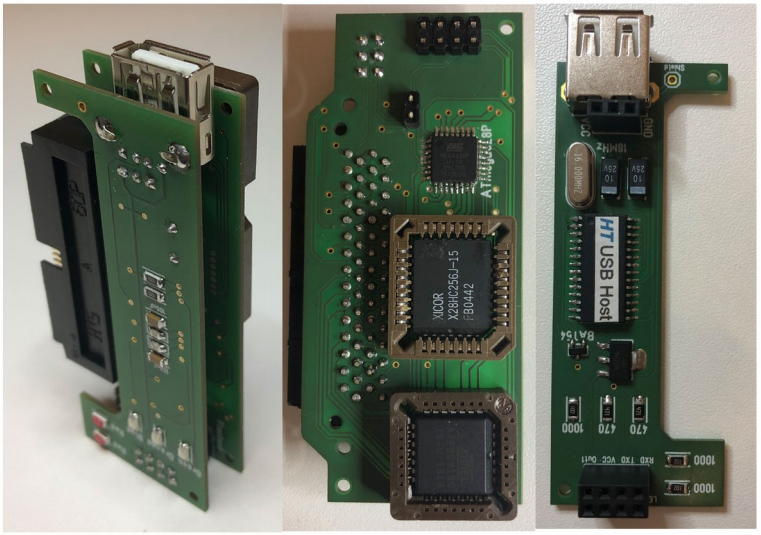
SHARP PC-1600
Modular Extension Platform - MEP
rev2 Manual



Backend + USB-Frontend | Backend | USB-Frontend

Modular Extension Platform - MEP

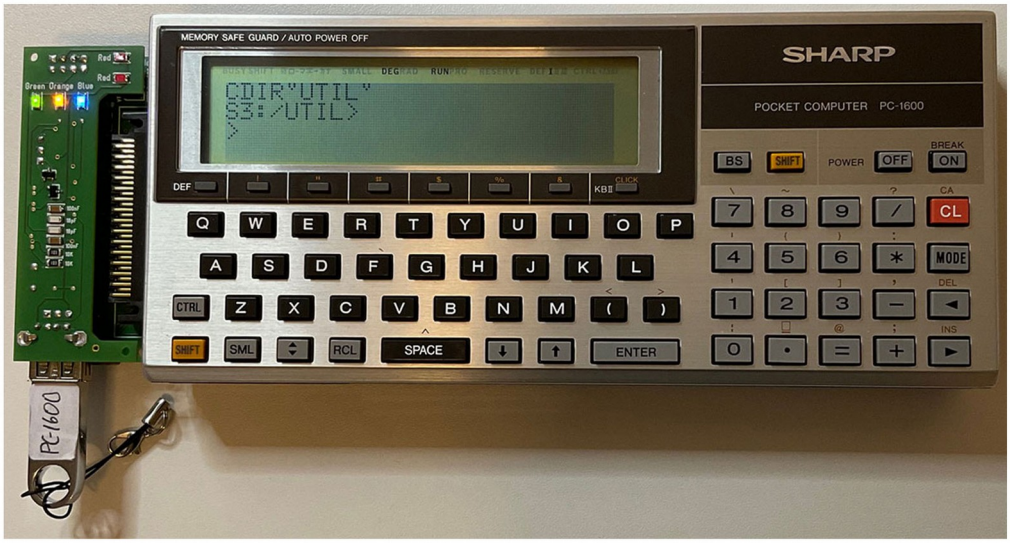
Fully programmable platform

Backend (Generic Bus Extension)
* ATmege328P microcontroller with PC-1600 port I/O support
* 16KB EEPROM for peripheral ROM extensions (e.g. additional BASIC tokens or assembler routines)
* Programmable Logic Device that contains all hardware adressing

Frontend (USB to UART)
* USB host controlller, e.g. for connecting an USB keyboard or an USB flash drive

Optional 3D-printed shell with brushed aluminium cover

MEP Application 1
USB-Stick Interface



MEP USB-Stick Driver Application

This MEP application is a USB flash drive interface that fully integrates with the PC-1600 operating system by supporting a native file device named "S3:" (or "Y:).

USB 2.0 and 3.0 flash drives (USB-sticks) formatted with FAT32 or exFAT are supported. The standard BASIC file commands are working with "S3: or "Y:" just like they do using "S1:" or "S2:": `FILES, [B]SAVE[,A], [B]LOAD, COPY, KILL, NAME, OPEN, CLOSE, PRINT#, INPUT#.`

The EEPROM of this MEP app provides a new BASIC command `CDIR` **to navigate through subdirectories on the flash drive in UNIX-style, e.g.** `CDIR"UTIL", CDIR"..", CDIR"../DEMO/PLOT".`

The file browser app DiskWorks (DW.BIN) is fully supported using drive letter "Y" (as long as no CE-1600F is connected).

This MEP app implements the standard IOCS-API for file devices so it can easily be accessed from within your own assembler programs.

Only one application at a time can be installed on the MEP.

MEP Application 2
USB-Keyboard Interface

MEP USB-Keyboard Application

This MEP-application is a USB keyboard interface supporting US, German and French standard keyboards

All PC-1600 keys are mapped, including function keys, cursor keys, SHIFT, CTRL, RCL, CL, INS, DEL, BS, MODE etc.

As a highlight the special characters of German and French keyboards (Umlaute, Accents etc.) are mapped to characters within the PC-1600 code table, so that you can write strings directly in German or French without CHR$(). Even keys that are meant to modify the next key are fully functional (see image).

The preconfigured EEPROM contains a BASIC-command “KBRD” to switch between country specific key mappings.

Only one application at a time can be installed on the MEP.

Table of Contents

Generic MEP Features

- Modular design with exchangeable frontend hardware. UART Rx/Tx communication between frontend and backend.
- Backend:
  - ATmega328P microcontroller with I/O-port communication (INP/OUT) to the PC-1600 with preloaded application code
  - 28C256 EEPROM with socket. Preloaded PC-1600 ROM extension
  - 22V10C PLD with socket. Preloaded addressing: ROM extension at #7,&4000, microcontroller port: &90
  - Onboard SPI socket for programming of the microcontroller
  - Compatible 60pin bus connector.
- USB-Frontend:
  - Hobbytronics USB-Host Controller with preloaded application
- Compatibility: **Only SHARP PC-1600**.

  PC-1500/A not supported – do not connect the module to a PC-1500/A !

Application Usage – USB-Stick Interface

In the following the terms USB-stick and flash drive are used synonymously.

When the MEP comes configured with this app, the module works as a USB flash drive interface that fully integrates with the PC-1600 operating system by providing a native file device named "S3:" or "Y:" respectively.

So this MEP app provides a seamless data and program exchange between a PC/MAC and the PC-1600 as well as a mass storage capability for the PC-1600.

This USB-stick interface app has been tested with different products with different USB standards (2.0, 3.0). However the USB-frontend relies on certain timings, so if you use very old or very slow USB-sticks, it may be possible that you experience read or write errors. USB 2.0 flash drives are recommended.

Before using this MEP app you need to format the flash drive with FAT32 (recommended) or exFAT. Depending on the capacity of the flash drive you need to partition it first, since these file systems do not support large memory capacities.

It is highly recommended to plug/unplug flash drives only when the PC-1600 and therefore the MEP is powered off. If the red LED is blinking after hot-plugging or unplugging the USB-stick, you definitely need to turn the computer off.
In the worst case the computer could crash or read or write errors may occur.
In that case you must perform a reset of the PC-1600.
If you avoid hot-plugging/unplugging of flash drives to/from the MEPs USB-port the system will always behave stable.

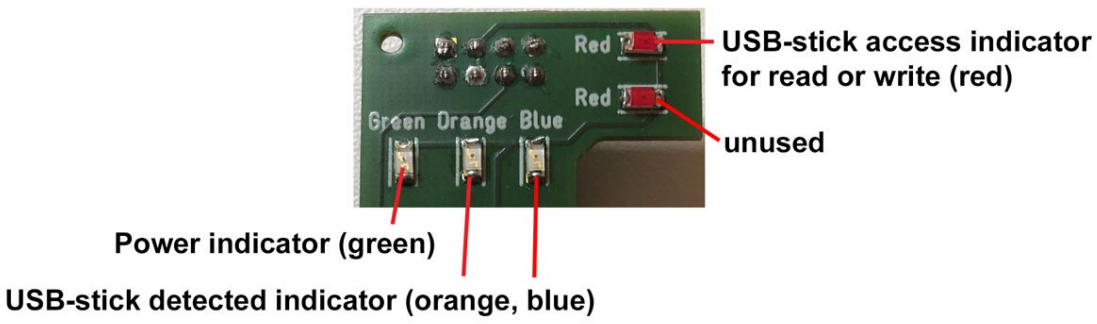
Assure that the SPI-programming jumper is attached to the backend (see Backend).

Assure that the switch at the frontend is in postion "SDO" (see USB-Frontend).

Carefully connect the assembled module (i.e. backend + USB frontend) to the PC-1600 60pin bus connector on the left side. Connect a formatted USB-stick and switch the computer on.

LED Indicators

Here is the meaning of the LEDs for the USB-stick interface app on the top of the USB-frontend:



Standard BASIC File Commands

You can now use the following standard BASIC file commands to access the connected flash drive. Instead of the device name S3 you can use Y as well:

- `FILES"S3:"`
  `FILES"S3:<search-pattern>"`
  Search patterns may include wildcards `*,?`
  Examples: `FILES"S3:"  FILES"S3:*.BAS"  FILES"S3:A*.???"`
- `[B]SAVE"S3:<filename>"[,A]`
  Examples: `SAVE"S3:TEST.BAS",A  BSAVE"S3:TEST.BIN",...`
- `[B]LOAD"S3:<filename>"`
  Examples: `LOAD"S3:TEST.BAS"  BLOAD"S3:TEST.BIN"`
- `COPY"<device>:<filename>"TO"<device>:<filename>"`
  S3 can be used as source- or target-device name or both
  Examples:
  ```
  COPY"S3:TEST.BAS"TO"S2:TEST.BAS"
  COPY"S2:TEST.BAS"TO"S3:TEST.BAS"
  COPY"S3:TEST.BAS"TO"S3:TEST1.BAS"
  ```
- `KILL"S3:<filename>"`
  Example: `KILL"S3:TEST.BAS"`
- `NAME"S3:<old-filename>"AS"S3:<new-filename>"`
  Example: `NAME"S3:TEST.TXT"AS"S3:TEST.BAS"`
- `OPEN"S3:<filename>" FOR [OUTPUT|INPUT] AS #<fileno>`
  `PRINT#<fileno>,<data>`
  `INPUT#<fileno>,<variables>`
  `CLOSE#<fileno>`
  Example:
  ```
  10 OPEN "S3:MYFILE1.TXT"FOR INPUT AS #1
  20 OPEN "S3:MYFILE2.TXT"FOR OUTPUT AS #2
  30 INPUT #1,I$:PRINT #2,I$
  40 CLOSE #1:CLOSE #2
  ```

The following BASIC file commands are not supported by this MEP app. If you use them with the device names "S3:" or "Y:" they yield an `ERROR 158`.

- `DSKF"S3:"`
  `SET"S3:<filename>",["P" | " "]`
  `OPEN"S3:<filename>" FOR APPEND AS #<fileno>`

Of course you can access the files on the USB-stick via a PC or MAC too (read & write).

A minor restriction of this MEP app is the fact, that only one file for read and one for write can be open simultanously. Setting the PC-1600 system variable `MAXFILES` to higher values than 2 has no effect on S3.

This MEP app is restricted to the 8.3 (FAT) file format like the PC-1600. Avoid longer file and directory names and usage of special characters or spaces.

You can operate the MEP flash drive interface app with the CE-1600P and CE-1600F, but then the device name "Y:" refers to the CE-1600F.

Directories – An Additional BASIC Command

When this app is installed on the MEP it provides an additional, non-standard BASIC command that gives access to (sub-)directories on the connected flash drive: `CDIR` (i.e. "change directory").

The syntax is:

CDIR"<path>"

There is no specification of a device since this command only operates on S3.

Like the `FILE` command the `CDIR` command outputs information to the LCD-display. In this case it's the prompt which tells the current selected (sub-)directory in UNIX-like notation.

Examples:
Here S3 is assumed to be the device name of the flash drive, assigned when formatted. Furthermore this example is a sequence of commands, starting in the root directory.

Command	Semantics	Prompt
`CDIR"."`	Show current dir (here: root)	`S3:/>`
`CDIR"UTIL"`	Relative path, one dir down	`S3:/UTIL>`
`CDIR"../GAMES"`	One dir up, one down	`S3:/GAMES>`
`CDIR"/DEV/ASM"`	Absolute path, two dirs down	`S3:/DEV/ASM>`
`CDIR".."`	One dir up	`S3:/DEV>`
`CDIR"/"`	Absolute path to root	`S3:/>`

The selected directory however acts like a context for the standard BASIC file commands (see above). So if you navigate to different directories on the flash drive, the `FILE` command will report the content of that directory only. This context concept holds for all standard BASIC commands and the IOCS file routine (see Assembler Programming - IOCS File API), since the PC-1600 operating system has no notion of directories. In consequence you cannot e.g. `LOAD` from a different directory than the currently selected one (e.g. `LOAD"S3:/UTIL/TEST.BAS"` is not possible).

This isolation of the directory concept from the PC-1600 OS is very important to maintain compatibility with existing PC-1600 programs and the OS itself. Consequently (sub-)directories are not 'seen' by the `FILE` command.

The directory structure itself has to be created on a modern computer, but the MEP flash drive app can navigate through that structure with the aid of the CDIR command.

So you can use a USB-stick as a structured mass storage for the PC-1600 !

Assembler Programming - IOCS File API

The MEP flash drive app provides a ROM extension for the PC-1600. This extension registers to the standard IOCS file routine of the PC-1600. In fact this is the only mandatory integration with the PC-1600 OS that has to be implemented by a PC-1600 peripheral file device. All standard BASIC file commands rely on that very same IOCS file routine, which is a very elegant and open design by the SHARP engineers of the 1980's.

By the way, this hooking to the standard IOCS file routine is the foundation of compatibility with existing PC-1600 file browser applications like DiskWorks (DW.BIN), which you can find here: https://www.sharp-pc-1600.de/Down_Maschine.html).

DiskWorks has no notion of a device called "S3:" but it can access "Y", which is the alternative name for the MEP file device :-)

The API of the standard IOCS file routine is a `CALL` to a specific ROM address:

```
FILE &01DE
```

Parameters: C-reg: function code, DE-reg: FileControl Block (FCB) pointer
Function codes:

```
&0F  OPEN FILE
&10  CLOSE FILE
&11  SEARCH FIRST
&12  SEARCH NEXT
&13  DELETE FILE
&14  SEQUENTIAL RD
&15  SEQUENTIAL WR
&16  CREATE FILE
&17  RENAME FILE
```

In order to access the MEP app through this API you need to set the 4-byte device name (FDVNO0..3) of the FCB to "`S3  `" or "`Y   `" respectively.

For further information about the IOCS file API and the structure of the FCB please refer to the PC-1600 Technical Reference chapters 3.3.1 and 3.3.2:
https://www.sharp-pc-1600.de/PDF/PC1600TechnicalReference.pdf

It is also possible to `CALL` the core of the `CDIR` command (see above), which is of course not part of the standard IOCS file routine:

```
CDIR #7,&4020
```

Parameters: DE-reg: path string, B-reg: size of path string
Returns: prompt string at `&FB10`, C-flag: success/error, BASIC error no in `&F89B`

The prompt string is limited to 26 characters (i.e. one LCD display line) and is terminated by `CR` (i.e. `&0D`).

Application Usage – USB-Keyboard Interface

When the MEP comes configured with this app, the module works as a USB keyboard interface for US, German and French standard keyboard layouts.

- Keyboard compatibility: Tested with Logitech standard keyboard layouts (US, German, French). Also tested with Microsoft wireless keyboard with USB-dongle.

Assure that the SPI-programming jumper is attached to the backend (see Backend).

Assure that the switch at the frontend is in postion "LED" (see USB-Frontend).

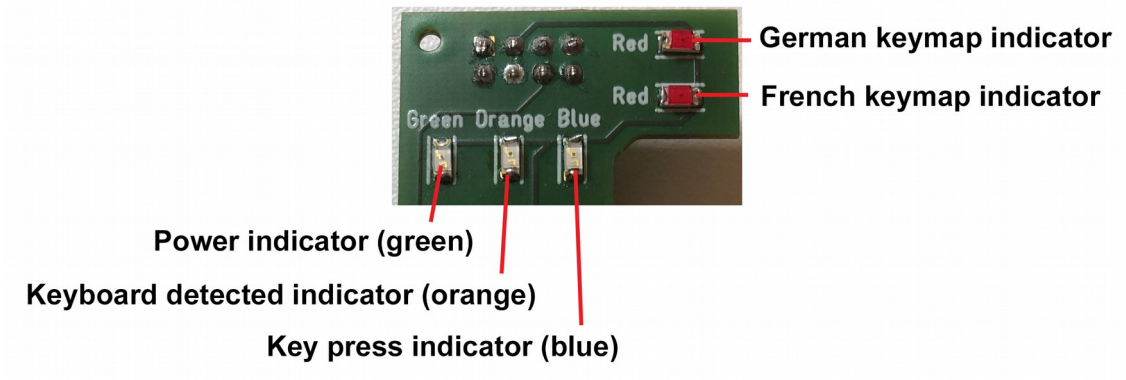
The key mapping can be switched by a BASIC-command. Additionally the default keymap can be selected at startup via a specific hardware setting.

Carefully connect the assembled module (i.e. backend + USB-frontend) to the PC-1600 60pin bus connector on the left side. Connect a USB-keyboard and switch the computer on.

You can now input characters on the native PC-1600 keyboard as well as on the connected USB-keyboard. Set the appropriate keymap for your keyboard (see Country Specific Keymaps).

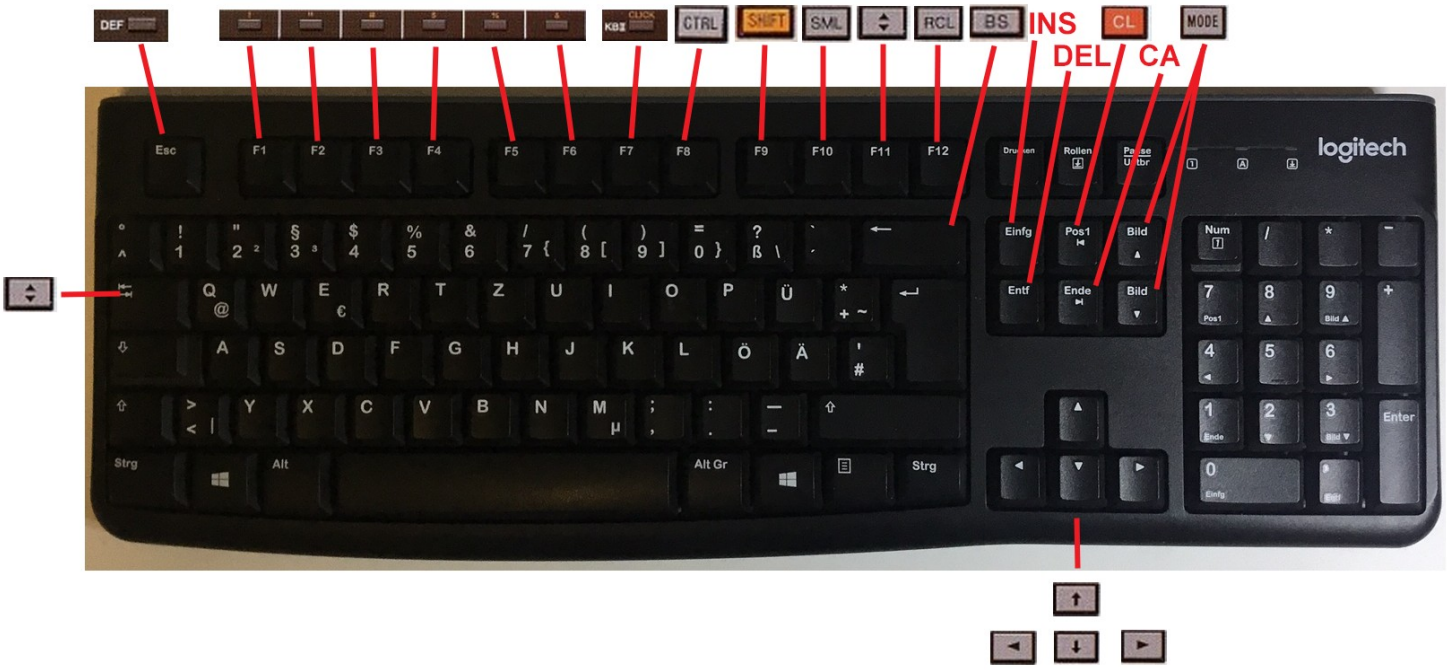
LED Indicators

Here is the meaning of the LEDs for the USB-keyboard interface app on the top of the USB-frontend:



Mapping of Special PC-1600 Keys

All country specific keyboard mappings share the same mapping for the PC-1600 special keys. This is as follows:



The special keys SHIFT, AltGr and CAPS-Lock on the USB-keyboard are not mapped directly to SHARP-keys but behave as usual, so they access the (country specific) respective SHIFT/AltGr-characters on the keyboard. Modifier keys, that are supposed to add an accent to the next character typed, are also working (as far as the PC-1600 code page supports the modified character).

Country Specific Keymaps

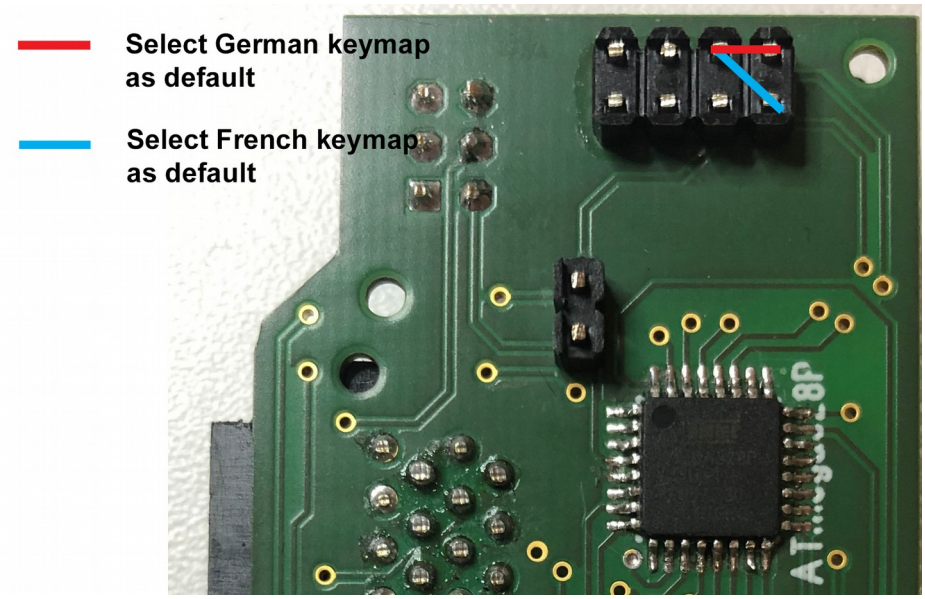
In order to change the country specific keymap, the modules ROM provides a respective BASIC-command. Type:

`KBRD"US"` to activate the keymap for a US keyboard layout

`KBRD"DE"` to activate the keymap for a German keyboard layout (QWERTZ)

`KBRD"FR"` to activate the keymap for a French keyboard layout (AZERTY)

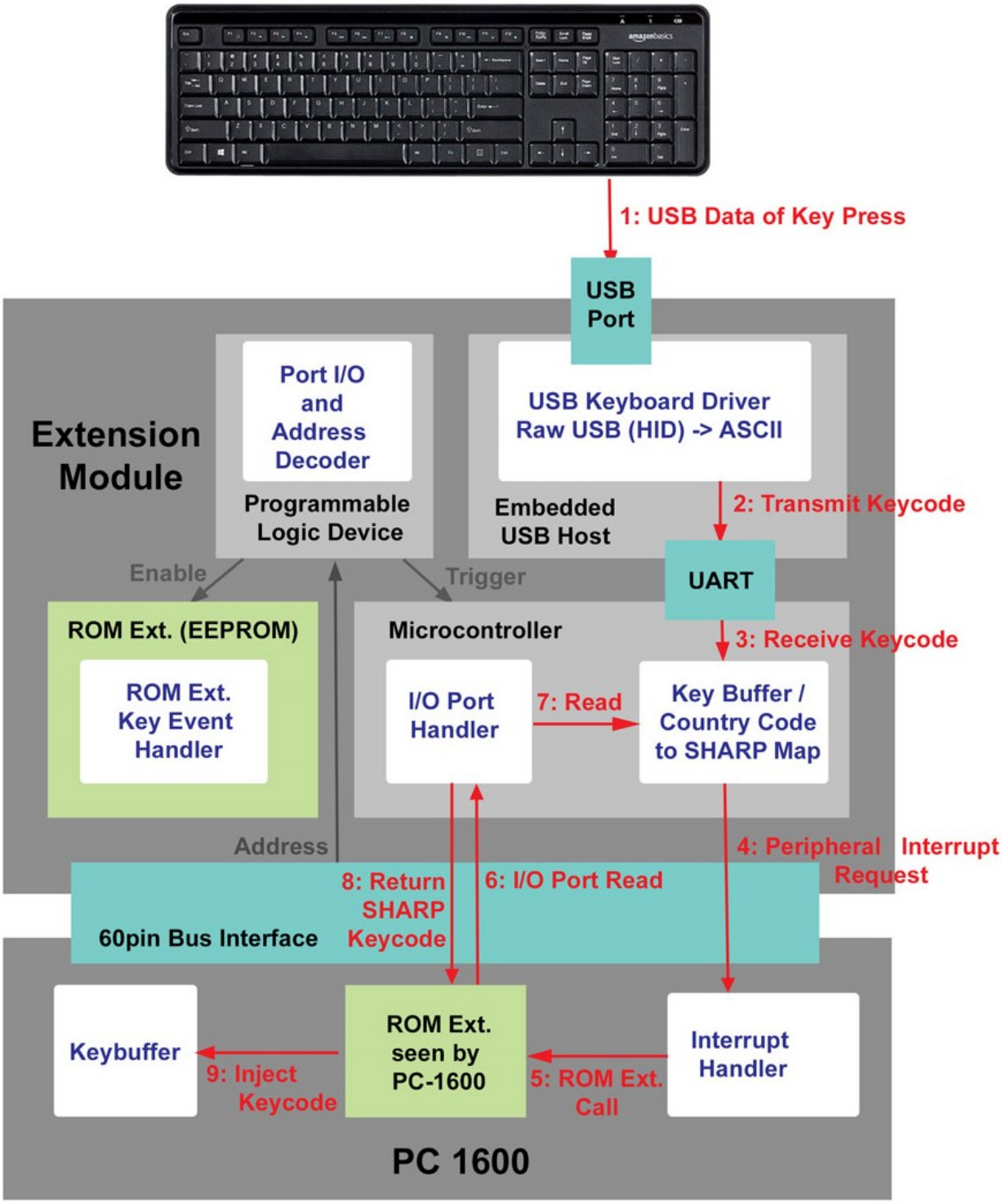
The default at startup is the US keymap. This can be changed via a hardware bridge, that you can add to the backend:



The "red" bridge (German keymap) pulls Inp0 to GND, while the "blue" bridge (French keymap) pulls Inp1 to GND. You can solder a fixed wire bridge to the backend or connect a DIP-switch, if you want a more flexible solution.
Be careful to not accidentally bridge other or multiple pins.

USB-Keyboard – Block Diagram & Message Flow

The following diagram gives on overview of the functional blocks and the message flow starting from a key press on the USB-keyboard up to displaying the respective character at the PC-1600 display:



USB-Keyboard - Known Issues

Due to the fact that the USB host (frontend) sends US-keycodes only and the country specific mapping to PC-1600 key codes has to be done in the backend, there are some key codes that are indistinguishable or conflicting for German and French keyboard layouts. This leads to some minor restrictions and workarounds.

US-keyboard layout:

- No issues.

German keyboard layout:

- ° mapped to ' (indistinguishable key codes)
- ä,ö,ü not sensitive for caps lock mode
- Numpad calculation keys (/,*,-,+) mismapped (don't use)

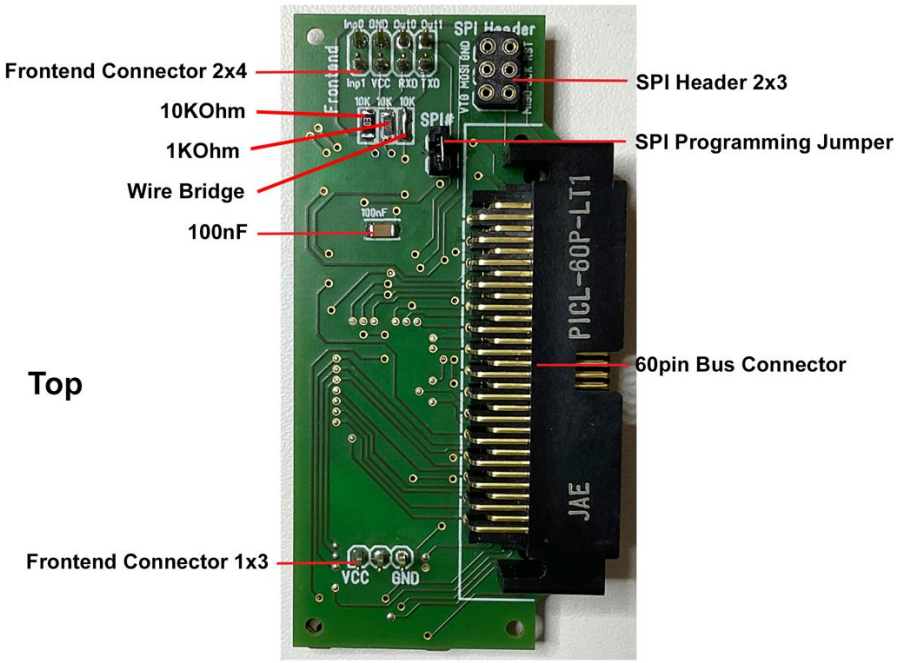
French keyboard layout:

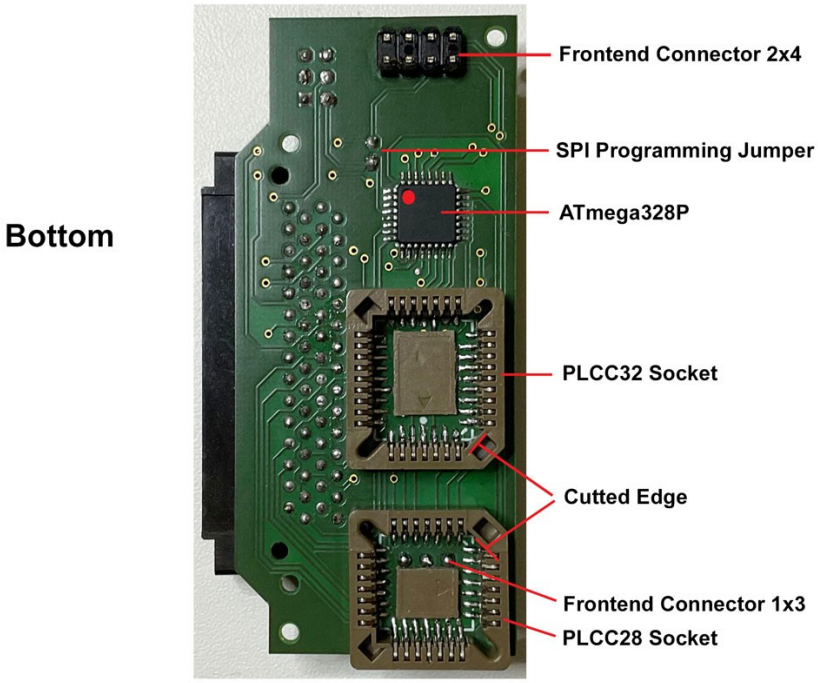
- * mapped to 3 (indistinguishable key codes). Workaround: AltrGr-$ mapped to *
- F3 mapped to } (indistinguishable key codes)
- Numpad mismapped (don't use)

MEP Assembly Instructions

The following sections just show the respective results – this does not imply a recommendation for the component soldering order.

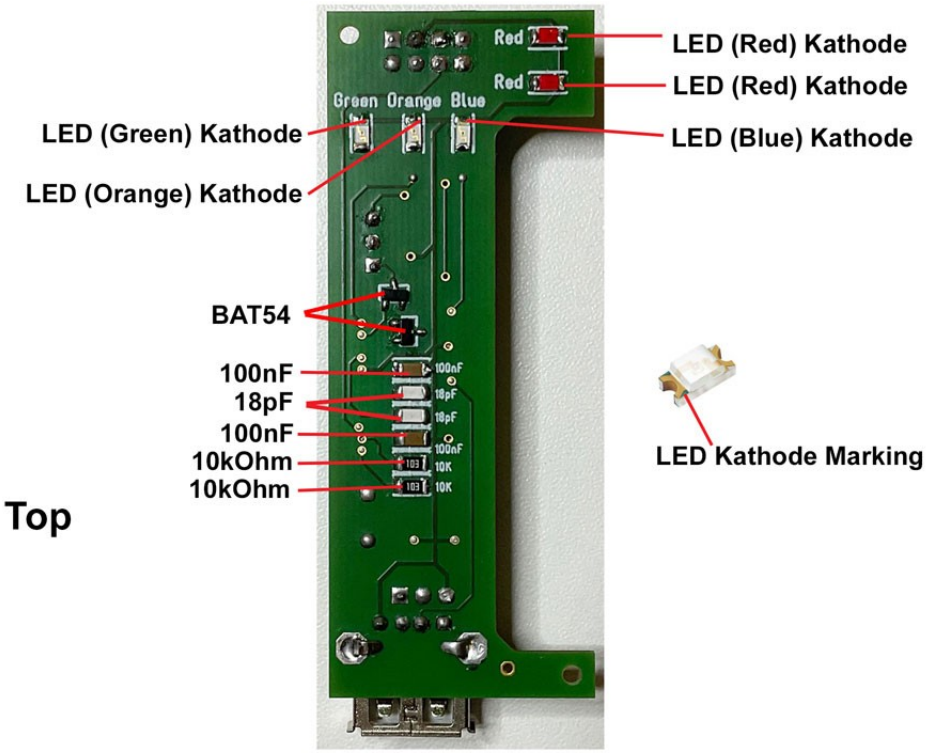
Backend

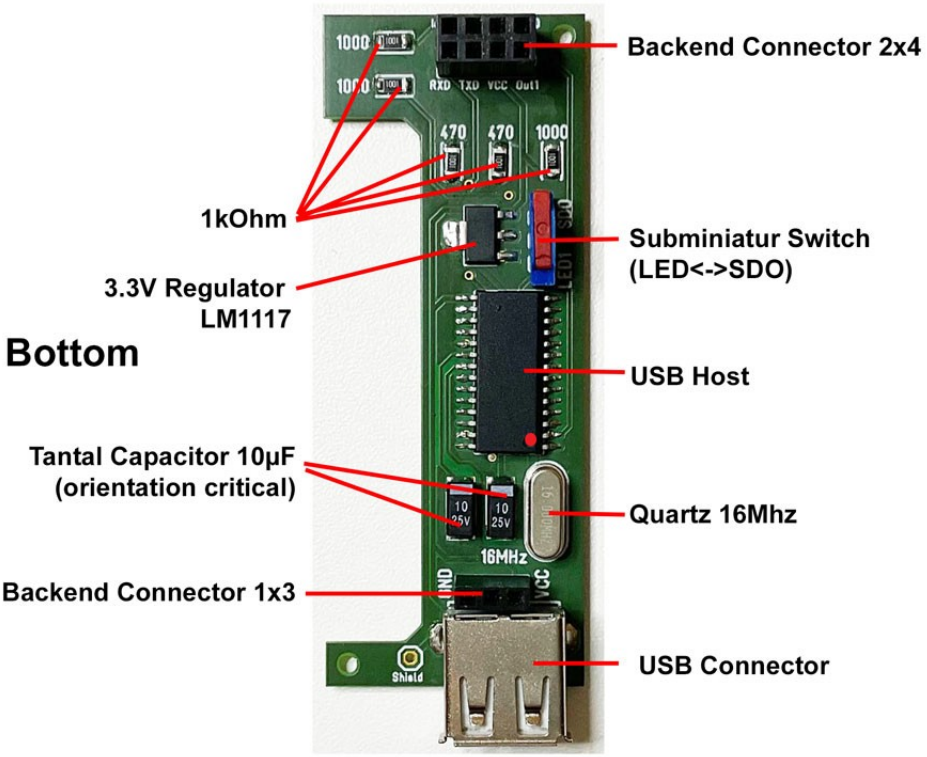




The PLCC sockets can either be soldered by soldering paste and hot air or by cutting off carefully the socket center plate and soldering the pins with a fine tip. In the latter case you can glue the center plate afterwards to the PCB like shown.

USB-Frontend

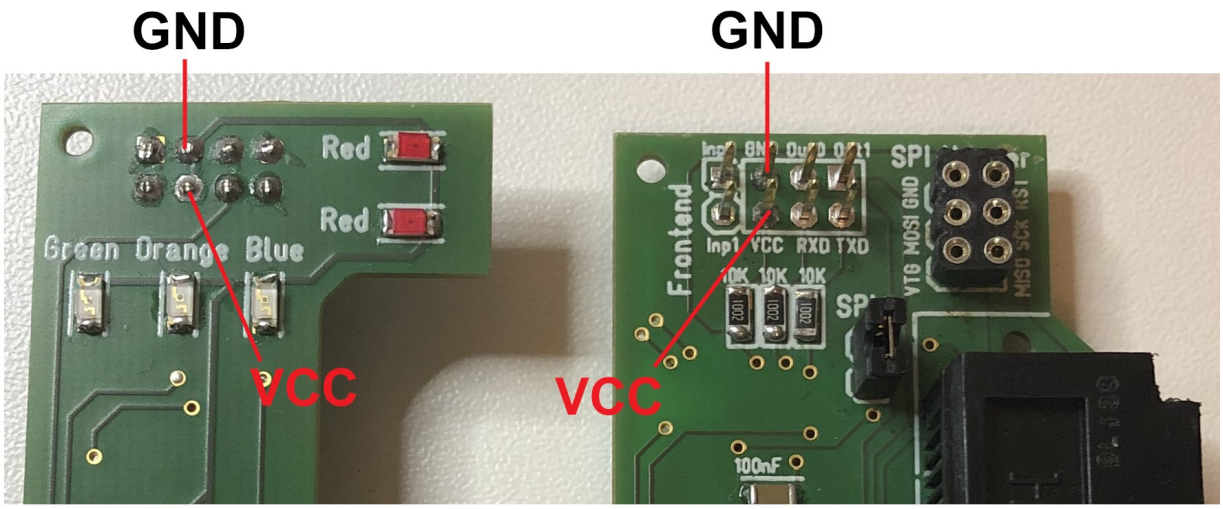




Tests

You can test the MEP backend and frontend separately.

The first and absolute mandatory test after assembly is to check for short circuits between VCC and GND in both the backend and the frontend:



Frontend **Backend**

USB-Frontend Test

If the keyboard subapp is installed on the frontend, you can test it by connecting an FTDI cable to your PC/MAC like shown in the section USB-Host Programming (Frontend). Start a terminal application (e.g. hterm or CoolTerm) and connect with the virtual COM-port of the FTDI cable.

In the terminal application set baudrate:9600, data:8, stop:1, parity:none.

Now connect an USB keyboard to the MEP frontend and type. You should see the respective key codes in the receive data window of the terminal app.

EEPROM and PLD Test (Backend)

A simple test for the PLD and the EEPROM would be to read the first two bytes of the ROM extension, which are constant for every PC-1600 ROM extension.

Connect the backend (only) to the PC-1600, switch the computer on and type:

```
HEX$(PEEK#(7,&4000))
```

This command must return 43

```
HEX$(PEEK#(7,&4001))
```

This command must return 16

Programming Guide

In this document the term *programming* refers to flashing pre-built, compiled software components to the MEP hardware. In contrast the term *development* refers to the process of creating such compiled software components, that form an application.

Applications and Subapplications

An application (app) for the MEP typically consists of two to four separate software components (subapps) that reside in different hardware components of the MEP, but can interact with each other. These subapps have to be programmed separately and by different means, which are described in the following sections.

Subapplication	MEP Hardware	Description
Peripheral ROM Extension	EEPROM (Backend)	Compiled Z80-assembler code. Provides a PC-1600 ROM extension header and optionally a peripheral interrupt handler, device I/O routines as well as a specific BASIC token table for the app. The ROM extension code is called by the PC-1600 OS and must follow its conventions.
Peripheral Data I/O	Microcontroller (Backend)	Compiled C code for the ATmega328P. Provides port I/O communication with the PC-1600 (INP, OUT commands). So this subapp is responsible for data exchange between the MEP and the PC-1600. On the other end it provides UART-communication with an arbitrary frontend via the Rx/Tx lines. In addition it can raise a peripheral interrupt request for the PC-1600.
USB Driver	USB Host (USB Frontend)	Compiled USB subapp from hobbytronics (e.g. keyboard driver, flash-driver, mouse-driver etc.) This 3rd party subapp communicates via UART with the Peripheral Data I/O component that resides in the MEP backend.
Adressing (mandatory)	PLD (Backend)	Compiled boolean adressing logic for Programmable Logic Devices (PLD). This subapp is mandatory and needs only to be changed, if a non-default addressing is required. The defaults are: Start address for the ROM extension: #7,&4000 Port address for data I/O: &90

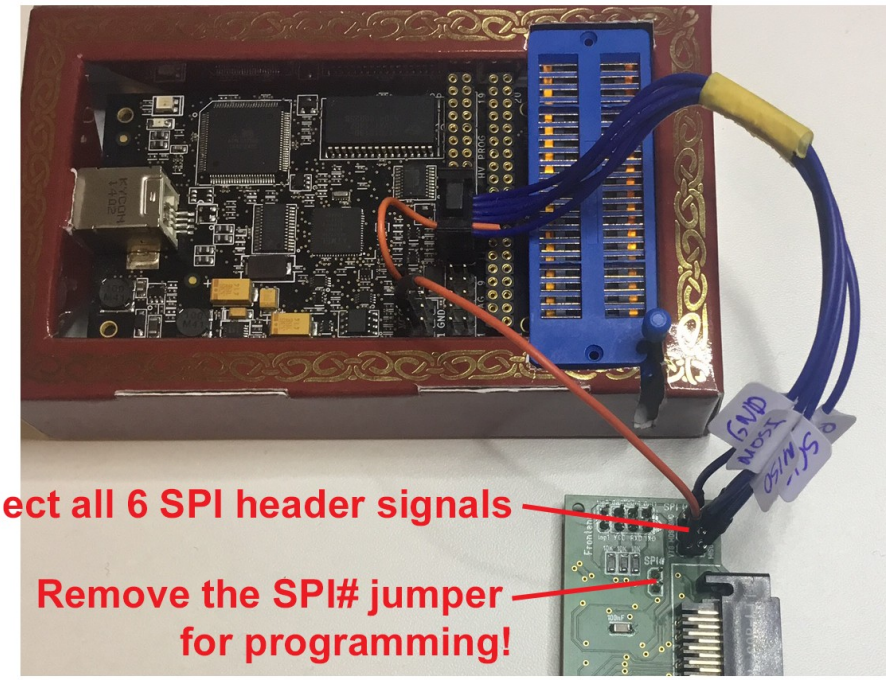
A typical interaction between a ROM extension and a data I/O subapp would be, that the ROM extensions reads and/or writes data from/to the I/O port and integrates with the PC-1600 OS. So this is control flow from the ROM extension to the data I/O subapp. The other way round would be a peripheral interrupt request, that can be raised from the data I/O subapp, which in turn leads to a call of a respective ROM extension routine, called by the PC-1600 OS. For an example see USB-Keyboard – Block Diagram & Message Flow.

Microcontroller Programming (Backend)

The onboard microcontroller is an ATmega328P. It can be programmed via the onboard SPI header and a respective development board. I recommend the AVR dragon togeher with AtmelStudio 7 (or above). The onboard ATmega328P does not come preinstalled with an Arduino bootloader.

The SPI header signal names are printed on the PCB. These are:

VTG, MISO, MOSI, SCK, GND, RST.



Don't forget to put the SPI# jumper back in place after programming!

EEPROM Programming (Backend)

The EEPROM is a 28(H)C256 (e.g. Atmel or Xicor). For programming you need a PLCC32 to DIP/DIL28 adaptor and a common EPROM burner that supports this chip family.

PLD Programming (Backend)

The PLD is either a GAL22V10C (Lattice) or an ATF22V10C (Atmel). In order to program the chip you need a PLCC28 to DIP/DIL24 adaptor and an burning device that is capable of flashing these types of ICs. Typically you wouldn't need a reprogramming of the PLD, unless you want to change the default addresses (#7,&4000 for the ROM extension and &90 as the I/O-port address of the ATmega328P as viewed from the PC-1600).

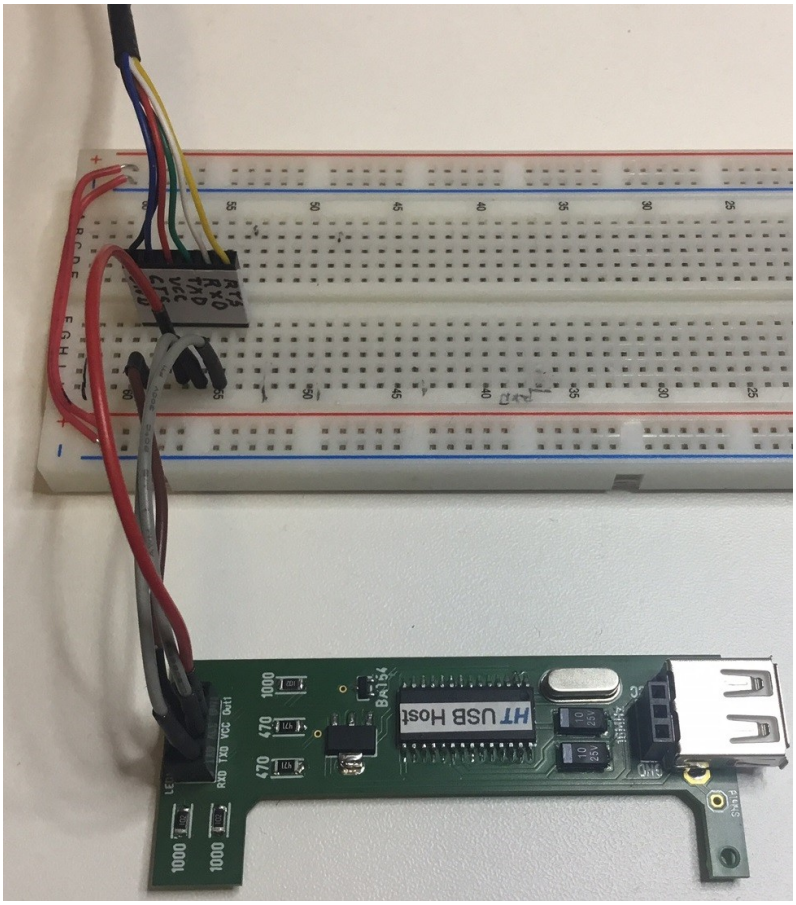
USB-Host Programming (Frontend)

The embedded USB-host IC is a PIC24FJ32GA002 microcontroller with a bootloader and application software from https://www.hobbytronics.co.uk/

In order to configure this IC or flash it with a different application provided by the manufacturer you need an FTDI USB to UART cable (5V), like shown.

Use the backend connector of the MEP frontend for connection. The frontend signal names are printed to the PCB:

FTDI Signal	MEP Frontend Signal
VCC (5V)	VCC
GND	GND
RXD	TXD
TXD	RXD



For the actual programming or configuration process, as well as available applications for the embedded USB-host, please consult the manufacturers website:

https://www.hobbytronics.co.uk/usb-host/usb-host-soic

Developers Guide

Please contact the author for specific infos or colaboration.

Copyright

Contact

www.silicium.org/forum -> PM to user spellbound