# FORTE

## High-Speed USB Programmer


## Reference Manual

ASIX s.r.o.
Na Popelce 38/17
150 00 Prague
Czech Republic

www.asix.net

support@asix.net

sales@asix.net

ASIX s.r.o. reserves the right to make changes to this document, the latest version of which can be found on the Internet.

ASIX s.r.o. renounces responsibility for any damage caused by the use of ASIX s.r.o. products.

# Table of Contents

# 1

# Introduction

This manual describes FORTE, a High-Speed USB programmer and its control software, both manufactured by ASIX.

Chapter 1 gives you ‘quick start’ instructions on how to start working with the programmer, offers examples of its connecting to applications and provides technical specifications.

Chapter 2 focuses on the installation of drivers and software updates.

Chapter 3 introduces the UP program, which is software used for controlling all ASIX programmers. You can find procedures there for setting up the programmer prior to programming, for actual programming and/or verification of devices. It also describes how to control the software from a command line or a DLL library.

Chapter 4 presents the JTAG SVF Player software used for programming devices with the JTAG interface using .svf/ .xsvf files.

Chapter 5 offers tips and tricks in case of experiencing difficulties with programming.

## 1.1 Abbreviations & Terms Used

| | |
|---|---|
| HVP | (High Voltage Programming) is a programming mode in which a higher voltage than the power-supply voltage is applied to pin P in the initial phase. |
| ICSP | (In-Circuit Serial Programming). The meaning of ICSP is identical with the meaning of ISP (In System Programming) in this manual, i.e. device programming done inside a system. |
| LVP | (Low Voltage Programming) is a programming mode in which none of the pins has higher than the power-supply voltage applied. |
| PDI | Program and Debug Interface |
| SBW | (SPY-BI-WIRE) MSP430 microcontroller interface |
| SWD | (Serial Wire Debug) ARM microcontroller interface |
| TPI | Atmel Tiny Programming Interface |
| VCC | If the text features VCC or VDD, they mean the power-supply voltage on the VDD pin, which can serve as either input or output depending on the particular application. |
| VPP | If the text features VPP, it means the programming voltage on pin P of devices with High Voltage Programming. |

The term “file” means a file with data to be programmed in context of this manual, in other cases a particular type of file is specified. A file with .hex extension means Intel-HEX file whilst a file with .bin extension means binary file.

# 2

# FORTE

Thank you for buying the FORTE programmer made by ASIX s.r.o. It was a wise decision. Feel free to contact our technical support in case of any questions or doubts.

## 2.1 Package Content

Your Forte package should include:

- FORTE programmer
- ICSPCAB16 cable
- ICSPCAB8 cable
- USB cable (type A - B)
- Info leaflet

## 2.2 Features

FORTE is a fast and flexible High-Speed USB In-System programmer suitable for programming a range of devices such as microcontrollers, EEPROM or Flash serial memory chips, CPLD, FPGA and many others.

- very fast programmer (30 MHz out, 15 MHz in/out)
- High-Speed USB 2.0 (480 Mbps) interface, programmer powered via USB
- embedded processor pro comprehensive functions
- synchronous and asynchronous programming, JTAG support
- programming voltage from 1.8 V [1] to 5.5 V
- feeds applications with 1.8 V to 5.5 V
- programming interface of 8×I/O + individually configurable pull-up/down resistors
- built-in fast input/output HW protection, independent of the status of the PC
- overcurrent protection on VDD and P sources
- overvoltage protection on VDD pin
- GO button for quick function selection
- more than one simultaneously running programmer per PC, command line support, support for Windows messages and for DLL
- Windows XP or later
- compact

[1] lower speeds can typically be programmed at less than 1.2 V.

# 2.3 Quick Start

Please install the drivers and the UP software prior to the first use of FORTE.

## 2.3.1 Windows

**Administrator rights are required** to run the software for the first time.

Start with installing the UP program. Its installer installs the USB driver for FORTE, too. You can download the installer from www.asix.tech/prg_up_en.html.

Once the installation is complete, connect the FORTE programmer to your computer.

The driver contained in the UP installer is intended for Windows 7 and later. For older Windows operating system versions the driver has to be downloaded from www.asix.net, from download section of the programmer and unpacked somewhere. After the programmer has been connected, the operating system asks for the driver. In "Found New Hardware" dialog the path to the unpacked driver has to be set.

The green ON-LINE LED should turn on after a few moments and the Windows Device Manager should present the programmer as correctly installed.

# 2.4 Use

FORTE is a fast and flexible High-Speed USB programmer suitable for programming of a range of devices such as microcontrollers, EEPROM or Flash serial memory chips, CPLD, FPGA and many others. It is equipped with overcurrent protection at the VDD and VPP sources and with overvoltage protection at the VDD pin.

The programmer is powered via USB. It can feed the application to be programmed with a voltage of 1.8 V to 5.5 V or it can utilize an external application's voltage during programming.

The programmer may run under Windows XP or later.

### Numerous Supported Devices

The list of supported devices includes:

- **Microchip PIC** microcontrollers – devices with serial programming, which include all PIC and dsPIC devices with the exception of several obsolete types.
- Microcontrollers with **ARM** core – such as ATSAM3N2A or LPC2148.
- **Atmel AVR** microcontrollers – all devices supporting "SPI Low Voltage Serial Downloading" such as ATtiny12, AT90S8535 or ATmega128.
- **Atmel ATxmega** microcontrollers – devices programmable via JTAG or PDI interface such as ATxmega32D4 or ATxmega128B1.
- **Atmel AVR32** microcontrollers – AT32UC3A1256, for example.
- **Atmel 8051** microcontrollers – devices that support ISP programming such as AT89S8253, AT89LP4052, AT89LP216 or AT89S2051.
- **Texas Instruments** microcontrollers – 16-bit MSP430, CC430 and CCxxxx with a flash memory, including their fuse programming.
- **Cypress** – PSoC microcontrollers.
- **Serial EEPROM** and **Flash** memory chips - I2C (24LCxx), Microwire (93LCxx) and SPI (25Cxx).
- Devices with **JTAG** interface, for which an SVF or an XSVF file can be created. These include CPLD (such as Xilinx XC95xx and CoolRunner), configuration memory for FPGA (such as Xilinx XC18Vxx and XCFxxS), microcontrollers (such as ATmega128) and others.

This, however, is not an exhausting list of possibilities. Additional types are regularly supplemented in response to customers' interest. New software versions are downloadable from the Internet for free.

## High Speed

Compared to the PRESTO programmer, FORTE features an embedded processor making it possible to perform complex operations with the device being programmed as well as offering a significantly faster output interface. Due to this, devices may be programmed up to the limits of their theoretical performance.

## USB Connection

FORTE is controlled and powered through a USB port. It communicates in the High-Speed mode (480 Mbps) / connected to a USB 2.0 port and in the Full-Speed mode / connected to a USB 1.1 port. This means that connecting the programmer is fast and easy, requiring only a single cable.

## Programming of Placed Devices

ISP (In-System Programming) or the special ICSP (In-Circuit Serial Programming) for the PIC microcontrollers is currently replacing the traditional method in which devices were first programmed and only then placed on a PCB (printed circuit board). Thanks to ISP even SMD devices with an extremely narrowly spaced pins can easily be programmed and their firmware upgraded in already assembled and finished devices.

## Programming of Autonomous Devices

Those who still need to program autonomous devices, i.e. devices not yet placed on a PCB can use the ISP2ZIF adapter featuring a ZIF (zero insertion force) socket.

## Programming Interface

The FORTE programmer features 8 independent I/O pins with individually configurable pull-up/down resistors.

Devices to be programmed are connected through a 16-pin ISP connector, which is backward compatible with the 8-pin ICSP connector for PIC microcontrollers. VPP can be set to any value between 6.5 V and 17 V.

Connection to a remote application through a 16-wire ribbon cable has been improved by interleaving its individual signals with ground.

## Power Supply From Application

The VDD output can serve as an input using the power from the application for feeding the output buffers, or it can become an output and as such to provide voltage for the application.

FORTE can power applications with a voltage from 1.8 V to 5.5 V. A controlled discharge resistor is incorporated for faster transitions between different programming phases when capacitors in the application need to be discharged all the way down to 0 V.

FORTE includes a built-in “voltmeter” at pins P and VDD.

## Overcurrent Protection

FORTE includes a fast-responding hardware overcurrent protection at pins P and VDD, which is independent of the state of the controlling PC.

## User Interface

The programmer status is clearly indicated by two LEDs. ON-LINE (green LED) informs of connecting to USB while ACTIVE (two-state yellow/red LED) signals an activity or a faulty state (error) of the programming interface.

The GO button significantly increases the operator’s comfort in repeated programming. It starts programming or other user-defined commands

## Software

The UP program is a basic software tool for working with FORTE, also compatible with the PRESTO programmer.

Apart from standard commands, UP provides numerous above-standard functions, which broaden the programmer’s applications and simplify its operation. These include the possibility to define projects, existence of adjustable parameters when running from the

command line providing for unattended programmer operation in routine programming, environment personalization incl. keyboard shortcuts, automatic generation of serial numbers, etc.

UP has been developed for Windows XP or later.

Modified drivers developed by FTDI are used for communication through USB.

Devices with the JTAG interface, for which an SVF or an XSVF file can be created, can be programmed by means of the JTAG SVF Player software.

The voltage at the VDD pin is permanently displayed in UP for greater comfort and for monitoring of the operation. The reset signal is conveniently controlled by a single button.

# 2.5 Controls and Connectors


*Fig. 2: The FORTE programmer*

FORTE features two LEDs, a button, a connector for linking to USB and a programming connector.

## 2.5.1 Programming Connector

The programming connector is a 16 way double row plug with pin No. 3 (in standard numbering) missing. Spacing between pins is 2.54 mm.

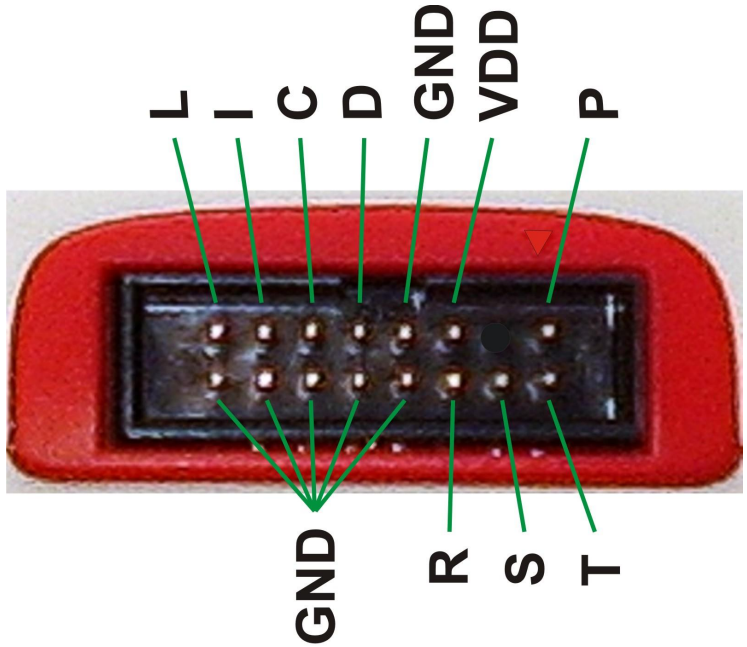


*Fig. 3: Programming connector*

| Pin | Type | Description |
|---|---|---|
| P | I/O, VPP | signal input/output or VPP output |
| VDD | PWR | power input/output |
| GND | PWR | signal grounding |
| D, C, I, L, T, S, R | I/O | signal input/output |

*Table 1: programming connector*

### 2.5.2 GO Button

This button simplifies work with the application being programmed. It triggers device programming or another pre-programmed function. For further information on settings see Setting the GO Button.

### 2.5.3 LED Indicators

#### ACTIVE

This two-state LED informs you of the equipment status.

The yellow color means that there is ongoing communication between the programmer and a device.

A red color signals an error state.

#### ON-LINE

The green LED informs you that FORTE is connected to a computer and that the computer and the programmer “understand each other”, i.e. drivers are installed and work correctly.

### 2.5.4 USB Connector

A standard USB (type B) connector is provided for connecting to a computer. The programmer uses the USB High-Speed (480 Mbps) interface for communication.

## 2.6 Connecting to Application

The programmer should be connected to an application to be programmed with an ICSPCAB8 or ICSBCAB16 cable. The cables are designed for 2.54 mm spacing.

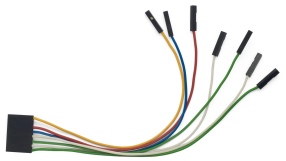

*Fig. 4: ICSPCAB8*

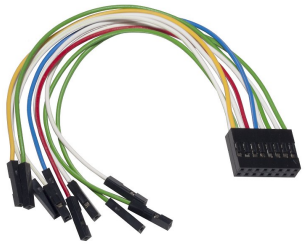

*Fig. 5: ICSPCAB16*

The connectors are specified in the next chapter.

### 2.6.1 Custom-made Connecting Cable

Should an application to be programmed have a non-compatible type of connector for linking to the programmer, the customer can make his/her own programming cable. Its length should not exceed 15 cm.

This is where interleaving of individual signals with GND in a 16-wire ribbon cable can be used conveniently. In order to utilize this ground-interleaving, all GND signals must be actively connected on the application's side.

The following table lists markings of connectors by FCI Electronics suitable for making a custom cable. Of course, it is possible to use any similar ones:

| FCI marking | Description |
|---|---|
| 65039-036LF | housing, 1 pin |
| 65039-029LF | housing, 1 x 8 pins |
| 65043-029LF | housing, 2 x 8 pins |
| 47217-000LF | pin |

*Table 2: ICSP cable - material list*

A cable with a cross-section between 0.1 and 0.3 mm2 may be used for making the custom connecting cable.

## 2.6.2 Programming in ZIF Socket

If programming of autonomous device is required, i.e. those that are only later connected to an application, it is possible by means of our optional ISP2ZIF accessory.

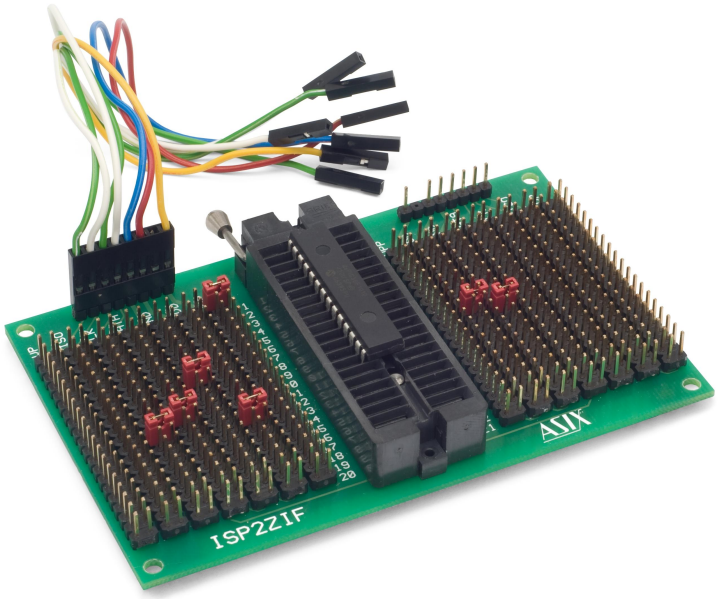


*Fig. 6: ISP2ZIF*

ISP2ZIF consists of a zero insertion force (=ZIF) socket and an ICSP connector for connecting to the programmer, which can also provide voltage for feeding the program circuitry.

## 2.6.3 Connecting Procedure

The correct procedure for connecting the programmer: first connect FORTE to the target application, then connect FORTE to the USB and finally turn on the application's power supply.

Please make sure that the GND of the application, the programmer and the USB are interconnected before signals and the power are applied.

**Important warning**

If an application is powered by a switched power source or is not grounded, a significant voltage difference may appear between the programmer's ground and the application's ground. This could damage the programmer.

The simplest way of connecting the GND prior to the other signals is to ground the application before connecting it to the programmer. This can, for example, be achieved by making the GND pin of the application's ICSP connector longer than the other pins. It will make sure that both grounds are interconnected first.

# Table of Connections

| Pin | AVR | AVR TPI | ATxmega PDI | 8051 |
|---|---|---|---|---|
| **P** | RESET | RESET | | RESET |
| **-** | | | | |
| **VDD** | VCC | VCC | VCC | VCC |
| **GND** | GND | GND | GND | GND |
| **D** | MOSI | TPIDATA | PDI_DATA | MOSI |
| **C** | SCK | TPICLK | PDI_CLK | SCK |
| **I** | MISO | | | MISO |
| **L** | | | | SS |
| **T** | | | | |
| **S** | | | | |
| **R** | | | | |

*Table 3: Connection list No.1*

| Pin | PIC | MSP430 | MSP430 SBW | TI CCxxxx |
|---|---|---|---|---|
| **P** | MCLR | TEST / VPP | VPP | RESET |
| **-** | | | | |
| **VDD** | VDD | VCC | VCC | VDD |
| **GND** | VSS | VSS | VSS | GND |
| **D** | PGD | TDI | SBWTDIO | Debug_data |
| **C** | PGC | TCK | SBWTCK | Debug_clock |
| **I** | | TDO | | |
| **L** | LVP | TMS | | |
| **T** | | | | |
| **S** | | | | |
| **R** | | RESET | | |

*Table 4: Connection list No.2*

| Pin | I²C | SPI | QUAD SPI[1] | Microwire | UNI/O |
|---|---|---|---|---|---|
| **P** | | - CS | - CS | CS | |
| **-** | | | | | |
| **VDD** | VDD | VDD | VDD | VDD | VCC |
| **GND** | GND | GND | GND | GND | VSS |
| **D** | SDA | SI | SIO0 | DI | SCIO |
| **C** | SCL | SCK | SCK | CLK | |
| **I** | | SO | SIO1 | DO | |
| **L** | | | SIO2 | ORG/(PRE) | |
| **T** | | | | | |
| **S** | | | SIO3 | | |
| **R** | | | | | |

*Table 5: Connection list No.3*

| Pin | PSoC | 1-Wire | ARM SWD | LPC UART |
|---|---|---|---|---|
| **P** | XRST | IO[2] | NRST | RESET |
| **-** | | | | |
| **VDD** | VDD | VDD | VDD | VDD |
| **GND** | VSS | GND | GND | VSS |
| **D** | ISSP-DATA | | SWDIO | RXD |
| **C** | ISSP-SCLK | | SWCLK | TXD |
| **I** | | | | ISP |
| **L** | | | | |
| **T** | | | | |
| **S** | | | | |
| **R** | | | | |

*Table 6: Connection list No.4*

| Pin | JTAG | HCS08 | HCSxxx | C2 |
|---|---|---|---|---|
| **P** | USR | RESET | | |
| **-** | | | | |
| **VDD** | VDD | VDD | VDD | VDD |
| **GND** | GND | VSS | VSS | GND |
| **D** | TDI | BKGD | Data | C2D |
| **C** | TCK | | Clock | C2CK |
| **I** | TDO | | | |
| **L** | TMS | | | |
| **T** | | | | |
| **S** | | | | |
| **R** | | | | |

*Table 7: Connection list No.5*

| Pin | RL78 | RX600 | AT89C51CC01UA |
|---|---|---|---|
| **P** | RESET | RES# | RESET |
| **-** | | | |
| **VDD** | VDD | VCC | VCC |
| **GND** | VSS | VSS | VSS |
| **D** | TOOL0 | RXD1 | RxD |
| **C** | | TXD1 | TxD |
| **I** | | MD | EA |
| **L** | | PC7/UB | PSEN |
| **T** | | | |
| **S** | | | |
| **R** | | | ALE |

*Table 8: Connection list No.6*

| Pin | AVR HVP | AVR UPDI | AVR UPDI+RST |
|---|---|---|---|
| **P** | RESET | UPDI / RESET | RESET |
| **-** | | | |
| **VDD** | VCC | VDD | VDD |
| **GND** | GND | GND | GND |
| **D** | SDI | | UPDI |
| **C** | SDO | | |
| **I** | SII | | |
| **L** | SCI | | |
| **T** | | | |
| **S** | | | |
| **R** | | | |

*Table 9: Connection list No.7*

| Pin | CH32V003 | SPD5118 | Z8F |
|---|---|---|---|
| **P** | NRST | | RESET |
| **-** | | | |
| **VDD** | VDD | VDDSPD | VDD |
| **GND** | VSS | GND | VSS |
| **D** | SWIO | HSDA | DBG |
| **C** | | HSCL | |
| **I** | | | |
| **L** | | HSA | |
| **T** | | | |
| **S** | | | |
| **R** | | | |

*Table 10: Connection list No.8*

[1] It is recommended to use a flat ribbon cable and to connect all its GND wires.

[2] An external pull-up resistor or possibly a Schottky diode should be connected for 1-Wire components – see chapter 1-Wire Interface.

## 2.6.4 Connection Examples

The following text presents examples of connections between FORTE and the device being programmed. We use a notation according to the manufacturer datasheet of each particular device.

## RL78 Microcontrollers

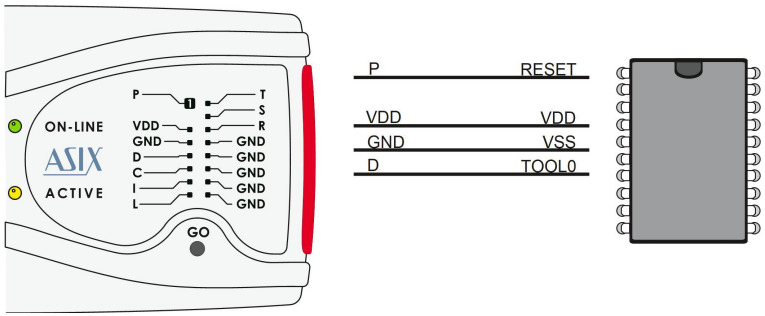


*Fig. 7: Renesas RL78 microcontroller*

## RX600 Microcontrollers

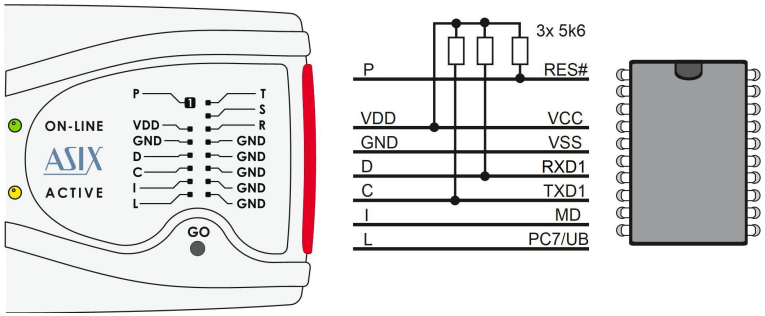


*Fig. 8: Renesas RX600 Microcontroller*

1) The RX600 MCUs are programmed in Boot mode.

2) For programming of OSIS(ID) and device locking the settings in the "FORTE programmer settings" window have to be used.

## PIC Microcontrollers

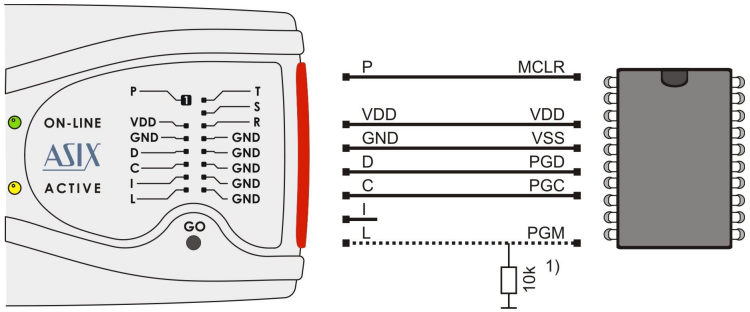


*Fig. 9: PIC microcontroller*

1) Not all devices have the PGM pin. The PGM pin can be connected to programmer's L pin, to VSS via a pull-down resistor (for HVP programming) or to VDD via a pull-up resistor (for LVP programming).

2) The whole device must be erased before programming if code/main memory protection (CP) or data memory protection (CPD) is active.

3) Some devices cannot be erased when CP or CPD is active if the power-supply voltage is less than 5 V.

4) If a microcontroller has more than one power-supply VDD or VSS pin, all of them must be connected including the AVDD and AVSS pins.

5) If the LVP mode is used, it is recommended to check whether the LVP fuse is still set after erasing the part.

6) Programming of PIC32 devices is supported by means of the ICSP interface.

7) Devices with an ICPORT fuse must have the dedicated ICSP port off for LVP programming.

8) PIC24 and dsPIC33 devices may be programmed using PE (Programming Executive) or using the standard method. Programming by means of PE is usually faster.

9) When the MCU debug mode is enabled using a fuse, the programmer cannot communicate with it.

# AVR Microcontrollers

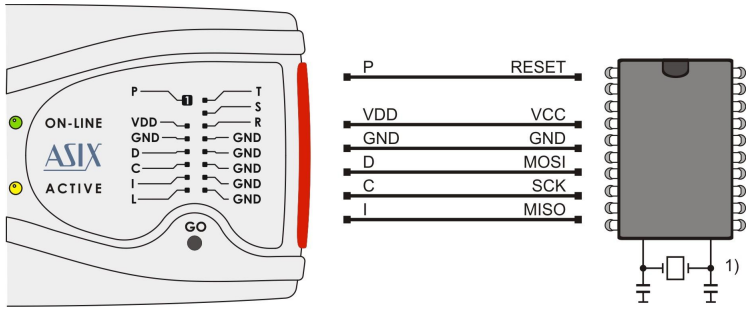


*Fig. 10: AVR microcontroller*

1) A source of the clock signal, which is set in the device or which will be set by fuses during programming must be connected to the device. A crystal must be connected if set up as the clock source.

2) Device fuses have been set up by the producer to the internal oscillator with a frequency of 1 MHz. In the initial programming, the device should be programmed with “Oscillator frequency” set up at “>750 kHz” or lower in the “**FORTE programmer settings**” window.

3) Not all AVR devices allow use of a crystal (e.g. ATtiny13, ATtiny15).

4) After the device fuses are correctly set up, right-click (i.e. using the right mouse button) inside the **Configuration** window and choose **Learn fuses**. This saves the fuses in the up.ini file or in the project if used. This is necessary due to the fact that .hex files for AVR microcontrollers themselves do not contain configuration fuses. If a device is programmed from the command line, a .ppr project file containing saved fuses needs to be used.

5) Ticking the **Open file with data memory automatically** option in the **File** menu loads data for the data memory simultaneously with the code/ main memory data.

6) Use the EESAVE fuse if preservation of data memory is required. If the EESAVE is active, choose **Program all except data memory** for programming, otherwise a warning appears at this place (blank check of data memory).

7) HPRAVR is an optional accessory for programming AVR microcontrollers in applications with the ISP10PIN standard connector on the device's side.

8) Some AVR devices have their ISP interface provided at different pins than the SPI interface. Further information can be found in the device data sheet (in the Serial Downloading section).

# AVR in HVP mode

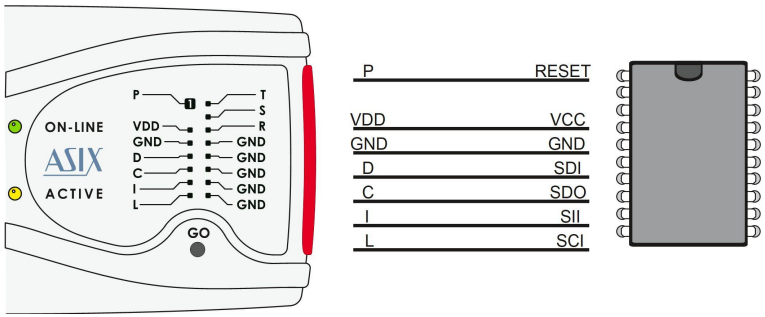


*Fig. 11: AVR in HVP mode (e.g. ATtiny841)*

1) Some devices require also other pins to be connected to GND, this can be found in a picture in the device datasheet in chapter "High-voltage Serial programming".

# AVR with TPI Interface (e.g. ATtiny10)

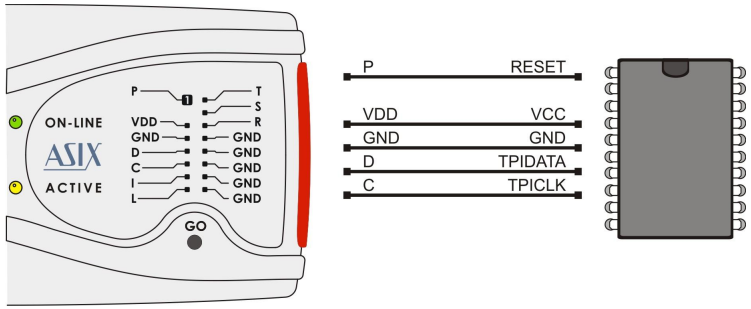


*Fig. 12: AVR microcontroller, TPI interface*

# ATxmega with PDI Interface

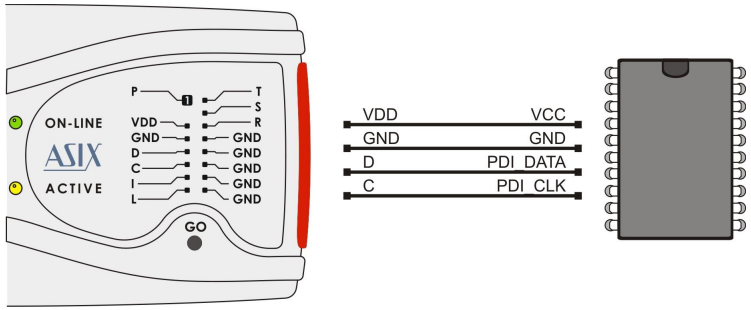


*Fig. 13: ATxmega microcontroller, PDI interface*

1) For programming that uses the JTAG interface, devices must be connected as described in the JTAG Interface section.

# AVR with UPDI and RESET pins

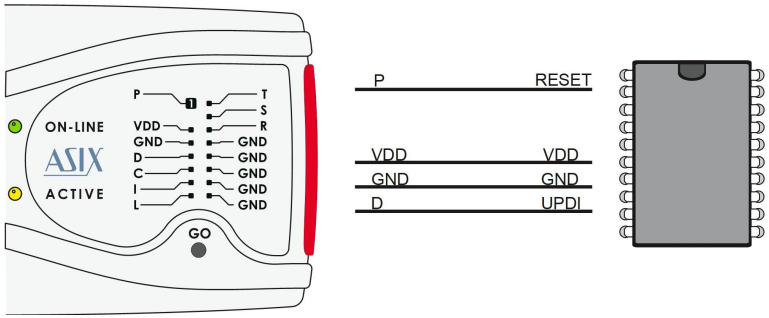


*Fig. 14: AVR microcontroller with separated UPDI and RESET pins*

# AVR with UPDI Interface

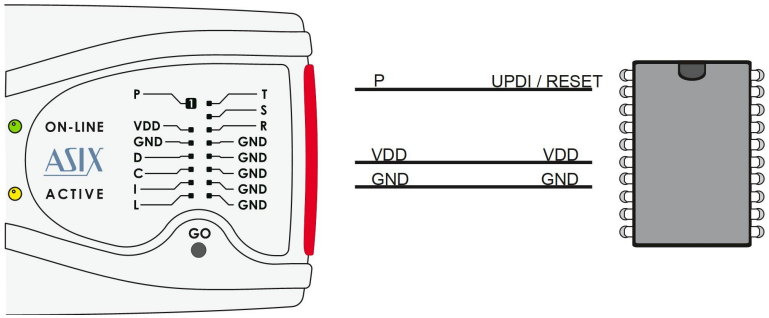


*Fig. 15: AVR microcontroller, UPDI interface*

1) On UPDI pin a 12 V pulse is used to enter the programming mode of ATtiny UPDI.

2) When the programmed device quits the programming mode or the UPDI interface is disabled during programming, a power-on reset of the programmed device has to be done.

3) For AVR with UPDI data files with memories mapped from zero are expected and their fuses save to a project file same as for older AVRs, because Atmel Studio creates the data files this way.

# Atmel 8051

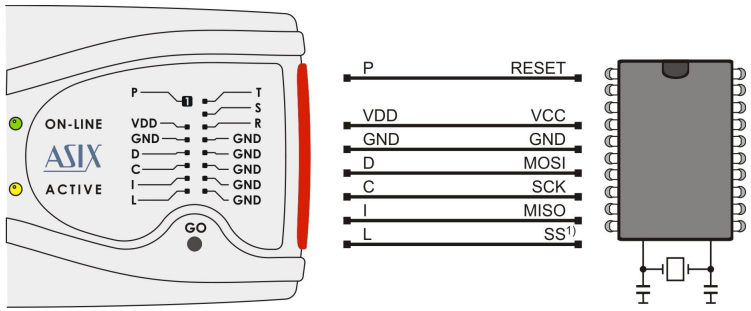


*Fig. 16: Atmel 8051 microcontroller*

1) The SS pin must be connected only for AT89LP2052 / 4052 / 213 / 214 / 216 / 428 / 828 / 6440 / 51RD2 / 51ED2 / 51ID2 / 51RB2 / 51RC2 / 51IC2.

2) AT89LP213, AT89LP214 and AT89LP216 have the inverse reset logic. Thus an appropriate resistor must be pulled-up to VCC.

3) FORTE can not program devices containing the letter “C” in their name, however, it supports devices with “S” in their name, of which some are compatible with the “C” types. For example, AT89C2051 is not supported, but AT89S2051 is.

4) The software assumes that while programming AT89LP52, the device’s POL pin is in logical 1. If POL is in logical 0, the **Inverse RESET** option should be activated in the program. For AT89LP51RD2, AT89LP51ED2, AT89LP51ID2, AT89LP51RB2, AT89LP51RC2 and AT89LP51IC2 the software assumes the POL pin in logical 0.

# AT89C51CC01UA

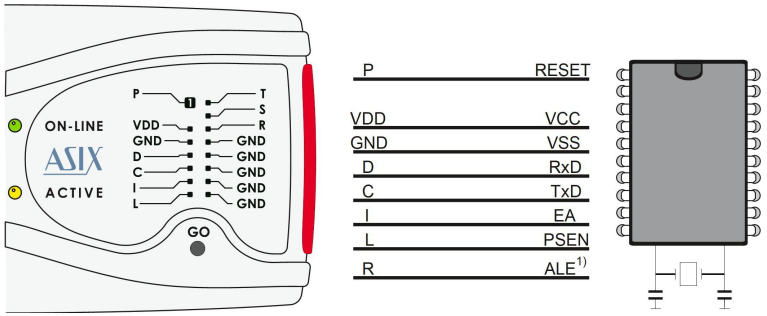


*Fig. 17: Mikrokontrolér AT89C51CC01UA*

1) The ALE pin do not have to be connected, when it is unconnected on the programmed device or when it is in log.1.

# Cypress PSoC

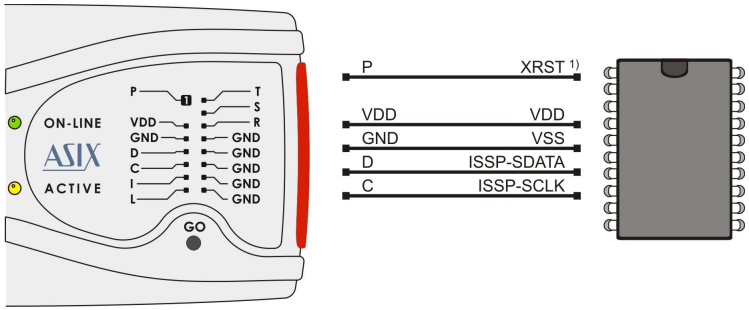


*Fig. 18: Cypress PsoC microcontroller*

1) The way of entering in the programming mode should be set in the **FORTE programmer settings** window. Devices without an XRST pin can only use initialization through the power-on reset (by power supply). Devices with an XRST pin may use both methods, but the method using the reset signal for initialization is better as it can be used in combination with an external power supply.

2) **Algorithm programming** in the **FORTE programmer settings** window should be set in accordance with the power supply cable used.

# MSP430 / CC430 with TEST Pin, JTAG Interface

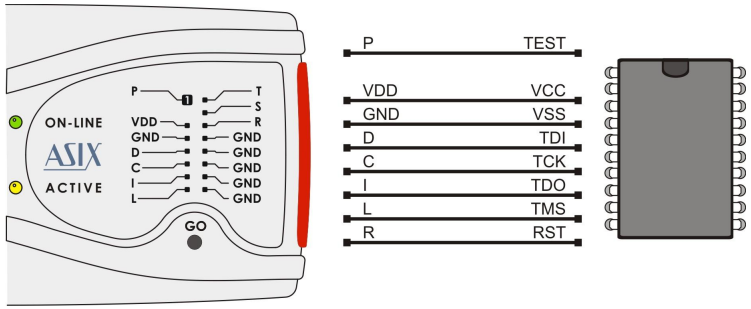


*Fig. 19: MSP430 / CC430 microcontroller, TEST pin, JTAG interface*

1) If the oscillator calibration values are saved in the information memory and this memory is not going to be re-programmed (erased) during programming, the device should be programmed with the **Cal Int. RC** (=calibrated internal RC oscillator) option selected in the **FORTE programmer settings** window. In the other cases **Not Cal Int. RC** (=not calibrated internal RC oscillator) should be selected.

# MSP430 / CC430 without TEST Pin, JTAG Interface

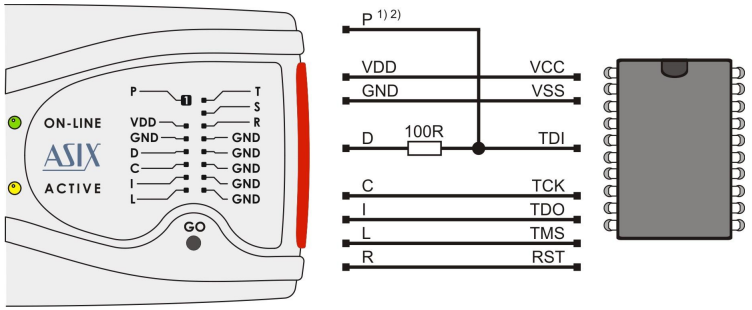


*Fig. 20: MSP430 / CC430 microcontroller, no TEST pin, JTAG interface*

1) Pin P feeds the device with 6.5 V during the fuse programming. If the fuse is not to be programmed, this pin does not have to be connected.

2) The MSP430F5xxx and CC430 devices lock in a different way, i.e. pin P stays disconnected. Even the 100 R resistor can be left out in such a case.

3) If the oscillator calibration values are saved in the information memory and this memory is not going to be re-programmed (erased) during the programming process, the device should be programmed with the **Cal Int. RC** (=calibrated internal RC oscillator) option selected in the **FORTE programmer settings** window. In other cases **Not Cal Int. RC** (=not calibrated internal RC oscillator) should be selected.

# MSP430 / CC430, SBW Interface

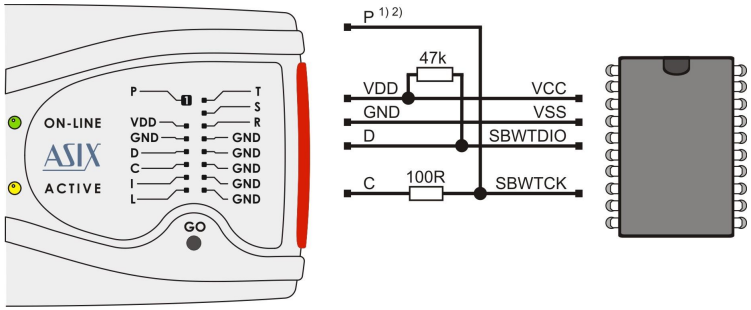


*Fig. 21: MSP430 / CC430 microcontroller, SBW interface*

1) Pin P feeds the device with 6.5 V during the fuse programming. If the fuse is not to be programmed, this ping does not have to be connected.

2) The MSP430F5xxx and CC430 devices lock in a different way, i.e. pin P stays disconnected. Even the 100 R resistor can be left out in such a case.

3) If the oscillator calibration values are saved in the information memory and this memory is not going to be re-programmed (erased) during the programming process, the device should be programmed with the **Cal Int. RC** (=calibrated internal RC oscillator) option selected in the **FORTE programmer settings** window. In the other cases **Not Cal Int. RC** (=not calibrated internal RC oscillator) should be selected. There is no need to select the oscillator for the MSP430F5xxx and CC430 devices.

4) **Speed** in the **FORTE programmer settings** window should be slow down if any external capacitor is connected to the device's **reset** pin.

5) The FORTE programmer also erases the Segment A of the information memory with the **Erase Segment A** option selected.

# TI (Chipcon) CCxxxx

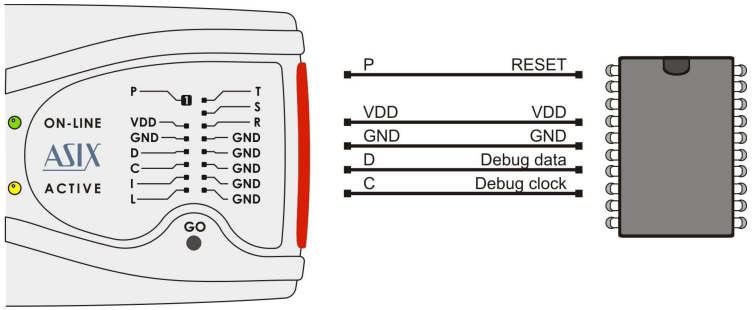


*Fig. 22: (Chipcon) CCxxxx microcontroller*

# STM8

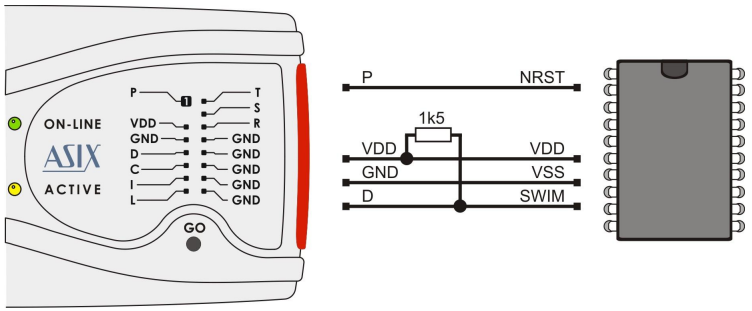


*Fig. 23: STM8 microcontroller*

# ARM with SWD interface

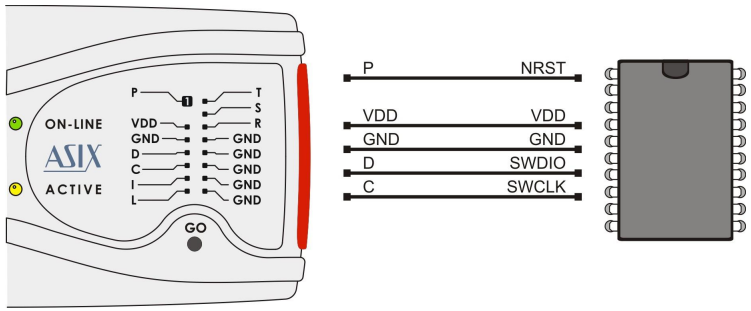


*Fig. 24: ARM with SWD interface*

1) ARM microcontrollers which do not have SWD interface are programmed via JTAG interface.
2) Renesas ARM MCUs with OSIS fuse will lock by programming of the changed OSIS value. When the correct OSIS value is filled for a locked MCU, then it is possible to communicate with the MCU. When the MCU is erased with the correct value filled, the OSIS lock will disable.

# LPCxxxx, UART interface

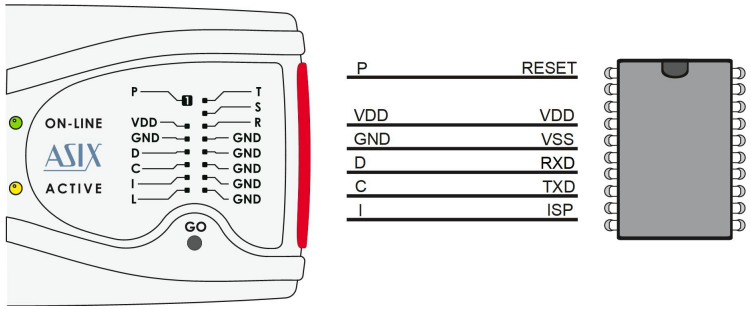


*Fig. 25: LPCxxxx, UART interface*

1) UART programming support is implemented for some LPCxxxx MCUs as an alternative interface to SWD.
2) Signals used as ISP are listed in the table:

| Microcontroller | ISP signal | Other signals |
|---|---|---|
| LPC1102 | - | - |
| LPC11xx | PIO0_1 | - |
| LPC11Cxx | PIO0_1 | PIO0_3 = Log.1 |
| LPC11Uxx, LPC13xx | PIO0_1 | PIO0_3 = Log.0 |
| LPC15xx | ISP_0 | ISP_1 = Log.0 |
| LPC17xx | P2.10 | - |
| LPC18xx | P2_7 | - |
| LPC5411x | PIO0_31 | PIO0_4 = Log.1,<br>PIO1_6 = Log.0 |

*Table 11: ISP signal*

# C8051 and EFM8 with C2 interface

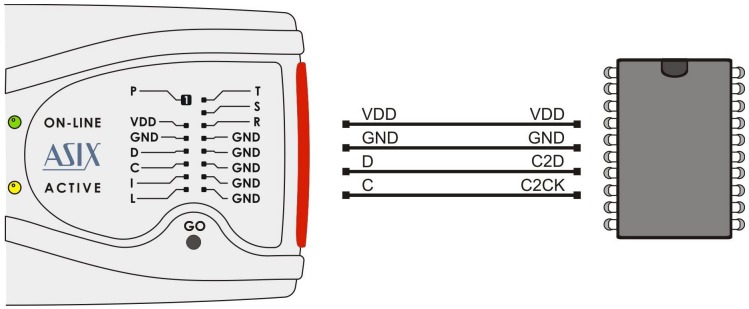


*Fig. 26: C8051 and EFM8 with C2 interface*

# HCS08

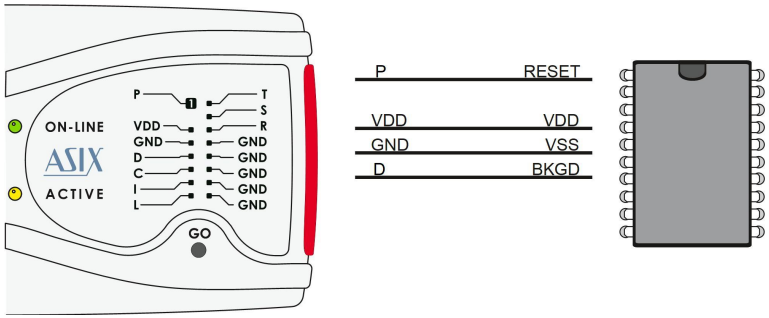


*Fig. 27: HCS08 microcontroller*

1) At some devices the programming mode cannot be entered using their RESET signal, then power-on reset is used. The programmer cannot do the power-on reset when UP is controlled using commandline parameters and external supply voltage is to be used, in such a case the user have to ensure that the BKGD pin is in log. 0 during connection of the supply voltage.

# CH32V003

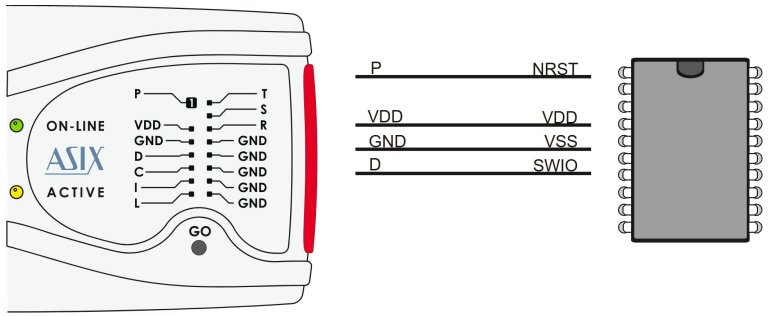


*Fig. 28: CH32V003 microcontroller*

# Z8F

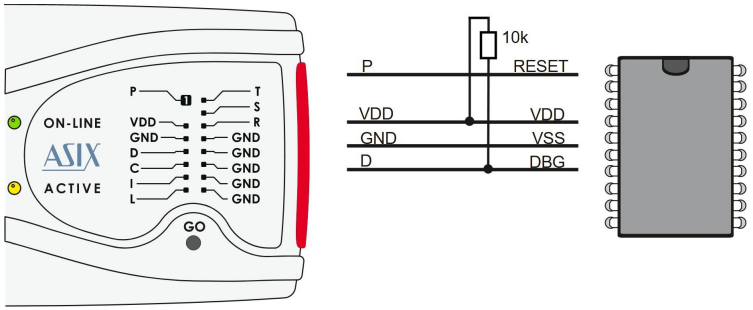


*Fig. 29: Z8F microcontroller*

# I2C Memory Chips

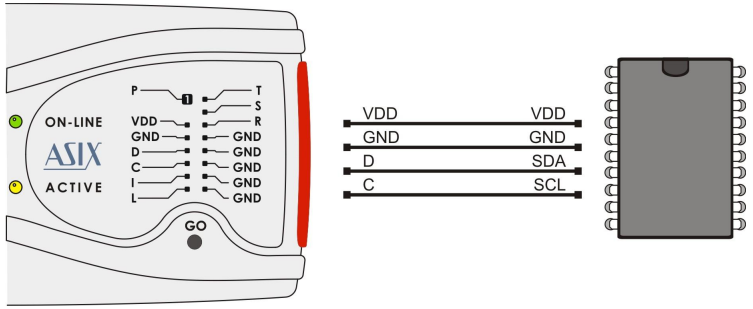


*Fig. 30: I2C memory chips*

1) The programmer uses an internal 2.2 kΩ pull-up resistor on the data wire (SDA) when working with an I2C device. This internal resistor can be disabled at "FORTE programmer settings" window.

2) If the device to be programmed is 24LC(S)21A or 24LC(S)22A, its VCLK pin must be connected to VDD during programming.

3) 34xx02 memory chips need “high” voltage at the A0 pin for commands protecting it against SWP and CSWP recording. The “high” voltage is generated at FORTE P pin, it must be connected to A0 pin in this case. The memory chip A0, A1 and A2 pins must be connected manually according to the selected protection mode.

4) 34AA04 memory needs “high” voltage at the A0 pin for configuration memory programming. The “high” voltage is generated at FORTE P pin, it must be connected to A0 pin in this case.

# SPI Memory Chips

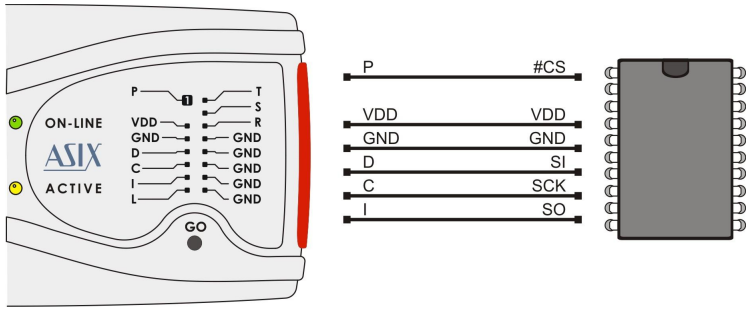


*Fig. 31: SPI memory chips*

1) Some devices have WP, HOLD or RESET pins. All of them must be connected to the required logic level in such a way as not to block the communication or device programming.

Different manufacturers mark the memory chips' SPI pins with different names. Some of them are listed in the following table:

| Name on picture | Atmel, SST | ST |
|---|---|---|
| DI | SI | D |
| DO | SO | Q |
| CLK | SCK | C |
| CS | CS, CE | S |

# QUAD SPI interface

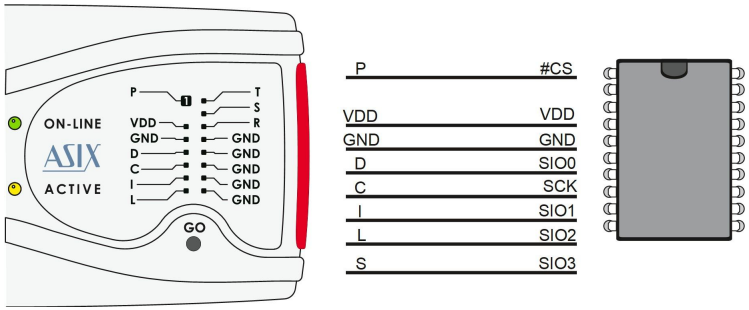


*Fig. 32: QUAD SPI interface*

1) It is recommended to use a flat ribbon cable and to connect all its GND wires.

# Microwire Memory Chips

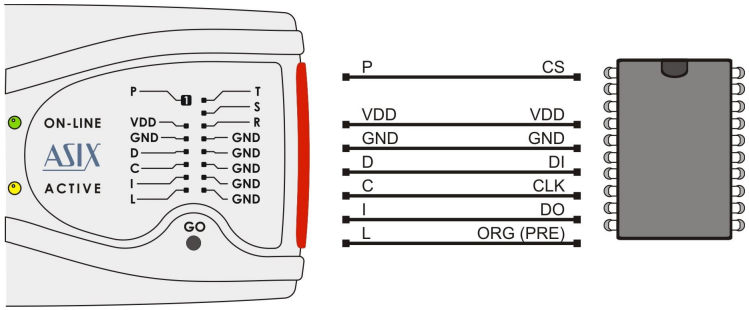


*Fig. 33: Microwire memory chips*

1) Pin L determines the memory organization as either 8-bits or 16-bits per word. The user selects the required organization in the UP program and FORTE then sets this pin to the corresponding logic level. If this memory pin is firmly connected to the correct logic level inside the application, pin L in the programmer remains disconnected.

2) If used in combination with the M93Sx6 memory chip, the programmer's L pin must be connected to the device's PRE pin to serve for selecting the protection register.

# UNI/O Memory Chips

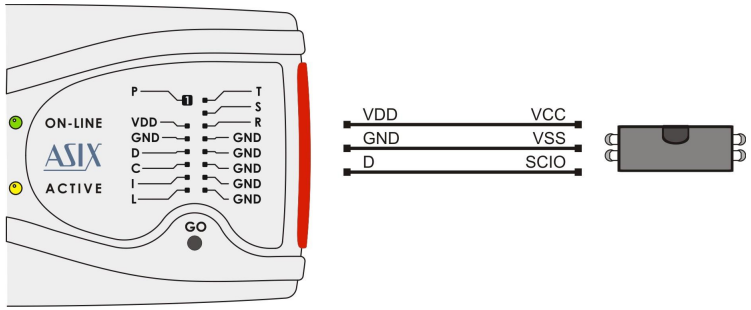


*Fig. 34: UNI/O serial memory chips*

# 1-Wire Interface

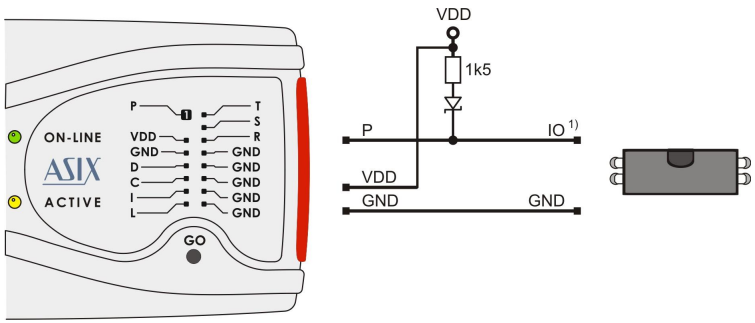


*Fig. 35: Components with 1-WIRE interface*

1) A Schottky diode is necessary only for devices requiring higher than the power-supply voltage for programming, such as DS2505 or DS2406, for example. A pull-up resistor is needed in any case.

2) For DS1821: In thermostat mode, the VDD pin of the device must be connected to the D pin of the programmer. However external supply voltage must not be connected to the D pin of the programmer. It may be connected to VDD pin of the programmer only. In this case the device must be programmed standalone!

3) For DS28E05: This device requires lower value of the pull-up resistor, e.g. 560R.

# JTAG Interface

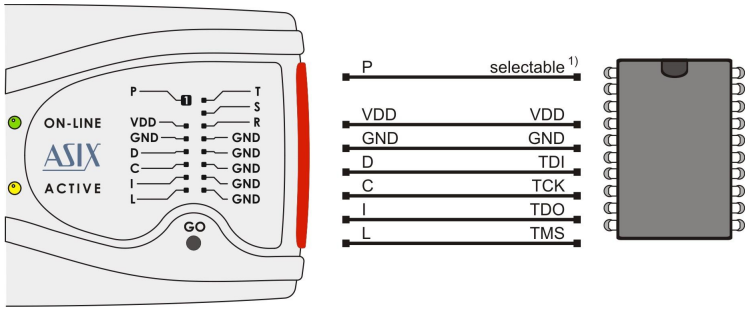


*Fig. 36: Components with JTAG interface*

1) Pin P is configurable in the JTAG Player utility. It can be set to keep the device in reset during programming. This is needed for ATmega devices, for example.

2) **The programmer always uses an external power-supply voltage for SVF or XSVF file programming by means of the JTAG Player utility.**

3) AVR32 microcontrollers are to be programmed from the UP software through the JTAG interface. The device must not be reset during programming.

4) ATxmega microcontrollers with the JTAG interface can be programmed from the UP software through this interface. Pin P is not needed for programming.

5) ARM microcontrollers can be programmed from the UP software through the JTAG interface.

   a) Devices which have SWD interface, e.g. most of Cortex-M3 core chips, are programmed via this interface, see its connection.

   b) Clock source which is connected to the chip and its frequency must be set in the **FORTE programmer settings** window.

   c) Using the programmer P pin is not mandatory for the programming, but it should be connected to the NRST pin of the device providing reset of the device after programming to launch the application program.

   d) **NXP LPC2xxx:** For proper operation, it is required to tie both /RESET and /TRST pins to FORTE P output. Further, it is necessary to provide external pull-down resistor of 4.7 kΩ to 10 kΩ on the /RTCK pin. The LPC2xxx MCUs execute on-chip bootloader after reset. The bootloader then examines pin P0.14 or P2.10 (depends on the part type, see the device datasheet) to decide whether to start user application on continue its operation (log. 1 for user application, log. 0 for the bootloader). Because of this, it is necessary to provide an external pull-up resistor on this pin to start user programmed application.

# HCSxxx

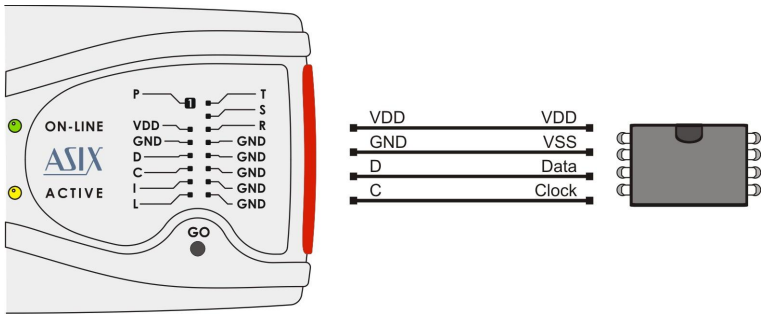


*Fig. 37: HCSxxx*

1) The Clock signal is usually on S2 pin, the Data signal is usually on PWM pin.

# SPD5118

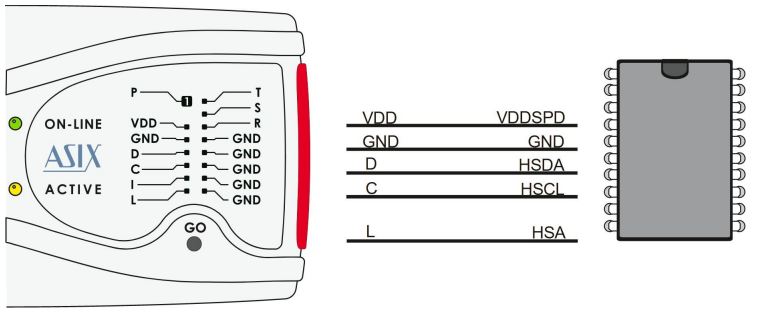


*Fig. 38: SPD5118*

1) In offline mode an iternal supply voltage from the programmer has to be used, because a POR with HSA pin connected to GND is needed. For this mode the HSA pin of the device can be connected to the L pin of the programmer. In normal mode the HSA pin do not have to be connected to the programmer. A write protection can be disabled in the offline mode only in registers of MR12, MR13.

# 2.7 Technical Specifications

## 2.7.1 Limit Values

| | | |
|---|---|---|
| Operating temperature | min. 0 °C | max. +55 °C |
| Storage temperature | min. -40 °C | max. +85 °C |
| Voltage at any pin[1] | min. -0.5 V | max. 6.5 V |
| Maximum current on I/O pin | 50 mA | |
| ESD protection (HBM model) | ±4 kV kontakt | |
| | ±8 kV vzduch | |

*Table 12: Limit values*

[1] Pin P configured as an output has a voltage within a range of +6.5 V to +17 V.

## 2.7.2 Operating Specifications

**Important Warning**
Failure to respect these parameters may damage the programmer or the connected computer.

| | |
|---|---|
| VDD feeding voltage supplied by programmer | 1.8 V to 5.5 V |
| VDD feeding voltage supplied by application | 1.8 V to 5.5 V |
| VDD external feeding voltage for communication at limited speed[1] | 1.2 V to 5.5 V |
| Maximum current drawn from VDD | 100 mA |
| Maximum current drawn from VPP | 100 mA at 7 V |
| | 10 mA at 17 V |
| Maximum current drawn from I/O pin | 4 mA @ VDD = 1.8 V |
| | 16 mA @ VDD = 4.5 V |
| Maximum current drawn from all I/O pins simultaneously | 100 mA |
| Allowed input voltage on pins | 0 to 5.5 V |
| P pin output voltage | Adjustable from 6.5 V to 17 V or logic levels |
| VIL input voltage | max. VDD x 0.3 V |
| VIH input voltage | min. VDD x 0.7 V |
| VOL output voltage | max. 0.55 V @ VDD=4.5 V @ I=4 mA |
| | typ. 0.1 V |
| VOH output voltage | min. 3.8 V @ VDD=4.5 V @ I=4 mA |
| | typ. VDD - 0.1 V |
| Resistance to short circuiting | permanent |

| | |
|---|---|
| Operating system | Windows[2] 32/ 64-bit |
| USB compatibility | USB 2.0 High-Speed (480 Mbps) |
| USB connector | type B |
| Dimensions | 112 x 64 x 22 mm |
| Weight | 60 g |
| Gross weight | 230 g |

*Table 13: Operating specifications*

[1] The programmer can typically communicate at a lower application's voltage at a reduced speed.
[2] Windows XP, Windows Vista, Windows 7, Windows 8, Windows 8.1, Windows 10, Windows 11

## 2.7.3 Declaration of Conformity and RoHS

Declaration of Conformity and RoHS directive documents are available at www.asix.tech in the Documents section.

# 3

# DRIVERS

This chapter deals with driver installation and updates.

## 3.1 Driver Installation

### 3.1.1 Windows Operating Systems

**The user must have administrator rights to install and run the UP program for the first time**. Standard user rights are sufficient for further use.

FORTE drivers are installed automatically as part of the UP program installation.

### Windows 7 and later

Start with installing the UP program. Its installer installs the USB driver for FORTE, too. You can download the installer from www.asix.tech/prg_up_en.html.

Once the installation is complete, connect the FORTE programmer to your computer. The green ON-LINE LED should turn on after a few moments and the Windows Device Manager should present the programmer as correctly installed.

### Older supported Windows versions

The driver contained in the UP installer is intended for Windows 7 and later. For older Windows operating system versions the driver has to be downloaded from www.asix.net, from download section of the programmer and unpacked somewhere. After the programmer has been connected, the operating system asks for the driver. In "Found New Hardware" dialog the path to the unpacked driver has to be set.

During the installation, the operating system will ask whether it should install the software, which has not passed the Microsoft compatibility test for Windows. Click **Continue Anyway**.

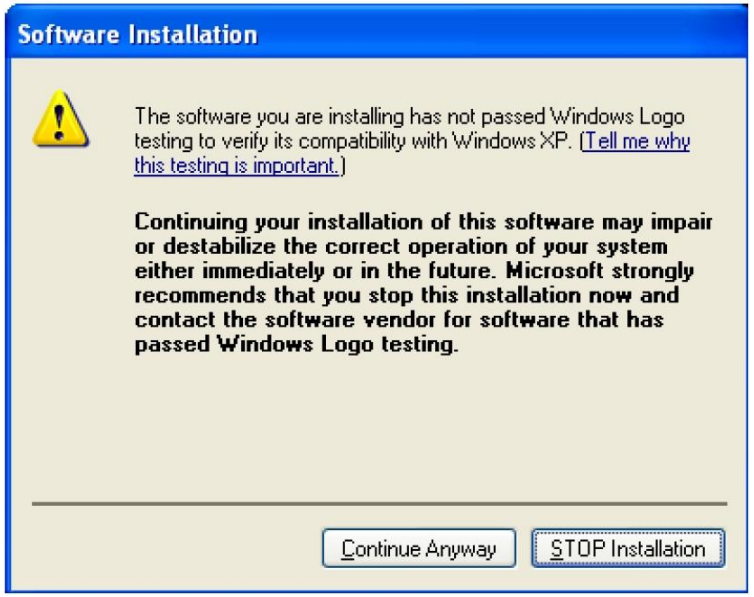


*Fig. 39: Compatibility test dialog window*

The green ON-LINE LED should turn on after a few moments and the Windows Device Manager should present the programmer as correctly installed.

## 3.2 Driver Updating

FORTE communicates with the PC through a USB circuit produced by FTDI www.ftdichip.com, which also develops drivers for these circuits.

The latest drivers are always included in the UP installation pack.

Drivers for the Windows operating system have been stable for a long time and therefore no further updates are usually needed, unless additional applications using the FTDI circuits on the given PC so require.

However, should you need to update your driver, the simplest way of doing so is to update the UP program.

Download the latest UP installer from the web and install the new version without risk of losing your personalized program setup data or data of your projects. The new version will replace the original one.

# 4

# Usage under Linux

**The support of ASIX products in Linux was discontinued.**

**Last version where we tested our products was UBUNTU 20.04.2 LTS and Wine 5.0.**

**Our customers informed us that our products could be successfully used up to Wine 6.0, but not in higher versions.**

The software for the programmers is capable of working under Wine. For USB access it uses libftd2xx.

## Step 1: Install libftd2xx and libftchipid

Install 32-bit versions of libftd2xx and libftchipid by FTDI, even if you use 64-bit kernel. The application is a 32-bit binary and requires 32-bit libraries.

The driver can be found on FTDI web at "Drivers/D2XX Drivers" section.

- Extract ***libftd2xx.so.1.1.0*** (in case of newer version replace 1.1.0 with the latest version) and ***libftchipid*** and copy the files ***libftd2xx.1.1.0.so*** and ***libftchipid0.1.0*** into the directory for 32-bit libraries (typically ***/usr/lib/i386-linux-gnu/***).
- ***ln -s libftd2xx.so.1.1.0 /usr/lib/i386-linux-gnu/libftd2xx.so.1*** (it is typically sufficient to run ldconfig to achieve this)
- ***ln -s libftd2xx.so.1.1.0 /usr/lib/i386-linux-gnu/libftd2xx.so.0*** (must be made manually)
- ***ln -s libftchipid.so.0.1.0 /usr/lib/i386-linux-gnu/libftchipid.so.0*** (it is typically sufficient to run ldconfig to achieve this)
- The library searches for device files in ***/dev/bus/usb***. Please ensure that ***/dev/bus/usb*** directory contains special files to access USB devices.
- Check that your device is recognized by the system (use command ***lsusb***).
- Check your access rights to the corresponding files in ***/dev/bus/usb*** (command ***ls -la /dev/bus/usb/***). Probably you will not have as a user the r+w access rights for these files.
- If you have not access rights and you are using udev:

Add a new file with udev rules to the directory ***/etc/udev/rules.d*** or ***/lib/udev/rules.d*** (Depending on your distribution). Suitable name for this new file is 51-asix_tools.rules. Insert the following lines to this file :

```
SUBSYSTEMS=="usb", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="f1a0", MODE:="0666", SYMLINK+="asix_presto"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="a600", ATTRS{idProduct}=="a000", MODE:="0666", SYMLINK+="asix_sigma"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="a600", ATTRS{idProduct}=="a003", MODE:="0666", SYMLINK+="asix_forte"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="a600", ATTRS{idProduct}=="a004", MODE:="0666", SYMLINK+="asix_omega"
```

VID and PID values can be determined using the listing of connected devices by the lsusb command.

## Step 2: Install wine

It is necessary to install 32-bit version of wine (for example ***wine-1.4:i386***).

The Wine versions over 6.0 are not supported.

## Step 3: Install lin_ftd2xx

The lin_ftd2xx is available at ASIX web.

Check environment variable *WINEDLLPATH*. It should point to directory where are 32-bit wine DLLs, typically */usr/lib/i386-linux-gnu/wine*. Install lin_ftd2xx by ASIX into this directory.

Installation of the Microsoft™ TrueType core fonts are recommended. These fonts may be obtained by installing msttcorefonts package from Ubuntu package repository.

## Note:

Library libftd2xx requires also access rights during opening of the programmer or logic analyzer to all FTDI serial devices to check that this is not the device it wants to open.

# 5

# UP SOFTWARE

UP is a name for control software designed for programmers made by ASIX. The program offers many advanced functions and enables an operator to control the programming process either from the software interface or remotely from the command line utilizing Windows messages and the DLL library. The program runs under Windows.

## 5.1 Abbreviations Used

***Menu → item*** bold italics followed by the arrow sign → make references to particular items in menus, names of tabs (cards) in a particular window.

## 5.2 Installation

Installation is very easy. The installation program can be downloaded from www.asix.tech/prg_up_en.html. Run the installer (UP_xxx_EN.EXE, where xxx represents the version number). There is no need to close other running applications. The installation process takes only a few seconds and requires you to press the Enter key a few times. No modifications to the operating system take place during installation, i.e. no computer restart is needed and the program can be used immediately after completion (by clicking its icon, for example). After the first start, the program asks which language it should use (English or Czech), which programmer to work with (e.g. FORTE) and which port the programmer is connected to.

If needed, the program can be uninstalled using the Add/ Remove Programs icon in the Control Panel or manually by deleting the corresponding directory and desktop shortcut(s).

A previous program version (if it exists) does not have to be removed before installing a newer version. Use of the latest available version is recommended.

## 5.3 Device Programming

The following sections describe the ways of programming devices and things to pay special attention to.

### 5.3.1 Programmer Selection

Before a device can be programmed, the programmer which will be used needs to be selected. Currently either FORTE or PRESTO can be chosen.

Select the programmer by going to ***Options → Select programmer*** or by double-clicking the programmer name displayed in the top right corner of the program window. The following dialog window opens:

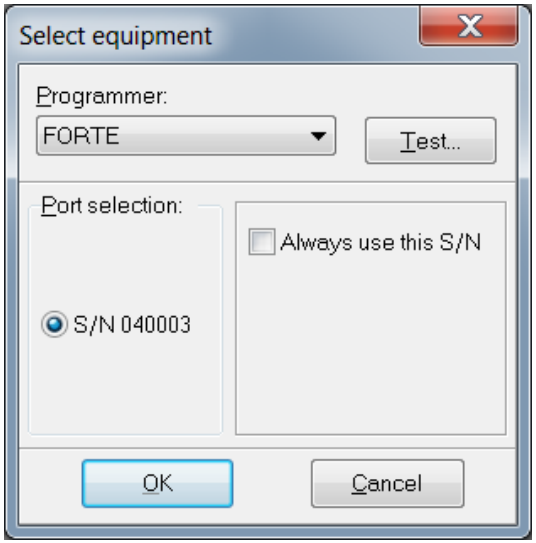


*Fig. 40: Programmer selection*

If the programmer is connected to a computer and no other program communicates with it, its serial number gets displayed. You can test communication with the selected programmer by pressing the **Test...** button.

If the **Always use this S/N** choice is set, the currently selected programmer is used even when in a loaded project file there is defined a different programmer serial number. Serial number defined on the CommandLine has priority before this choice.

## 5.3.2 Projects

It is recommended to use projects for device programing in UP.

Projects save all settings directly associated with particular device programming such as device type, required voltage, verification method, name of file containing the programming data and many other important pieces of information.

A new project can be created by choosing ***File → New project***.

This is followed by selecting a device type, a file with programming data, settings of the device's configuration word, applied voltage and other important parameters.

configuration word, applied voltage and other important parameters.

Once all required selections are made, save the project by choosing ***File → Save project...***

An existing project can be opened by choosing ***File → Open project....***

## 5.3.3 Device Type Selection

Select the device type by choosing ***Device → Select device*** or by double-clicking the device name displayed in the top right corner of the program window.

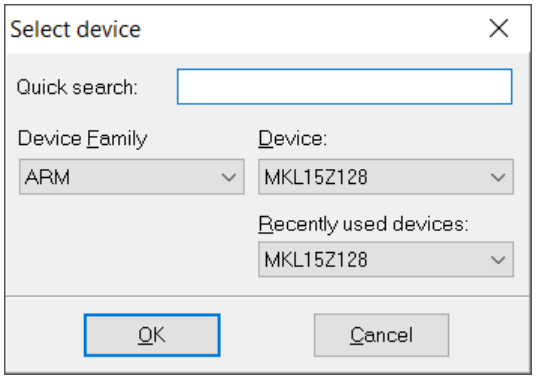


*Fig. 41: Device selection*

A filter is available for sorting devices by families and another one for Quick search. Just start typing an important part of the device's name in its field. This radically reduces the list of devices and speeds up the whole selection process.

The Quick search filter allows use of the question mark as a wild-card for a part of the name, for example **PIC18?20**.

There is also a list of 10 **Recently used devices** where it is possible to select one.

## 5.3.4 Program settings

It is advisable to fine-tune the program behavior by adjusting the settings to suit your particular needs. This can be done by choosing ***Options → Program settings***.

There is a great number of possible settings. If you are not sure that all the settings are correct, use the **Load defaults** button, which returns all the setting to the original (default) state.

If you are updating UP to a new version, all the existing settings will be preserved.

A detailed description of all settings can be found in Menus, so we will pay attention to only a few important settings for now.

# Delay for VDD switching on/off when supplied from programmer

If you supply power to your application from the programmer during programming and UP announces overcurrent on VDD pin due to charging capacitors in the application, it may be advisable to enlarge the switch on time in ***Program settings... → Programming → Delay for VDD switching on/off when supplied from programmer***. This might be necessary to give the programmer enough time to charge the large capacitors in the application.

At the same place it si possible to set the discharge time, when the program announces, that the VDD was not fully discharged.

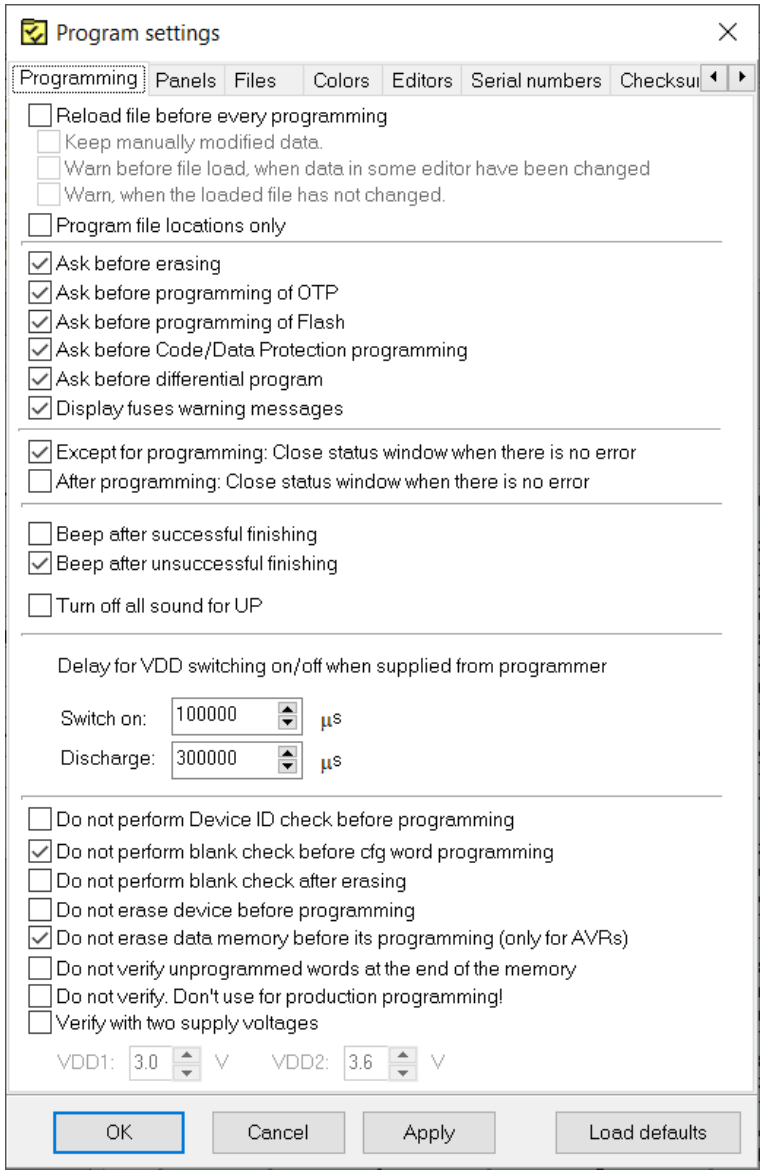


*Fig. 42: Programming settings*

# Production Programming Settings

If you want to perfectly verify your production to make sure that the device is correctly programmed, you can use the ***Program settings → Programming → Verify with two supply voltages*** option and to apply the lowest and the highest voltage allowed for the device as the limits. Some manufacturers recommend this option for the production programming. However, it is only available if the application is supplied with power from FORTE.

***Program settings → Panels → Show mass production counter in status bar*** could be a useful tool visibly monitoring numbers of successfully and unsuccessfully programmed devices. The counter can be reset in ***Device → Program → Mass production → Counter reset***.

***Program settings → Serial numbers → Log to file*** can be used as another convenient assistant. With this option selected, information on the programming progress of individual devices gets recorded (logged) in a selected file.

If you wish to automatically include the serial number during production programming, you have a great range of possibilities concerning what the number should look like and where it should be positioned in the memory. All serial number settings can be found on the ***Serial numbers*** tab.

For further information on serial numbers see Serial Numbers.

# Settings for Programming During Development

If you are frequently changing the content of the programming data such as during application development, for example, you can simplify your work by using the ***GO*** button, which typically serves for programming and verification of the whole memory content. The function of this button is programmable and can be set up under the ***Options → Key shortcuts*** menu in the ***GO button*** section.

The use of the GO button for programming should be accompanied by ***Options → Program settings → Programming → Reload file before every programming*** option.

***Options → Program settings → Programming → Keep manually modified data*** causes that manually modified data are not rewritten using the automatical file reload before programming.

***Options → Program settings → Programming → Warn before file load, when data in some editor have been changed*** causes that a warning appears if data have been changed in any editors and automatic file reloading option is set.

***Options → Program settings → Programming → Warn, when the loaded file has not changed*** causes that a warning appears if the file loaded before programming has not changed.

The following option can also be useful: ***Program settings → Programming → After programming: Close status window if there is no error***.

There are special cases when developers do not have the device's voltage supply pin available in an application powered by an external source. In order to feed the programmer's output circuits, power supply from the programmer must be switched on in such a case and its voltage set at the same value as used in the application. Yet as some voltage still “sneaks” into the programmer from the application through the programming pins, the programmer “sees” that there is some voltage present and refuses to activate its own output voltage. For such a case the following option is available: ***Program settings → Others → Allow internal and external supply voltages collision***.

**WARNING: Activating this option in other situations could damage the programmer!**

# Programmer Settings

Each time a particular programmer is selected (FORTE or PRESTO), the programmer settings window is displayed allowing the user to set the voltage used and some other important programming options.

If you are using an external power source for feeding the application, the **During programming** option must not be checked.

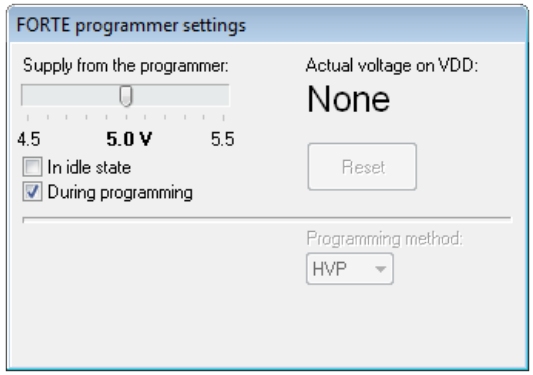


*Fig. 43: Programmer settings*

The current voltage is continuously displayed in the top right corner of this window.

Should an error i.e. overload occur, a warning appears at this place.

The output voltage in the idle state if it is turn on can be changed only if it is allowed under ***Options → Program settings → Others → Allow to change supply voltage level when it is on*** (FORTE only).

If a voltage is present at the VDD pin, you can stop the running microcontroller application by pressing the **Reset** button and restart it by pressing the same button again.

# Fuses and Working with Them

Characteristics of the device (fuse) to be programmed can be set in the **Configuration** window. Changes in fuses can be saved by choosing ***File → Save*** or ***File → Save project....***

If the given .hex file of device includes definitions of its fuses, fuses are saved in it.

Fuses are saved together with a project in the case of Atmel microcontrollers and memory chips. If you are working with these, right-click the **Configuration** window once you have set the fuses and use the **Learn fuses** option.

If the ***Options → Program settings → Programming → Reload file before every programming*** option is active, the software reloads current file after the **Program** button is pressed. If, however, fuses are not saved in current file and no project is created from which fuses could be used, the configuration memory initializes into the default state before each programming procedure. This can be avoided by deactivating the ***Program settings → Files → Erase configuration memory before file reading*** option.

Many devices have specific requirements for fuse settings. Further information on how to correctly set their fuses can be found in the particular device data sheet.

## 5.3.5 Programming

Select the file that you want to program by choosing ***File → Open....***

Once the file opens, you can see the current code/main memory, data memory and configuration memory (fuses). If you cannot see these windows, activate them under the **View** menu.

The memory being programmed can be manually modified at any time by simply marking the required location and rewriting its value from the keyboard. If you

want to save such a modification, choose ***File → Save, File → Save as... or File → Export data memory to file....***

**Important warning**

It is recommended to check the programmer settings and fuse settings before starting the actual programming as an error in these parameters could damage the device or even the programmer.

The programming process is triggered by choosing ***Device → Program*** or by clicking the ***Program*** button.

In some devices, the system checks the Device ID (device electronic signature) and the code/data protection bits before launching the programming process. If the ID does not match the selected device type, a warning message is displayed.

Displaying of this warning message is frequently caused by a fault in the interconnection of the device and the programmer.

If things are in order, the programmer performs the following operations: erases the device, checks the erasure, programs the device and checks the programmed device.

If only a particular memory of a microcontroller needs to be programmed, it can be done by choosing the corresponding item in the ***Device → Program*** menu or by clicking the pull-down arrow next to the ***Program*** button on the button bar. Depending on the device type, the following options may be available: program code/ main memory, program data (EEPROM) memory, program configuration memory or program all.

### Differential Programming

The ***Device → Program*** menu offers a possibility of differential programming, provided the device to be programmed supports it. If selected, the existing memory content is read first and only cells that differ are then programmed.

Differential programming is useful for development during which content of the programming data changes very often, but the changes are tiny. As only changed cells are overwritten, the differential programming is advisable for devices with a low number of writing cycles. It can also be faster than the conventional re-writing of the whole memory content.

## 5.4 Further Features

The following section focuses on selected additional functions of UP available for device programming.

### 5.4.1 Setting the GO Button

ASIX programmers feature a GO button, which allows the user to trigger the programming process without a computer mouse or keyboard.

The function of the GO button can be set under the ***Options → Key shortcuts*** menu in the ***GO button*** field to suit user's needs.

The UP software must be running if the user wants to use the GO button, but may be minimized on the screen.

Additional settings associated with the GO button can be found in Settings for Programming During Development.

### 5.4.2 Mass Production

The mass production function is available under the ***Device → Program → Mass production*** menu. It can also be called up by clicking the arrow next to the ***Program*** button on the toolbar.

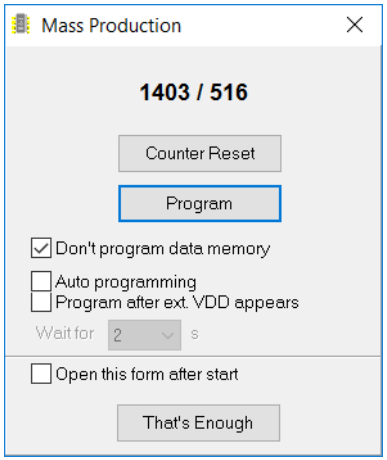


*Fig. 44: Mass production*

The actual programming can be triggered by clicking the ***Program*** button in the ***Mass production*** dialog window.

The function of this button is identical to the ***Program all*** or ***Program all except data memory*** options depending on the state of the ***Don't program data EEPROM*** checkbox in the ***Mass production*** window.

This dialog window also displays a counter of devices programmed. Depending on the program settings, the counter can also be displayed in the status bar. For further information on settings see Production Programming Settings.

The counter displays the number of programmed devices in both the mass production mode and the standard mode.

Pressing the **Counter reset** button zeroes all the mass production counters. This operation cannot be undone.

The choice "Auto programming" causes that the device is programmed after it has been connected, the connection test is done by reading the Device ID.

The programming can be launched by connecting external power supply (VDD).

When the "Open this form after start" setting is enabled, the form opens after program start or after project file open, if it is used.

## 5.4.3 Serial Numbers

The Serial Numbers function programs the serial number or another sequence of characters in the selected memory location.

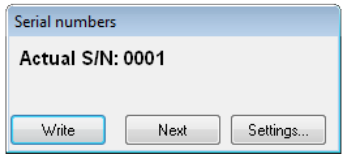


*Fig. 45: Serial numbers*

Once the serial numbers are activated and their type set up under ***Options → Program settings → Serial numbers***, a window with the current serial number opens offering the possibility to manually write a serial number in the memory HEX editor, to specify where it should be written and if the system should proceed to the next sequential number.

Serial numbers can be:

- **Computed**

The computed serial numbers must always be written in the same selected position in the device such as the code/main memory, data memory or an ID position, for example. The serial number is always expected to be a number in decimal or hexadecimal format and may be coded as a 4-bit combination (one to four characters per word) or as an ASCII character (one or two ASCII characters per word). If the code/main memory is used for Microchip microcontrollers, you can choose RETLW instructions. Then RETLW instructions are written in the given address of the code/main memory together with a parameter corresponding with the serial number.

- **Taken from File**

A single serial number can be distributed in more than

one device memories (the serial number itself directly to the program, the equipment address to the data memory, and the serial number again to ID positions in order to be able to read it from a locked device, for example).

**Note:** A word is understood as one memory position.

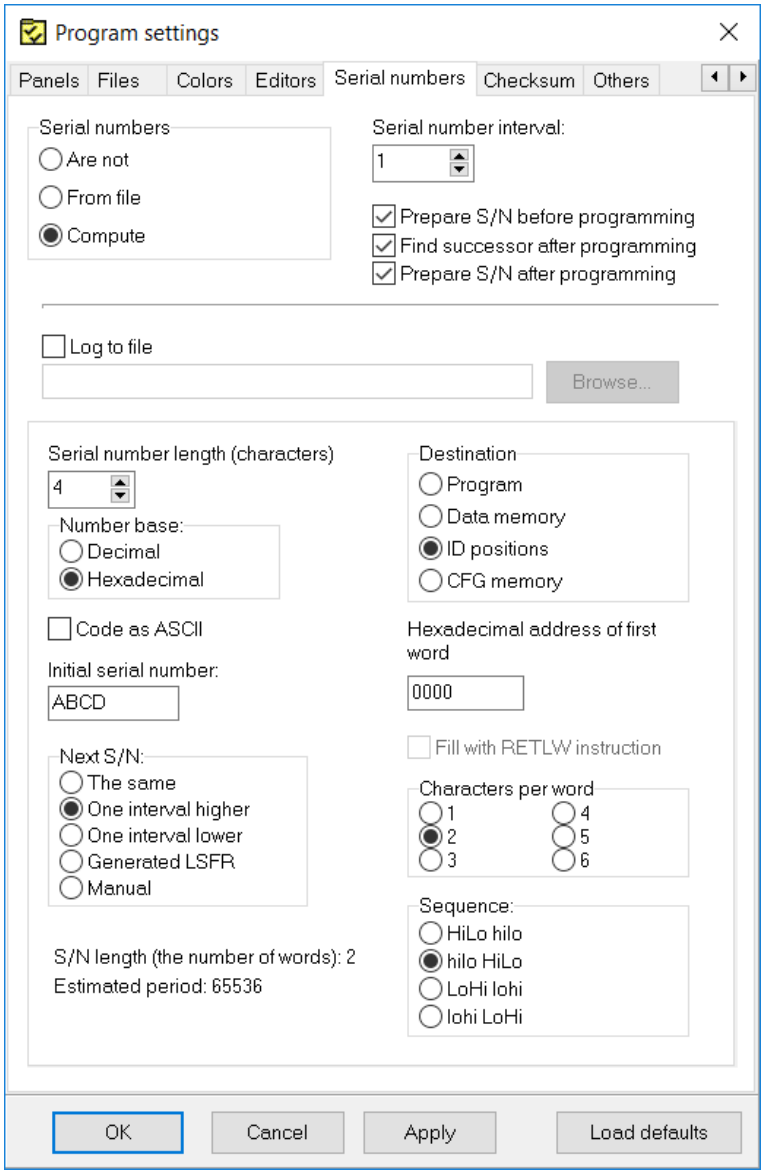


*Fig. 46: Serial number settings*

A file for logging the programmed serial numbers can be selected under ***Options → Program settings → Serial numbers***.

The numbers themselves are logged in case of calculated serial numbers while their labels are logged in case of serial numbers taken from a file - see Format of Files with Serial Numbers.

Except for the serial numbers also the date, time and the result of the programming are writen to the file. For the devices for which the revision reading is supported, the revision number is also written to the file.

# Format of Files with Serial Numbers

Files containing definitions of serial numbers are text files that can easily be created in third-party programs. *.SN or *.TXT are recommended file extensions.

A serial number record has the following form:

```
[comment] label: data record, data record, ..., data record;
```

Semicolons are obligatory at the end of records.

- **Comment** is any string containing no colon ':'. Comments are optional. If a colon cannot be found in the whole serial number record, the record is ignored (understood as a comment). It is also possible to write a comment behind two slashes '//'.

```
This is just a comment;
// Also this is a comment.
```

- **White character** is a space, tabulator or end of line (CR+LF).
- **Label** is a string identifying the serial number. This string is compulsory. The label must not contain white characters, colons or semicolons.

## Data Record

A data record consists of an address and data items in a chain following this address.

Each item can be written in hexadecimal form (e.g. 2100) or a numeral base in which the number is written in can be explicitly defined.

For example, b'10101010' means the same as h'AA', d'170' or just AA. 'A' means d'65' (ASCII character itself).

## Example record for PIC16F628A:

2100 05 55 54 means to save data 05h, 55h, 54h in addresses 00 to 02 of a data memory.

The code/main memory which stores the serial number can also be specified by the word “CODE.” or “PROG.” or just “P.”.

The data memory should be specified by the word “DATA.” or “EE.” or just “E.”.

“ID” or just “I” is used for the memory of ID positions.

These words are always followed by an address inside the specified memory.

**Example:**

EE.00 05 55 54 means to save data 05h, 55h, 54h in addresses 00 to 02 of a data memory.

## Notes

- * There is no specifier for the configuration memory, it would make no sense.

dsPIC – addresses of 24-bit words are to be written in for all addresses (i.e. an internal dsPIC address of 24h is 12h here). For a data (EEPROM) memory addresses of 16-bit words are to be written in, i.e. as they go one by one through the microcontroller.

- Autonomous memory chips ($I^2C$, SPI) have only the code memory. If a non-existent memory is specified, an error is reported.

## Example of File with Serial Numbers

```
Comment at the beginning;
sn1:    0000 34 45 56 67,
        2100 01 02 03 04; serial number 1
```

```
sn2:    0000 45 56 67 78, 2100 02 02 03 04;
sn3:    0000 56 67 78 89, 2100 03 02 03 04;
note

sn4:    0000 67 78 89 9A, 2100 04 02 03 04;
sn5:    0000 78 89 9A AB, 2100 05 02 03 04;
sn6:    0000 78 89 9A AB, 2100 06 02 03 04;
sn7:    0000 78 89 9A AB, 2100 07 02 03 04;
sn8:              code.0001 3F00 3F01 3F02 3F03,
data.0002 'x' '4' '2';
sn9:    prog.0001 3F00 3F01 3F02 3F03,
        e.0002 'x' '4' '3';
```

### 5.4.4 Calibration Memory Support

Some devices contain a calibration memory containing factory-preset device calibration. The loss of its content can cause a fault in the device functionality. For this reason we have tools allowing users to work with the calibration memory.

#### Working with Calibration Memory When Erasing a Device in UV Eraser

Calibration data should be saved prior to erasing a device. To do so, choose ***File → Save calibration data...***

ou can retrieve it back by choosing ***File → Read calibration data...***

The program contains a function for verifying whether a device has been erased correctly: ***Device → Blank check***. When this command is used, the program displays data from the calibration memory.

#### Working With Calibration Memory in Devices With Flash Memory

When these devices are erased, the content of their calibration memory is preserved.

If you really need to erase the calibration memory for some reason, you can do so by choosing ***Device → Erase → Erase all (including calibration)***.

**Advice**

New Flash devices with a calibration memory (such as PIC12F629) contain so called 'bandgap bits', which form part of the device calibration. These bits can be found in the configuration word and are erased too if the command ***Device → Erase → Erase all(including calibration)*** is executed!

## 5.5 Program Controls

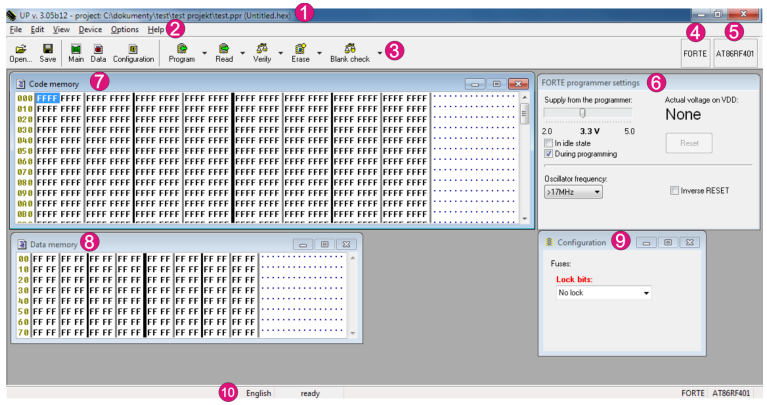


*Fig. 47: Controls of UP software*

1) Title bar with the open project name or file
2) Menu

3) Toolbar
4) Currently selected programmer
5) Currently selected device
6) Programmer settings window
7) HEX editor window for code/main memory
8) HEX editor window for data memory
9) Configuration window
10) Status bar

### 5.5.1 Toolbar

A toolbar is a panel with quick selection buttons located under the program menu (see Program Controls).

If you want to remove the toolbar, simply uncheck the Display icons and descriptions on the toolbar button options in the Settings menu as described in Program settings.

### 5.5.2 Status Bar

The status bar is a panel in the bottom part of the window (see Program Controls). It presents information on the programmer, the device, changes to the file since the last saving, etc.

Individual parts of the status bar react to double-clicking and to the context menu that opens after right-clicking.

If you want to hide or re-display the status bar, choose ***Program settings → Panels → Display the status bar in the lower part of the window***.

### 5.5.3 Menus

The following texts describe in detail individual items in the UP software menus.

Menu commands can be triggered with mouse by clicking on a particular menu item or from the keyboard by pressing the  key together with the shortcut, i.e. the underlined letter in the menu.

The menu bar consists of the following menus (categories):

- File Menu
- Edit Menu
- View Menu
- Device Menu
- Options Menu
- Help Menu

## File Menu

### File → New

*Keyboard shortcut:* Ctrl+N

The program creates a new empty file. If the currently open file has not been saved, the program prompts the user to save it.

### File → Open...

*Keyboard shortcut:* Ctrl+O

The program opens the Windows standard dialog listing the files saved on the disc. The supported file formats are described in Appendix C: Intel-HEX File Format. Files with extensions .hex or .a43 are loaded as Intel-HEX, others as binary. There are also filters available making it possible to open all files as .hex or as .bin.

### File → Open next file...

The program imports another .hex or .bin file with an optional offset. This function is useful if the user needs to load another file into device's memory. Files with extensions .hex or .a43 are loaded as Intel-HEX, others as .bin.

For example, this command provides for merging of several .hex files (e.g. bootloader + program).

When in **Open next file** dialog the user selects **Autoload next time**, the file will be loading after the main data file automatically. Then the file is listed in the **File → Open next:** menu. Using this menu item the autoload can be disabled again.

This way it is possible to autoload one file.

## File → Open next:

When this menu item is enabled, it shows that the automatic load of next file is enabled. By clicking this menu item it is possible to disable the automatic file load.

## File → Reload actual file

*Keyboard shortcut:* Ctrl+R

The program re-opens (i.e. re-reads from the disc) the currently open file. This command is useful if you know that the file on the disc has changed and you want to load these changes into the program.

If you are using ***Program settings → Files → Do automatic check for newer versions of actual file***, the program automatically notifies the user of a change in the currently open file and offers its reloading.

## File → Save

*Keyboard shortcut:* Ctrl+S

The program saves the file on a disc. If you want save your file under a different name than the current one, use the ***Save As...*** command instead.

The program can skip unused locations of the memory. See Program settings for details.

## File → Save as...

The program saves the open file on a disc under a new name using the Windows standard dialog.

The program can skip unused locations of the memory while saving or not to save certain selected locations. See Program settings for details.

## File → Import data memory from file...

Using the Windows standard dialog, the program reads and imports the content of a data (EEPROM) memory saved in a different file. This allows the user to load a data memory if this has not been saved in the same file as the code/main memory (this concerns ATmega8 microcontrollers, for example).

**Important warning**

Such a file, regardless of its content, is read from the zero address as if it contained only the data (EEPROM) memory. This means that a file generated normally by a compiler cannot be correctly read using this command.

## File → Open file with data memory automatically

With this option selected, the program automatically loads the file for the data memory alongside with loading the file for the code/main memory. This option is active only if a separate file is loaded for the data memory.

## File → New project

*Keyboard shortcut:* Shift+Ctrl+N

This command creates a new project.

It is especially advisable to use project files if you frequently switch between programming different device types or if you are using several different programmers. A project file contains all the corresponding settings and provides for loading of all of them in one step.

## File → Open project...

*Keyboard shortcut:* Shift+Ctrl+O

Using the Windows standard dialog, the program opens an already existing project file saved on a disc. If an additional file had been opened together with the project,

such a file opens as well.

## File → Save project...

*Keyboard shortcut:* Shift+Ctrl+S

Using the Windows standard dialog, the program saves the current project under a new name. Saving of the project under the same name is done automatically as well as saving the program settings, for example.

In the **Save project** dialog using **Load project unlocked** option it is possible to set that the project will not be locked after load.

Using **Add note** option it is possible to add a note to the project file, which will be shown after load of the project.

## File → Close project

*Keyboard shortcut:* Shift+Ctrl+W

The program terminates the work with the currently open project, saves the project file on the disc and returns to the state in which it was before the new project was opened.

## File → Recent projects

This function remembers the last 10 opened projects. Clicking the name of one of them opens the project.

## File → Read calibration data...

Using the Windows standard dialog, the program opens a file with the calibration data and loads the data in the memory.

## File → Save calibration data...

Using the Windows standard dialog, the program creates a file with the device's calibration data previously read from the device connected to the programmer. This calibration data can be reloaded using ***Read calibration data*** if the device gets erased in the meantime.

For further information on the program's support of work with the calibration memory see Calibration Memory Support.

## File → Export to bin...

This command saves binary data from the code/main memory, from the data memory or from ID positions in a selected file.

16 or 8 bits word width can be selected for the data to be saved.

## File → Exit

*Standard Windows keyboard shortcut:* Alt+F4
*Keyboard shortcut:* Alt+X

### Warning

If closing of the program is forced by the Turn Computer Off command and the program does not receive the user's confirmation, the system closes it forcibly after some time without saving the open file or settings.

If the program is currently working with the hardware, it refuses all system's requests to turn off and can be seen as “not responding” by the system.

This command closes the program. If the open file has changed, the program prompts the user to save the changes.

# Edit Menu

## Edit → Fill with value...

The program fills a memory location with a specified value. This command is used especially for erasing (filling with ones) or zeroing (filling with zeros) a particular location. Yet it can be filled with any value or with random data.

When the ***Fill with value...*** command is called, the program presets the selected memory according to the

active window. If a memory location was highlighted before calling this command, the program presets the marked location for filling in.

A memory location can be selected by holding the Shift key and clicking by mouse or moving by means of the cursor keys (arrows). For further information on highlighting a location see HEX Editor Windows.

## Edit → Text insert...

This command saves a text in ASCII or in hexadecimal format into a selected memory location. Ends of lines can be coded as NULL, CR, LF or CR+LF characters.

Individual bytes can be inserted as well as saved in RETLW instructions (this applies only to the code/main memory working with Microchip microcontrollers). Note: Microchip PIC RETLW instruction means return with a constant value in the work register – this instruction is frequently used for creating tables.

If the ***Text insert...*** command is called, the program presets the selected memory and the starting cell according to the current window and the selected cell.

## Edit → Fill selected location with RETLW

This command fills the selected Microchip microcontroller's memory location with RETLW.

The command can be called only from an open HEX editor. It is also available through the context menu (right mouse click) in the editor.

A memory location can be selected by holding the Shift key and clicking by mouse or moving by means of the cursor keys (arrows). For further information on highlighting a location see HEX Editor Windows.

# View Menu

## View → Code/main memory

This command opens or closes the code/main memory HEX editor window. For more on HEX editors see HEX Editor Windows.

## View → Data memory

This command opens or closes the data memory HEX editor window. For more on HEX editors see HEX Editor Windows.

## View → Boot memory

This command opens or closes the boot memory HEX editor window. For more on HEX editors see HEX Editor Windows.

## View → Configuration memory

This command opens or closes the configuration memory HEX editor window. For more on HEX editors see HEX Editor Windows.

## View → Console

This command opens or closes the console, where the UP can write details about programming.

## View → Display code/main memory

*Keyboard shortcut:* Alt+F10

This command displays the code/main memory HEX editor. If it is already displayed, it moves it to the front (the editor gets focus). For more on HEX editors see HEX Editor Windows.

## View → Display data memory

*Keyboard shortcut:* Alt+F11

This command displays the data memory HEX editor. If it is already displayed, it moves it to the front (the editor

gets focus). For more on HEX editors see HEX Editor Windows.

## View → Display configuration memory

*Keyboard shortcut:* Alt+F12

This command displays the configuration memory HEX editor. If it is already displayed, it moves it to the front (the editor gets focus). For more on HEX editors see HEX Editor Windows.

## View → Display programmer form

*Keyboard shortcut:* Ctrl+P

The programmer form is always visible. This function moves the programmer form to the front (the editor gets focus).

# Device Menu

## Device → Program

*Keyboard shortcut:* Shift+F5

This command opens a sub-menu with programming options. Some items may be inaccessible for certain device types.

### ▸ Program all

*Keyboard shortcut:* F5

This command erases the device, checks the erasure, programs it and verifies the programming result in the whole device. The Device ID check and the Code/Data Protection check are performed prior to this operation. The behavior of this command is influenced by the program settings - see Program settings.

### ▸ Program all except data memory

A keyboard shortcut can be assigned in ***Options → Key shortcuts***.

This command works the same as ***Program all*** with the exception that it does NOT erase, program or verify the data memory.

Devices without a data memory do not have this command available and their programming is done using the ***Program all*** command.

This command cannot be used in some cases that use Code or Data Protection. For these, the program offers the command ***Erase whole device and program also data memory*** (with data currently in the editor).

### ▸ Program code/main memory

A keyboard shortcut can be assigned in ***Options → Key shortcuts***.

This command erases the code/main memory, checks the erasure, programs and checks the programming result.

### ▸ Program data memory

A keyboard shortcut can be assigned in ***Options → Key shortcuts***

This command erases the data memory, checks the erasure, programs and checks the programming result.

### ▸ Program configuration memory

A keyboard shortcut can be assigned in ***Options → Key shortcuts***

This command programs the configuration memory and ID positions (if device contains them) and checks the programming result

### ▸ Program differentially

*Keyboard shortcut:* Ctrl+F5

This command programs the device applying the differential method. This means that it reads the device and programs only those cells the content of which differs from the editor.

This command is available only for those devices that support such treatment (not all devices do).

If a device has the Code/Data Protection active, differential programming makes no sense. The program performs complete programming instead, including device erasure.

### ▸ Differential program data memory

A keyboard shortcut can be assigned in ***Options → Key shortcuts***

This command programs the data memory applying the differential method. Its function is identical with differential programming of the code/main memory.

This command is available only for those devices that support such treatment (not all devices do).

If a device has the Code/Data Protection active, differential programming makes no sense. The program performs complete programming instead, including device erasure.

Differential programming of a data memory should be used for AVR microcontrollers if the user needs to reprogram the data memory only without previously erasing the device.

### ▸ Mass Production

A keyboard shortcut can be assigned in ***Options → Key shortcuts***

This command opens a window for easy programming of several identical devices with the same or very similar program(s) (except their serial numbers and the like). For further information see Mass Production.

## Device → Read

*Keyboard shortcut:* Shift+F6

This command opens a sub-menu with device reading options. Some of them may be not available for certain device types.

### ▸ Read all

*Keyboard shortcut:* F6

This command reads the content of the whole device.

### ▸ Read all except data memory

A keyboard shortcut can be assigned in ***Options → Key shortcuts***

This command reads the content of the whole device except the data memory.

### ▸ Read code/main memory

A keyboard shortcut can be assigned in ***Options → Key shortcuts***

This command reads the content of the code/main memory.

### ▸ Read data memory

A keyboard shortcut can be assigned in ***Options → Key shortcuts***

This command reads the data (EEPROM) memory.

### ▸ Read configuration memory

A keyboard shortcut can be assigned in ***Options → Key shortcuts***

This command reads the configuration memory and ID positions (if device contains them).

### ▸ Read address

This function enables reading of data from user filled address, it supports MCUs ARM via SWD interface.

## Device → Verify

*Keyboard shortcut:* Shift+F7

This command opens a sub-menu with device memory content verification options. Some of them may be not available for certain device types

### ▸ Verify all

*Keyboard shortcut:* F7

This command compares the content of all device memories with the current content of HEX editors.

### ▸ Verify all except data memory

A keyboard shortcut can be assigned in ***Options → Key shortcuts***

This command compares the content of all device memories except its data memory with the current content of HEX editors.

### ▸ Verify code/main memory

A keyboard shortcut can be assigned in ***Options → Key shortcuts***

This command compares the content of the code/main memory with the current content of code/main memory's HEX editor.

### ▸ Verify data memory

A keyboard shortcut can be assigned in ***Options → Key shortcuts***

This command compares the content of the data memory with the current content of data memory's HEX editor.

### ▸ Verify configuration memory

A keyboard shortcut can be assigned in ***Options → Key shortcuts***

This command compares the content of device's configuration memory and ID positions (if device contains them) with settings in the Configuration window.

## Device → Erase

*Keyboard shortcut:* Shift+F8

This command opens a sub-menu with device memory erasing options.

Erasure verification (blank check) is automatically performed after each erasing. This verification can be skipped by choosing ***Options → Program settings → Programming → Do not perform blank check after erasing***. This can save time with some devices.

### ▸ Erase all

*Keyboard shortcut:* F8

This command erases the whole device.

### ▸ Erase code/main memory

A keyboard shortcut can be assigned in ***Options → Key shortcuts***

This command erases the code/main memory. It cannot be used if Code/Data Protection is active.

### ▸ Erase data memory

A keyboard shortcut can be assigned in ***Options → Key shortcuts***

This command erases the data memory and verifies it. It cannot be used if Code/Data Protection is active.

## Device → Blank check

*Keyboard shortcut:* Shift+F9

This command opens a sub-menu with device erasure verification options. Some of them may be not available for certain device types.

### ▸ Blank check all

*Keyboard shortcut:* F9

This command verifies whether the device is correctly and completely erased.

### ▸ Blank check all except data memory

A keyboard shortcut can be assigned in ***Options → Key shortcuts***

This command verifies whether the device except the data memory has been correctly erased.

### ▸ Blank check of code/main memory

A keyboard shortcut can be assigned in ***Options → Key shortcuts***

This command verifies whether the code/main memory is correctly erased.

### ▸ Blank check of data memory

A keyboard shortcut can be assigned in ***Options → Key shortcuts***

This command verifies whether the data memory is correctly erased.

### ▸ Blank check of configuration memory

A keyboard shortcut can be assigned in ***Options → Key shortcuts***

This command verifies whether the configuration memory and ID positions (if device contains them) are correctly erased.

## Device → Select device

*Keyboard shortcut:* F4

This dialog allows you to select a device type to be programmed. Some memory types need to have their data organization selected after the device type selection.

Only devices supported by the currently selected programmer are displayed in the dialog window. If you want to select a device not supported by the current programmer, select a different programmer type first.

For further information on selecting a device see Device Type Selection.

## Device → Device info

Shows a window with information about connecting the selected device to the programmer.

# Options Menu

*Keyboard shortcut:* Shift+F10

The ***Options*** menu holds all the possible settings of the UP program. There is a great number of setting parameters. If you are not sure that all settings are correct, you may use the ***Load defaults*** button, which returns all the settings to the initial (factory) state.

> Advice
> ***Program settings → Load defaults*** button renews the state of all settings on all tabs, i.e. even the color settings, for example.

## Options → Program settings → Programming

*Keyboard shortcut:* Shift+F10

This tab provides for setting all the general programming parameters.

Settings concerning the programmer and the communication port are described in Programmer Selection.

A special Device → Select device window is available for setting the device type.

### ▸ Reload file before every programming

The program reads the current file with programming data from the disc before each and every request to program a device if this option is checked.

If the use of serial numbers is also activated requesting reading the current file prior to programming, then the file is read first and only then is the current serial number written.

### ▸ Keep manually modified data

When this choice is active, the file load before programming does not rewrite manually changed data. This choice affects only behavior of the file reload before programming.

### ▸ Warn before file load, when data in some editor have been changed

When before load of the file before programming it is found that data in some editor have been changed, the program shows a warning.

### ▸ Warn, when the loaded file has not changed

Program shows a warning, when the content of the loaded file has not changed from the previous programming.

### ▸ Program file locations only

The programming changes only the positions contained in the file. First it reads the programmed device content, then it reads the file content which replaces the data read from the device. The addresses which are not contained in the file hold the value read from the device. Finally the memory is programmed.

### ▸ Ask before erasing

With this option checked, the program asks for user confirmation before erasing a device.

### ▸ Ask before programming of OTP / Flash / Code/Data Protection / differential

A set of options determining which user confirmation dialogs will or will not be displayed by the program.

The program asks just once except the ***Code/Data Protection programming***. If the program needs to ask the user for additional information before actual programming (such as **Device has Code or Data Protection. Do you want to erase it entirely?**), it

does not ask for further programming confirmation after answering the first dialog.

### ▸ Display fuse warning messages

The user can choose if warning messages should be displayed for some fuses. It is recommended to leave this option on.

### ▸ Except for programming: Close status window

This option closes the status window if no error occurs during erasing, blank check, programming verification and reading.

### ▸ After programming: Close status window

This option closes the status window if no error occurs during programming and the subsequent verification.

### ▸ Beep after successful finishing

With this option active, the program calls up the standard Windows system beep if the operation (such as erasing, programming, etc.) went smoothly.

### ▸ Beep after unsuccessful finishing

With this option active, the program calls up the standard Windows system beep if an error or warning occurs during the operation (such as erasing, programming, etc.).

### ▸ Turn off all sound for UP

With this option on, the program does not produce any sounds.

### ▸ Delay for VDD switching on/off when supplied from programmer

This option is important for programming PCB-placed devices via an ICSP cable. It determines the length of the delay after voltage is connected to or disconnected from a device.

Both PRESTO and FORTE feature an overcurrent protection. This means that a test for excessive current of approximately 100 mA in power supply voltage is performed after connecting power to the device and expiration of the delay time set in **Switch on**.

Similarly the delay time set in **Discharge** specifies how late after switching the power supply off a check should be performed if no voltage is present any longer at the pin.

If a blocking capacitor is connected to the device's power supply pins (recommended), the voltage at the pin changes more slowly. This could cause problems during programming, but they can be eliminated by prolonging the charging and discharging time.

Too long a time increases the risk of damaging the device if connected incorrectly while too short a time could cause programmer's circuits to still detect excessive current flowing to the application's capacitors. A formula for an approximate determination of the necessary time can be found in Appendix B: Use of ICSP.

### ▸ Do not perform Device ID check before programming

This option switches off the Device ID check before programming.

### ▸ Do not perform blank check before cfg word programming

Most rewritable devices can overwrite the configuration word without erasing the whole device content. Skipping the blank check of the configuration word utilizes this possibility, i.e. the program ignores the not erased word.

This option does not apply to the whole-device programming in which the device is erased completely. It concerns only the configuration word and its overwriting.

### ▸ Do not perform blank check after erasing

This option speeds up the programming process and is useful especially for debugging. The danger is that an incorrectly erased device is subsequently incorrectly programmed and the fault only gets detected later. On the other hand, a device is erased incorrectly only once in several hundred attempts.

### ▸ Do not erase device before programming

The device is not erased before programming.

### ▸ Do not erase data memory before its programming

This option only concerns programming of Atmel AVR devices e.g. ATmega8. Data memory of this family devices does not require to be erased before programming. If this option is not activated, whole device will be erased before programming.

### ▸ Do not verify unprogrammed words at the end of the memory

If there is a location at the end of the code/main memory containing only default values, it is not verified.

This option speeds up the verification of the programmed memory as the content of the “empty” memory at its end usually does not matter.

### ▸ Do not verify

This option completely switches off the verification of a programmed device. Such off-switching can radically speed up the programming process during development.

This option must not be activated for production as correctness of the programmed content would not be guaranteed.

### ▸ Verify with two supply voltages

This option is available only for FORTE and only for the internal power supply from the programmer. It allows you to perform the verification at two different power supply voltages defined by the user.

Some device manufacturers recommend to verify the content after production programming at two different voltages corresponding with the approved range of the device's power supply voltages.

## Options → Program settings → Panels

*Keyboard shortcut:* Shift+F10

This part of the Options menu allows you to alter the application's appearance. The user can set where and how some control components are to be displayed.

### ▸ Display selected device on toolbar

In addition to the status bar, the selected device is also displayed in the toolbar.

### ▸ Display selected programmer on toolbar

In addition to the status bar, the selected programmer is also displayed in the toolbar.

### ▸ Display the status bar in the lower part of the window

This option determines whether the status bar is to be displayed or not.

### ▸ Display icons on toolbar buttons

Icons of individual tools are displayed in the toolbar.

### ▸ Display descriptions on toolbar buttons

Tool names are displayed next to individual tools in the toolbar.

If the ***Descriptions on toolbar buttons to the right*** option is used, the total toolbar height is reduced to a half.

### ▸ Show mass production counter in status bar

This option displays the mass production counter in the status bar. The counter shows the number of devices programmed and the number of devices programmed successfully.

If you use the ***Count all actions option***, the counter includes all actions performed with the device (such as reading, erasing, data memory programming, etc.).

***Show values description*** displays explanatory notes to individual counter items for easier understanding.

If you want to exclude operations that triggered a warning from the correct operations, use the If ***action ends with warning assume as erroneous*** option.

The choice ***Reset counters on project open*** causes, that the counters are reset on any project file load, including automatic project load during program start.

***Counter style*** allows you to choose whether the counter should display the **good/bad** or **good/total** figures.

The ***Counter reset***, button can be used for zeroing the counters while their initial values can be set in the ***Preset counter*** values fields.

## Options → Program settings → Files

*Keyboard shortcut:* Shift+F10

This tab holds all the settings for reading data from and saving data to files.

### ▸ File save style

The ***File save style*** panel offers the possibility not to always save all areas of all editors in the file, but rather only some. UP then asks questions based on the saving style settings before saving different project editors.

### ▸ Do automatic check for newer version of actual file

This option helps especially in debugging a program. The program re-loads the file if it detects a change in the file modification date.

### ▸ Never ask and never save changes to data file

When this option is enabled and data in an editor have been changed, before opening of an other file, before closing of the application etc. the application will not ask and will not save the file which is currently open.

### ▸ Check device type when loading .hex file

If the device type has been saved in the corresponding Intel-HEX file and it does not agree with the currently selected device type, the program points out this discrepancy.

### ▸ Save device type into .hex file

The program adds one more line below the end of file command specifying the type of device for which the file is being saved.

Such a modified file does not comply with the Intel-HEX format, but most programs working with the Intel-HEX format ignore this line.

For further information about the Intel-HEX format see Appendix C: Intel-HEX File Format.

## ▸ Warn when loaded HEX does not contain CFG memory data

If the file does not contain configuration memory data and the data are expected for the chosen device, a warning message is displayed.

## ▸ Warn when loaded HEX is not aligned to word size.

If the file is not aligned to the word size of the memory of the device, a warning message is displayed.

## ▸ Binary file loading and saving style

This panel provides for the setting of how .bin files are to be loaded and saved if a device with more than one byte per word is selected.

Available options are: the program always asks before loading or saving a .bin file (**Ask if to load/save .bin as Big or Little Endian**), or without asking, the program always loads files as Little Endian (**Never ask, load/save as Little Endian**) or as Big Endian (**Never ask, load/save as Big Endian**).

## ▸ Save unused locations to .hex file

If not all positions are saved, the final file is smaller, but it can cause difficulties as a cell is considered an “empty position” if it contains only ones (e.i. FFFh, 3FFFh, etc.). Yet this could be a meaningful instruction (3FFFh is Microchip PIC addlw -1 instruction, for example).

As UP always saves files by larger blocks, (by eight or sixteen bytes), the danger of losing an instruction by such saving is rather small in reality.

If you use memory initialization prior to loading a .hex file (which is recommended), even a program loaded from a reduced .hex file will be correctly programmed as the missing instructions are created “automatically”.

## ▸ Clear code/main / data memory / ID positions before file reading

This location is filled with ones and only then a file is loaded. This way all positions that are not saved in the .hex file get erased.

This option is also important if no “empty” locations are saved in the .hex file (see ▸ Save unused locations to .hex file).

## ▸ Erase configuration memory before file reading

With this option activated, the configuration memory is initialized before a file for code/main memory is loaded.

If no fuses are saved in the file, it is recommended to deactivate this option. At the beginning of the work, the user can set the content of the configuration memory and then the fuses do not have to be set again if the file is repeatedly loaded.

## ▸ Read data memory not from the file but from the device

If you want to make sure that the content of the data memory does not get overwritten, use the ▸ Program all except data memory option. If, however the ▸ Program all option is selected by accident (by unintended pressing of the GO button, for example), the content of the data memory would be erased.

For such cases, this option of **Read data memory not from the file but from the device** is available. The program fills in the given location with the memory content of the device connected to the programmer.

### Advice

This option could cause unexpected actions during work with the programmer such as switching the application on or starting the UP program.

### ▸ Read ID positions not from the file but from the device

If you want to preserve the content of the ID positions, use the **Read ID positions not from the file but from the device** option. The program fills in the given location with the memory content of the device connected to the programmer before starting the programming itself.

> **Advice**
> This option could cause unexpected actions during work with the programmer such as switching the application on or starting the UP program.

### ▸ Save fuses in UP instead of data file

This option allows to save fuses in ini file or UP projekt file even for devices which by default have their fuses saved in a data file.

### ▸ Project storing style

This panel allows you to set how projects should be saved when exiting UP. Available options are: save automatically, ask before saving or keep the original file (i.e. do not save).

### ▸ Load last project on start-up

This option allows you to set if on the next UP start it will open the project file, which was open on program closing.

## Options → Program settings → Colors

*Keyboard shortcut:* Shift+F10

This is where you can change and save colors of HEX editors so that they suit your needs and your aesthetic preferences.

A front color, a background color and a font can be selected for all the listed text segments.

## Options → Program settings → Editors

*Keyboard shortcut:* Shift+F10

### ▸ Code/main memory editor: show words as bytes

Individual words can be displayed by bytes in case of devices with 16-bit-long words.

### ▸ Code/main memory editor 8 words wide

This option narrows the editor from the original sixteen cells down to eight. The option is suitable especially for small monitors. It can change automatically when a different device type is selected.

The same option can be set for others memories:

### ▸ Data memory editor 8 words wide

### ▸ Boot memory editor 8 words wide

### ▸ Show only the lowest byte of word in ASCII

When this option is activated, the program shows the ASCII translation of only the lowest byte of word, which can be useful especially for PIC MCUs.

### ▸ Mask ID positions while reading from device, from file, etc.

Some manufacturers' specifications recommend saving only masked data (with only four bits usable) in ID positions. With this option activated, the program always applies this bit mask when reading ID positions from different sources.

### ▸ Mask ID positions during direct user input

With this option activated, the program applies the bit mask to each user-modification of ID positions. For further information see ▸ Mask ID positions while reading from device, from file, etc..

### ▸ Configuration memory editor: show cfg word instead of fuses

This option is recommended for advanced users only. It is not saved in the UP configuration file for safety reasons.

Direct editing of fuses is understood as direct writing of a configuration word value in the hexadecimal form.

When a “non-translatable” configuration word is entered, the program leaves the unrecognized items unchanged unless the user changes them him/herself. This typically concerns CP fuses having several bits but only two values.

## Options → Program settings → Serial numbers

*Keyboard shortcut:* Shift+F10

For further information on working with serial numbers see Serial Numbers.

### ▸ Serial numbers

This tab allows you to choose whether serial numbers are to be used and if so then whether they should be read from a file or computed and whether they should be automatically written to the corresponding location.

### ▸ Prepare S/N before programming

This option prepares a serial number in the selected memory location before the actual device programming.

### ▸ Find successor after programming

Once one programming procedure is successfully completed, the system proceeds to the following serial number. If this option is not activated, it is still possible to proceed to the following number manually by clicking the **Next** button on the Serial numbers tab – see Serial Numbers.

### ▸ Prepare S/N after programming

This option causes that a serial number is prepared in the selected memory location after the device programming.

### ▸ Serial number interval

This defines by what value the following serial number will be larger or smaller.

### ▸ Log to file

By activating this option and by selecting a log file in the field, information about correctly and incorrectly programmed devices as well as the programming time is saved in the selected file.

### ▸ After project load set actual SN according to the last in the log

When this option is enabled, the last programmed SN is read from the log file and when the programming is logged as errorless the next SN is computed from the read value else the last SN is set again.

The log file has to exist and the last record has to contain a SN.

This function can be useful when the serial numbers are enabled and saving of UP project files is disabled, in such a case the actual SN will not update in the project file on program close.

## ▸ Serial number length (the number of characters)

This field determines how many characters the serial number can hold. For example, 4 characters provide for the serial numbers 0001 to 9999 in base 10 (i.e. in decimal form).

## ▸ Number base

This option defines the base of the serial number characters. Base 10 and base 16 are available (i.e. the decimal and hexadecimal forms).

## ▸ Code as ASCII

With this option activated, serial numbers are legible as ASCII characters.

## ▸ Initial serial number

This field defines the initial value from which serial numbers are to be computed.

## ▸ Next S/N

This field defines how the following serial number is to be generated.

If **The same** is selected, the serial number does not change. If **One interval higher** is selected, the following number will be larger by the value of ▸ Serial number interval while if **One interval** lower is selected, it will reduce by that value.

The **Generated LSFR** option generates the numbers in a pseudo-random sequence. This means that the sequence will always be the same if the same **Initial serial number** and the same **Serial numbers step** is used.

The **Manual** option shows a field, in the serial numbers window, where the serial number in the hexadecimal form can be entered before programming. When this option is selected, the serial number can be entered using the /sn commandline parameter as well.

## ▸ Destination

This panel determines the memory location in which the serial numbers are to be stored. **Code/main memory, data memory** and **ID positions** are available as options.

## ▸ Hexadecimal address of first word

This address defines where the first word of the serial number is to start in the selected memory type.

## ▸ Fill with RETLW instruction

Individual words of the serial number can be filled with RETLW instruction in Microchip microcontrollers. This option is available only if the serial number is located in the code/main memory.

## ▸ Characters per word

This panel determines how many serial number characters form one word. Options of 1 to 4 characters per word are available.

## ▸ Sequence

This panel sets how individual serial number characters and words are to be ordered.

For example, if you have a base 10 serial number consisting of four characters organized as 2 characters per word and you are working let's say with 1234 as the serial number, it will be saved in the following ways:

```
HiLo hilo: 12 34
hilo HiLo: 34 12
LoHi lohi: 21 43
lohi LoHi: 43 21
```

# Options → Program settings → Checksum

### ▸ Show checksum in status bar

After enabling of this setting the checksum of the code memory will be shown on the status bar.

When MD5 checksum algorithm is selected, the checksum is shown after placing the mouse on MD5 text in the status bar.

Note: Doubleclick on the checksum value in the status bar will cause the value recalculation.

### ▸ Write checksum to log file

Enables writing of the loaded data file checksum to a log file.

Note: The checksum is recalculated on data file load. This function have to be enabled before the file load, else the checksum value will not be written to the log file.

### ▸ Checksum algorithm

Using this choice the checksum calculation algorithm can be chosen.

## Options → Program settings → Others

*Keyboard shortcut:* Shift+F10

### ▸ Update check settings

Here you can set if the program should ask for your permission to connect to the Internet and check for updates upon each start or not.

For further information see Updating UP.

### ▸ Allow internal and external supply voltages collision

**Warning**
A collision of power supply voltages can damage the programmer or the application!

If you approve the collision of supply voltages, it allows the programmer to connect its internal voltage to pin VDD at times when it can already see a voltage present there. This could damage both the programmer and/or the application being programmed.

This option has been included for very specific cases. One of them can be a request to program an application, which does not have any VDD pin. Then the application must be powered from its own source but the programmer's output buffers must be powered from the USB. Yet as the voltage at the application's data signals “sneaks” through the protective diodes into the output section of the programmer, the programmer can see a certain small voltage present at VDD and refuses to connect its own internal voltage to it.

### ▸ Do not show warning if internal 5 V is switched on with 3.3 V device

**Warning**
High a voltage at the device can damage the programmer or the application being programmed.

This option can be used to suppress the warning if a higher voltage is switched on at the VDD output than the value allowed for the particular device.

If there is a voltage convertor integrated in the application between the device being programmed and the programming interface, it might be useful to use this option of programing at a higher voltage than what the device has been designed for.

This option is available only for PRESTO.

### ▸ Allow to change supply voltage level when it is on

In order to protect the application connected to FORTE from damage, it is typically forbidden to change the output voltage level if the voltage is currently connected to the application. After activating this option, the output voltage can be changed at any time within the range that

the device being programmed can cope with.

### ▸ Allow external supply voltage for devices requiring VPP before VCC.

For some devices their manufacturer requires the programming voltage to be connected before the supply voltage of the device. When the external supply voltage is used, it does not comply with the specification. In this case a warning is displayed, which can be permanently disabled using this setting.

Disabling of this warning should be well considered. This warning is displayed for devices for which the external supply voltage violates the device manufacturer specification.

### ▸ When using Windows Messages disable other warnings

When controlled via Windows Messages the UP software does not show any warnings, similarly like in quiet mode on the commandline.

### ▸ Pin T during programming

With this setting it is possible to set a logical level during programming on the T pin. The selected logical level will be visible on the pin only when the supply voltage is available at the VDD pin of the programmer.

In case that the T pin is used for programming, this function will not be available.

This option is available only for FORTE.

### ▸ Pin T after programming

With this setting it is possible to set a logical level, which will appear on the T pin after programming. The selected logical level will be visible on the pin only when the supply voltage is available at the VDD pin of the programmer.

In case that the T pin is used for programming, this function will not be available.

This option is available only for FORTE.

## Options → Select programmer

Settings concerning the programmer and the communication port are described in Programmer Selection.

## Options → Language selection...

*Keyboard shortcut:* Ctrl+L

A file with a different language localization can be selected using the Windows standard dialog. This means that one program installation can easily switch between different user-interface languages.

## Options → Keyboard shortcuts...

*Keyboard shortcut:* Ctrl+K

Using this dialog window you can define or change keyboard shortcuts for most commands the programmer is capable of performing.

This window also provides for setting the required behavior of the ***GO*** button.

## Options → Lock project

After a project file has been opened, it is locked against unintentional changes. To unlock or lock it again this function can be used.

When in project save dialog it was selected to **Load project unlocked**, after load it will not be locked.

# Help Menu

## Help → Help on program

*Keyboard shortcut:* F1

This command opens the help that you are are just reading.

### Help → List of supported devices

This command displays a list of devices currently supported by the latest version of UP.

### Help → Check Internet for updates

The program connects to the Internet and checks if you are using the latest version.

### Help → ASIX website

This command opens www.asix.cz, the website of ASIX s.r.o. where you can find the latest drivers and manuals for ASIX products.

### Help → About

Basic information on the program and a contact to technical support.

## 5.5.4 Programmer Settings Window

All important settings concerning the programmer and the applications to be programmed are displayed in the programmer settings window.

The window appearance changes in accordance with the selected programmer and device.

## FORTE Programmer Settings Window

### Power supply from the programmer

This trackbar allows the user to set the voltage level supplied to the application by the programmer.

### In idle state

If this option is checked, the programmer supplies power to the application even when not programming.

### During programming

If this option is checked, voltage from the programmer will be used during programming.

### Reset

This button allows you to switch levels on the device's ***reset*** pin between the level required for reset and high impedance.

***Reset*** is active if voltage is present on the power supply pin.

### Settings Associated with RX600 Microcontrollers

#### ▸ Protect with ID

If this option is active, it programs also the OSIS(ID) value during Configuration memory programming, this way it is possible to lock the device bootloader access.

#### ▸ Allow using Configuration Clearing (erases TM, ID)

The Configuration Clearing erases a TM (Trusted Memory) space. When the TM is in use, the use of this option should be well considered.

The use of this option is the only possibility how to erase OSIS(ID).

#### ▸ Baud Rate

Using this option it is possible to set a communication speed between the programmer and the programmed device.

### Settings Associated with PIC Microcontrollers

### ▸ Programming method

- **HVP** Traditional programming will be applied using a voltage of 8 V to 13 V at pin P.
- **LVP** Programming utilizing the microcontroller's LVP pin will be applied; only logical values of 0 or 1 are present at programmer's pin P.

### ▸ Use PE

Programming utilizing the ***Use PE*** option can be applied to PIC24 and dsPIC33 devices. PE stands for Programming Executive and it is a programming method frequently proving to be faster. The Programming Executive does verification after programming, that is why UP does not verify after PE programming.

## Boot memory programming

This allows to select the boot memory region to be programmed or checked.

# Settings Associated with AVR and 8051 Microcontrollers

## Oscillator frequency

An external oscillator or a functional internal oscillator must be connected in the course of AVR microcontroller programming. The frequency setting must match the frequency at which the device's oscillator is really running, after eventual dividers. The maximum speed of communication with the microcontroller then depends on the frequency of this oscillator.

## Faster Programming with Slow Clock

If this option is active, once the device is erased, the fuses are programmed at the maximum frequency of the internal oscillator. This allows the programmer to communicate with the device at a faster speed. At the end of programming, the required value of the communication memory including the clock speed is programmed.

This option has its effect only if the whole device is programmed.

## Inverse Reset

With this option activated, the programmer generates an inverse reset signal.

This option is appreciated if the application includes a reset circuit that needs an inverse signal at the input compared to the output signal sent to the microcontroller, and if the programmer is connected via this reset circuit.

## Write RC osc Adjustment

When this option is set, programmer will write the value defined in the Configuration window to "RC osc Adjustment" fuse. When it is not set, the value read from the device will be written back to the fuse.

## HVP

If this option is checked, the programmer uses the “high” voltage at pin P for communication with the device.

This allows you to program a device with the external RESET signal switched off.

# Settings Associated with CH32V003 Microcontrollers

### ▸ Fast mode

When this option is set, programmer uses double speed for communication with the microcontroller.

# Settings Associated with I2C Memory Chips

## I2C Bus Speed

Select the maximum possible speed of the $I^2C$ bus. The programmer switches on the internal pull-up of 2.4 kΩ while working on the $I^2C$ bus.

### I2C Memory Address

Select the address of the $I^2C$ memory on the bus.

## Settings Associated with SPI Flash Chips

### ▸ Start address

Sets the address range for work with the memory. ***Start address*** is the first address, where the selected operation is executed.

### ▸ End address

Sets the address range for work with the memory. ***End address*** is the last address, where the selected operation is executed.

# PRESTO Programmer Settings Window

## In idle state

If this option is checked, the programmer supplies power to the application even when not programming.

## During programming

If this option is checked, voltage from the programmer will be used during programming.

## Settings Associated with PIC Microcontrollers

### MCLR Pin Control

The logical value present at pin P1 (VPP) can be controlled by the **Run**, **Stop**, **Tristate** and **Reset** buttons in the idle state if a voltage is present.

The **Reset** button generates a resetting pulse.

### Programming Method

- **HVP** Traditional programming is applied with 13 V present at VPP.
- **LVP** Programming utilizing microcontroller's LVP pin is applied, only logical values of 0 or 1 are present at programmer's pin P.

### Algorithm Programming

- **Auto** An algorithm is selected according to the voltage currently present at VDD.
- **Ucc=5 V** The algorithm for fast 5 V programming is used.
- **Ucc=2.7 to 5.5 V** The algorithm for slow programming is used with the benefit of working at all power supply voltages.

### Use PE

Programming utilizing the ***Use PE*** option can be applied to PIC24 and dsPIC33 devices. PE stands for Programming Executive and it is a programming method frequently proving to be faster. The Programming Executive does verification after programming, that is why UP does not verify after PE programming.

### Boot memory programming

This allows to select the boot memory region to be programmed or checked.

## Settings Associated with AVR and 8051 Microcontrollers

### Oscillator Frequency

An external oscillator or a functional internal oscillator must be connected in the course of AVR microcontroller programming. The frequency setting must match the frequency at which the device's oscillator is really running, after eventual dividers. The maximum speed of communication with the microcontroller then depends on the frequency of this oscillator.

### Faster Programming with Slow Clock

If this option is active, once the device is erased, the fuses are programmed at the maximum frequency of the internal oscillator. This allows the programmer to communicate with the device at a faster speed. At the end of programming, the required value of the communication memory including the clock speed is programmed.

This option has its effect only if the whole device is programmed.

### Inverse Reset

With this option activated, the programmer generates an inverse reset signal.

This option is appreciated if the application includes a reset circuit that needs an inverse signal at the input compared to the output signal sent to the microcontroller and if the programmer is connected via this reset circuit.

### HVP

If this option is checked, the programmer uses the “high” voltage at pin P for communication with the device.

This allows you to program a device with the external RESET signal switched off.

## Settings Associated with I2C Memory Chips

### I2C Bus Speed

Select the maximum possible speed of the $I^2C$ bus. The programmer switches on the internal pull-up of 2.2 kΩ while working on the $I^2C$ bus.

### I2C Memory Address

Select the address of the $I^2C$ memory on the bus.

## Settings Associated with SPI Flash Chips

### ▸ Start address

Sets the address range for work with the memory. ***Start address*** is the first address, where the selected operation is executed.

### ▸ End address

Sets the address range for work with the memory. ***End address*** is the last address, where the selected operation is executed.

# 5.5.5 HEX Editor Windows

HEX editors are used for displaying the content of a memory chip to be programmed.

Different colors are used in HEX editors to differentiate between states of different cells to make it easy to see which cells have been read from a file, which have been successfully programmed, etc.

Colors may be user-assigned. This is especially recommended for workstations with displays/screens displaying only a limited number of colors.

## Selecting an Area

An area can be selected in a HEX editor by holding the Shift key down and pressing the cursor keys (i.e. arrows).

Once a required area is selected, it can be filled with a required value. The values can also be filled with RETLW instruction. These options are available in the context menu (opens by mouse right-click).

## Code/Main Memory Editor

*Menu:* ***View → Display code/main memory***

*Keyboard shortcut to open the window:* F10
*Keyboard shortcut to close the window:* Esc

The code/main memory editor displays the code/main memory content or the content of the memory itself in case of serial EEPROM chips (24xx, 93xx,...).

## Data Memory (EEPROM) Editor

*Menu:* ***View → Display data memory***

*Keyboard shortcut to open the window:* F11
*Keyboard shortcut to close the window:* Esc

The data memory editor is used for displaying the content of the additional memory in some devices (typically EEPROM).

Not all devices have an additional memory. This means the editor may not be available for some devices.

## Configuration Memory Editor

*Menu:* ***View → Display configuration memory***

*Keyboard shortcut to open the window:* F12
*Keyboard shortcut to close the window:* Esc

The configuration memory editor displays settings that are to be programmed in a device, but are not part of any of the above memory types.

The configuration memory editor content depends on the selected device type. It is necessary to get acquainted with the device's data sheet in order to get a closer explanation of individual options in this window.

Not all devices need configuration data. This means the editor may not be available for some devices.

### Tips for Advanced Users

Even though the configuration memory may be presented as a set of settings, in reality it is nothing more than a memory that can be approached cell by cell. Due to this, it is possible to display the memory in this way.

This can be achieved by activating ***Options → Program settings → Editors → Configuration memory editor: show cfg word instead of fuses*** option or by double-clicking the configuration memory window.

The device's ID positions (do not mistake for Device ID) can also be found in the configuration memory window. ID positions can be programmed with values identifying the device, such as the serial number, for example. ID positions can always be read - even if the device is locked against reading.

According to the recommendation of the Microchip, ID positions should not be programmed with any value; only a certain number of bits (typically 4) should carry data for identification while other bits should be programmed with the default value.

This can be achieved by activating ***Options → Program settings → Editors → Mask ID positions....***

# 5.6 Running UP from Command Line

The UP program can alternatively be controlled from the command line.

The program itself makes sure that it always runs in one instance only (with one class name). Should a second instance (with the same class name) be started, parameters from the command line are transferred to the first instance for execution. Only the /p, /pdiff, /blank, /verify, /erase, /read, /noe, /eeonly, /noboot, /boot, /code, /cfg parameters are transfered to the running instance. UP can be started in more instances using different class name, see **/wnd** in List of Parameters.

On the commandline it is recommended to use project file, it contains important settings for programming. Some project settings can be modified using commandline parameters. Commandline parameters have higher priority than the settings from the project defined on the commandline.

## 5.6.1 List of Parameters

UP.EXE [{**/ask** | **/q** | **/q1**}]
[{**/e** ***File_with_eepromm.hex*** | [**/noe**]}] [{**/p** |[**/pdiff**]| [**/o**]} ***File.hex*** | ***File.ppr***] [**/df** ***File.hex***] [**/part** ***device_name***] [**/eeonly**] [**/erase**][**/w**[**nd**] ***up_window_class***] [**/cfg**] [**/devid**] [**/blank**] [**/verify** ***File***] [**/read** ***File***] [**/s** ***SN_programmer's***][**/progname** ***name***] [**/noboot**] [**/boot**] [**/code**] [**/getpartrev**] [**/sn** ***serial_number***] [**/conf** ***file***]

### Legend

- Text presented here in **bold** is to be keyed in in the command line exactly as printed here.
- Text presented here in *italics* is to be replaced by the corresponding parameter. For example, *file_name* should be replaced with the real name of a file to be opened.
- Text in braces {} separated by the | sign represents a selection of one of the presented options. For example, { A | B } means “choose either A or B”.
- Text in square brackets [] presents an optional parameter – it can be keyed in but does not have to.
- Text in quotation marks “ ” is a mnemonic.

**/ask** To be used linked with /p. The program always asks if it is to continue before launching the actual device programming even if the program settings specify not to ask. The dialog displays the selected device type.

**/q /quiet** “Quiet” mode. The program does not ask any questions. If it needs to display a dialog, the execution finishes with an error. See UP program return codes. An external application can monitor UP work flow by ProgressBar value reading, see Work flow monitoring.

**/q1** “Quiet” mode 1. It works same like the /q parameter, but shows the Status form, which is closed after programming regardless of errors. UP does not provide with the ProgressBar value when the /q1 parameter was used.

**/e** *file* “EEPROM” file. For keying in the name of a file with data for data memory, if needed. If the file name contains space(s), it must be enclosed by inverted commas.

**/noe** “No EEPROM”. The program skips the device data memory programming. If this parameter is used for MSP430 microcontroller programming, the information memory is erased and programmed.

**/p** *file* To “program”. The keyed in file will be programmed. The file can be a data file, e.g. hex file, or UP project file. If the file name contains space(s), it must be enclosed by inverted commas.

**/pdiff** *file* To “program the difference”. The keyed in file gets programmed by the differential algorithm. If the file name contains space(s), it must be enclosed by inverted commas.

**/o** *file* To open. The keyed in file opens. The file can be a data file, e.g. hex file, or UP project file. This parameter is optional. If the file name contains space(s), it must be enclosed by inverted commas.

**/df** *file* When there is UP project file specified after the /p or /o parameter and user wants to use the project and to change data file only, the /df parameter with the new data file name can be used for this purpose. If the file name contains space(s), it must be enclosed by inverted commas.

**/eeonly** This executes the selected operation only with the data (EEPROM) memory or only with the information memory in the case of MSP430.

**/part** name This selects the required device in UP.

**/erase** This erases the device.

**/wnd** class name A different window class name. More than one instance of the UP program can be started using this parameter (running then simultaneously). Each new instance of the running UP must have a different class name.

**/cfg** If this parameter is used in combination with parameter **/p**, only the configuration memory gets programmed. This is useful for programming AVR microcontrollers, for example, which can be switched to a faster oscillator and then programmed much faster.

**/devid** If this parameter is used in combination with parameter **/p**, the software only checks the device's Device ID.

**/blank** This checks device erasure (blank check) and returns an error code if the result is not OK.

**/verify** *file* This verifies the device.

**/read** *file* This reads the device and saves the read content to the file.

**/s** *programmer_SN* This makes it possible to select a programmer by its serial number. The serial number is to be keyed in the way it is displayed in the UP program and printed on the programmer bottom. Example: 016709 or A6016709. When * sign is used instead of the serial number, then any available programmer of the selected type is used.

**/progname** *name* This makes it possible to select a programmer by its name such as PRESTO or FORTE for example.

**/noboot** This skips programming of the MSP430 boot memory.

**/boot** This performs the selected operation only with the MSP430 boot memory.

**Note**

PIC32 microcontrollers also have a boot memory. This, however, uses variables and a data memory form for programming, so it works with parameters for a data memory and not with parameters for a boot memory. A similar situation can be found in the information memory of MSP430 and CC430 microcontrollers.

**/code** This performs the selected operation only with code or main memory.

**/getpartrev** Only reads the revision of the device and returns revision + 0x10000 as the error code.

**/sn** serial_number Using this parameter it is possible to enter the value of the serial number, which is then written to the address in accordance with the serial numbers settings. In the settings, the serial numbers have to be configured as Computed, Manual. The number is entered as hexadecimal, e.g. 1234ABCD.

**/conf** *file* When UP closes, it saves its console content to the file, but only when the operation was started from the commandline.

**Note:** When no parameter requesting an operation is used, the UP starts and keeps running. When a parameter requesting an operation is used and the software is not running, the operation is executed and the software closes if it does not need user interaction.

# Using a Project File

When the user programs different devices, it can happen that the program is set up differently than expected. **Whenever it is possible, it is recommended to use project files** (.PPR) that hold all the program settings needed for programming and paths to the data files.

# Examples of Use

File Opening

```
up.exe project.ppr

up.exe "C:\My Documents\Recent Projects\PIC
\My latest project\project.ppr"
```

Device Programming

```
up.exe /p project.ppr

up.exe /p "C:\My Documents\Recent Projects\PIC
\My latest project\project.ppr"
```

## 5.6.2 Program Return Codes

| | |
|---|---|
| 0 | Problem-free execution. |
| 1 | File error. File not found or incorrect file format, for example. |
| 2 | Equipment error. Communication test failed, communication error. |
| 3 | Programming preparation error. device cannot be erased, etc. |
| 4 | Programming error. |
| 5 | Verification error. |
| 6 | Programming failed due to a need to communicate with user. |
| 7 | Device ID error. |
| 8 | Not supported. |
| 9 | Error of the serial number entered using the /sn parameter. |
| 10 | Error, device protected. |
| 0x10000 + const | Revision of the device read using /getpartrev parameter. |

**Note:** The return value can be found in the *%errorlevel%* variable if working with batch files.

## 5.6.3 Work flow monitoring

When the application was started in the quiet mode (with /q parameter) on the commandline, an external application can monitor its work flow by reading of the value of the main ProgressBar, it is being continuously saved to a Named Shared Memory.

The UP software shares a variable of size of one Integer. It can be accessed using OpenFileMapping and MapViewOfFile Windows functions. The name of the shared variable is the UP window class name plus _Progress string. E.g., when the name is set to up1 using the /w parameter, then the name of the shared variable is up1_Progress.

In the installation directory of the UP, in an example_ProgressBar subdirectory, there is a C language example of how to read the ProgressBar value.

# 5.7 Running UP by Means of Windows Messages

The UP program can be controlled by means of messages of the Windows operating system. The running instance of UP executes the requested command immediately after message receipt.

Messages must be sent to "up v1.x" class window. The message type is always of WM_USER.

Commands are identified by “wParam” while parameters by “lParam”.

## 5.7.1 List of Commands

Unless specified otherwise, the command returns to its return value:

| | |
|---|---|
| 0 | error, failed |
| 1 | everything worked OK |

Commands with a wParam of 1, 2, 3, 4, 5, 6, 7 and 24 are thread blocking.

| wParam | lParam | Description |
|---|---|---|
| 0 | 0 | does not perform anything, returns 1 |
| | 1 | SetForegroundWindow() |
| | 2 | Maximize, SetForegroundWindow() |
| 1 | any | programs everything, same return code as from the command line |
| 2 | any | programming excl. data memory, same return code as from the command line |
| 3 | see below | programming incl. erasing, same return code as from the command line |
| | bit 0 = 1 | code/main memory |
| | bit 1 = 1 | data memory (EEPROM) |
| | bit 2 = 1 | configuration memory |
| | bit 3 = 1 | boot memory (MSP430, CC430) |
| 4 | see below | reading |
| | bit 0 = 1 | code/main memory |
| | bit 1 = 1 | data memory (EEPROM) |
| | bit 2 = 1 | configuration memory |
| | bit 3 = 1 | boot memory (MSP430, CC430) |
| 5 | see below | differential programming |
| | bit 0 = 1 | code/main memory |
| | bit 1 = 1 | data memory (EEPROM) |
| | bit 2 = 1 | configuration memory |

| | | |
|---|---|---|
| | bit 3 = 1 | boot memory (MSP430, CC430) |
| 6 | see below | verification |
| | bit 0 = 1 | code/main memory |
| | bit 1 = 1 | data memory (EEPROM) |
| | bit 2 = 1 | configuration memory |
| | bit 3 = 1 | boot memory (MSP430, CC430) |
| 7 | see below | erasing, same return code as from the command line |
| | bit 0 = 1 | code/main memory |
| | bit 1 = 1 | data memory (EEPROM) |
| | bit 3 = 1 | boot memory (MSP430, CC430) |
| 8 | see below | BlankCheck of memory, same return code as from the command line |
| | bit 0 = 1 | code/main memory |
| | bit 1 = 1 | data memory (EEPROM) |
| | bit 2 = 1 | configuration memory |
| | bit 3 = 1 | boot memory (MSP430, CC430) |
| 15 | any | presses the GO button, returns 1 |
| 16 | see below | asks if the programmer supports the function |
| | 0 | MCLRControl_Run |
| | 1 | MCLRControl_Stop |
| | 2 | MCLRControl_Reset |
| | 8 | current voltage at pin VDD |
| 17 | see below | performs the programmer function |
| | 0 | MCLRControl_Run |
| | 1 | MCLRControl_Stop |
| | 2 | MCLRControl_Reset |
| | 8 | current voltage at pin VDD<br>**return value PRESTO:**<br>0 – unknown level<br>1 – 0 V<br>2 – approx. 2 V<br>3 – approx. 5 V |
| | | 4 – more than 6 V<br>**return value FORTE:**<br>-1 - measuring error<br>xx - measured voltage × 10<br>example: 33 means 3.3 V |
| 24 | address | reads data from address, returns error code, data are returned with wParam = 25 |
| 25 | any | returns last data read with wParam of 24 |
| 32 | see below | UP program initialization |
| | bit 0 = 1 | reload settings (reload project/ini file or registry) |
| | bit 1 = 1 | reload language file |
| | bit 2 = 1 | recreate programmer (like programmer was changed) |
| | bit 3 = 1 | reload programmer settings (like port settings) |
| | bit 4 = 1 | reload selected device |
| | bit 5 = 1 | reload current file (.hex, .bin, ...) |
| | bit 6 = 1 | recreate all dialog windows (adjust their size when reloading device) |
| | 0x 0100 | refresh device specific windows |
| | 0x 0200 | refresh all editors |
| | 0x 0300 | refresh project captions |
| 33 | 1 | save all project settings |
| 48 | see below[1] | saves current file (same as Ctrl+S), returns 0 when there was no error else returns a nonzero value |
| | bit 0 = 1 | code/main memory |
| | bit 1 = 1 | data memory (EEPROM) |
| | bit 2 = 1 | configuration memory |
| | bit 3 = 1 | boot memory (MSP430, CC430) |
| 56 | 0 | returns the handle of the UP main form |

*Table 14: Commands WM_USER*

A message of the WM_CLOSE type closes the UP program.

## Example of use

```
var
    window: HWND;
begin
    window := FindWindow('up v1.x', nil);
                    Result := SendMessage(window,
WM_USER, 0, 0);
end.
```

[1] Only parameter lParam=1 or lParam=2 are allowed for AVR microcontrollers.

# 5.8 UP_DLL.DLL Library

Thanks to this library strings can be exchanged with the UP program.

As the library needs to communicate with UP, the program must be running. UP_DLL cannot work on its own.

```
unit up_dll;
interface

Function UP_LoadFile (FileName: PChar; style: in
teger): integer; stdcall;
(*
* Load File (with extension .hex or .ppr);
* Loading of .ppr file can result in loading .he
x file too;
* Result codes are same like on command
* line.
*
* Style |
= 1; UP will be quiet on file load errors
* Style |
= 2; UP will do no previous file saving
*
*)

Function UP_GetStrValue(ValueName: PChar; Value:
 PChar; Size: integer): integer; stdcall;

Function UP_GetIntValue(ValueName: PChar; var Va
lue: integer): LongBool; stdcall;

Function UP_SetStrValue(ValueName: PChar; Value:
 PChar): LongBool; stdcall;

Function UP_SetIntValue(ValueName: PChar; Value:
 integer): LongBool; stdcall;

Function UP_LoadFile_Wnd(WndClass:PChar; FileNam
e: PChar; style:integer):integer; stdcall;

Function UP_SetStrValue_Wnd(WndClass:PChar; Valu
```

```
eName: PChar; Value:PChar): BOOL; stdcall;

Function UP_SetIntValue_Wnd(WndClass:PChar; Valu
eName: PChar; Value:integer): BOOL; stdcall;

Function UP_GetStrValue_Wnd(WndClass:PChar; Valu
eName: PChar; Value: PChar; Size: integer): inte
ger; stdcall;

Function UP_GetIntValue_Wnd(WndClass:PChar; Valu
eName: PChar; var Value: integer): LongBool; std
call;

(*
* All these functions are used for changing
* internal settings of UP in runtime.
* UP_GetIntValue, UP_SetStrValue,
* UP_SetIntValue returns nonzero if
* successful
* UP_GetStrValue returns amount of
* characters to copy into Value string
* including null terminator
* If Size is less than requied size, no
* characters are copied.
*)

Function UP_GetChecksum(Recalculate: integer; Ch
ecksum: PChar; Size: integer): BOOL; stdcall;

Function UP_GetChecksum_Wnd(WndClass:PChar; Reca
lculate: integer; Checksum: PChar; Size: integer
): BOOL; stdcall;

(*
* These functions return checksum value from the
 UP, where:
* Recalculate = 0; only returns the checksum
* Recalculate = 1; checksum will be recalculated
 first
* Checksum; the returned checksum as string
* Size; the length of the Checksum array, the ca
ller have to fill
* WndClass; the UP window class name
*)

implementation

function UP_LoadFile; external 'up_dll.dll';

function UP_GetStrValue; external 'up_dll.dll';

function UP_GetIntValue; external 'up_dll.dll';

function UP_SetStrValue; external 'up_dll.dll';

function UP_SetIntValue; external 'up_dll.dll';

function UP_LoadFile_Wnd; external 'up_dll.dll';


function UP_SetStrValue_Wnd; external 'up_dll.dl
l';

function UP_SetIntValue_Wnd; external 'up_dll.dl
l';

function UP_GetStrValue_Wnd; external 'up_dll.dl
l';
function UP_GetIntValue_Wnd; external 'up_dll.dl
l';

function UP_GetChecksum; external 'up_dll.dll';
function UP_GetChecksum_Wnd; external 'up_dll.dl
l';

end.
```

For further information see Appendix A UP_DLL.DLL.

## 5.9 Running More Than One Instance of UP

If you need to connect more than one programmer to one computer, a separate instance of UP must be running for each programmer connected.

UP should be running in just one instance for standard use. Each additional instance just sends commands from

the command line to the already running first instance or makes it visible in another way.

However, UP can be started and run in several instances if used together with the parameter **/w** and a different window class name. Those programs (instances) having the same window class name will communicate with each other.

Parameters for the program controlled from the command line are described in Running UP from Command Line.

### Example of use

The first instance of UP can be started the standard way from the start menu.

Additional instances can be started from the command line as **up.exe /w "another up"**, for example.

## 5.10 Access of More UP Instances to One Programmer

Only one program or utility can access one programmer at any one time.

The operating system takes care of the access rights of programmers connected to the USB interface.

If UP is running, the operating system does not allow any other program to access the selected programmer as UP needs to continually check the GO button state and the voltage on the feeding pin.

UP can temporarily be prohibited to access the programmer and the programmer freed this way for another application. This can be done by choosing ***Options → Select programmer***. As long as this dialog is open, another program can access the programmer. The data the system works with in HEX editors does not get lost by closing this dialog window.

## 5.11 Updating UP

If a newer program version is available on the web, it is recommended to update the existing version as it may eliminate known bugs or make the programming algorithm for a particular device more efficient.

New versions also add newly supported types of devices.

Updates are free and are very easy. You can download a new program version from https://www.asix.tech/prg_up_en.html and simply install it over the previous one by running the installer and clicking the **Next** button till it finishes.

You do not have to worry about losing the settings you created in your current program version. The installation preserves all of them.

You can set an automatic check for newer versions on the web in *Options → Program settings → Others*.

## 5.12 Appendix A UP_DLL.DLL

This appendix deals with the names of settings and values of functions in the UP_DLL.DLL library.

Please look at the sample batch files in UP installation folder for better understanding.

The following information is provided only for experienced users and does not provide any guarantee.

### 5.12.1 Data Types

| | |
|---|---|
| string | is string |
| integer | signed 32bit value |
| boolean | accessed like integers; 0 is false, other value is true |

## 5.12.2 List of UP variables

### Prog.LoadFileBfgProg
boolean

If true, current file is reloaded before device (or its part) is programmed.

### Prog.LoadFileBfgProgWarnMod
boolean

If true, the program warns when it is set to reload current file before programming and data in some editor have been manually modified.

### Prog.LoadFileBfgProgWarnOnNoChange
boolean

If true, the program warns when it is set to reload current file before programming and the content of the loaded file has not changed from the previous programming.

### File.AutoCheck
boolean

If true, current file is periodically tested for changes.

### File.LoadOnModify
boolean

If true, when change is detected, question pops up.

### FileLoad.ClearData

### FileLoad.ClearCfg

### FileLoad.ClearID

### FileLoad.ClearCode
boolean

If true, the contents of code memory is erased (in UP memory) before new file is loaded;

all cells not stored in the file will have its default (blank) state.

### Part.Name
string

Selected device name

### Prog.Name
string

Selected programmer name

Possible values are PREST, FORTE.

### Prog.PortBase
integer

Serial number of programmer

### Prog.UseOnlyThisSN
boolean

If true, the SN saved in the up.ini is always used regardles of the SN contained in a project file.

### LanguageFile
string

Relative path to used languange file

### Project.File
string

Project file path

### Project.Present
boolean

### Project.Template
boolean

If true, the user is asked for project name before its saving.

## HexFile.File
string

Opened current file path

## HexFile.Present
boolean

## HexFile.IHex
boolean

If true, the file with extension different from .HEX or .BIN is loaded as HEX file.

## HexFile.Template
boolean

If true, the user is asked for name before saving.

## HexFile.SaveVoid
boolean

If true, empty cells are saved, too.

## HexFile.AskForBinEndian
boolean

If true, the user is asked if the loaded BIN file should be loaded as Little or Big endian.

## HexFile.BinLittleEndian
boolean

If true, the loaded BIN file is loaded as Little endian.

## HexFile.LoadDataAuto
boolean

If true, with file is loaded also a separate file for data memory.

## HexFile.LastDataFile
string

Path of a separate file for data memory.

## HexFile.DataIHex
boolean

If true, the file for data memory with extension different from .HEX or .BIN is loaded as HEX file.

## Prog.QBfrEraseFlash
boolean

Question before erasing flash devices

## Prog.QBfrProgFlash
boolean

Question before programming flash devices

## Prog.QBfrProg
boolean

Question before programming OTP devices

## Prog.QBfrDiffProg
boolean

Question before differential programming (of flash devices)

## Prog.QBfrProgCP
boolean

Warning before programming device with some kind of protection

## Prog.CloseStatOnGoodAct
boolean

If true, status window will be automatically closed after read/verify etc... without errors.

## Prog.CloseStatOnGoodProg
boolean

If true, status window will be automatically closed after programming without errors.

## Prog.BeepOnGoodProg
boolean

If true, program makes a sound on successful programming.

## Prog.BeepOnBadProg
boolean

If true, program makes a sound on unsuccessful programming.

## Prog.SoundsOff
boolean

If true, all sounds of the program are switched off.

## Prog.SkipBlankForCfg
boolean

If true, no blank check of part is performed before programming of configuration space.

## Prog.SkipBlankCheck
boolean

If true, no blank check of device is performed before device programming.

## Prog.SkipErase
boolean

If true, no erasing is performed before device programming.

## Prog.SkipEraseData
boolean

If true, no erasing is performed before only data memory programming.

## Prog.SkipLastFFVerify
boolean

If true, verification of empty FF positions at the end of the programmed memory is not preformed.

## Prog.SkipVerify
boolean

If true, no verification is performed after programming.

## Prog.DoubleVerify
boolean

If true, verification on two supply voltages is performed after programming. Supported by FORTE with internal supply voltage only.

## Prog.DoubleVerifyV1
integer

Defines supply voltage size for the first verification, the value is in volts x 10.

## Prog.DoubleVerifyV2
integer

Defines supply voltage size for the second verification, the value is in volts x 10.

## Serial
integer

| | |
|---|---|
| 0 | no serial numbers |
| 1 | serial numbers are from external file |
| 2 | serial numbers are calculated |

## Serial.Step
integer

Step of serial numbers

## Serial.File
string

File name of external file with serial numbers

## Serial.File.Next
string

Label of serial number

## Serial.Length
integer

If serial number is computed, serial number length (digits)

## Serial.Actual
(unsigned) integer

If serial number is computed, actual computed serial number (if decimal, coded as BCD)

## Serial.ASCII
boolean

If serial number is computed, if true, serial number is stored to device as ASCII characters.

## Serial.SaveTo
integer

| | |
|---|---|
| 1 | code/main memory |
| 2 | data memory |
| 4 | ID positions |

## Serial.Retlw
boolean

If serial number is computed, if true, memory cells are filled with retlw instructions.

## Serial.Addr
integer

If serial number is computed, address where to save

## Serial.CPW
integer

If serial number is computed, chars per word

## Serial.Base
integer

If serial number is computed, base of serial number, can be only 10 or 16

## Serial.Succ
integer

next serial number is

| | |
|---|---|
| 0 | same |
| 1 | incremented |
| 2 | decremented |
| 3 | random (LSFR) |

## Serial.Order
integer

| | |
|---|---|
| 0 | HiLo hilo |
| 1 | hilo HiLo |
| 2 | LoHi lohi |
| 3 | lohi LoHi |

## Serial.Write.BeforeProg
boolean

If true, current serial number is "written" into opened HEX editors just before programming the device.

## Serial.Write.AfterProg
boolean

If true, current serial number is "written" into opened HEX editors after successful programming.

## Serial.Succ.AfterProg
boolean

If true, next serial number is generated after successful programming.

## Serial.LogSN
boolean

If true, the result of programming is logged to a selected file.

## Serial.LogFile
string

File name of a file where the result of the programming will be logged.

## ICSP.LongTime
boolean

If true, longer times for switching Vcc are taken.

## ICSP.LongTime.Time.SwOn
integer

Time to wait after Vcc is switched on in microseconds.

## ICSP.LongTime.Time.SwOff
integer

Time to wait after Vcc is switched off in microseconds.

## SpecSettings.PREST.Power
integer

| | |
|---|---|
| 0 | idle power supply is None / External |
| 1 | idle power supply is Internal 5 V |

## SpecSettings.PREST.ProgPower
integer

| | |
|---|---|
| 0 | power supply during programming is External 2.7 to 5.5 V |
| 1 | power supply during programming is Internal 5 V |

## SpecSettings.PREST.i2cSpeed
integer

| | |
|---|---|
| 0 | 100 kHz |
| 1 | 500 kHz |
| 2 | 1 MHz |
| 3 | Maximal |

## SpecSettings.PREST.i2cAddr
integer

| | |
|---|---|
| 0 | first suitable address or N/A |
| 1 | second suitable address |

etc...

## SpecSettings.PREST.LVP
integer

| | |
|---|---|
| 0 | HVP method |
| 1 | LVP method |

## SpecSettings.PREST.PICAlg
integer

| | |
|---|---|
| 0 | automatic selection |
| 1 | assume VDD = 5 V |
| 2 | assume VDD < 5 V |

## SpecSettings.PREST.UsePE
boolean

If true, PE (programming executive) is used for programming of PIC MCUs.

## SpecSettings.PREST.PIC32MZ_BootProg
integer

| | |
|---|---|
| 0 | Lower Boot alias |
| 1 | Boot Flash 1 and 2 |

## SpecSettings.PREST.PSoCAlg
integer

| | |
|---|---|
| 0 | automatic selection |
| 1 | assume VDD > 3.6 V |
| 2 | assume VDD =< 3.6 V |

## SpecSettings.PREST.PSoCRSTInit
integer

| | |
|---|---|
| 0 | Reset mode |
| 1 | Power cycling |

## SpecSettings.PREST.MSP430osc
integer

| | |
|---|---|
| 0 | Calibrated internal RC oscilator |
| 1 | Not calibrated internal RC oscilator |

## SpecSettings.PREST.MSP430speed
integer

| | |
|---|---|
| 0 | Normal |
| 1 | Slow |
| 2 | Slowest |

## SpecSettings.PREST.MSP430EraseSegmentA
boolean

If true, MSP430 Segment A is erased.

## SpecSettings.PREST.SPI_Flash_Freq
integer

| | |
|---|---|
| 0 | 3 MHz |
| 1 | 1.5 MHz |
| 2 | 750 kHz |

## SpecSettings.PREST.AVRXTAL.CLK

## SpecSettings.PREST.AVRXTAL.RPT
integers

represent maximum AVR oscillator frequency

values can be found in *.lng files at item

MainForm.PRESTSpecForm.ComboAVRXTAL.xxx.Items
where xxx is minimum divisor of system clock of selected AVR's SPI module. This is 2 for new AVRs, 3 and 4 for older AVRs and 24 for Atmel's 8051 arch. processors.

These settings can be found in ini file too at [SpecSettings.PREST], XTALRpt and XTALClk.

## SpecSettings.PREST.AVRXTAL.AutoClk
boolean

If true, faster programming with slow clock is used for AVR MCUs.

## SpecSettings.PREST.AVRRSTInverse
boolean

If true, the reset signal polarity is assumed to be inverse in comparison with standard polarity of the reset signal of the selected chip. It is supported with AVR and 8051 devices.

## SpecSettings.PREST.AVRHVP
boolean

If true, high voltage method is used for AVR MCUs, it is supported for AVR TPI interface only.

## SpecSettings.FORTE.UVCCLevel
integer

Defines the size of the internal supply voltage provided by

FORTE programmer. The value is in volts x 10.

## SpecSettings.FORTE.Power
boolean

If true, idle power supply is provided by FORTE programmer.

## SpecSettings.FORTE.ProgPower
boolean

If true, the supply voltage during programming is provided by FORTE programmer.

## SpecSettings.FORTE.LVP
integer

| | |
|---|---|
| 0 | HVP method |
| 1 | LVP method |

## SpecSettings.FORTE.UsePE
boolean

If true, PE (programming executive) is used for programming of PIC MCUs, when programmed by FORTE programmer.

## SpecSettings.FORTE.PIC32MZ_BootProg
integer

| | |
|---|---|
| 0 | Lower Boot alias |
| 1 | Boot Flash 1 and 2 |

## SpecSettings.FORTE.PSoCAlg
integer

| | |
|---|---|
| 0 | automatic selection |
| 1 | assume VDD > 3.6 V |
| 2 | assume VDD =< 3.6 V |

## SpecSettings.FORTE.PSoCRSTInit
integer

| | |
|---|---|
| 0 | Reset mode |
| 1 | Power cycling |

## SpecSettings.FORTE.MSP430osc
integer

| | |
|---|---|
| 0 | Calibrated internal RC oscilator |
| 1 | Not calibrated internal RC oscilator |

## SpecSettings.FORTE.MSP430speed
integer

| | |
|---|---|
| 0 | Normal |
| 1 | Slow |
| 2 | Slowest |

## SpecSettings.FORTE.MSP430EraseSegmentA
boolean

If true, MSP430 Segment A is erased.

## SpecSettings.FORTE.MSP430Interface
integer

| | |
|---|---|
| 0 | JTAG |
| 1 | SBW |

## SpecSettings.FORTE.ARMFreq
integer

The value specifies the frequency of connected oscillator or crystal. This value is used for programming of ARM7TDMI.

## SpecSettings.FORTE.ARMOscType
integer

| | |
|---|---|
| 0 | Crystal |
| 1 | Ext. Clock |

2 Int.RC 32 kHz

### SpecSettings.FORTE.ARMCommFreq
integer

Selects communication frequency between FORTE programmer and the programmed chip. The frequencies are same as stated in the "Communication frequency" ComboBox in UP. 0 means the highest frequency, 1 is lower and so on.

### SpecSettings.FORTE.SPI_Flash_Freq
integer

Selects communication frequency between FORTE programmer and the programmed chip. The frequencies are same as stated in the "Communication frequency" ComboBox in UP. 0 means the highest frequency, 1 is lower and so on.

### SpecSettings.FORTE.AVRXTAL.DELAY
integer

represents maximum AVR oscillator frequency

value can be found in *.lng files at item

MainForm.PRESTSpecForm.ComboAVRXTALPresto2.xxx.Items where xxx is minimum divisor of system clock of selected AVR's SPI module. This is 2 for new AVRs, 3 and 4 for older AVRs and 24 for Atmel's 8051 arch. processors.

This setting can be found in ini file too at [SpecSettings.FORTE], XTALClk.

### SpecSettings.FORTE.AVRXTAL.AutoClk
boolean

If true, faster programming with slow clock is used for AVR MCUs.

### SpecSettings.FORTE.AVRRSTInverse
boolean

If true, the reset signal polarity is assumed to be inverse in comparison with standard polarity of the reset signal of the selected chip. It is supported with AVR and 8051 devices.

### SpecSettings.FORTE.AVRHVP
boolean

If true, high voltage method is used for AVR MCUs, it is supported for AVR TPI interface only.

### SpecSettings.FORTE.ATxmegaInterface
integer

0 PDI

1 JTAG

### SpecSettings.FORTE.i2cSpeed
integer

0 100 kHz

1 400 kHz

2 1 MHz

3 Maximal

### SpecSettings.FORTE.i2cAddr
integer

0 first suitable address or N/A

1 second suitable address

etc...

# 5.13 Appendix B: Use of ICSP

**ICSP** (In-Circuit Serial Programming) is a method of PIC microcontroller programming making it possible to program devices already placed on PCBs.

Two different algorithms may be used for PIC microcontroller programming: HVP (using programming

voltage on pin -MCLR/VPP) or LVP (using the LVP pin).

The LVP programming algorithm can be disabled in the device's configuration word. Microcontrollers have the LVP algorithm enabled from the production, therefore their PGM input needs to be treated during the first programming (PGM input must be in log.0 for the time of programming by means of the HVP algorithm).

**Note:** Not all devices have the PGM pin.

## 5.13.1 Pins Used for Programming

This chapter describes how to treat pins in the ICSP programming mode in accordance with the programming algorithm.

### HVP Algorithm

- PGM pin (if the device has one) must be maintained in log. 0!!
- -MCLR/VPP must be separated from the resetting circuits (with a 10 kΩ resistor, for example).
  A programming voltage is supplied to this pin P(VPP) for programming. The leading edge and the voltage level at VPP must not be influenced by the application. The PRESTO programmer supports only fixed voltage of 13 V on pin P(VPP) in contrast to adjustable voltage within a range of 6.5 V to 17 V supported by FORTE. Please check limit voltage value on pin -MCLR/VPP of the PIC device to be programmed by PRESTO.
- RB6 and RB7 pins must not be influenced by the application during programming.

### LVP algorithm (without VPP)

- RB6, RB7, PGM and -MCLR/VPP pins must not be influenced by the application during programming. All pins are in various logical levels during programming.

### Loading of Different Programmer Pins

The maximum current drawn from I/O pins, from pins P(VPP) and VDD can be found in the Technical Specification.

OTP (One-Time Programmable) devices have a considerably higher current consumption on pin P(VPP) than devices with a FLASH memory. Therefore the application should not have any additional current consumption at pin P(VPP) in case of OTP.

Data pins may have their signals changing at a speed of several MHz, so the application must not influence the signals in any significant way.

## 5.13.2 Power Supply Options

Connecting the shared data and power supply ground (GND) is obviously necessary for all cases. The microcontroller being programmed can be fed:

- externally from an application
- internally from a programmer

The external power supply from the application cannot be used for some types of microcontroller having the -MCLR/VPP pin alternatively configurable as an I/O.

The internal power supply can be used only in cases where the application does not draw too much current from programmer's feeding pin (VDD). The maximum allowed current consumption can be found in the Technical Specification.

Both PRESTO and FORTE programmers feature an overcurrent protection. PRESTO has a software protection in which an output gets disconnected by the running UP program having detected an overload. FORTE has a hardware protection, which does not depend on the control software.

The programmer switches the power supply off if the

maximum load is substantially exceeded for a certain time (this time is adjustable). PRESTO checks for the overload state the whole time the power supply is connected.

Support for an external power supply is integrated in both PRESTO and FORTE hardware. The programmer feeds the input and output circuits with a voltage connected to pin VDD. The voltage may be lower than 5 V.

**Advice**
When designing the process, please pay special attention to the type of device to be programmed – if it can be not only operated but also programmed at a voltage lower than 5 V.

# Power Supply Capacities in Application

If there are capacities present at the application's power-supply pin that would slow down the power on/off switching and application is fed from programmer during programming, longer charging/discharging times need to be set in UP. In other case UP will probably announce overcurrent on VDD pin.

An approximate time that should be set in the program:

$t[\mu s] = 2.5 \times C[\mu F] \times R[\Omega]$.

The equivalent resistance is presented in table following table:

| Programmer | Charging current | Discharging current |
|---|---|---|
| PRESTO | Corresponds to 50 Ω | Corresponds to 1 kΩ |
| FORTE | Corresponds to 50 Ω | Corresponds to 27 Ω |

*Table 15: Equivalent resistance of pin VDD*

**Example**

The necessary charging time for an application with a 33 µF capacitor programmed with PRESTO:

$2.5 \times 33 \times 50 = 4125$ µs

discharging:

$2.5 \times 33 \times 1000 = 82.5$ ms.

For further information on the setting see ▸ Delay for VDD switching on/off when supplied from programmer.

**Notes**

- An error may occur if UP cannot program the calibration word or if a fault occurs during Device ID reading, UP warns about an overcurrent at VDD, etc. This may be eliminated by prolonging the charging/ discharging time in the ▸ Delay for VDD switching on/ off when supplied from programmer option - up to several seconds.
- If UP points out an overcurrent error at VPP, a shorter ICSP cable may help (its maximum length is 15 cm).

## 5.13.3 ICSP Connector

All ASIX programmers use a unified connector with 2.54 mm spacing of pins for programming with the ICSP algorithm.

This connector has 8 pins with 7 signals. Not all signals are always needed for the programming itself.

For further information on connecting a device to the programmer see Connection Examples.

| Pin No. | Signal | Programming connector |
|---|---|---|
| 1 | -MCLR | P / VPP |
| 2 | | not used (key) |
| 3 | VCC | VDD |
| 4 | GND | GND |
| 5 | RB7 | D / DATA |
| 6 | RB6 | C / CLOCK |
| 7 | | not used |
| 8 | RB3/RB4/RB5 | L / LVP |

*Table 16: ICSP connector*

The following drawing illustrates the recommended connection of -MCLR/VPP pin found in Microchip microcontrollers for device programming through the ICSP interface.

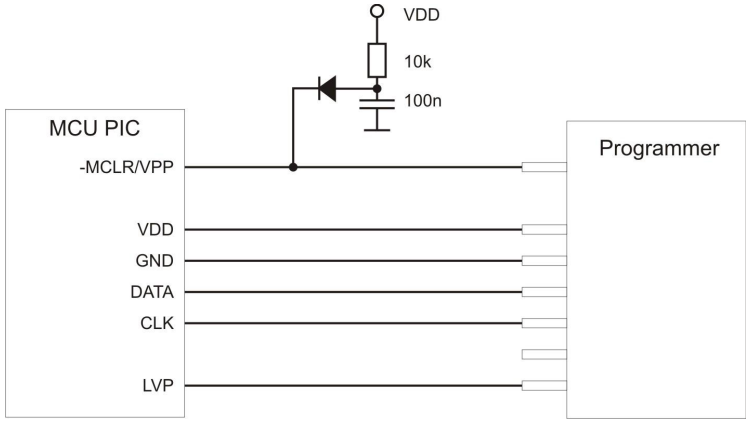


*Fig. 48: Recommended connection of -MCLR/VPP pin*

# 5.14 Appendix C: Intel-HEX File Format

This appendix describes the format of Intel-HEX files used by UP for data reading and saving. .hex is the typical extension for such a file.

## 5.14.1 Supported Alternatives of HEX Files

UP supports 2 basic alternatives of Intel-HEX files:

- "basic", sometimes also 8-bit Intel-HEX File (e.g. Microchip MPASMWIN generates this file with parameter INHX8M).
- "extended", sometimes also 32-bit Intel-HEX File (e.g. Microchip MPASMWIN generates this file with parameter INHX32).

## 5.14.2 Description of Intel-HEX File Format

Intel-HEX are text files consisting of lines.

Each line has the following structure:

***:LLAAAATTDDDD...CC***

| | |
|---|---|
| „:“ | Each line in a file must start with this character (colon, 0x3A). |
| LL | Length of record (number of DD fields). |
| AAAA | Address of record's first byte. |
| TT | Record type. The types may be:<br>00 - Data<br>01 - End of file. Each file must finish with this record.<br>02 - Extended segment address. (32-bit Intel-HEX only)<br>04 - Extended linear address. (32-bit Intel-HEX only) |

There are also other types like 03 and 05, which are ignored by UP while reading and not used for saving.

| | |
|---|---|
| DD | Record data. The number of bytes must be exactly LL. |
| CC | Checksum. The check sum is computed as a binary supplement to the sum of all values on the line. |

### Data Record

A line with the configuration memory of a 14-bit device is used here as an example.

***:02400E00413F30***

| | |
|---|---|
| 02 | Record length. The configuration memory size is one word = 14 bit = 2 byte (byte alignment). |
| 400E | Record address. The configuration memory address is the word 2007h addressed by bytes, i.e. 400Eh. |
| 00 | Record type = data |
| 413F | Record data. Configuration word =3F41h |
| 30 | Check sum. 30 = 02 + 40 + 0E + 00 + 41 + 3F = xxD0; neg D0 = 30 |

### End of File

The only acceptable alternative of the End of file line is:

***:00000001FF***

### Extended Linear Address

Only files that need to address more than 64 kB of the addressed space contain this line.

For example, processors of the PIC18F family have their configuration memory saved at the address 0x 30 00 00 00.

If this address needs to be used, it is necessary to insert a line in the .hex file specifying the extended linear address, i.e. the upper 16 bites of the address. The lower 16 bits are read from the line with the data record.

***:020000040030CA***

This line selects the configuration memory of PIC18F family devices.

The extended segment records specify the segment, i.e. bits 19-4 of the address, which are then added to addresses from the data records (offset).

# Saving Device Type in .hex File

Users frequently mistook the selected device type for the device type for which the .hex file was saved. To eliminate this, UP contains a function allowing you to save the device type in a file.

The program adds one more line below the End of File: **#PART=....** The absolute majority of programs working with Intel-HEX files ignore this line, but such a modified file cannot be considered as complying with the Intel-HEX format.

# 6

# up_control.dll library

The up_control.dll enables user to control the UP software using functions contained in the library. It contains basic programming functions.

From the view of the user the UP software is running invisibly.

There is more information concerning the library available in a separate document.

# 7

# FORTE.DLL Library

Functions implemented in forte.dll enable setting and reading of logical levels on single pins of FORTE programmer. This way it is possible to make various communication protocols.

Except for functions which enable controlling of the single pins, the library contains also functions prepared for communication via SPI, $I^2C$ and 1-Wire buses, functions for supply and programming voltage controlling and supply voltage and GO button reading.

The functions implemented in the **forte.dll** library are described in detail in a separate document devoted to this library.

# 8

# JTAG PLAYER

JTAG PLAYER is a utility designed for programming devices with the JTAG interface by means of PRESTO or FORTE programmers.

This utility is not part of the UP software installation pack but it can be downloaded separately from https://www.asix.tech/prg_jtag-svf-player_cz.html

JTAG PLAYER is localized only in English.

## 8.1 JTAG Device Programming

Having started JTAG PLAYER, select your programmer in ***Options → Select Programmer***.

If you cannot see your connected programmer in the dialog window, check if the green ON-LINE LED on the programmer is on. If so, make sure thatanother software is not communicating with the programmer.

Program the device by choosing ***File → Open&Process***. The file to be programmed must be in SVF/XSVF format. For more information on the file format see SVF File or XSVF File.

If an error occurs during the programming process, check the connection between the programmer and the application. In addition, check the presence of a feeding voltage.

**Important warning**
A power supply voltage from an external source must be present during programming that utilizes the JTAG interface. The programmer does not feed the application in this case.

### 8.1.1 SVF File

SVF (Serial Vector Format) is a file used for describing high-level operations on the IEEE 1149.1 bus.

Serial Vector Format (**.svf) is the recommended file format for all the testing and programming except for Xilinx CPLD XC9500. The Xilinx Serial Vector Format (**.xsvf) is recommended for Xilinx CPLD XC9500.

## Examples of How to Create SVF Files

This chapter presents methods of creating an SVF file for different types of device.

### Atmel AVR (e.g. ATmega128) Programming

Generate an SVF file using the **avrsvf.exe** program available at the ATMEL website in the “Tools & Software of AVR 8-bit RISC MCUs” section.

**Example:**

```
avrsvf -datmega128 -s -e -ifmyfile.hex -pf -vf -ovmyfile.svf -mp
```

This example shows how to create an SVF file from the *myfile.hex* file. The final SVF file is then used by jtagplay.exe for erasing, programming and verification. **Run avrsvf -h** for more information.

The device must be in the reset state during programming.

**Notes**

Some AVR devices do not support page by page programming. In such a case, the SVF file must be created without the -mp parameter.

### Lattice CPLD Programming

SVF file can be created from a .JED file using ispVM System. This program is part of the ispLEVER Classic system, which is available at the Lattice website.

### Altera CPLD Programming

The QUARTUS II program by Altera can generate SVF files if it is set up so in the menu.

Such an SVF file, however, cannot be used as it is due to a wrong Silicon ID. According to Altera, SVF files generated by their software are assumed only for ATE (Automatic Test Equipment) programmers and Altera does not consider supporting others.

However, such an SVF file can be manually modified to suit our needs. In order to do so, delete the "CHECKING SILICON ID" section in the SVF file or mark it as a non-executable comment.

## State of .svf File Implementation

The SVF file support has been implemented in accordance with the "Serial Vector Format Specification, Revision E" available at www.asset-intertech.com/support/svf.html with the following limitations:

- PIO a PIOMAP commands are not implemented.
- HDR+SDR+TDR / HIR+SIR+TIR length is limited to 2^31 bits.
- Supported TCK frequencies are 3 MHz, 1.5 MHz, 750 kHz and fragments of 1 MHz starting at 500 kHz for PRESTO. FORTE adds 15 MHz, 10 MHz and 5 MHz on top of these.
- RUNTEST MAXIMUM max_time SEC parameter is ignored.
- RUNTEST run_count is limited to 2^31/3 (approximately 715 million).
- RUNTEST min_time SEC is limited to 2^31/3 ms (approximately 715 seconds).
- TRST and RUNTEST SCK – these commands share the same configurable P/VPP pin of both the PRESTO and FORTE programmers and can never be used together.

# 8.1.2 XSVF File

XSVF (Xilinx Serial Vector Format) is a file used for describing high-level operations on the IEEE 1149.1 bus extended by Xilinx.

XSVF is recommended for Xilinx CPLD XC9500.

## Examples of How to Create XSVF Files

### Xilinx CPLD Programming

To create an XSVF file, use the iMPACT program available at the Xilinx website.

Choose ***Prepare Configuration Files → Boundary-Scan File → XSVF File*** in the ***Operation Mode Selection*** dialog window that opens after the iMPACT program starts. Run all operations (Erase, Program, Verify, Test...) the same way as if a programmer was connected (e.g. Xilinx Parallel Cable). Then save the new file and close iMPACT. Running the created XSVF file will perform the recorded operations.

We **do not recommend** using an SVF file for programming devices of the Xilinx XC9500 family, as the algorithm for XC9500/XL/XV device programming cannot be correctly described in the SVF format.

## State of XSVF File Implementation

The XSVF file support has been implemented in accordance with the "XAPP503, Appendix B: XSVF File Format" specification available at the Xilinx website with the following limitations:

- XSETSDRMASKS, XSDRINC and XSIR2 commands are not implemented.
- Only the XSVF file format is recommended for programming the Xilinx XC9500/XV/XL families.

We seriously recommend using an SVF file for all architectures except XC9500/XV/XL. For XC9500/XV/XL programming, the use of the XSVF format is recommended as the SVF format does not include the XREPEAT command, which is necessary for XC9500/XV/XL programming.

**Warning:** Executing a file containing the XREPEAT command may be very slow.

### 8.1.3 Programming Connector

The following table describes the functions of PRESTO's and FORTE's pins for programming by means of the JTAG interface.

| PRESTO | FORTE | Function |
|---|---|---|
| VPP | P | SCK (System Clock)/User defined state during file execution and after. |
| - | - | Not connected pin (key) |
| VDD | VDD | Feeding of I/O buffers |
| GND | GND | Ground of I/O buffers |
| MOSI | D | JTAG TDI (Test Data In) |
| CLOCK | C | JTAG TCK (Test Clock) |
| MISO | I | JTAG TDO (Test Data out) |
| LVP | L | JTAG TMS (Test Machine State) |

*Table 17: JTAG programming connector*

## 8.2 Settings

This chapter explains the meaning of different settings in ***Options → Program Options***.

### 8.2.1 Default TCK signal frequency

This frequency of the TCK clock is used until the JTAG Player reaches the first FREQUENCY command in the SVF file or until the FREQUENCY with the "default" value is reached.

The XSVF file format does not support the FREQUENCY default command. Therefore frequency defined by **Default TCK Signal Frequency** parameter is applied to all operations.

The maximum frequency is 3 MHz for the PRESTO programmer and 15 MHz for FORTE.

If the *Ignore FREQUENCY commands* option is selected, the programmer applies only the frequency set by the user and ignores all FREQUENCY commands.

### 8.2.2 Fast Clocks Option (FORTE only)

This option is available only for the FORTE programmer.

According to the JTAG specification, the signal at TDI is sampled at TCK's rising edge. However, should a higher frequency be required (around 5 MHz or more), it may be useful to change the moment of sampling from the rising edge to the falling edge by shifting it by 1/2 of the TCK period. To do so, choose **Fast Clock Option**.

### 8.2.3 RUNTEST without run_count (SVF only)

The programmer should stay in the specified state for the specified time and generate the clock signal on TCK while executing an SVF file.

The specified time can be exceeded, but it slows the programming process down. Even though it is not supported by the SVF specification, many programmable devices allow stopping the TCK clock during this time.

The programmer's ability to keep accurate time needs to be considered. A high level of accuracy cannot be achieved if the maximum frequency is used (the programmer can guarantee respecting the min_time SEC parameter only). A higher accuracy can be achieved with slow clock (~100 kHz). If the clock signal is not used at all, the programmer is capable of following the min_time SEC parameter almost exactly.

Three possibilities are available considering these facts:

- no clock on TCK
- slow clock on TCK (~100 kHz)
- default speed clock on TCK

Example: "RUNTEST 3E-3 SEC;" means "Generate clock on TCK for at least 3 ms".

### 8.2.4 RUNTEST Timing Multiply (both SVF and XSVF)

Recommended values:

- for accurate timing specified in the SVF or XSVF file: 0% (no added time)
- for XC9500(XL) family: 100% or more
- for Atmel AVR (e.g. ATmega128): 25%

### 8.2.5 RUNTEST with run_count and no timing (both SVF and XSVF)

This command should be interpreted as the minimum frequency on TCK.

Yet some SVF file generators (such as Xilinx iMPACT, for example) use this command as waiting time and assume a frequency of 1 MHz. For such cases the use of "interpret as RUNTEST min_time with scale 1 MHz" setting is recommended.

**JTAG Player Behavior When Reaching RUNTIME Command With MINTIME Specification**

(This concerns only SVF files as the RUNTEST alternative, which is usable in XSVF XRUNTEST run_count cannot specify time.)

- The RUNTEST command containing run_count and specifying min_time is executed at the currently running TCK frequency. Therefore the command can take much more time than what is specified in min_time.
- The RUNTEST command containing run_count and specifying max_time is executed at the currently running TCK frequency. As the programmer cannot respect the “deadline” specified in the max_time parameter, this parameter is ignored.

### 8.2.6 VPP PRESTO / P FORTE pin usage while running test (file) / after test completion

The selection of the VPP / P pin function: TRST or SCK as described in the SVF file or user-defined output levels. (This is useful to keep the device in the reset state during file execution.)

### 8.2.7 Default Settings

The JTAG Player features several default settings intended primarily for use in connection with FORTE, but not with PRESTO. Feel free to change these settings if the default values do not suit your application.

#### Default Settings for FPGAs

***Default TCK frequency:*** 15 MHz; Ignore FREQUENCY commands

***Fast Clock Option (FORTE only):*** 5 MHz and above

***RUNTEST without run_count (SVF only):*** default speed clock on TCK

***RUNTEST timing multiply (both SVF and XSVF):*** 0%

***RUNTEST with run_count and no timing (both SVF and XSVF):*** interpret as RUNTEST min_time with scale 1 MHz

***VPP PRESTO / P FORTE pin usage while running test (file):*** Tristate

***VPP PRESTO / P FORTE pin usage after test completion:*** Tristate

#### Default Settings for XC9500

***Default TCK frequency:*** 5 MHz; Ignore FREQUENCY commands

***Fast Clock Option (FORTE only):*** 5 MHz and above

***RUNTEST without run_count (SVF only):*** slow clock on TCK (~100 kHz)

***RUNTEST timing multiply (both SVF and XSVF):*** 100%

***RUNTEST with run_count and no timing (both SVF and XSVF):*** interpret as RUNTEST min_time with scale 1 MHz

***VPP PRESTO / P FORTE pin usage while running test (file):*** Tristate

***VPP PRESTO / P FORTE pin usage after test completion:*** Tristate

#### Default Settings for AVR:

***Default TCK frequency:*** 1 MHz; Ignore FREQUENCY commands

***Fast Clock Option (FORTE only):*** 5 MHz and above

***RUNTEST without run_count (SVF only):*** default speed clock on TCK

***RUNTEST timing multiply (both SVF and XSVF):*** 25%

***RUNTEST with run_count and no timing (both SVF and XSVF):*** interpret as RUNTEST min_time with scale 1 MHz

***VPP PRESTO / P FORTE pin usage while running test (file):*** Tristate

***VPP PRESTO / P FORTE pin usage after test completion:*** Tristate

# 8.3 Running JTAG Player from Command Line

SVF & XSVF JTAG Player can be run from the command line for more comfort especially during debugging. The following parameters apply:

***jtagplay.exe [-p] [-f filename] [-i inifile] [-c] [-cc] [-s serial]***

| | |
|---|---|
| -p | automatically executes the file specified in the -f filename parameter |
| -f *filename* | specifies SVF / XSVF file to be executed |
| -i *inifile* | ini file containing settings |
| -c | closes the program if the file has been executed without errors |
| -cc | closes the program even if the file has been executed with error(s) |
| -s *serial* | uses PRESTO or FORTE programmer of the specified serial number, when instead of the serial number there is the "-" (dash) sign, it will use any connected programmer regardless of its serial number |
| -forte | uses FORTE, not PRESTO |

## Return Codes

The jtagplay.exe program returns the following return codes:

| | |
|---|---|
| 0 | execution of the last file had no errors |
| 1 | an error occurred during execution of the last file |
| 2 | execution of the last file could not be started |

# 9

# MultiUP

MultiUP is utility usable for programming with up to 4 ASIX programmers at once. MultiUP uses program UP.

The utility is installed together with the UP program.

MultiUP can simply, using one button, run programming of selected task, which is deffined with the UP program project and other settings.

To increase speed the utility loads the programming data only once, it is suitable for using in production.

## 9.1 Programming settings

In the "Options/Programming settings" it is possible to select programmers to be used, UP program project to be programmed, whether they will program the common way or using the differential programming and the memories to be programmed.

To improve clarity it is possible to select a name for each of the programmers.

Each of the programmers can program a different project or all of them can be the same project.

All of the programmers do not have to be of the same type (PRESTO, FORTE), but the selected project have to use the same type of the programmer.

In the Options it is also possible to select Keyboard shortcuts for running work of the praticular programmers and to set the console saving during the program closing.

The parameters configured in the Options can be used once or they can be saved to MPPR file for other using.

## 9.2 Programming

For programming of predefined tasks it is advisable to open MultiUP project file in menu "File/Open MPPR file". The project can be crated in the "Options" menu.

Afterwards the particular programmings can be run separately using the "Run" button on the appropriate programmer panel or all the active programmings can be run at once using the "Run all" button, eventually the associated keyboard shortcuts can be used.

## 9.3 CommandLine

It is possible to provide the MPPR project path on the commandline, it will be loaded on the utility start.

Example:

multiup.exe C:\data\project.mppr

# 10

# TROUBLE-SHOOTING

This chapter describes how to proceed if experiencing difficulties with device programming. It also describes testing utilities for FORTE designed to check if the programmer is in good condition or damaged. Please go through all the tips and tricks before you send your programmer to a service shop.

The testing utilities are not part of the UP software installation, but they can be downloaded separately from https://www.asix.tech/prg_testers_cz.html. They are localized only in English.

## 10.1 Tips and Tricks

Should you experience difficulties with programming, we recommend checking the following points:

- Check the microcontroller and programmer connections according to the description in Connecting to Application. Even though it may seem trivial, it is recommended to check the connection twice and to assume the second time that the link does not always lead where you want it to or it leads to more than one point.
- If a crystal or other auxiliary devices are used for programming, (typically pull-down or pull-up resistors), check if their values and connections are correct.
- It is advisable (and often necessary) to have blocking capacitors at the power supply of the device to be programmed.
- If the device to be programmed has more than one power-supplying pin, all of them must be really fed. It is often recommended to check their voltage with a voltmeter.
- Use of the latest version of UP is recommended. If you are still using an older version, an update is recommended.
- Are you programming a new device or has it been previously programmed? If so, the problems may be caused by unsuitably set fuses in the device.
- The length of the ICSP cable should not exceed 15 cm.
- Make sure that there is no additional capacity present at the programming (data) pins. Consider if circuits on the programming pins could cause too large a load for programming.
- It is important that the crystal or the communication speed settings reflect the reality in devices that need that settings.
- Some devices may be programmed using several different methods. Programming problems may be caused by selecting a different programming interface or a different programming mode than the ones that the application is prepared for.

If none of the above tips help to eliminate the problem and if it absolutely sure that the device to be programmed is not faulty, the fault may lie in the programmer.

To easily find out whether the programmer is in order or not, you can use the testing utilities described in the following chapters.

## 10.2 FORTE Tester

The FORTE Tester utility easily tests the “health state” (usability) of the FORTE programmer right at the customer's site.

In order to run it successfully, only one FORTE

programmer may be connected to the control PC and its drivers must be correctly installed. This is confirmed by the green ON-LINE LED being on.

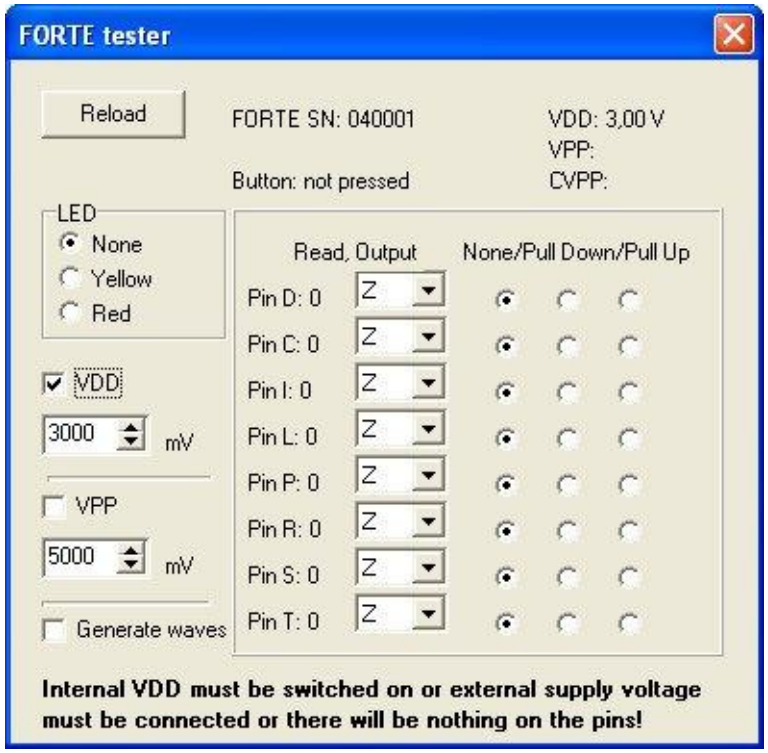


*Fig. 49: FORTE Tester*

After you start the tester, make sure that the programmer's output buffers are fed. You can either switch on the **VDD** and set its voltage to the required level or connect an external feeding voltage to the VDD pin. The level of detected voltage is displayed in the lower part of the window.

Basic communication with the programmer can easily be recognized by the **Active** diode being on. You can choose its color to be yellow or red.

Programmer's serial number is displayed in the top part of the window. This number is necessary for any communication with the service shop. You can also find this number on the bar code label on the programmer bottom.

Space for displaying possible error messages concerning overcurrent or overvoltage detected is provided in the window under the serial number.

The line below provides information on the GO button.

The programing voltage level on pin P can be verified by switching the **VPP** option on and setting the required voltage level. The detected voltage is displayed in the top right corner of the window. The level of the current detected on pin P is displayed immediately below the voltage. As the P output features a built-in potential divider of approximately 5 kΩ, a non-zero value of the current is displayed even with the output pin P disconnected caused by feeding this potential divider.

If outputs are in a state of high impedance (Z), information on input logical levels at these pins is displayed. If a pull-up resistor is connected to a selected pin, the system starts reading log. 1. If a pull-down resistor is connected, log. 0 is read.

Outputs can be linked individually to log. 1 or log. 0. Output signal quality can be verified by a voltmeter.

The **Generate waves** option sends a periodical signal of approximately 1 kHz simultaneously to all outputs.

Neither reading of inputs nor setting of outputs works if there is no voltage at pin VDD.

If everything seems to be in order, but the device still fails to get programmed, it is recommended to check the application connection according to instructions for connecting a microcontroller to the programmer.

If in doubt as to whether the programmer is in order or faulty, do not hesitate to contact our technical support.

# 11

# HPRAVR

HPRAVR is an optional accessory for PRESTO and FORTE programmers for programming AVR microcontrollers in applications with the ISP10PIN standard connector on the device's side and with the ICSPCAB8 cable on the programmer's side. The ISP10PIN connector is typically used on boards with microcontrollers of the AVR type such as STK500, for example.

## 11.1 Usage

Connect the HPRAVR adapter to the application's connector. Make sure that pin 1 of the HPRAVR adapter is connected to pin 1 of the ISP10PIN connector in the application (Pin 1 is marked with a red dot at HPRAVR while it may be marked in different ways on the connected application – see information in the corresponding manual).

Now interconnect the adapter and FORTE with an ICSP cable. Pin 2 in the cable is used as the key. The following picture illustrates a typical interconnection of the PRESTO programmer with an application through the HPRAVR adapter:

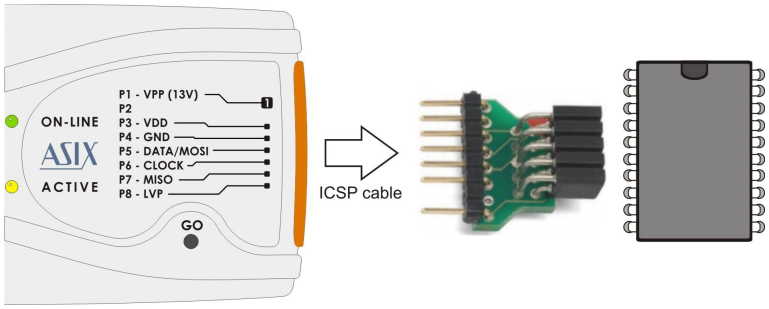


*Fig. 50: Use of HPRAVR*

**Important Warning**

Pin No. 1 of the HPRAVR adapter is marked with red color. Please double check the position of the pin No. 1 in the application as the application could be damaged if incorrectly connected.

| | | | |
|---|---|---|---|
| MOSI | 1 | 2 | VCC |
| NC | 3 | 4 | GND |
| RST | 5 | 6 | GND |
| SCK | 7 | 8 | GND |
| MISO | 9 | 10 | GND |

*Fig. 51: ISP10PIN, top view*

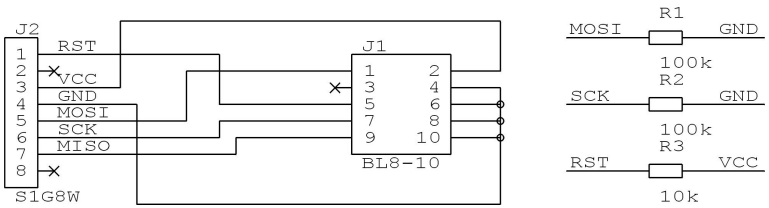


*Fig. 52: HPRAVR adapter diagram*

# 12

# Document history

| Document revision | Modifications made |
|---|---|
| 2013-07-03 | Newly created document |
| 2013-12-18 | Minor text corrections |
| 2014-02-28 | The NXP LPC2xxx family was added in the JTAG Interface chapter |
| | New command line parameter [/read] |
| 2014-04-25 | The STM8 family was added in the Connection Examples chapter |
| 2014-07-01 | The ARM SWD interface was added in the Connection Examples chapter |
| | CE certificate was replaced by Declaration of Conformity |
| 2014-09-23 | A note about reset of MCU ARM SWD was cleared. |
| | Described new option - Show only the lowest byte of word in ASCII |
| 2014-11-21 | Updated list of variables for up_dll.dll |
| 2014-11-25 | New settings of the UP program have been added. |
| 2015-02-03 | Added View/Console menu description. |
| | Added description of PIC32MZ_BootProg variable. |
| | Small text fixes. |
| 2015-03-12 | The name of item Delay for VDD switching on/off has been changed |
| | Added description of the checksum settings |
| 2015-04-07 | Updated description how to create SVF file for Lattice. |
| 2015-05-19 | A chapter about forte.dll was added. |
| | The C8051 and EFM8 C2 interface was added in the Connection Examples chapter. |
| | Definition of input voltage on pins was added. |
| | Description of w=32 parameter for Windows messages was added. |
| 2015-08-05 | Description of the programmer driver installation has been changed. |
| | Added new /getpartrev parameter. |
| | Added new commandline errorcodes. |
| 2015-09-21 | Added description of the setting of the warning when the loaded file is not aligned to a word size. |
| | Fixed links. |
| 2015-11-19 | Added description of setting of the T pin during and after programming. |
| 2016-01-22 | The serial numbers logging description has been updated. |
| 2016-03-08 | Added description of the Display programmer form function. |
| | Added description of the setting which disables the VPP before VCC warning. |
| 2016-05-20 | The PE description has been updated. |
| | Added description of the start and end address for the SPI Flash memories. |
| 2016-08-10 | Description of the HCSxxx connection has been added. |
| 2016-10-10 | Added description of the /sn commandline parameter and manual serial numbers description. |
| | Added the commandline error code 9 description. |
| 2016-12-15 | Description of usage of the programmers at Linux has been added. |
| 2017-02-02 | Added description of auto programming in the Mass production window. |
| | Added description of Keep manually modified data. |
| 2017-02-10 | The Linux driver instalation chapter has been completed. |
| 2017-03-13 | A link to lin_ftd2xx has been added. |
| 2017-03-23 | Added description of /q1 commandline parameter. |
| 2017-05-09 | Added description of "Load last project on start-up" option. |

| | |
|---|---|
| 2017-06-29 | Added chapter about MultiUP utility. |
| 2017-10-30 | Description of the AVR UPDI connection has been added. |
| | Added description of "When using Windows Messages disable other warnings" option. |
| | Added description of "Write checksum to log file" option. |
| 2017-11-22 | Added description of RL78 connection. |
| 2018-02-19 | HPRAVR chapter moved. |
| 2018-03-28 | Added description of "Device info" menu item. |
| 2018-05-17 | Added RX600 connection description. |
| | Added description of Renesas RX600 settings. |
| 2018-06-07 | Added AT89C51CC01UA connection decription. |
| 2018-07-30 | Added a note to Debug fuse of MCU PIC. |
| | Added a note to AVR UPDI files mapping. |
| 2018-08-10 | Added description where Linux FTDI driver can be found. |
| | Added a note to commandline parameters. |
| 2019-01-25 | Description of saving fuses to ini function has been added. |
| | Write RC osc Adjustment function description has been added. |
| 2019-01-31 | Added description of how to read the ProgressBar value in the quiet mode. |
| 2019-02-26 | Added a note for DS28E05. |
| | Added description of Read address function |
| | Added description of Windows Messages wParam of 24 and 25. |
| 2019-04-23 | Added description of /df parameter, description of /p and /o parameters has been updated. |
| 2019-06-26 | Updated description of /s commandline parameter. |
| | Added description of "Lock project" function. |
| | Added description of function setting actual SN from log file. |
| 2019-07-26 | Added HCS08 connection description. |
| 2019-09-20 | Company address has been changed. |
| 2019-10-24 | HCS08 note has been modified. |
| | Added chapter about up_control.dll |
| 2019-12-18 | Added AVR HVP connection description. |
| 2020-01-10 | Added Autoload next file function description. |
| 2020-03-31 | Added description of /conf parameter. |
| | Added list of parameters which are transferred to already running UP instance. |
| | Added description of the file locations only programming. |
| | Added description of the option switching off the Device ID check. |
| 2020-05-26 | Added description of "Load project unlocked" function. |
| | Added description of how to use project on the commandline. |
| 2020-08-18 | Modified AVR UPDI connection. |
| 2021-01-11 | Driver installation description has been updated. |
| | SDA internal pull-up disable option has been described. |
| | UP software installation description has been updated. |
| | JTAG Player -s parameter description has been updated. |
| 2021-02-25 | Fixed Windows Message w=48 description. |
| | Fixed Windows Messages table. |
| | Added a description of a Recently used devices list. |
| 2021-03-15 | Some typing errors have been fixed. |
| 2021-12-23 | Added a picture and a description of QUAD SPI interface. |
| 2022-01-07 | Added a note concerning 34AA04 memory. |
| | Rules settings were changed for Linux installation. |
| 2022-01-14 | Description of comments in serial numbers files has been extended. |
| 2022-05-23 | Added LPC UART connection description. |
| 2022-06-28 | Projects description at commandline chapter has been modified. |

| | |
|---|---|
| 2022-09-14 | Added connection description of AVR MCU with separated UPDI and RESET pins. |
| 2023-01-12 | Added an OSIS fuse description for Renesas ARM. |
| | The chapter about CE and RoHS has been modified. |
| | Added a note concerning Wine version. |
| 2023-03-13 | Added a note to showing the MD5 checksum. |
| | Added description of UP_GetChecksum and UP_GetChecksum_Wnd v up_dll.dll functions. |
| 2023-05-15 | Added CH32V003 connection description. |
| 2023-08-24 | CH32V003 Fast mode function has been described. |
| 2023-11-02 | Addded description of "Never ask and never save changes to data file" function. |
| 2023-11-06 | Added an information about ProgressBar read example. |
| 2023-12-06 | Added a note that Linux support was discontinued. |
| 2023-12-14 | Added SPD5118 connection description. |
| 2024-01-05 | Added Zilog Z8F connection description. |
| 2024-02-16 | Z8F connection picture has been fixed. |
| 2024-08-22 | Added a new return code for commandline when the device is protected. |
| 2025-02-20 | Added description of Add note option for project. |
| 2025-03-28 | Added table of LPCxxxx ISP signals. |
| | In the Connecting to Application chapter added a link to a chapter where connectors are described. |
| 2025-05-13 | LPC5411x added to the table of ISP signals. |