WAcouSense

Command Shell

Overview

The MCU FW provides a **command shell** (*command line interface*). It is used to control the MCU, to enable/disable features, to debug or to monitor the FW, to configure the system...

The command shell is available via:

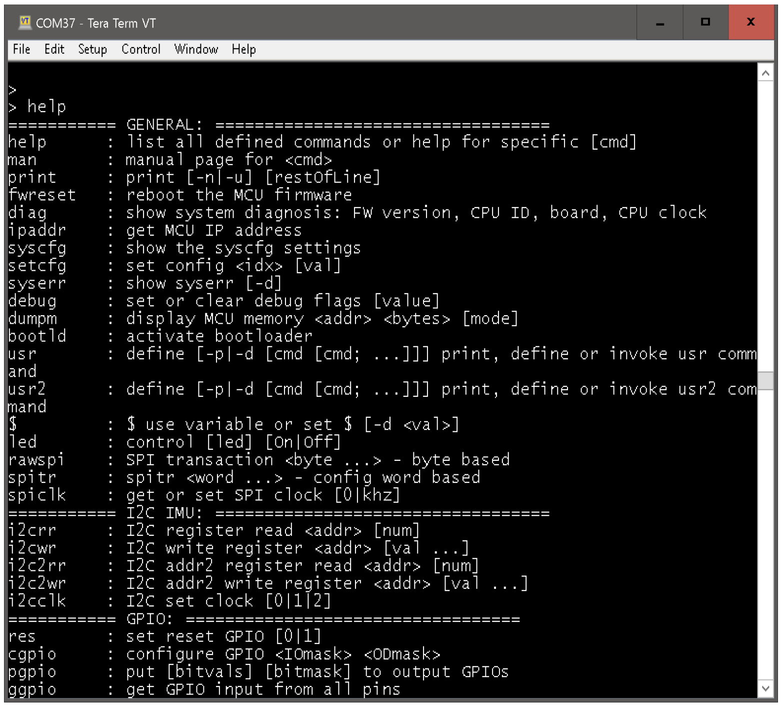
- **UART terminal** connection (e.g. using TeraTerm) – *main use*
- via **Web Browser** (MCU provides a web page to enter commands in Browser)
- **Python script** (acting like a TELNET session, *exclusive* to Web Browser!)

UART command shell

When USB-C cable for power of HW setup is provided – it provides also a USB VCP UART port. Check the “Device Manager” which COM port is USB VCP UART.

Connect via UART terminal program (on PC) to MCU:

baudrate: any
8 bit, no parity, no flow control



```
>
> help
=========== GENERAL: ==================================
help      : list all defined commands or help for specific [cmd]
man       : manual page for <cmd>
print     : print [-n|-u] [restOfLine]
fwreset   : reboot the MCU firmware
diag      : show system diagnosis: FW version, CPU ID, board, CPU clock
ipaddr    : get MCU IP address
syscfg    : show the syscfg settings
setcfg    : set config <idx> [val]
syserr    : show syserr [-d]
debug     : set or clear debug flags [value]
dumpm     : display MCU memory <addr> <bytes> [mode]
bootld    : activate bootloader
usr       : define [-p|-d [cmd [cmd; ...]]] print, define or invoke usr comm
and
usr2      : define [-p|-d [cmd [cmd; ...]]] print, define or invoke usr2 com
mand
$         : $ use variable or set $ [-d <val>]
led       : control [led] [On|Off]
rawspi    : SPI transaction <byte ...> - byte based
spitr     : spitr <word ...> - config word based
spiclk    : get or set SPI clock [0|khz]
=========== I2C IMU: ==================================
i2crr     : I2C register read <addr> [num]
i2cwr     : I2C write register <addr> [val ...]
i2c2rr    : I2C addr2 register read <addr> [num]
i2c2wr    : I2C addr2 write register <addr> [val ...]
i2cclk    : I2C set clock [0|1|2]
=========== GPIO: ==================================
res       : set reset GPIO [0|1]
cgpio     : configure GPIO <IOmask> <ODmask>
pgpio     : put [bitvals] [bitmask] to output GPIOs
ggpio     : get GPIO input from all pins
```

Figure 1: UART command shell with command **help**

Syntax

The **help** command displays all available shell commands.

<par> a mandatory parameter to provide: a value without the < and >,

	e.g.: **res 1**
[par] or [0|1]	an *optional* parameter (can be omitted): if a parameter is omitted (always at the tail) it is taken as value 0, e.g.: **debug 1** **debug** #using 0 as parameter a value, e.g. 1, must be specified on command (0 can be omitted only)
[-o]	an *optional* option (minus plus one charater): the option must be always the first given after command keyword, e.g.: **tofc -i**

Multiple commands on single line

Several commands can be entered on a singe line: separate each (complete) command with a semicolon ; , e.g.:

print display FW version; diag; syscfg

Comments on command line

When a command line contains a comment sign, the # , the rest of line is ignored (as a comment), e.g.:

print hello #this is comment

Commands

Commands are grouped into different categories (and separated on **help**):

- general commands, independent of a particular HW connected, e.g.:
  help, **diag**, **syscfg**, **print**, **led**, …
  just related to the MCU FW and the MCU board itself (not to a particular HW)
- interface specific commands, to use I2C, SPI interfaces, e.g.:
  i2crr, res, **pgpio**, **rawspi**, **spitr**, …
- SD Card related commands, e.g.:
  sdinit, **sddir**, …
- auxiliary commands, e.g.:
  msdelay, **picoc**
- expert commands (not intended to be used in general), e.g.:
  test, **syshd**, **pmicw**, …
- and HW specific, e.g. sensor commands, e.g.:
  tofc, **mic**, …

Helpful general commands

help [cmd]	display entire list of commands or just single line help for particular **cmd**
diag	display FW version, board info, …
syscfg	display system configuration, e.g. network config, I2C slave addresses (for sensors)
setcfg <idx> [val]	set a system configuration value at idx with val
ipaddr	display MCU IP address (got via DHCP or as STATIC fall back)
udpip <PCIPAddr>	set the host destination IP address: needed for streaming of any sensor data and audio to a host PC (destination must be know)
man <cmd>	display man page for a particular command
sdexec <script>	execute a script file on SD card with shell commands
usr, usr2	with option **-d**: define a user command line, without option: execute the stored **usr** command
print <rest_of_cmd>	print a message with the rest of command (until **;**), useful when executing scripts from SD card file

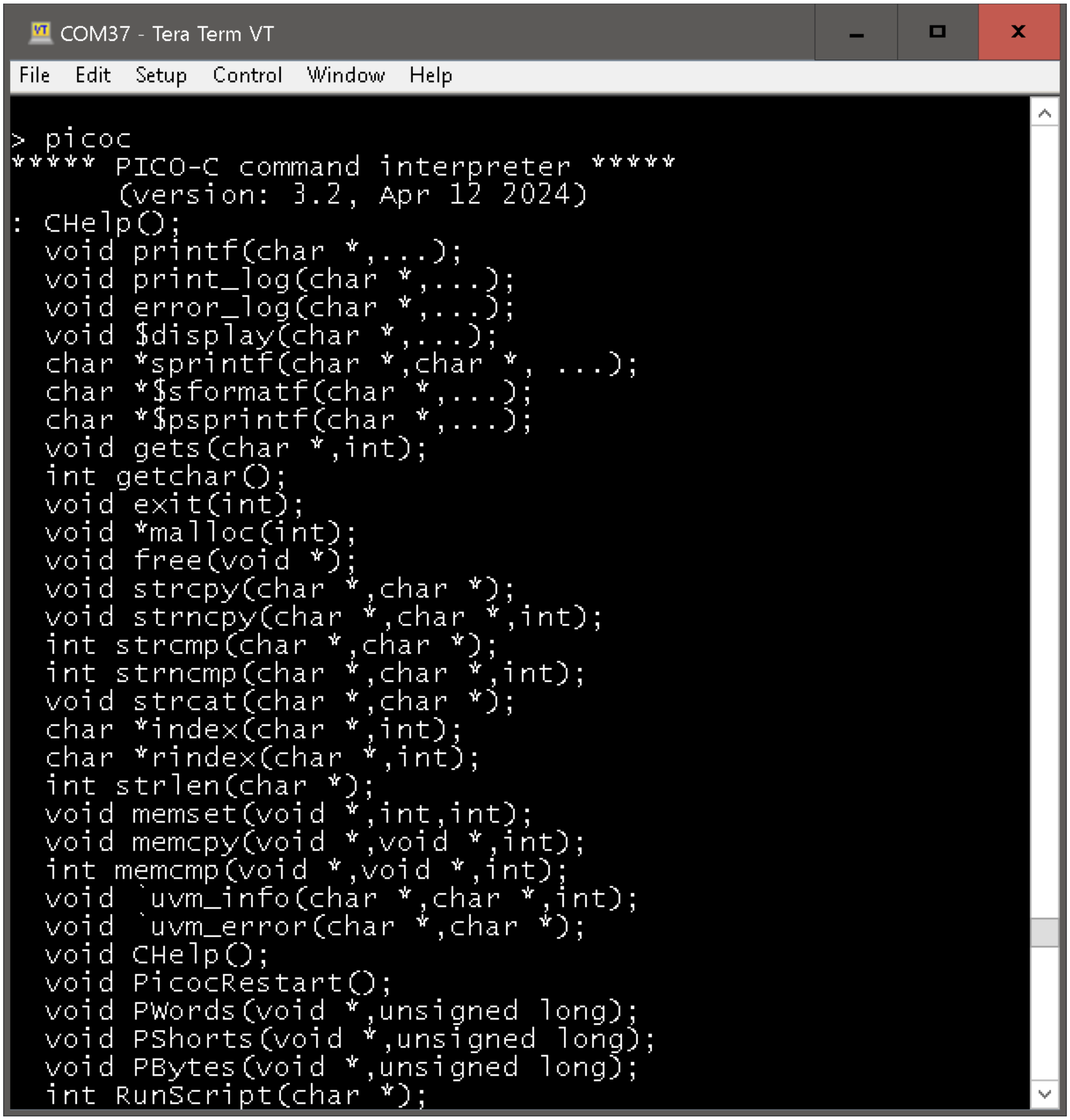
Sensor commands

mic <db>	enable MIC streaming via **VBAN**, **udpip <PCIPAddr>** needed before, **db** is amplification factor: **0** disables MIC streaming (*mute*)
tofc [-i]	without option: enable thread to stream TOF sensor data via network, **udpip <PCIPAddr>** needed before (just once), **-i** displays on UART only (no network streaming)

Pico-C

Entering the command **picoc** changes to take any command line input as a **C-code** statement (with semicolon ; to finish C-code). See the different prompt used as colon **:** .

Pico-C works **only** on UART command shell (not the other options to send commands, e.g. Python network script or Web Browser). If launched, e.g. in Web Browser or via Python command script – all is now on UART shell command.



```
> picoc
***** PICO-C command interpreter *****
      (version: 3.2, Apr 12 2024)
: CHelp();
  void printf(char *,...);
  void print_log(char *,...);
  void error_log(char *,...);
  void $display(char *,...);
  char *sprintf(char *,char *, ...);
  char *$sformatf(char *,...);
  char *$psprintf(char *,...);
  void gets(char *,int);
  int getchar();
  void exit(int);
  void *malloc(int);
  void free(void *);
  void strcpy(char *,char *);
  void strncpy(char *,char *,int);
  int strcmp(char *,char *);
  int strncmp(char *,char *,int);
  void strcat(char *,char *);
  char *index(char *,int);
  char *rindex(char *,int);
  int strlen(char *);
  void memset(void *,int,int);
  void memcpy(void *,void *,int);
  int memcmp(void *,void *,int);
  void `uvm_info(char *,char *,int);
  void `uvm_error(char *,char *);
  void CHelp();
  void PicocRestart();
  void PWords(void *,unsigned long);
  void PShorts(void *,unsigned long);
  void PBytes(void *,unsigned long);
  int RunScript(char *);
```

For more details how to use Pico-C – see the separate documentation.

SD Card

The SD Card is only used for **command shell** scripts or for **Pico-C** scripts.

It is not (yet) used to store binary files, e.g. sensor data (it can be added/extended).

The SD Card has to be formatted with a FAT file system (FAT16 or **FAT32**, on a PC first).

Before using the SD Card, in order to execute command shell scripts or Pico-C scripts, it has to be enabled:

```
sdinit 1                    #sdinit 0 will release SD Card
```

Afterwards, all the SD Card commands become available, e.g.:

```
sddir
sdexec <filename>           #command shell script
RunScript(char *);          //Pico-C script execution
```

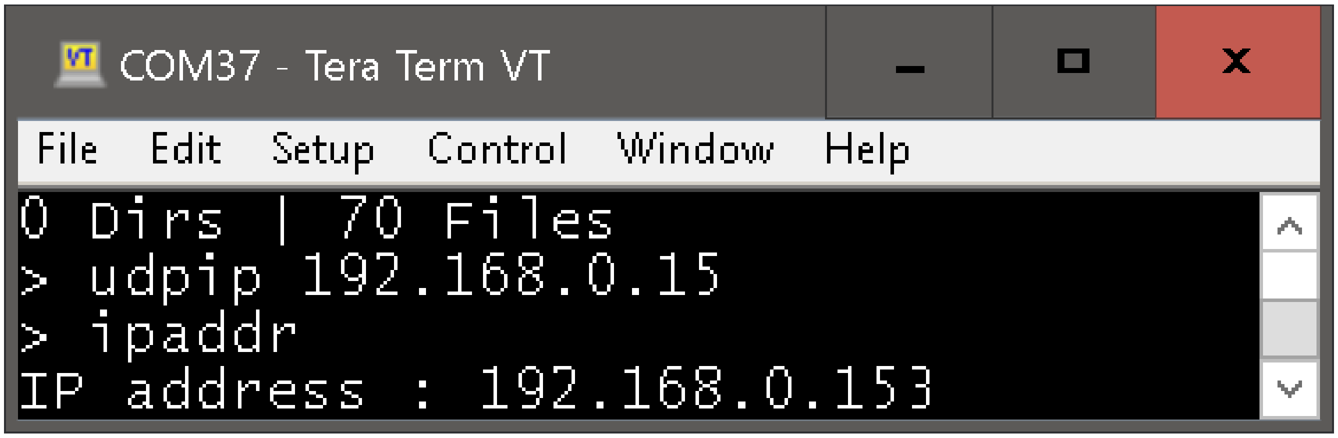
Transfer files to/from SD Card

It is not necessary to take the SD Card out just to copy/transfer files from/to it on a host PC.

There is a **TFTP** daemon running, so that files can be transferred to/from SD Card using the (existing) network connection (without to remove the SD Card).

Enable SD card and network:

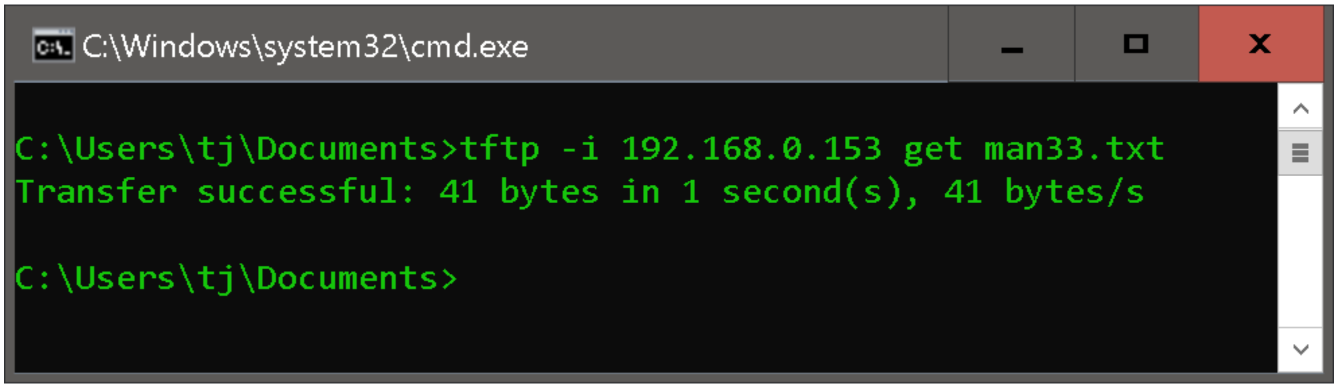
```
sdinit 1
udpip 192.168.0.15        #set the PC IP address as destination
```



```
0 Dirs | 70 Files
> udpip 192.168.0.15
> ipaddr
IP address : 192.168.0.153
```

Start on PC command line **TFTP** and transfer a file (here: copy from MCU SD Card to host PC):

```
tftp -i 192.168.0.153 get <filename>        #MCU IP address and filename on SD Card
```



```
C:\Users\tj\Documents>tftp -i 192.168.0.153 get man33.txt
Transfer successful: 41 bytes in 1 second(s), 41 bytes/s

C:\Users\tj\Documents>
```