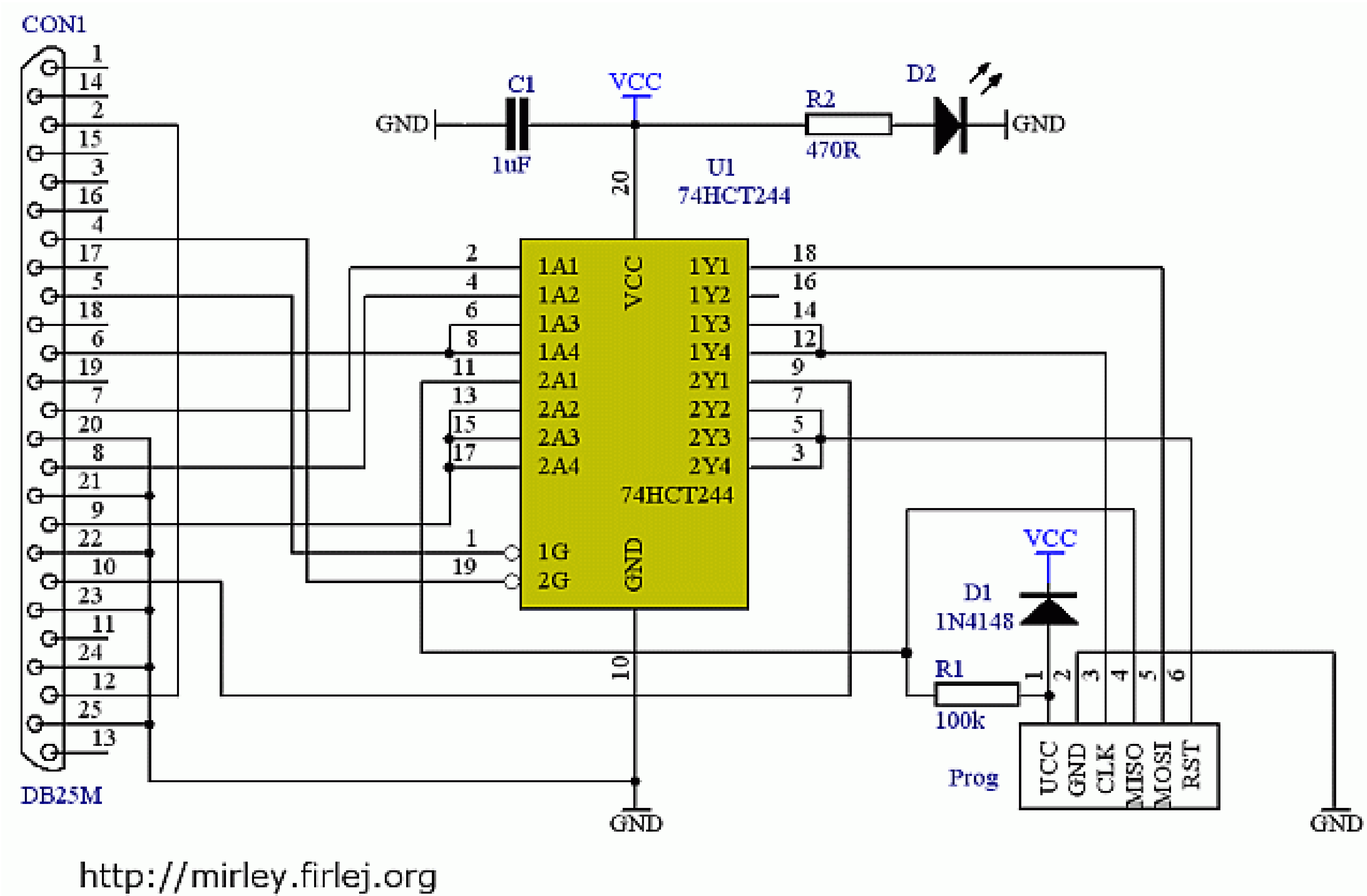
| Command | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte ... |
|---|---|---|---|---|---|
| Program Enable[1] | 1010 1100 | 0101 0011 | xxxx xxxx | xxxx xxxx | |
| Chip Erase | 1010 1100 | 100x xxxx | xxxx xxxx | xxxx xxxx | |
| Write Code **Byte** | 0100 0000 | xxxx A11 A10 A9 A8 | A7 A6 A5 A4 A3 A2 A1 A0 | D7 D6 D5 D4 D3 D2 D1 D0 | |
| Read Code **Byte** | 0010 0000 | xxxx A11 A10 A9 A8 | A7 A6 A5 A4 A3 A2 A1 A0 | D7 D6 D5 D4 D3 D2 D1 D0 | |
| Write Code **Page**[2] | 0101 0000 | xxxx A11 A10 A9 A8 | A7 A6 A5 0 0000 | Data 0 ... Data 31 | |
| Read Code **Page**[2] | 0011 0000 | xxxx A11 A10 A9 A8 | A7 A6 A5 0 0000 | Data 0 ... Data 31 | |
| Write User Fuses[3] | 1010 1100 | 0001 F3 F2 F1 F0 | xxxx xxxx | xxxx xxxx | |
| Read User Fuses[3] | 0010 0001 | xxxx xxxx | xxxx xxxx | xxxx F3 F2 F1 F0 | |
| Write Lock Bits[4] | 1010 1100 | 1110 0x LB2 LB1 | xxxx xxxx | xxxx xxxx | |
| Read Lock Bits[4] | 0010 0100 | xxxx xxxx | xxxx xxxx | xxxx xx LB2 LB1 | |
| Write User Signature **Byte** | 0100 0010 | xxxx xxxx | xxx A4 A3 A2 A1 A0 | D7 D6 D5 D4 D3 D2 D1 D0 | |
| Read User Signature **Byte** | 0010 0010 | xxxx xxxx | xxx A4 A3 A2 A1 A0 | D7 D6 D5 D4 D3 D2 D1 D0 | |
| Write User Signature **Page**[2] | 0101 0010 | xxxx xxxx | xxxx xxxx | Data 0 ... Data 31 | |
| Read User Signature **Page**[2] | 0011 0010 | xxxx xxxx | xxxx xxxx | Data 0 ... Data 31 | |
| Read Atmel Signature Byte[5] | 0010 1000 | xxxx xxxx | xxx A4 A3 A2 A1 A0 | D7 D6 D5 D4 D3 D2 D1 D0 | |

# Dedicated and universal programmers

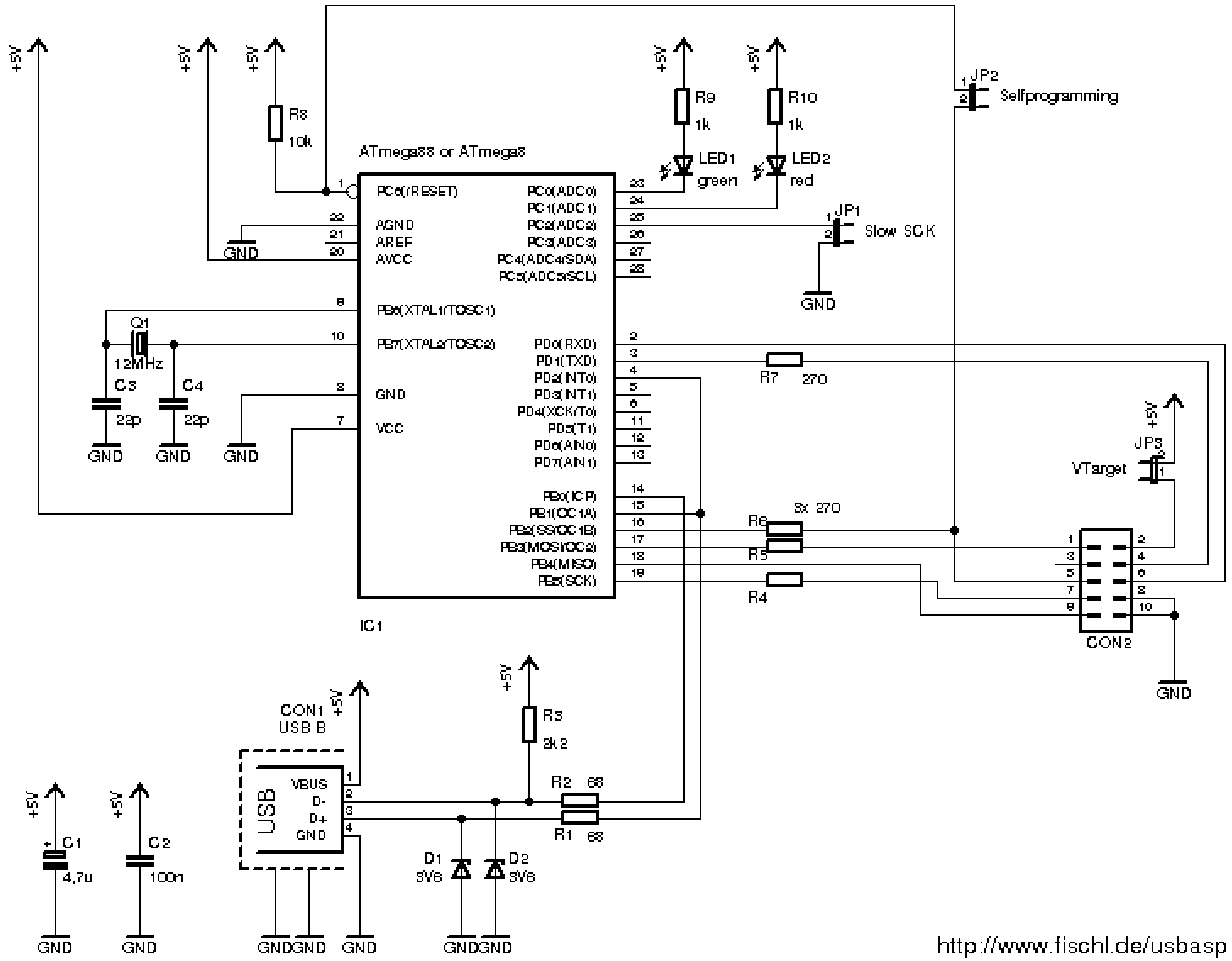

# Almost the simplest (and dangerous) ISP programmer, using ancien LPT port



# STK200


CON1
DB25M
C1
1uF
GND
VCC
R2
470R
D2
U1
74HCT244
1A1
1A2
1A3
1A4
2A1
2A2
2A3
2A4
1Y1
1Y2
1Y3
1Y4
2Y1
2Y2
2Y3
2Y4
1G
2G
VCC
GND
D1
1N4148
R1
100k
Prog
UCC
GND
CLK
MISO
MOSI
RST
http://mirley.firlej.org

# USBAsp


ATmega88 or ATmega8
IC1
PC6(/RESET)
AGND
AREF
AVCC
PB6(XTAL1/TOSC1)
PB7(XTAL2/TOSC2)
GND
VCC
PC0(ADC0)
PC1(ADC1)
PC2(ADC2)
PC3(ADC3)
PC4(ADC4/SDA)
PC5(ADC5/SCL)
PD0(RXD)
PD1(TXD)
PD2(INT0)
PD3(INT1)
PD4(XCK/T0)
PD5(T1)
PD6(AIN0)
PD7(AIN1)
PB0(ICP)
PB1(OC1A)
PB2(SS/OC1B)
PB3(MOSI/OC2)
PB4(MISO)
PB5(SCK)
R8 10k
R9 1k
R10 1k
LED1 green
LED2 red
JP1 Slow SCK
JP2 Selfprogramming
JP3 VTarget
Q1 12MHz
C3 22p
C4 22p
R7 270
3x 270
R6
R5
R4
CON2
CON1 USB B
USB
VBUS
D-
D+
GND
R3 2k2
R2 68
R1 68
D1 3V6
D2 3V6
C1 4,7u
C2 100n
+5V
GND
http://www.fischl.de/usbasp

# STLinkV2

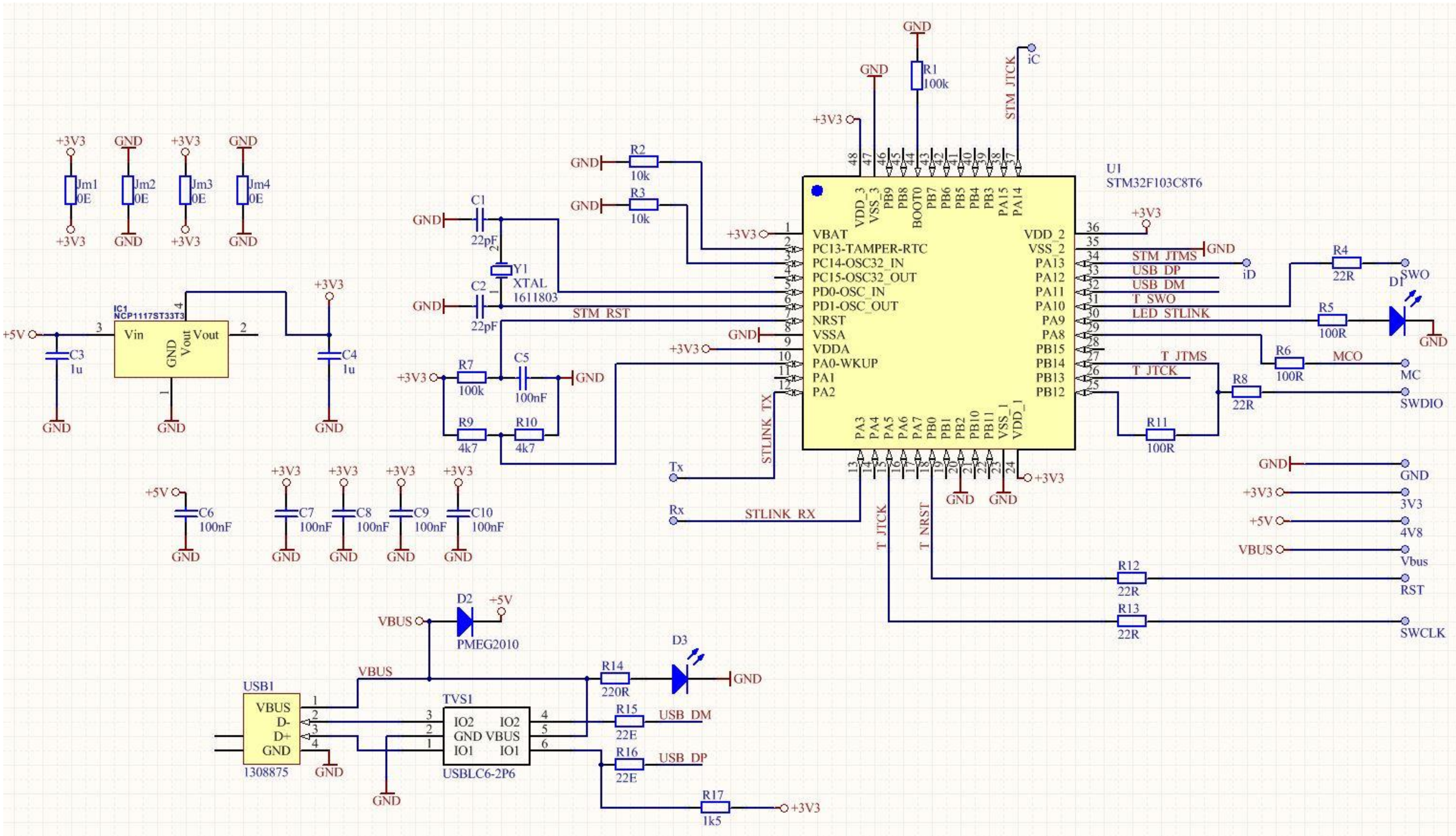



Image source: https://e.pavlin.si/2016/02/22/st-link-v2-on-a-single-sided-diy-pcb/