# Lecture 5: Practical application

**Hardware, Internals and Programming of AVR Microcontrollers in Assembler**

**by**
**Gerhard Schmidt**
**Kastanienallee 20**
**D-64289 Darmstadt**

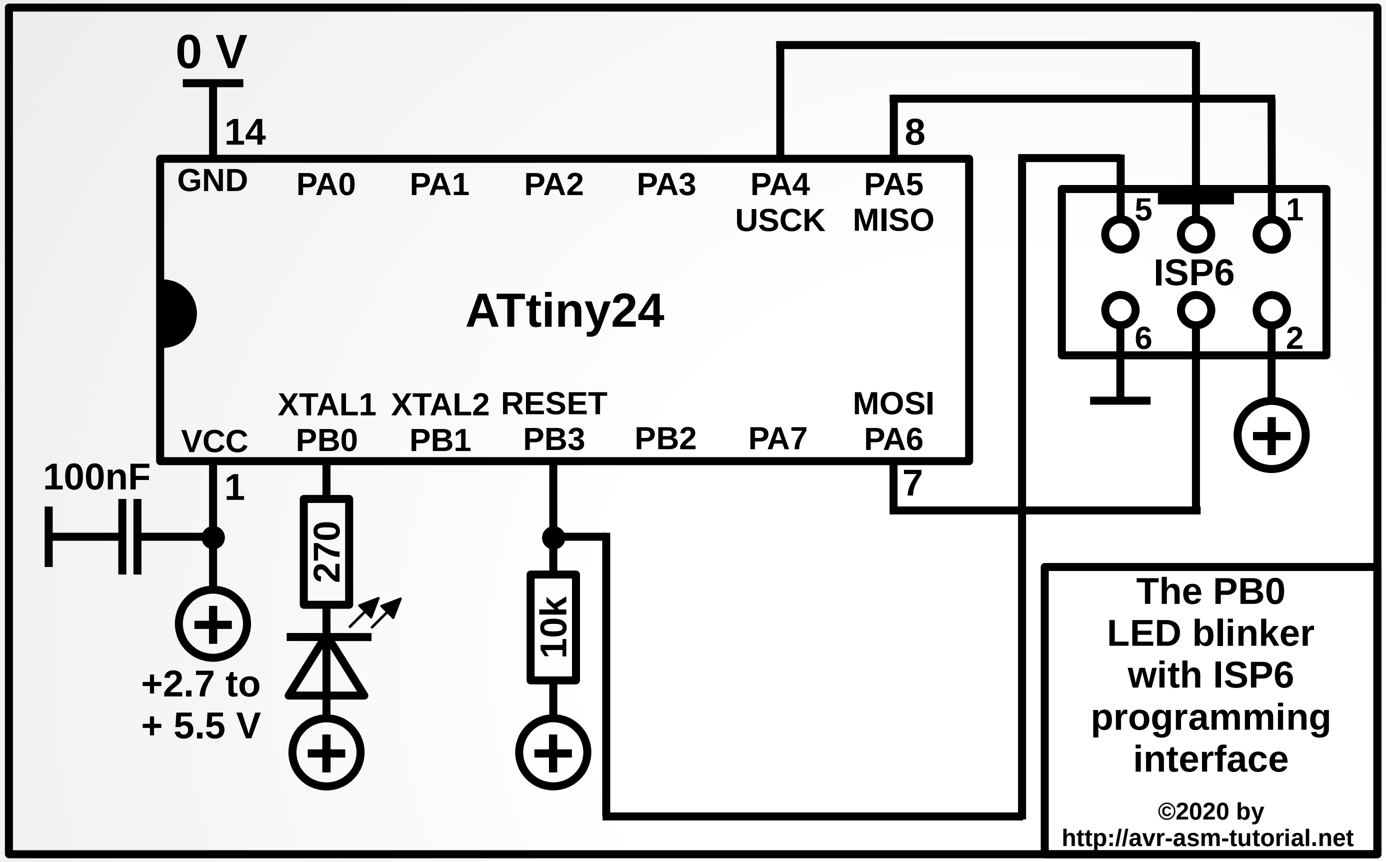
## The ISP programming interface

- Writing binary code to the controller requires access to its internals.
- A convenient method to gain access is the In-System-Programming interface of all AVRs.
- ISP (In-System-Programming) works via a standardized 6-pin plug. Via this the programmer hardware can access the internals.
- A second programming method works with +12 V on the RESET pin (High-voltage serial program-ming, HV programming). This is not used here.

# Practical applications

- With this lecture you will learn how the controller is programmed and practically does for what your source code has teached him to do.
- You will need
  - a) a hardware programmer (I use a Diamex ProgS2, but there are several others in the market),
  - b) a program that can write hex files to the programmer (I use ATMEL‘s Studio 4.19, an alternative is AvrDude),
  - c) some electronic components (a breadboard and wires, a 14-pin IC socket, a 6-pin box connector, two 6-pin IDC sockets plus 10 cm 6-pole flat cable, a resistor of 10 kΩ, a resistor of 270Ω, a red standard LED and a 100nF ceramic capacitor).
- First get familiar with the breadboard and its internal wiring.

# The schematic of the PB0 blinker



- This is the schematic.
- The ATtiny24 is shown from the top.
- The ceramic capacitor on pin 1 suppresses rectangles from the CMOS internals on the +5V line.
- The LED is connected as sink load with a current-limiting resistor.
- The 10kΩ ties the RESET pin to +5V.

- The In-System-Programming interface ISP6 is connected with USCK, MISO, MOSI, RESET and the operating voltage.
- The programmer delivers +5V, if so enabled. If not: connect a 4.5V or 3.7V batterie to the GND- and +-connections.

## The ISP programming interface

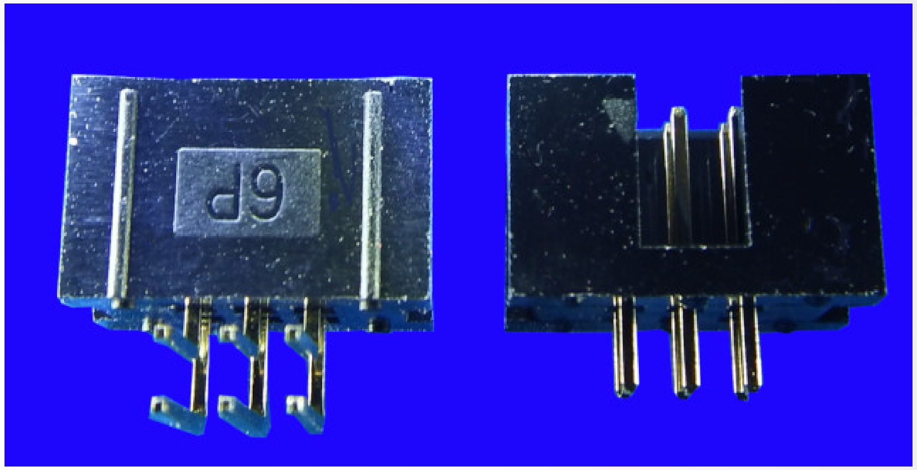
- Via a 6-pin plug the following signals allow access with programmer hardware:
  1. The RESET pin (pin 4 of the ATtiny24, pin 5 of the ISP6 plug) is tied LOW, which initiates the serial communication between the programmer and the controller.
  2. The SCK or USCK pin (pin 8 of the ATtiny24, pin 3 of the ISP6) clocks the device‘s communication.
  3. MOSI (pin 7 of the ATtiny24, pin 4 of the ISP6) outputs serial data (from the programmer to the device).
  4. MISO (pin 8 of the ATtiny24, pin 1 of the ISP6) inputs serial data (from the device to the programmer).
- When programming is completed, the RESET pin is disconnected, the 10k pulls it high and the controller immediately starts at address 0000.

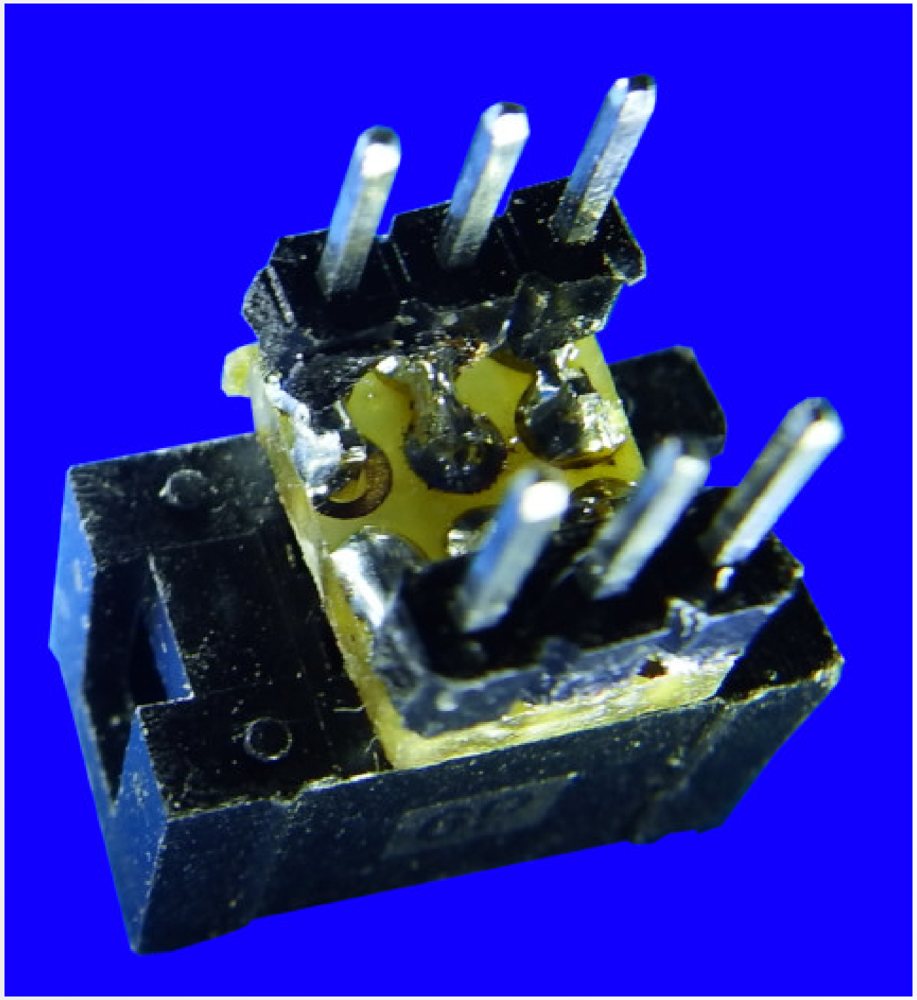
## Advantages of ISP

- The device can be programmed with the standard operating voltages, no extra voltage supplies are necessary.
- The device can remain in its place, no removal or disassembling of the device is necessary.
- Via the 6-pin ISP plug the whole system can be sourced (e.g. with the 5V from the USB plug), if the programmer enables that.
- Quick and reliable re-programming of the controller can be achieved with very low effort and nearly no extra hardware.

# The ISP6 connector

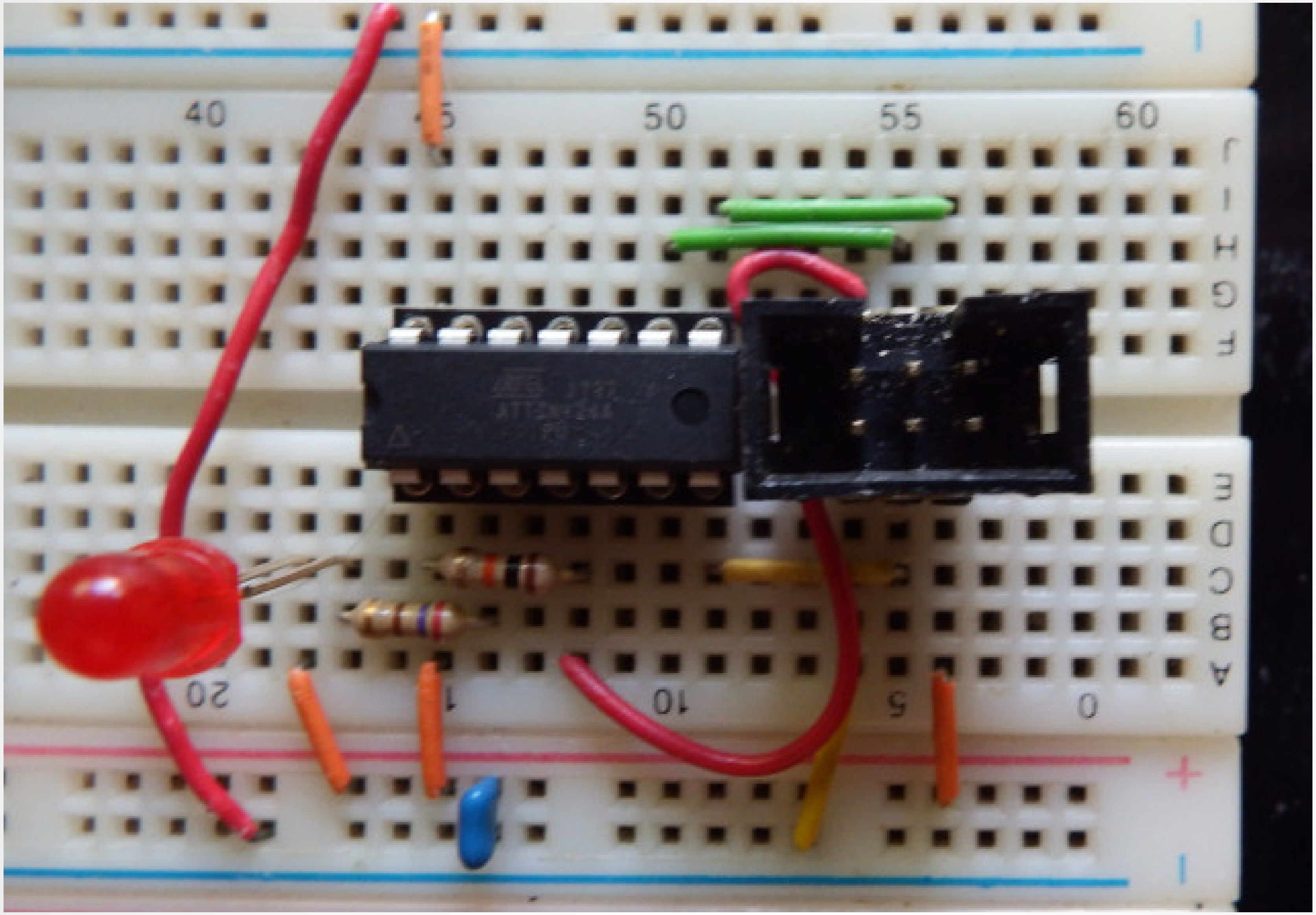
- These are the 6-pin box connectors. We need only a straight one.





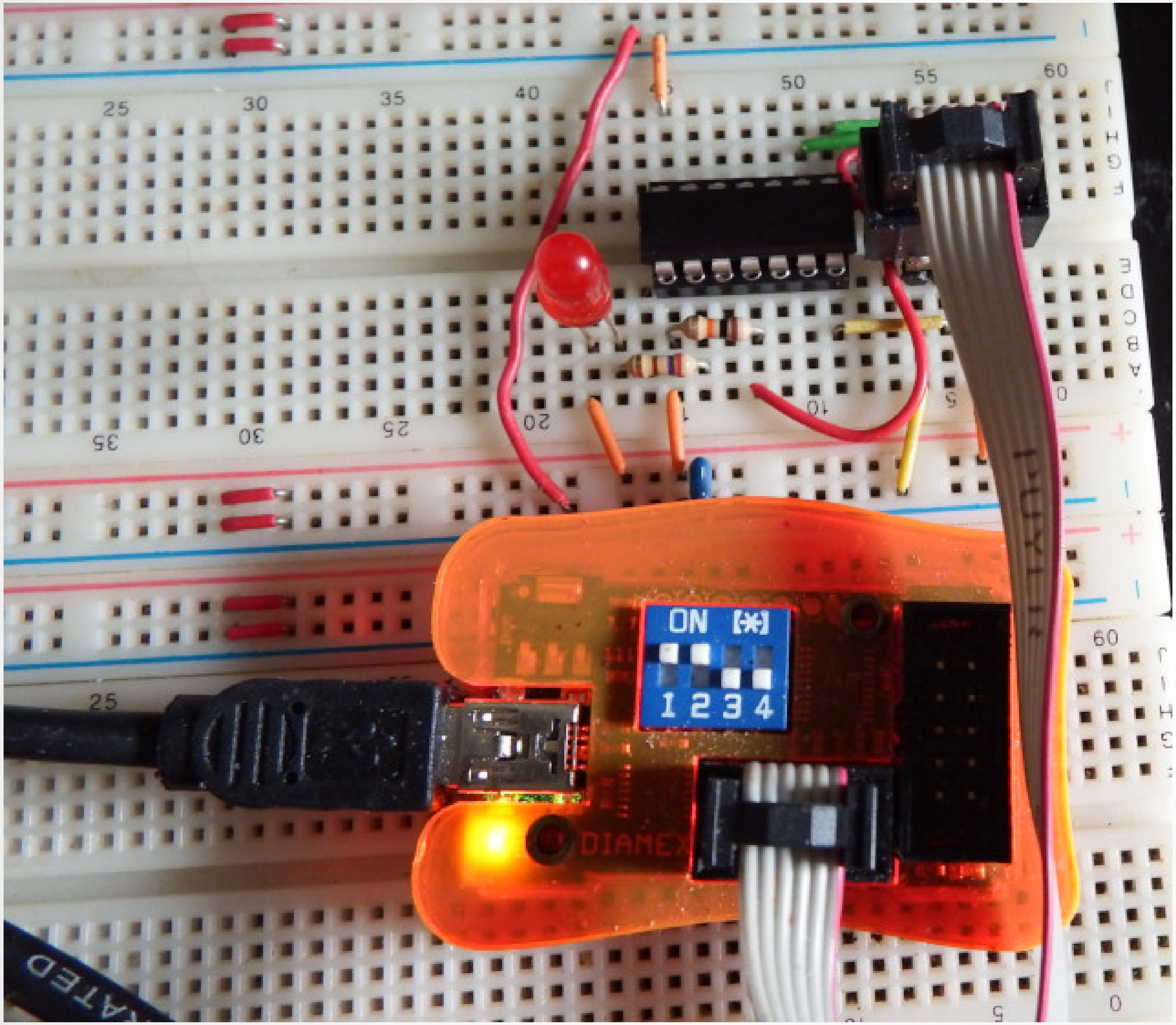
- In order to fit this to the breadboard we need to solder this to a small 12-hole PCB and add two 3-pin plugs, like shown here.

# The breadboard programmer



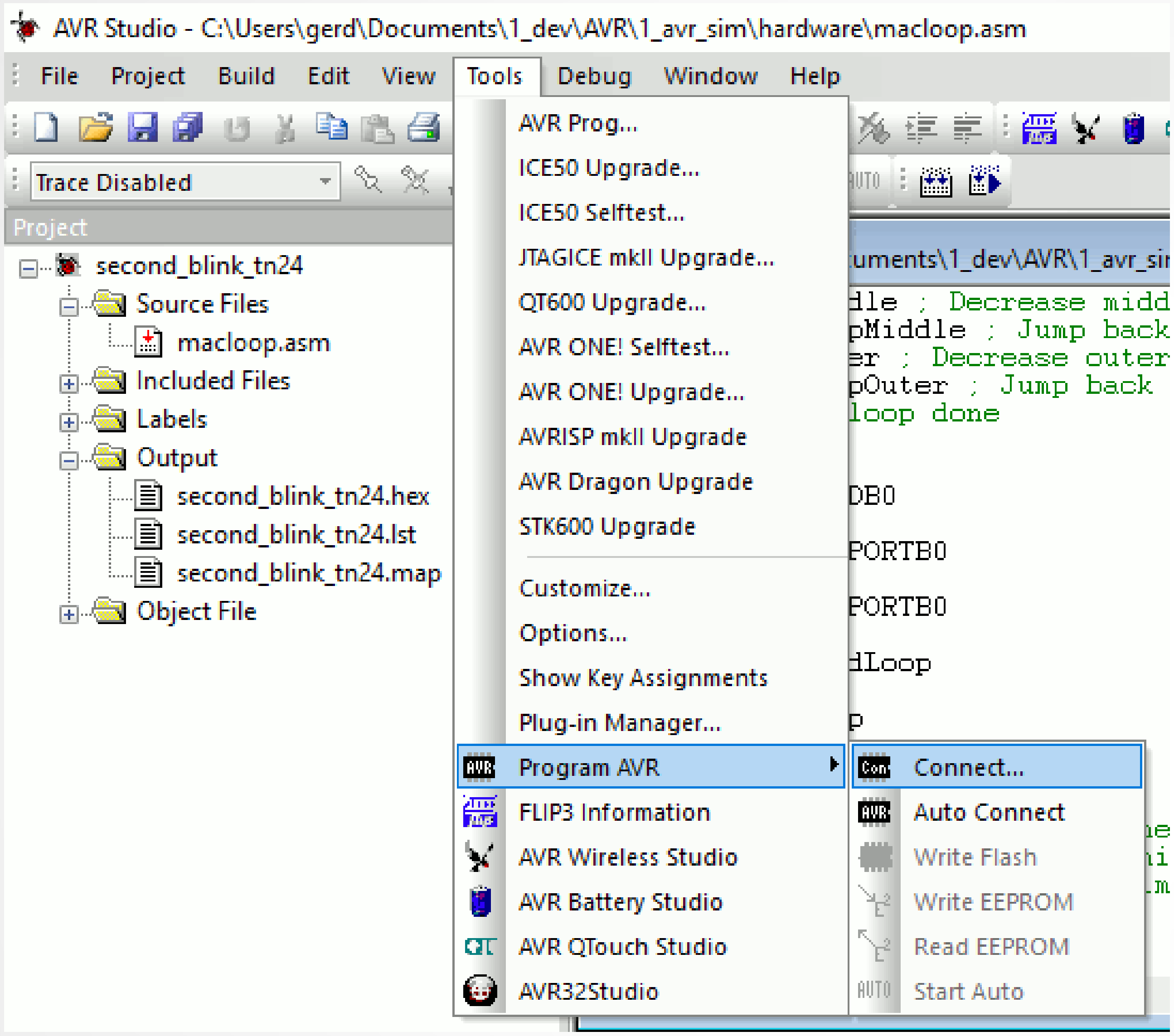
- This is all on the bread-board.
- The upper and lower (–) line are to be connec-ted.
- The ISP6 delievers the operating voltage.

# Connecting the programmer



- The programmer (here a Diamex ProgS2) is connected with a 6-pin flat cable with the ISP6 plug.
- This allows programming of the ATtiny24 on the board.
- The two switches provide the 5V through the ISP6 line.

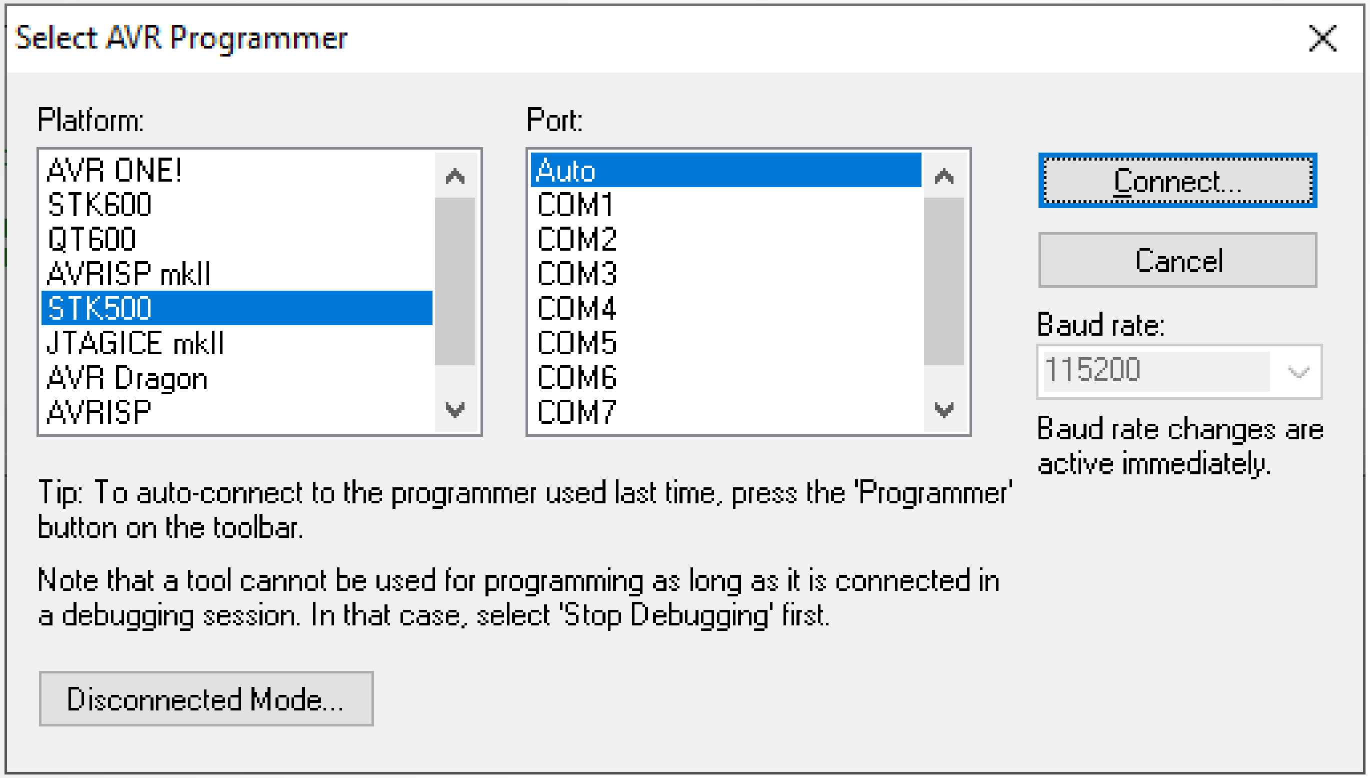
# Programming with the Studio



```
uments\1_dev\AVR\1_avr_si
dle ; Decrease midd
pMiddle ; Jump back
er ; Decrease outer
pOuter ; Jump back
loop done

DB0

PORTB0

PORTB0

dLoop

p
```

- If you use the Studio 4.19 (like shown here) you‘ll have to download it, to install it and to install the Jungo USB driver from ATMEL Norway (under Win10 the one for Studio 7, not the one that comes with Studio 4.19, disable driver signatures).
- In the Tools menu select Program AVR and connect to your programmer.

# Identifying the programmer


Select AVR Programmer

Platform:

- AVR ONE!
- STK600
- QT600
- AVRISP mkII
- STK500
- JTAGICE mkII
- AVR Dragon
- AVRISP

Port:

- Auto
- COM1
- COM2
- COM3
- COM4
- COM5
- COM6
- COM7

Baud rate: 115200

Baud rate changes are active immediately.

Tip: To auto-connect to the programmer used last time, press the 'Programmer' button on the toolbar.

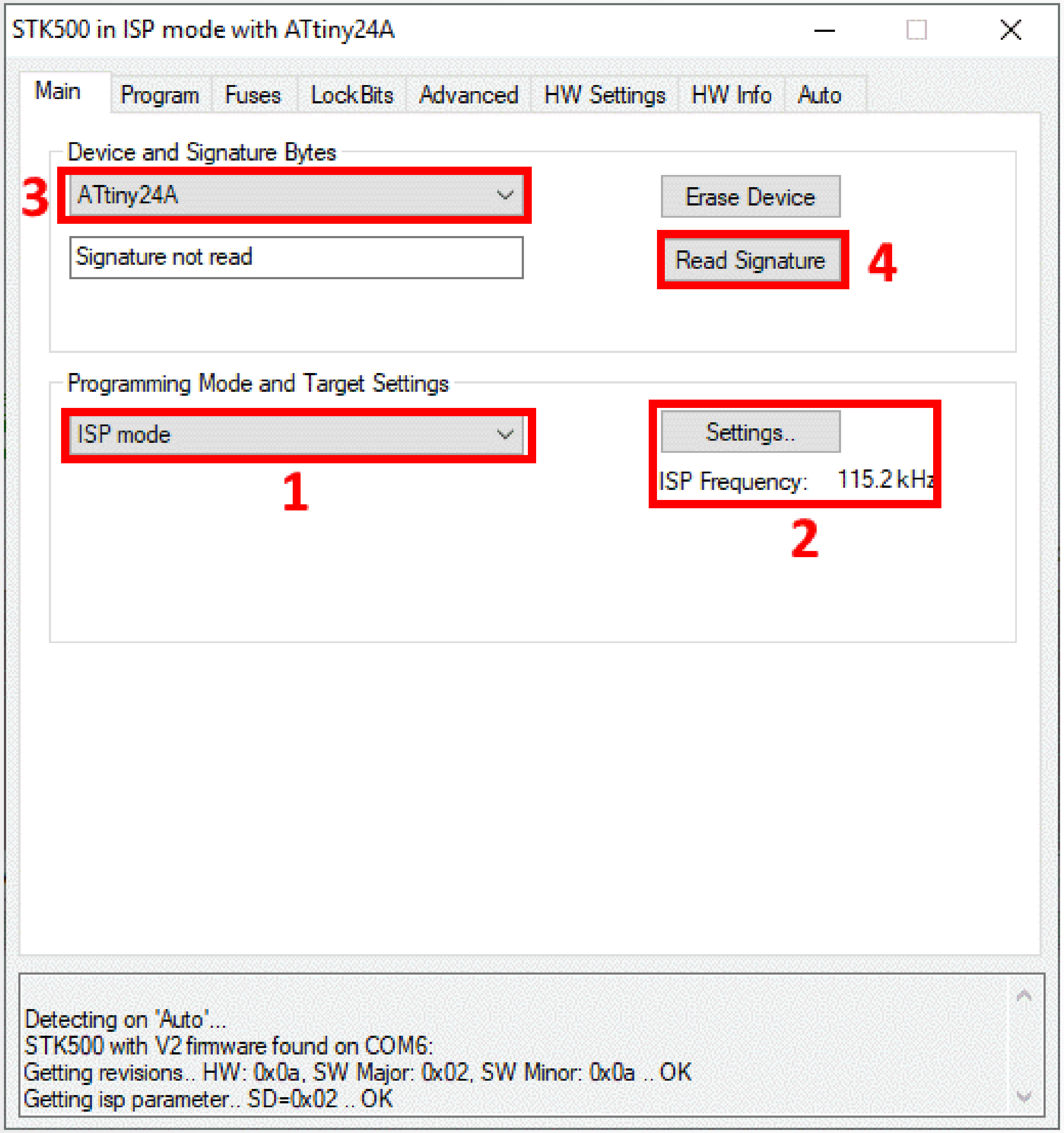
Note that a tool cannot be used for programming as long as it is connected in a debugging session. In that case, select 'Stop Debugging' first.

- **Select the mode that the programmer is working with.**
- **Most programmers work like an STK 500 board and have a USB-to-Serial converter.**

- **Select the mode that the programmer is working with.**
- **Most programmers work like an STK 500 board and have a USB-to-Serial converter.**

- **If your Windows installs the Serial converter to a COM interface beyond COM8, first replace it by a lower COM number.**

# Setting ISP mode

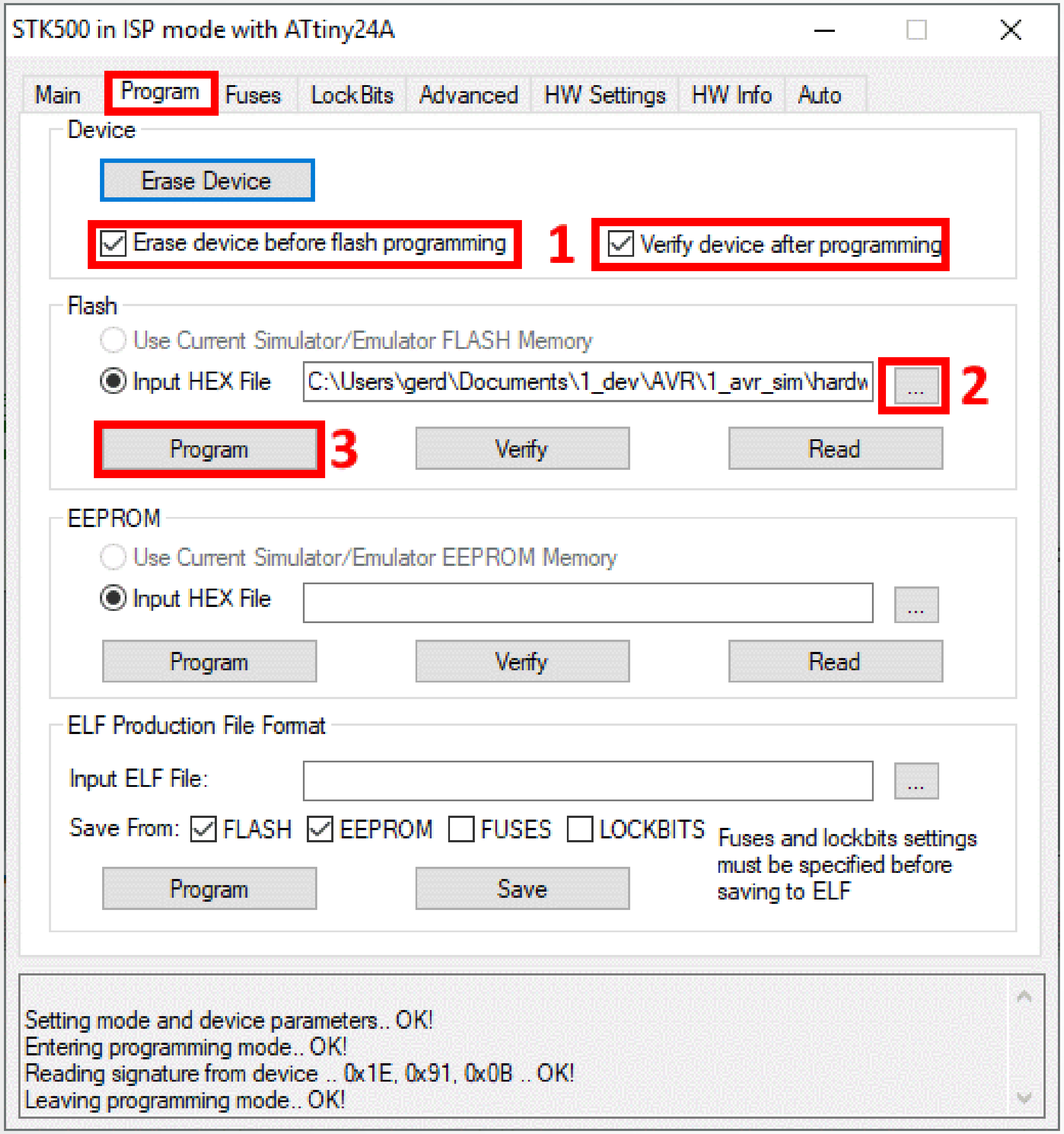

STK500 in ISP mode with ATtiny24A

ISP Frequency: 115.2 kHz

```
Detecting on 'Auto'...
STK500 with V2 firmware found on COM6:
Getting revisions.. HW: 0x0a, SW Major: 0x02, SW Minor: 0x0a .. OK
Getting isp parameter.. SD=0x02 .. OK
```

- **In the Main tab, make sure that the ISP mode is selected. If not select this from the dropdown list.**
- **Next ensure that the ISP frequency is lower than ¼ of the device's clock frequency (1 MHz). If not click Settings and select another frequency.**
- **Next set the device type and select it from the dropdown list.**
- **At last click Read Signature. The result should match.**

# Programming the device



- [✓] Erase device before flash programming
- [✓] Verify device after programming

Flash: Use Current Simulator/Emulator FLASH Memory [● Input HEX File] C:\Users\gerd\Documents\1_dev\AVR\1_avr_sim\hardw

EEPROM: Use Current Simulator/Emulator EEPROM Memory [● Input HEX File]

Input ELF File:

Save From: [✓] FLASH [✓] EEPROM [ ] FUSES [ ] LOCKBITS

Fuses and lockbits settings must be specified before saving to ELF

```
Setting mode and device parameters.. OK!
Entering programming mode.. OK!
Reading signature from device .. 0x1E, 0x91, 0x0B .. OK!
Leaving programming mode.. OK!
```

- **Then move to the Program tab.**
- **First make sure that the two settings are correct.**
- **Then select the hex file that holds your program (this can either produced by the Studio with the menu Build or with avr_sim).**
- **If correct, press the Program button and view the messages on the window. It should display correct verification.**

## The LED is blinking

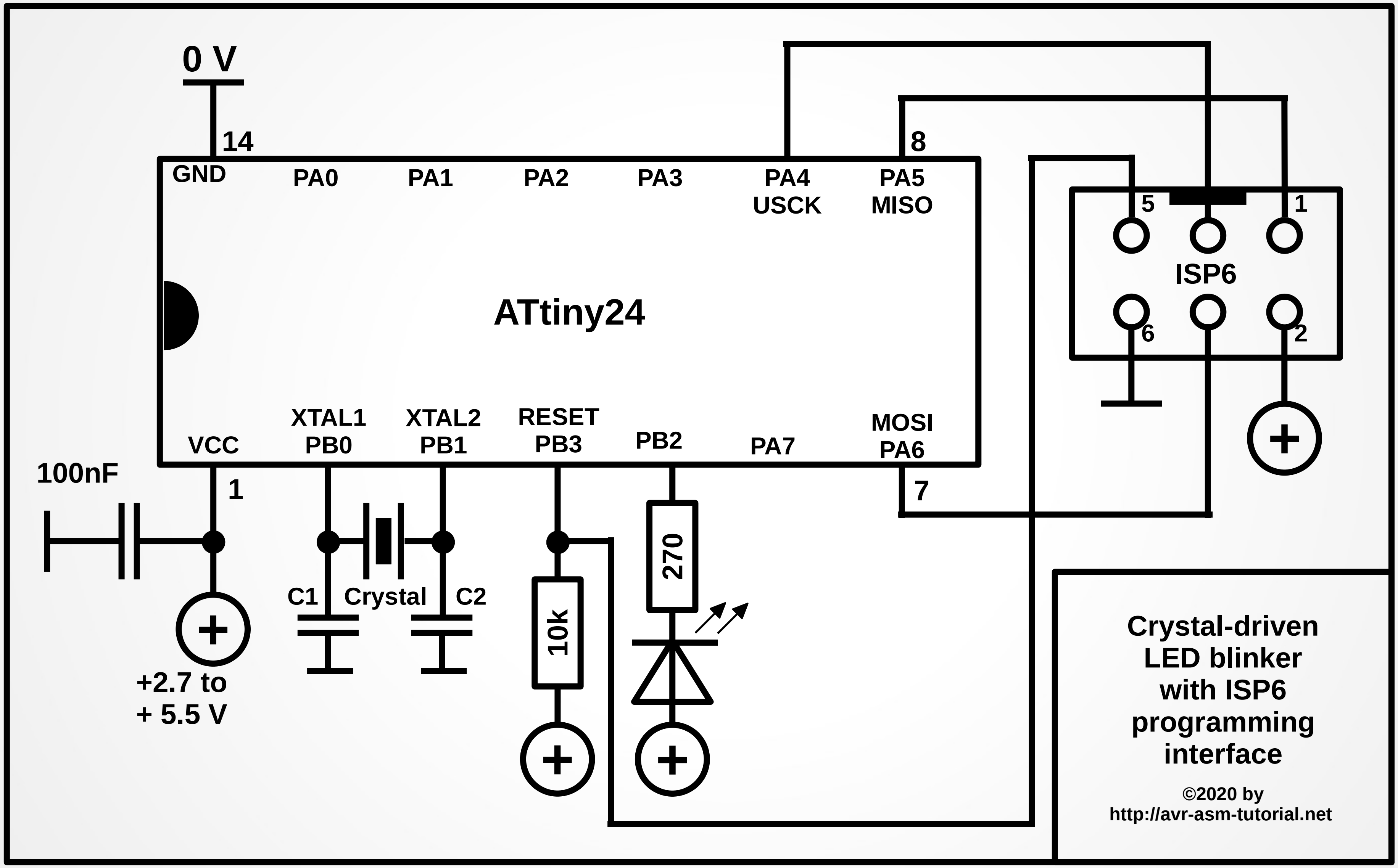
- Immediately after the verification ended, the LED should blink in a seconds rhythm.
- If not: there are many different errors that can be made, such as:
  - false wiring of the ISP6 plug (the programmer reports errors when trying to connect),
  - the ATtiny24 was installed in reverse direction (the USB of the computer reports over-current),
  - missing operating voltage or out of voltage range,
  - a defective LED.

## A high-precision LED blinker

- **The internal RC oscillator in the ATtiny24 is not very reliable: it has an error rate of 3% and more.**
- **If you need the seconds signal to be more exact, you‘ll have to clock the controller with a crystal.**
- **The following shows how to do that and teaches you about the fuses of the controller.**

# Exact clocking – with a crystal

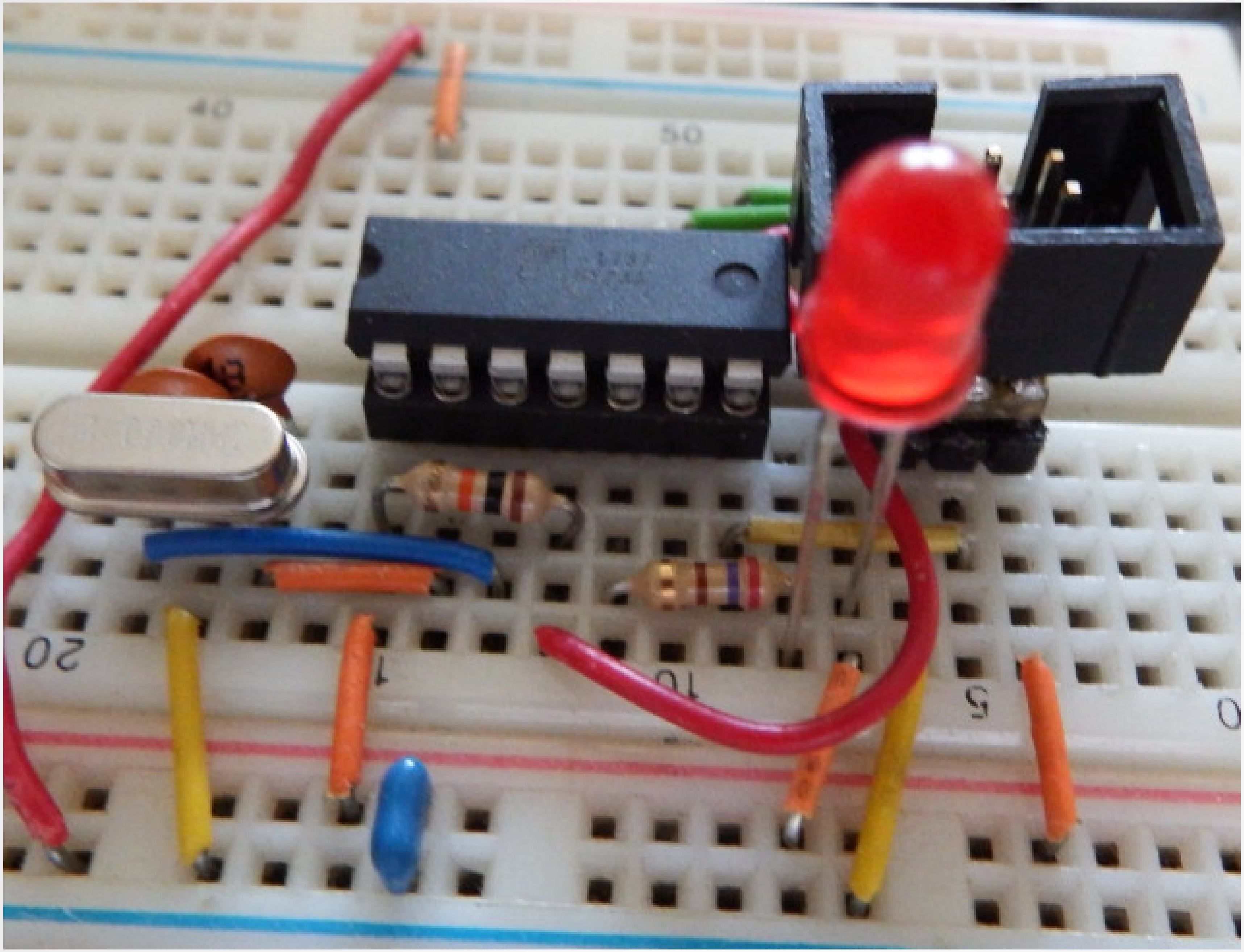
- **This is the crystal-driven ATtiny24:**



## Some hints on the schematic

- **A general rule: each pin of a controller can serve multiple functions. Not GND and VCC, but most others.**
- **An example: PB3 serves by default as RESET input and is required to program the flash memory of the chip and to change internal fuses via ISP.**
- **Here, PB0 and PB1 change their function: they are part of an internal crystal oscillator. XTAL1 is input and XTAL2 is output. But: PB0 and PB1 can not be used any more (so the LED is relocated to PB2).**
- **The generated oscillator signal then clocks the ATtiny24. It runs as exact as the crystal is (for a 4 MHz crystal: 20 to 30 ppm = 4 MHz +/- 80 to 120 Hz).**
- **This change can be (and has to be) done by changing fuses: internal switches that change the hardware of the controller.**

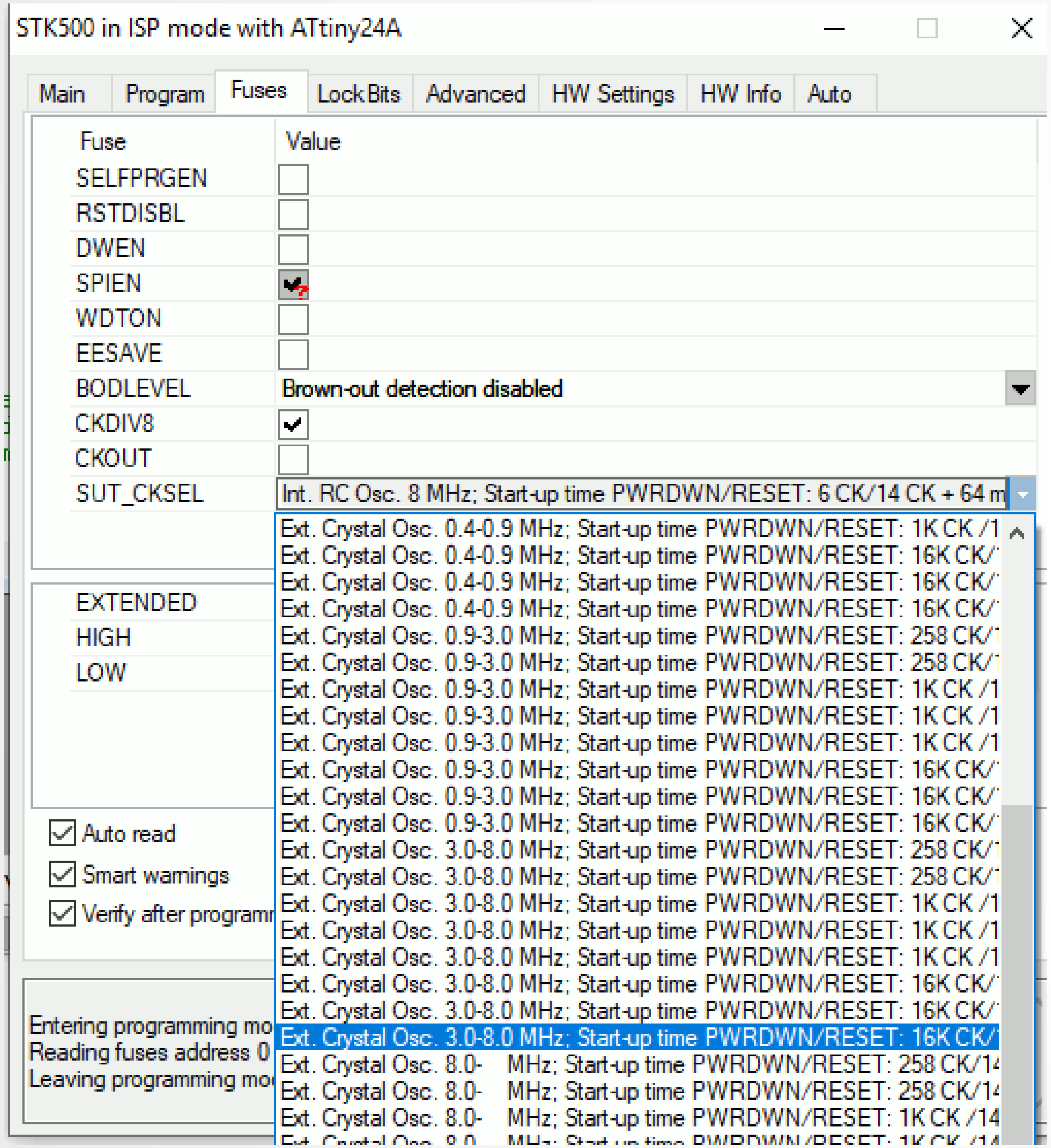
# The xtal driven LED blinker



The crystal has 4 MHz, the two capacitors 18pF.

The LED and its current limiting resistor is now connected to PB2.

# Changing the clock fuse



STK500 in ISP mode with ATtiny24A

| Fuse | Value |
|---|---|
| SELFPRGEN | ☐ |
| RSTDISBL | ☐ |
| DWEN | ☐ |
| SPIEN | ✓ |
| WDTON | ☐ |
| EESAVE | ☐ |
| BODLEVEL | Brown-out detection disabled |
| CKDIV8 | ✓ |
| CKOUT | ☐ |
| SUT_CKSEL | Int. RC Osc. 8 MHz; Start-up time PWRDWN/RESET: 6 CK/14 CK + 64 m |

Ext. Crystal Osc. 0.4-0.9 MHz; Start-up time PWRDWN/RESET: 1K CK /1
Ext. Crystal Osc. 0.4-0.9 MHz; Start-up time PWRDWN/RESET: 16K CK/
Ext. Crystal Osc. 0.4-0.9 MHz; Start-up time PWRDWN/RESET: 16K CK/
Ext. Crystal Osc. 0.4-0.9 MHz; Start-up time PWRDWN/RESET: 16K CK/
Ext. Crystal Osc. 0.9-3.0 MHz; Start-up time PWRDWN/RESET: 258 CK/
Ext. Crystal Osc. 0.9-3.0 MHz; Start-up time PWRDWN/RESET: 258 CK/
Ext. Crystal Osc. 0.9-3.0 MHz; Start-up time PWRDWN/RESET: 1K CK /1
Ext. Crystal Osc. 0.9-3.0 MHz; Start-up time PWRDWN/RESET: 1K CK /1
Ext. Crystal Osc. 0.9-3.0 MHz; Start-up time PWRDWN/RESET: 1K CK /1
Ext. Crystal Osc. 0.9-3.0 MHz; Start-up time PWRDWN/RESET: 16K CK/
Ext. Crystal Osc. 0.9-3.0 MHz; Start-up time PWRDWN/RESET: 16K CK/
Ext. Crystal Osc. 0.9-3.0 MHz; Start-up time PWRDWN/RESET: 16K CK/
Ext. Crystal Osc. 3.0-8.0 MHz; Start-up time PWRDWN/RESET: 258 CK/
Ext. Crystal Osc. 3.0-8.0 MHz; Start-up time PWRDWN/RESET: 258 CK/
Ext. Crystal Osc. 3.0-8.0 MHz; Start-up time PWRDWN/RESET: 1K CK /1
Ext. Crystal Osc. 3.0-8.0 MHz; Start-up time PWRDWN/RESET: 1K CK /1
Ext. Crystal Osc. 3.0-8.0 MHz; Start-up time PWRDWN/RESET: 1K CK /1
Ext. Crystal Osc. 3.0-8.0 MHz; Start-up time PWRDWN/RESET: 16K CK/
Ext. Crystal Osc. 3.0-8.0 MHz; Start-up time PWRDWN/RESET: 16K CK/
**Ext. Crystal Osc. 3.0-8.0 MHz; Start-up time PWRDWN/RESET: 16K CK/**
Ext. Crystal Osc. 8.0- MHz; Start-up time PWRDWN/RESET: 258 CK/14
Ext. Crystal Osc. 8.0- MHz; Start-up time PWRDWN/RESET: 258 CK/14
Ext. Crystal Osc. 8.0- MHz; Start-up time PWRDWN/RESET: 1K CK /14

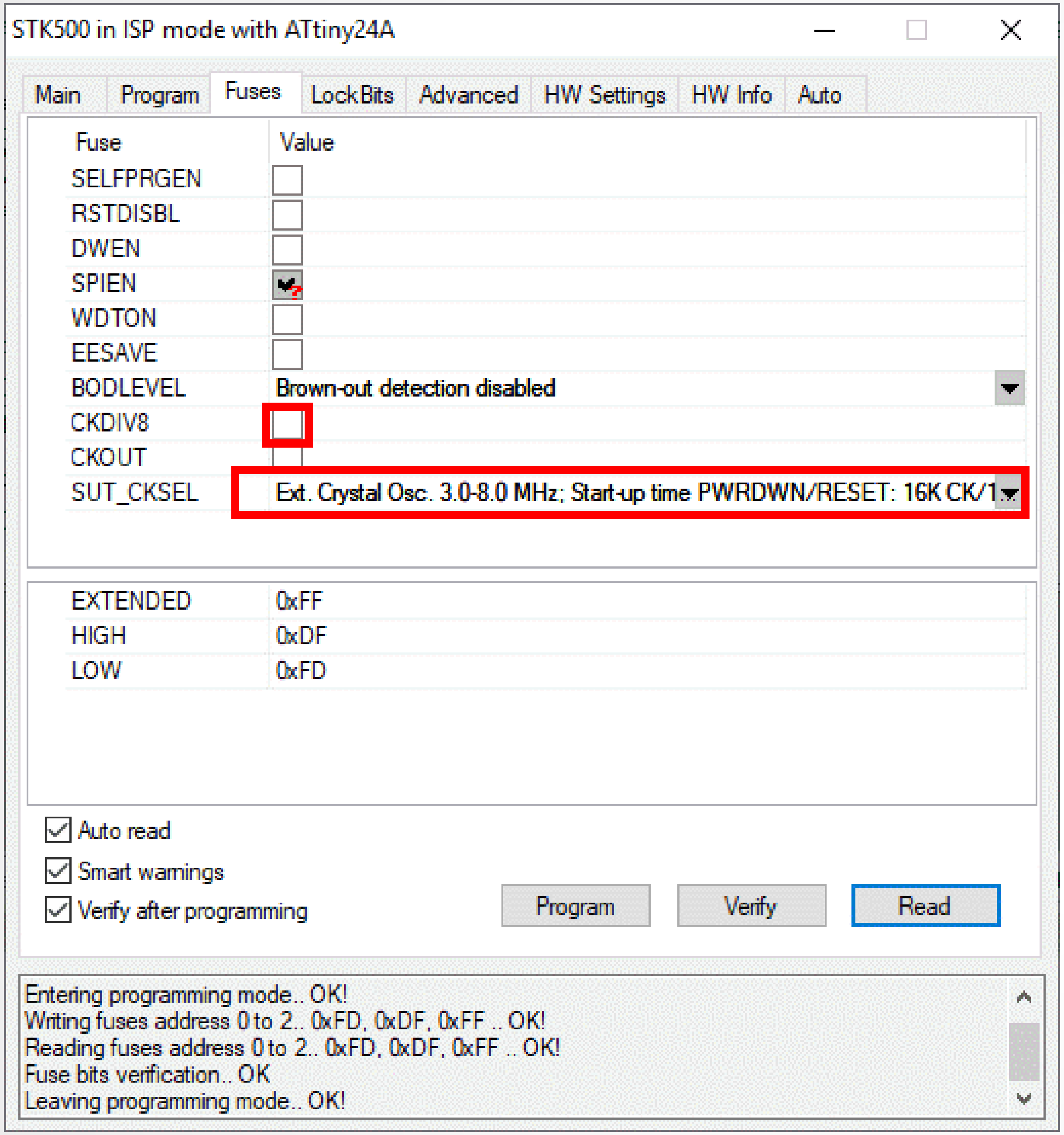
| Fuse |
|---|
| EXTENDED |
| HIGH |
| LOW |

☑ Auto read
☑ Smart warnings
☑ Verify after programm

```
Entering programming mo
Reading fuses address 0
Leaving programming mo
```

- To change the clock fuse, change to the Fuses tab.
- From the dropdown list in SUT_CKSEL select Ext. Crystal Osc. 3.0-8.0 MHz.
- Choose the longest duration from the different start-up times.
- Before pressing Program see the next page.

# Clearing the CLKDIV8 fuse



| Fuse | Value |
|---|---|
| SELFPRGEN | ☐ |
| RSTDISBL | ☐ |
| DWEN | ☐ |
| SPIEN | ✓ |
| WDTON | ☐ |
| EESAVE | ☐ |
| BODLEVEL | Brown-out detection disabled |
| CKDIV8 | ☐ |
| CKOUT | ☐ |
| SUT_CKSEL | Ext. Crystal Osc. 3.0-8.0 MHz; Start-up time PWRDWN/RESET: 16K CK/1 |

| | |
|---|---|
| EXTENDED | 0xFF |
| HIGH | 0xDF |
| LOW | 0xFD |

☑ Auto read
☑ Smart warnings
☑ Verify after programming

```
Entering programming mode.. OK!
Writing fuses address 0 to 2.. 0xFD, 0xDF, 0xFF .. OK!
Reading fuses address 0 to 2.. 0xFD, 0xDF, 0xFF .. OK!
Fuse bits verification.. OK
Leaving programming mode.. OK!
```

- **The crystal oscillator should now appear in the SU_CKSEL field.**
- **Then clear the CKDIV8 fuse. If that would be forgotten, the controller would run with 1/8th of the crystal‘s frequency, 4 MHz / 8 = 500 kHz.**
- **Now you can press Program.**
- **What would happen, if the crystal is not installed? Already verification would fail. In that case either repair the error or attach an RC oscillator with 1 to 4 MHz to XTAL1 and clear the fuse.**

## AVRs are extremely flexible

- With their clever designed hardware AVRs can
  - be tailored to any electronic system,
  - exchange douzends of CMOS/TTL devices in a system by one single IC,
  - be used in small and very small quantities at an extremely low price (from less than 1 € up to 6 € per piece),
  - be programmed within the system, easing debugging and functional changes,
  - ideally be used by amateurs as well as on an industrial scale.

## Questions and tasks in Lecture 5

Question 5-1: The instructions PUSH register throws the register content onto the stack and POP register recalls the last value from the stack.

What happens with a POP when the PUSHed values are exhausted (Which value comes back? Which position has the stack pointer?).

Bonus question: What happens if you PUSH a value to an unitialized stack (the stack pointer is at zero after a RESET).

## Questions and tasks in Lecture 5 - Continued

Task 5-2: Design a schematic and program source code that blinks a 2-pin red/green LED with 0.9 seconds in green and 0.1 seconds in red.

Hint: Forward voltages for such LEDs are around 2.0 Volt. Think about using Zener diodes.

## Questions and tasks in Lecture 5 - Continued

Task 5-3: Design a schematic and write a program that outputs four bits in binary format on PA0 to PA3 (ones: LED on, zeroes: LED off), then waits for a second and counts up.

Bonus question: What happens if 0x10 is reached after 16 seconds? Is it necessary to restart the counter?