

WAcouSense

Command Line and **Pico-C** usage

Overview

This document describes how to use the UART command line and the “**Pico-C**” C-code interpreter. It is based on “how to use the IMU sensor” (as an HW extension) on the Portenta-H7 MCU with Breakout Board and running the “*WAcouSense*” FW.

Command Line interface

The MCU FW provides an USB VCP UART via USB-C: the host should see a COM port and being able to open an UART terminal (e.g. TeraTerm):

<https://sourceforge.net/projects/tera-term/>

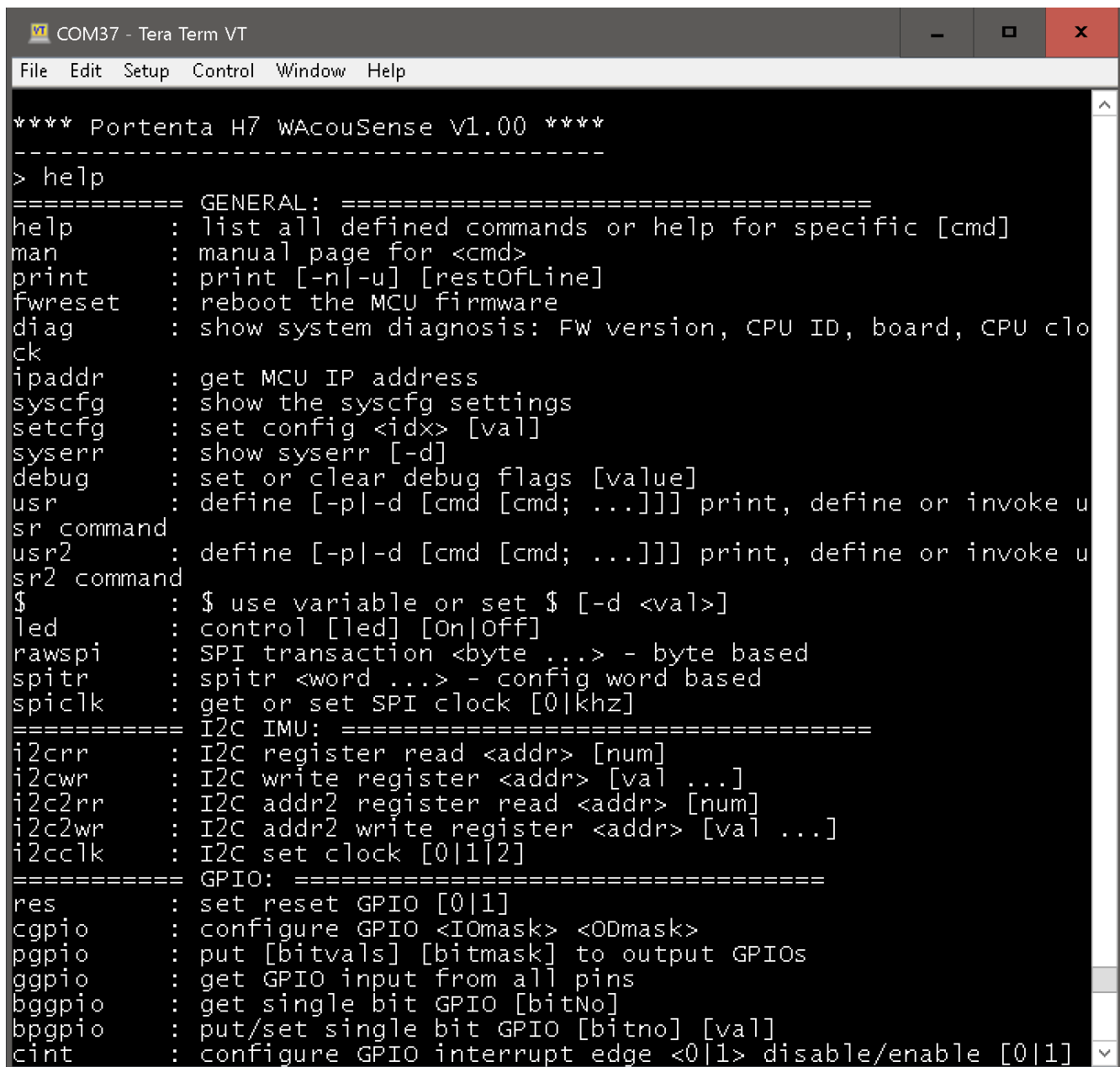
Find the COM port associated with the USB VCP provided by the MCU (use “Device Manager” in order to see all COM ports).

Open this COM port, with any baud rate and you should get a command prompt.

Enter command

help

in order to see all available commands:



```
COM37 - Tera Term VT
File Edit Setup Control Window Help

**** Portenta H7 WAcouSense v1.00 ****
-----
> 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]
usr       : define [-p|-d [cmd [cmd; ...]]] print, define or invoke user command
usr2      : define [-p|-d [cmd [cmd; ...]]] print, define or invoke user command
$         : $ 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
bggpio    : get single bit GPIO [bitNo]
bpgpio    : put/set single bit GPIO [bitno] [val]
cint      : configure GPIO interrupt edge <0|1> disable/enable [0|1]
```

Figure 1: UART command line and command **help** entered

Option:

When you have connected the MCU with PC via an Ethernet Cable (RJ-45, see separate “network setup guidelines”) - you can also use a Web Browser to access the command line interpreter.

It works in a similar way, instead of using a UART terminal tool, to access MCU and fire commands, but from any Web Browser and enter commands in command line:

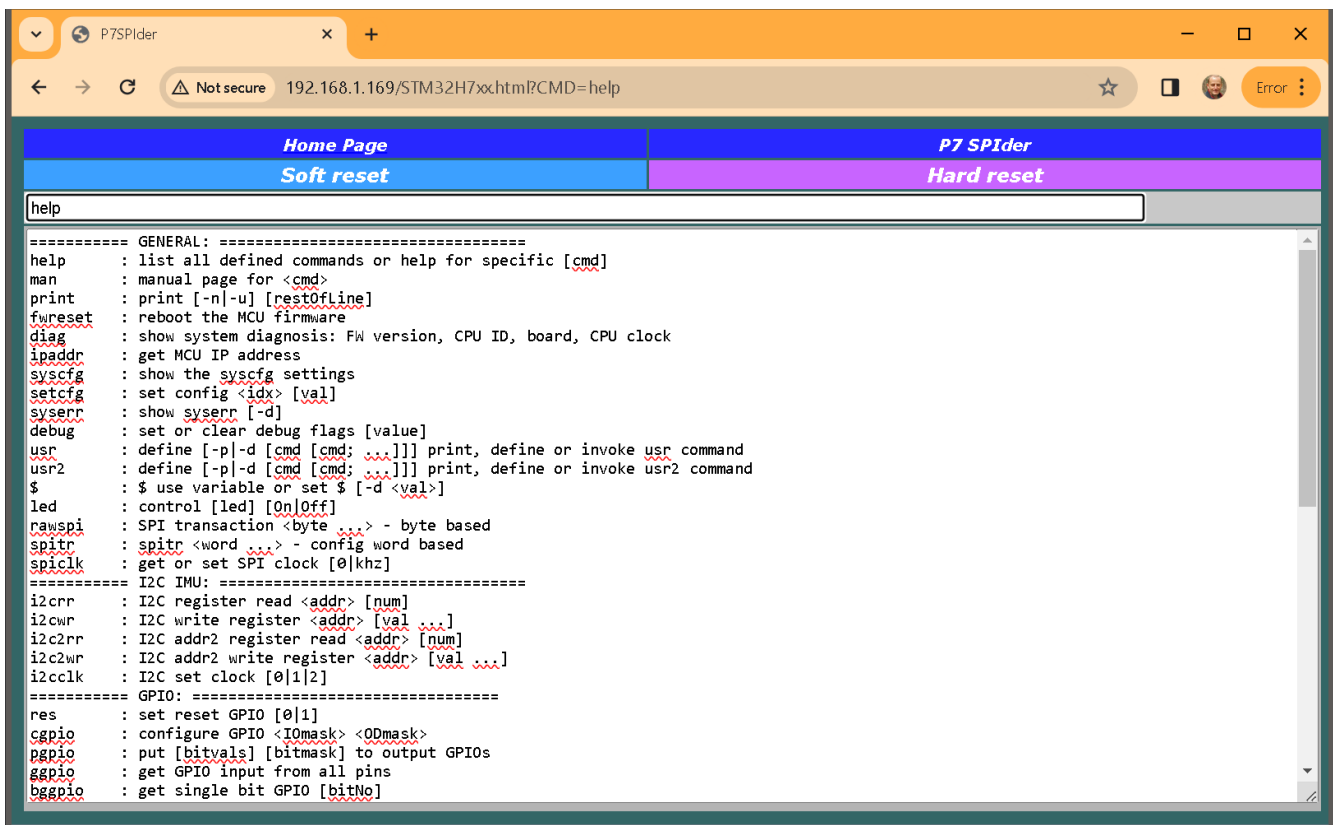


Figure 2: access MCU via Web Browser to enter commands

The commands provided also a way to write and read via I2C external HW connected, here the IMU sensor as: LSM9DS1

You might need the datasheet for this chip, e.g. to know how to enable, how to read the ChipID... which registers are provided by this external chip:

<https://www.st.com/resource/en/datasheet/lsm9ds1.pdf>

Except the I2C communication with the IMU chip – nothing else is (currently) provided by the MCU FW to do anything with the IMU sensor.

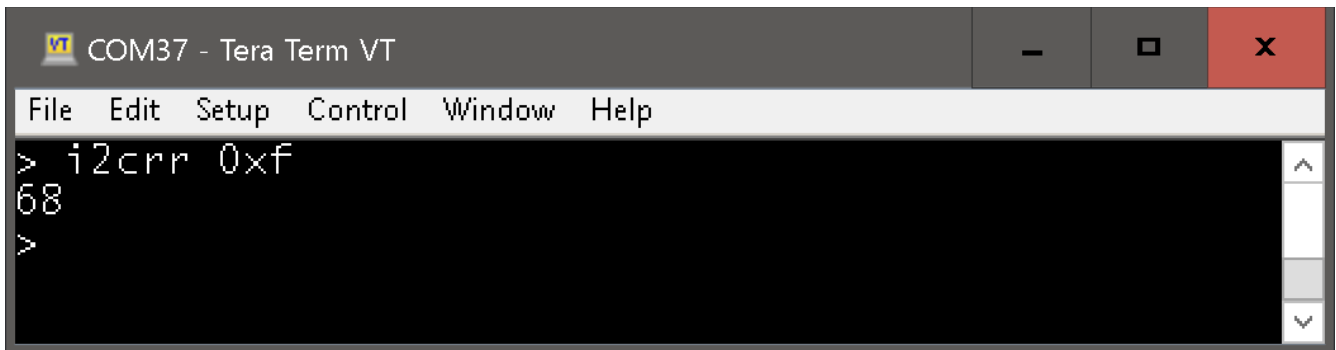
Read the ChipIDs

In order to check if the IMU is properly connected and can be accessed via I2C (e.g. to configure, read sensor data), check first if the ChipIDs can be read.

The IMU sensor used is a 9-DOF, with accelerometer and gyro (A/G) and a magnetic compass (M). Both units use a different I2C slave address, therefore there are two different sets of commands (see later below in example commands, e.g. **i2crr** vs. **i2c2rr**).

Read the A/C ChipID

i2crr 0xf

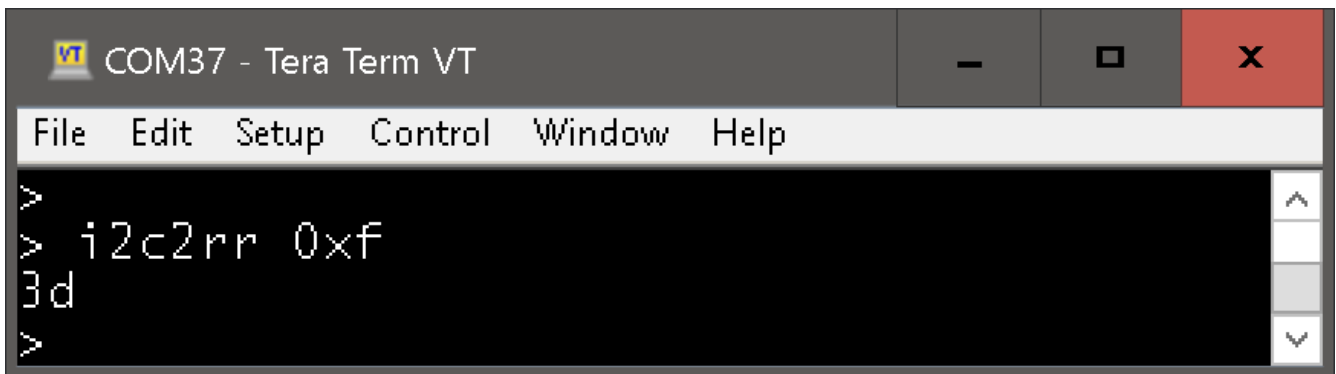


```
COM37 - Tera Term VT
File Edit Setup Control Window Help
> i2crr 0xf
68
>
```

Figure 3: read ChipID of A/G sensor: response should be 0x**68**

Read the M ChipID

```
i2c2rr 0xf
```



```
COM37 - Tera Term VT
File Edit Setup Control Window Help
>
> i2c2rr 0xf
3d
>
```

Figure 4: read ChipID of M sensor: response should be 0x**3d**

Read the temperature in A/G sensor

In order to do anything with the sensors – they have to be enabled. Default is: they are in deep sleep. So, before reading any sensor data – the sensor has to be enabled.

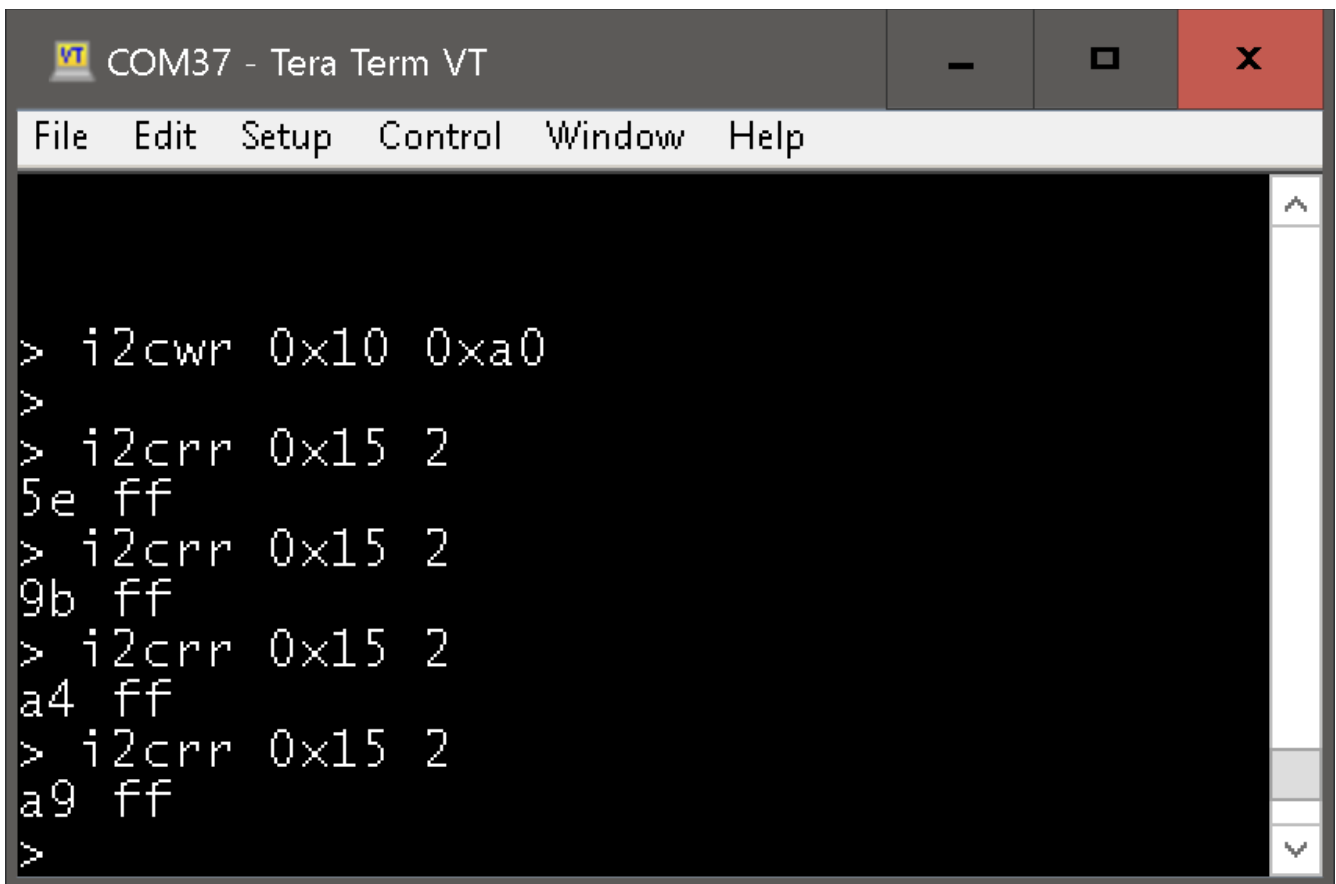
We use as demonstration here: “how to read the temperature sensor” in A/G part of the chip.

Enable the IMU chip in order to start measurements:

```
i2cwr 0x10 0xA0
```

In order to read the temperature value, afterwards:

```
i2crr 0x15 2
```



```
> i2cwr 0x10 0xa0
>
> i2crr 0x15 2
5e ff
> i2crr 0x15 2
9b ff
> i2crr 0x15 2
a4 ff
> i2crr 0x15 2
a9 ff
>
```

Figure 5: enable A/G and read temperature sensor

You get a value back, e.g. 0xA9 0xFF

This is a two-complements value, as little endian, so that the value is 0xFFA9.

The temperature is based on a 25C reference value (which would give 0x0000). So, the 0xFFA9 is a negative value as:

$$0x10000 - 0xFFA9 = 0x57 \text{ (as difference)}$$

Divide this value by /16 = 3 (as 3C below 25C temperature).

Subtract from 25C this 3 = 22C. This is the (internal) temperature read from the A/G chip.

Now you can configure the A/G sensors, read their values, etc.

Pico-C

The MCU FW contains a C-code interpreter (**Pico-C**): you can enter C-code like command lines, e.g. launch functions calls. You can also write Pico-C scripts, e.g. stored on SD Card.

For this Pico-C C-code – you do not need any compiler, the FW does not need to be updated. It works like a **script engine**, for C-code language used. Pico-C runs internally as part of the MCU FW.

In order to do the same IMU configuration with a Pico-C scripts, see below the example.

Pico-C Limitations

Pico-C is not a full-blown C-code interpreter. There are some restrictions and limitations:

- `#include "file.h"` - does not work on command line:
use `RunScript("file.h");`
- `typedef` is not implemented
- an *array of struct* is not possible to define and use, but regular “simple type” arrays are supported
- there is a pre-processor, e.g. to define macros, use `#ifdef` etc. But it is not full blown, e.g. not able to resolve macros in a recursive way or to use string commands and substitutions (via `##`).

Start the Pico-C

Enter on command line:

`picoc`



```
> picoc
***** PICO-C command interpreter *****
      (version: 3.2, Feb 24 2024)
:
: CHelp();
ERROR: :1: 'CHelp' is undefined
: CHelp();
void printf(char *,...);
void print_log(char *,...);
void error_log(char *,...);
void $display(char *,...);
char *sprintf(char *,char *, ...);
char *$sprintf(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 *);
int ExitValue();
void mssleep(unsigned long);
```

Figure 6: start **Pico-C** (see welcome message) and use **CHelp()** ; in order to see supported function calls

Remark:

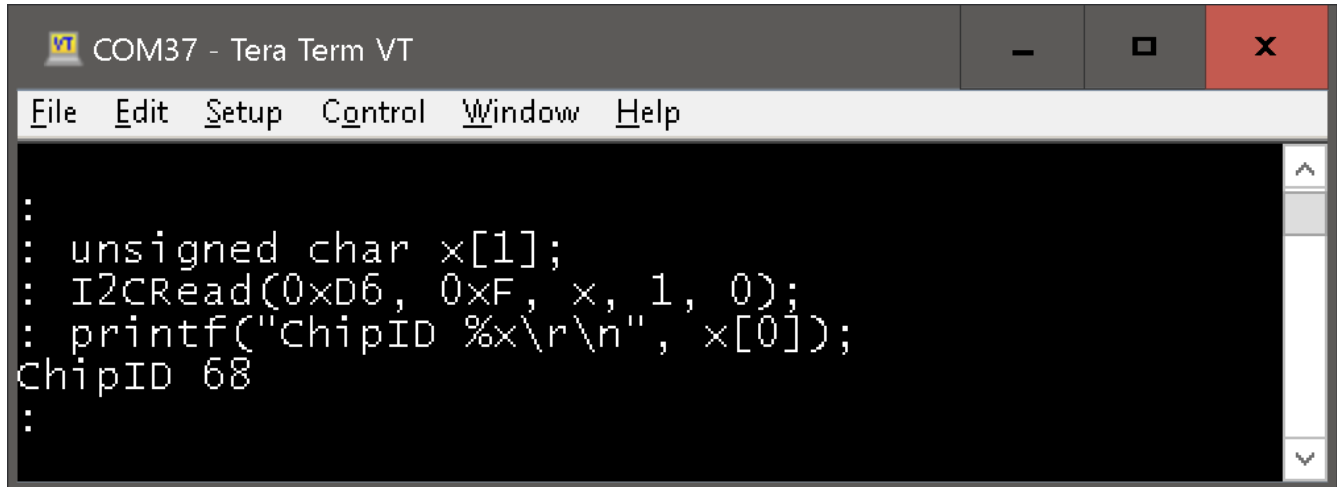
Now, you have to enter **valid C-code statements**. Important is: every C-code line is completed with a semicolon. The syntax is the same as in any C-Code for a C-compiler.

You can exit Pico-C and return to regular command line via:

```
exit(0);
```

You can use also comments, e.g. enclosed in `/* and */` or like C++ via `//rest of line`.

In order to read the ChipID enter these commands:

A screenshot of a terminal window titled "COM37 - Tera Term VT". The window has a menu bar with "File", "Edit", "Setup", "Control", "Window", and "Help". The terminal content shows a C program snippet:

```
:  
: unsigned char x[1];  
: I2CRead(0xD6, 0xF, x, 1, 0);  
: printf("chipID %x\r\n", x[0]);  
chipID 68  
:
```

Figure 6: Read IMU ChipID via **Pico-C**

Pico-C scripts

Instead to demonstrate all the C-code instructions how to enable IMU and read the temperature sensor – we use a script.

The script is stored on an SD card:

- have a Micro-SD-Card, formatted as FAT32
- copy the script below as a file, e.g. “`picoc_ReadTemp.c`”, to SD card
- Initialize the SD-Card on UART command line:

```
sdinit 1
```

You should see the root folder content (and also this file for Pico-C as “`picoc_ReadTemp.c`”)

- When you have launched already Pico-C and now all command line entered strings are C-code-like, you can still enable SD-Card from Pico-C via:

```
ShellCommand("sdinit 1");
```

- start Pico-C via

```
picoc
```


You can launch a Pico-C script via C-command line:

```
RunScript("picoc_Temp.c");
```

You could enter all the Pico-C C-code line also interactively (but a script from SD-Card is more convenient).

Here the script content (or for command line to enter):

```
/*
 * read the temperature sensor in A/G IMU
 * using Pico-C
 */
#define AG_SLAVE_ADDR    0xD6
#define CHIPID_REG       0x0F
#define ENABLE_REG       0x10
#define TEMP_REG         0x15

unsigned char rx[2];
unsigned char GetChipID() {
    I2CRead(AG_SLAVE_ADDR, CHIPID_REG, rx, 1, 0);
    return rx[0];
}

unsigned char chipID;
chipID = GetChipID();
printf("chipID: %x\r\n", chipID);

void EnableAG() {
    unsigned char x[2];
    /* a bit special: x[] needs register address and byte to write */
    x[0] = ENABLE_REG;    //enable register
    x[1] = 0xA0;          //enable value
    I2CWrite(AG_SLAVE_ADDR, x, 2, 0);
}

unsigned short GetTemp() {
    unsigned char cid;
    cid = GetChipID();
```

```

if (cid != 0x68) {
    printf("cannot find A/G chip ID\r\n");
    return 0; //0 as wrong temperature
}
else
{
    signed short temp;
    I2CRead(AG_SLAVE_ADDR, TEMP_REG, rx, 2, 0);
    temp = rx[0];
    temp |= rx[1] << 8;
    temp = 25 + (temp / 16);
    return temp;
}
}

/* use the defined function */
unsigned short t;
EnableAG(); /* enable the A/G sensor */
mssleep(1000); /* wait a bit until first measurement is complete */

void DisplayTemp() {
    int i;
    for (i = 0; i < 10; i++) {
        t = GetTemp();
        printf("temperature: %d[C]\r\n", t);
        mssleep(1000); /* every 1sec a new display */
    }
}

```

DisplayTemp();

When this script was running once, you can launch the defined functions afterwards also on command line, e.g.:

DisplayTemp();

tjaeckel
02/25/2024