olso net Communications

Pawel Gburzynski

Installation and quick start

September 2023

Other related documents available from Olsonet Communications:

[picos]	Programming under PicOS
[vuee]	VUE^2: the Virtual Underlay Execution Engine
[side]	SIDE/SMURPH: a Modeling Environment for Reactive Telecommunication Systems, Reference Manual
[pip]	PIP: an integrated SDK for PicOS
[picomp]	PiComp: the PicOS Compiler
[vneti]	VNETI: Versatile NETwork Interface
[mkmk]	mkmk: the PicOS Makefile Maker
[mspdebug]	MSPDebug man page
[serial]	UART Communication via VNETI (TCV)
[emspcc11]	EMSPCC11 brochure
[mspgcc]	mspgcc: A port of the GNU tools to the Texas Instruments MSP430 microcontrollers

Preamble

Each of the four Olsonet software packages mentioned in this note: PICOS, SIDE, VUEE, and PIP can be fetched individually from https://github.com/senserf/. A link to a specific package looks like this:

https://github.com/senserf/*pkgname*

where *pkgname* can be picos, side, vuee, or pip.

In addition to the documents provided with the packages, you may also find interesting and relevant various supplementary materials, including technical notes, slide presentations, and miscellaneous papers available from Olsonet's web site:

http://www.olsonet.com

This link:

http://www.olsonet.com/software

points to the collection of supplementary software. All the third-party items included there (or pointed to from there) are available (free of charge) from their official sites under their respective licenses. All those licenses hold retaining in full the rights of their authors.

For obscure historical reasons, SIDE is sometimes called SMURPH (this is how it used to be called in the past). Sometimes we also call it SMURPH/SIDE. Don't let it confuse you: SMURH, SIDE, as well as SMURPH/SIDE all refer to the same thing.

Here is a brief explanation of what each of the four packages brings in:

PICOS [picos] contains the source code of the PicOS operating system, libraries, documents, and sample applications (praxes) organized into separate directories (projects). Theoretically, you wouldn't need any of the remaining three packages, if you only wanted to develop software for real-life hardware using manual (command-line tools). But you would still need the toolchains for the respective CPU architectures (i.e., the MSPGCC compiler toolchain for MSP430 and the ARM toolchain for CC13XX).

VUEE [vuee] brings in the set of add-ons to SMURPH/SIDE to create a virtual environment for emulated execution of PicOS praxes. It also provides GUI to VUEE models. VUEE is useless without SMURPH/SIDE and PICOS.

SIDE [side] is an independent simulation/emulation package for networks and reactive systems (so it is useful on its own). It provides the low-level vehicle for building and executing VUEE models of (networked) PICOS praxes. Having the three packages mentioned so far, i.e., PICOS, VUEE, and SIDE, will allow you to develop PicOS praxes for real-life devices, as well as execute them virtually, using command-line tools.

PIP [pip] is an integrated SDK gluing PICOS, VUEE, and SIDE together. It provides a project view of the PicOS praxes and automates the procedures for their editing, configuring, compiling, uploading into physical nodes, debugging, and running them virtually as VUEE models. PIP includes elvis, which is a vi-compatible text editor used by the SDK. The editor was originally written by Steve Kirkendall and adapted by us to collaborate with PIP.

Hardware

You need:

A PC or laptop capable of running Windows or Linux. A virtual machine (Linux on Windows) is fine and recommended. The programs have been tested on Windows 10 + Cygwin and recent Ubuntu (under Oracle VM Virtual Box). These days, we no longer bother to test 32-bit systems, so I cannot vouch for them. The messy Cygwin environment should be probably abandoned, and we should all switch to Linux (e.g., Ubuntu or Fedora) which can be comfortably run in a virtual machine on Windows. That would simplify things even more (and shorten this document a bit). Although we still use Windows 10 + Cygwin (old habits die hard), the Linux setup is friendlier and cleaner, and practically no less capable than the Cygwin environment.

The remaining hardware items are needed if you want to work with real devices. They are not necessary, if you only want to compile and play with VUEE models of the networks.

A JTAG programmer suitable for the target device. We have had some experience with these programmers for MSP430/CC430: MSP-FET430UIF, EZ430-RF, Tiny USB JTAG, but others will likely work, especially when used with the official flashing software from TI, like UNIFLASH, MSP430-FLASHER, or SmartRF Flash Programmer (the last one is only available for Windows). For CC1350, you will be using XDS110 (this is the standard TI debugger interface for CC1350 that comes with LAUNCHXL-CC1350) or the DevPack Debug interface for CC1350STK.

All the programming devices mentioned above work under both Windows and Linux. Note that the XDS100 programmer that comes “integrated” with LAUNCHXL-CC1350 (aka the Launchpad) can be decoupled from the board and used as an independent, standalone programmer for other CC1350-based devices.

PIP can be interfaced to any (reasonable) command-line flash programming software by the user [pip]. And, obviously, GUI software can be used independently of PIP (if its integration poses a problem). Notably, the UNIFLASH tool will let you program basically everything manufactured by TI (on Windows as well as on Linux) and it can even be used to generate dedicated, project-oriented, command-line programming tools whose integration with PIP should pose no problems and can easily be done by the user, as outlined in [pip].

A USB to serial dongle is generally useful (needed) to provide the UART interface for some devices. Note that, e.g., LAUNCHXL-CC1350 is already equipped with a UART-USB interface (which comes together with the XDS110 debugger). For MSP430/CC430 devices, we have been using TTL232R3V3 (it works both under Windows and Linux out of the box). It can be purchased at several places, including the manufacturer: http://www.ftdichip.com/. The connector on our MSP430 EMSPCC11 device (the so-called Warsaw board) has been especially designed for that dongle. If you want to try other solutions, please keep in mind that PicOS UART drivers don't use the CTS/RTS flow control signals, i.e., only two pins on the connector (besides ground) are needed: pin 4 = RX and pin 5 = TX. Those pins are connected to the microcontroller pins and require 3V logic. Another advantage of the USB dongle is that it provides a handy external power supply for the EMSPCC11 board (5V on pin 3 downregulated to 3V). The dongle also works with CC1350, including LAUNCHXL-CC1350 detached from the XDS110 debugger. When connecting the dongle with loose wires steer clear of the 5V USB power supply wire. It can fry the microcontroller if accidentally connected to one of its (3V) pins.

On all recent versions of Windows, the FTDI dongle needs no driver installation.

The requisite cables: a USB cable, a JTAG cable, and so on.

MSPGCC (for MSP430/CC430)

These software components are required for programming real devices based on MSP430/CC430 microcontrollers. In addition to our platform, you need a GCC compiler and toolchain for MSP430.

A bit of history is needed to explain how things stand on this front. The GNU compiler for MSP430 (together with the associated toolchain) was launched ages ago as a GNU open-source project (based on GCC) and made available for Linux and Windows systems [mspgcc]. In 2013 the project was adopted by Texas Instruments which basically meant that, while officially remaining an open-source project, it ceased to be supported (or even influenced) by the community. For a long time, we preferred to cultivate and use the last community version of the GCC compiler making sure that it ran on all the modern systems. The last (vanilla) community version of MSPGCC (in a binary edition for Windows) is available from https://sourceforge.net/projects/mspgcc/. We provide our private version of that final release (slightly tweaked and prepackaged) which still works fine for us. Until recently, the code generated by it was shorter than that produced by the TI version. This has changed, and the TI version is now the winner. Considering that the TI version is being actively maintained by the company (even though with less vigor than their commercial compilers) we suggest using that version as the standard compiler for our platform. It can be obtained from http://www.ti.com/tool/msp430-gcc-opensource (both for Windows and Linux). At

http://www.olsonet.com/software we provide a link to ready binaries of our old (legacy) community version tested under Windows (Cygwin) and Ubuntu. We recommend installing both versions. They do not interfere, and one handy item that comes with our version is MSPDebug [mspdebug], a convenient flash loader which can also be used as a GDB proxy for low-level debugging. It is easy to switch between the two compilers with no interference. The way to switch the toolchain or change the parameters of tools is described in [mkmk] (Section 3.2) and in [pip] (Section 8).

ARM toolchain (for CC1350)

These software components are required to program real devices (as opposed to their VUEE models) based on CC1350 microcontrollers. The most recent toolchain can be downloaded from this link:

https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads

Depending on your Linux flavor, the tools also come as packages, and you may find it the preferred way of installing them. I prefer to use the tarball from the above link, as it covers everything in a single go, even though the stuff does not formally register as a package in the system.

Just in case, here is the list of related packages for Ubuntu (H/T Nick Boers):

- binutils-arm-none-eabi
- gcc-arm-none-eabi
- gcc-arm-none-eabi-source
- libnewlib-arm-none-eabi
- libnewlib-nano-arm-none-eabi
- libstdc++-arm-none-eabi-newlib

I am not sure if you need them all. Here is the list for Fedora's dnf:

- arm-none-eabi-binutils-cs.x86_64
- arm-none-eabi-gcc-cs.x86_64
- arm-none-eabi-newlib.noarch

You should also install OpenOCD which can be obtained from this GIT repo:

https://sourceforge.net/p/openocd/code/ci/master/tree/

This is a flasher-debugger interface (with support for XDS110) that will allow you to connect to CC1350 devices from GDB (H/T Nick Boers). You need at least version 0.11.0-1 of OpenOCD (previous versions had no CC1350 support). Ready binaries (xPack OpenOCD) for Linux are available for download from here:

https://github.com/xpack-dev-tools/openocd-xpack/releases

for simple and easy installation. Prebuilt binaries for Windows (which can be comfortably used under Cygwin) are available here:

https://gnutoolchains.com/arm-eabi/openocd/

Ubuntu/Fedora

The following instructions assume some expertise in Linux (Ubuntu or Fedora). After the installation of a recent vanilla system (we assume it is a 64-bit version), make sure that the following packages are installed: g++ (implies gcc), make, perl, libtinfo5, tclsh, tk, p7zip-full. The default configuration of preinstalled packages may vary with the distribution. In any case, a missing package will show up when referenced; so just look things up and add them when a problem occurs and try again.

Step 1 (not needed for programming and running VUEE models only):

Set up the toolchains. **For MSP430/CC430**, install the TI version of the compiler. Fetch the package from:

http://www.ti.com/tool/msp430-gcc-opensource

Select the complete installer version for Linux (including the support files). First unpack the package into a local temporary directory, e.g.:

```
p7zip -d msp430-gcc-full-linux-x64-installer-3.3.1.2.7z
```

The version number may differ in your case. Then become root and execute the resulting file. When prompted for the target directory, type something like /usr/local/msp430-ti. When the installation is done, add /usr/local/msp430-ti/bin (or wherever you decided to install the toolchain) to your shell PATH.

Then install our legacy version of the toolchain. For that fetch the prepackaged compiler from the link at http://www.olsonet.com/software and unpack it like this:

```
tar xvfj mspgcc_v4_ubuntu_64.tar.bz2
```

This will create a subdirectory in wherever you are. Move to that directory, become root, and execute:

```
./install.sh
```

The installation script stores the package in /usr/local/msp430 and additionally creates this file: /etc/ld.so.conf.d/mspgcc.conf configuring a shared library needed by MSPDebug. As before, add /usr/local/msp430/bin to the shell PATH.

The two toolchains do not interfere because the names of their programs differ. The TI programs are named msp430-elf-…, while the names of programs from our version of the toolchain don't have the "elf" part in the middle (e.g., msp430-elf-gcc versus msp430-gcc).

If you ever want to uninstall the compilers, just remove the directories (nothing is hidden about the installation). For our version, this command:

```
rm -rf /usr/local/msp430 /etc/ld.so.conf.d/mspgcc.conf
```

completely uninstalls the toolchain.

For CC1350, install the ARM toolchain. For the non-packaged version, the most natural way to do that is to fetch the compressed file from the link mentioned above, become root, and go like this:

```
cd /usr/local
bzcat path_to_the_compressed_file | tar -xvf -
```


This will put the toolchain in a subdirectory in /usr/local. Its name looks like gcc-arm-none-eabi-10.3-2021.10 (the version number may differ in your case). Make a link to it from /usr/local/gcc-arm-none-eabi for easier reference. Then add /usr/local/gcc-arm-none-eabi/bin to your shell PATH.

Install the UNIFLASH tool from Texas Instruments. It is an authoritative flash loader for all devices manufactured by TI. Unpack the file retrieved from the TI site and run the thing as root. You may see complaints about missing libraries. UNIFLASH will install despite that (and may in fact work), but for peace of mind you may prefer to install those. Look up (on the Web) the packages containing them. I had to install: libusb-0.1.-4 and libgconf-2-4 to make the installation complete smoothly.

You may also want to install MSP430-FLASHER which is a TI command-line utility for flashing MSP430/CC430 devices.

The actual executable of the UNIFLASH loader is not obvious to locate. To invoke the loader from a command line, you execute the script named uniFlashGUI.sh in the loader's installation directory. It makes sense to link to this script via some more natural name, like uniflash, e.g., from your BIN directory. The default configuration of loaders in PIP assumes that the PATH-visible name of the UNIFLASH executable is uniflash.

Also install OpenOCD (see above). In addition to enabling debugging with GDB, it can serve as an alternative (and less cumbersome) loader/flasher than UNIFLASH. Make sure that the executable (openocd) is on the PATH.

Step 2:

Unpack our software (see **Preamble**) into any directory within your HOME path. No root access is needed for this step. Each package should go to its own subdirectory named PICOS, VUEE, SIDE, PIP in the directory in which you have decided to put them. The structure should be flat. Each package assumes that its co-requisites are located in the same nearest superdirectory.

Make sure that you have a directory named bin or BIN (either will do) in your home directory and that it is mentioned in your shell PATH.

Then go to directory PICOS and execute:

```
./deploy
```

This will configure SIDE, set up a few symbolic links for VUEE, and copy some scripts to your bin (or BIN) directory. Note that deploy (as well as most other scripts used by the platform) is programmed in Tcl.

Install the modified elvis editor that comes with PIP. Move to the PIP directory, become root, and execute:

```
./instelvis
```

You may want to check first if you have the (standard) elvis package already installed. Use the package manager to find out if you have a package named elvis. Most likely you don't (it doesn't get installed by default, and is not extremely popular these days), but if you do, remove it first.

Windows + Cygwin

For this kind of setup (if you really need it), you must start by installing Cygwin from http://www.cygwin.com/. Follow the instructions, i.e., download the installer (setup.exe) and so on. Install the 64-bit version of Cygwin (I am assuming you have a 64-bit machine).

When installing Cygwin, select the recommended setting “For all users”. To avoid problems with missing items, you may just install *everything*. If you want to economize (I do these days), then you will have to keep adding things if they appear missing. No big deal. Just make sure to install initially X11, gcc, g++, make, Tcl, and Tk.

Then install the toolchains as explained above (selecting the Windows version). Make sure that the bin directories of the toolchains are appended to the system PATH. On Windows 10, go to Advanced system settings and click Environment variables…. Select “Path” in the lower pane and click Edit. Then click New for each directory to be added to the list. You may want to reboot (or at least restart Cygwin) to make sure that the change has been noticed. Execute:

```
echo $PATH
```

in a Cygwin terminal to see if the new additions are there.

The rest, including the exercises outlined below, is exactly as for Linux, assuming that you are running X11 under Cygwin and are entering commands from an xterm. Please read the introduction to PIP [pip].

For CC1350 (under Cygwin) you will be able to take advantage of a handy flashing tool that comes with SmartRF Flash Programmer. The package consists of two flash programmers: GUI + command line. The way to integrate the latter with PIP is explained in Section 4 in [pip].

The VM appliance

I have created a VM appliance (please ask for its availability) which contains all the essential components needed for development and experiments with both architectures nicely preset and preconfigured. It can be used as a reference for setting up a similar system for production. I will try to keep the appliance up to date. It boots into an Oracle VM with at least 1GB of RAM. The single user defined within the system is named micro and the password is micro as well.

Having played with the appliance for a while in the “seamless” display mode of the Virtual Machine, I must conclude that it (almost) obviates any residual need for Cygwin that people attached to Windows-only tools (programmers) may still feel. The windows displayed by the Virtual Machine on the Windows desktop provide for a much friendlier and less kludgy interaction with the platform than Cygwin while making the Windows tools still available, if needed.

The appliance works on x64 systems. To use it, you must install Oracle VM VirtualBox which can be obtained at this link:

https://www.virtualbox.org/

If you are new to the idea, please read the information available from the above site. Manuals and tutorials are available there as well as in numerous other places on the net. When installing the VirtualBox, make sure to install the so-called extension pack as well.

Setting up the system

The operation of installing the appliance is straightforward and boils down to executing File→Import Appliance … from the VM VirtualBox Manager. This will set up a virtual machine which on startup will automatically login into the single user named micro.

System content

The system has been derived from a minimal Ubuntu Desktop installation by throwing in the packages required for the platform programs and the toolchains to function. The following platform components mentioned above have been installed (in their recent versions as of the date of this document):

- MPGCC toolchains: TI + Legacy (2 versions) in directories /usr/local/ti/msp430-gcc (the TI version) and /usr/local/msp430 (the legacy version).
- ARM toolchain in directory /usr/local/gcc-arm-none-eabi.
- UNIFLASH loader from TI in directory /usr/local/ti/uniflash. The executable is in /usr/local/ti/uniflash/uniFlashGUI.sh. Strictly speaking, I am not sure if I have the right to preinstall TI software in my VM appliance, so don't brag about this.
- OpenOCD in directory /usr/local/xpack-openocd.

The shell PATH defined for the user (micro) includes paths to all the relevant executables (bin directories), except for uniFlashGUI.sh (which doesn't come in a bin directory).

Disk space

The disk space on the appliance is limited. You may grow it; the procedure is beyond the scope of this document, but several guides are available on the network. Note that you may easily exchange files between the VM and your host operating system (Windows). There is a link named C in the home directory of the micro user that (intentionally) points to your C: drive under Windows. You must set up shared folders for the VM (Settings→Shared Folders) for it to work. Use Folder Path: C:\ and Folder Name: C_DRIVE. Also click the Auto-mount box.

The directory PG in the home directory contains links to stuff on my Windows filesystem (which I use to set things up in the appliance and keep them up to date). Feel free to remove that directory as you won't be able to use those links.

Accessing USB devices from a virtual machine:

For a system running in a VM VirtualBox, you will often have to transfer the USB devices (programmers, UART dongles) used by the platform to the jurisdiction of the virtual machine. A moderate bit of tweaking is required to make UNIFLASH work. First, I had to set the USB controller in the VM to USB 3.0. Then, a programmer device, when plugged in, must be transferred to the virtual machine. This can be done manually, whenever a device is plugged in, but (at least in some circumstances) it makes better sense to do it through USB device filters associated with the VM (available from the VM Manager through Settings→USB), so the delegation happens automatically as soon as the device shows up. One scenario where this is critical is when the XDS100 (for CC1350) needs to have its internal firmware replaced (by UNIFLASH), which will almost certainly happen on the first use of the programmer. Without a special precaution, the USB device will change its parameters (including the name!) half-way through the procedure causing the operation to fail because the VM will lose the device from its sight.

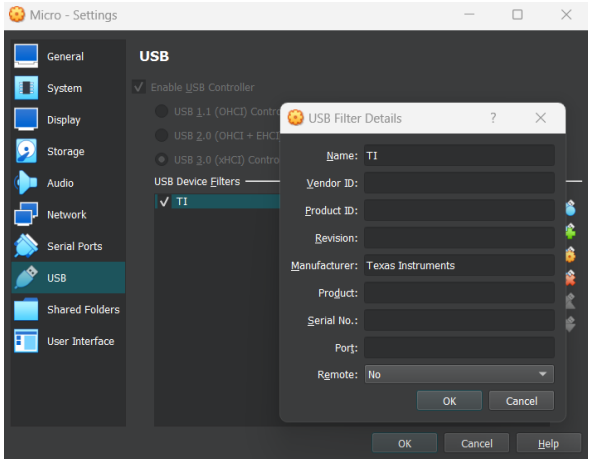

Figure 1. A USB filter for a VM.

It seems that setting a crude filter, e.g., using the Manufacturer string “Texas Instruments” as the device qualifier (see Figure 1) does the trick. The filter can be removed (or made subtler) after the programmer has been reprogrammed. Even better, a blanket “catch all” filter, looking like the one in Figure 1 with the Manufacturer field empty, will intercept all new USB devices as they are being plugged or re-plugged. Leaving this kind of filter permanently might be reckless but activating it momentarily to resolve intermittent problems will be quite effective.

The recommended way of working with USB devices is to have them unplugged before the VM is started and only plug them in after the VM (the system running in it) is fully up. The filter will only apply to the devices that are plugged in while the machine is operational; thus, for as long as you don’t plug/unplug any devices needed by the host system, having a “catch all” filter is not a bad idea. The VM will be claiming all USB devices plugged in while it is up. Later, when things have settled, the filter can be made less aggressive. Note that any filter, once defined, can be easily disabled/enabled.

Exercises

Here we go through a quick drill of compiling a ready project and uploading it into a device. The purpose of this exercise is to check whether things work. The PIP document [pip] presents more elaborated and educational examples with due explanations.

We assume that:

1. The platform has been properly installed. If you are doing this exercise from the VM appliance, this can be taken for granted.
2. You have a device (node) known by PicOS as one of the configured “BOARDs”. In the remainder of this exercise, we assume that the device is either EMSPCC11 (aka the Warsaw board), for MSP430, or CC1350 LAUNCHXL (aka the Launchpad) for CC1350. The tests may work with other boards.
3. You have one of the supported flash programmers, e.g., MSP-FET430UIF (from Texas Instruments) for MSP430. You don’t need a programmer for the Launchpad which comes with its own.

4. You have a means to connect the node's UART to the PC, such that it appears as a /dev/ttyUSBx device. For EMSPC11, this is probably a TTL232R3V3 dongle from FTDI. Strictly speaking, this item is not absolutely required for a test. Note that the Launchpad needs no extra interface: all you need is a USB cable.

Make sure that you (as a Linux user) can control the USB devices. No need to worry about this if you are on the VM appliance, which has things set up right. If you are doing this on your own system and are not sure, the solution boils down to making yourself a member of the user group in charge of the serial devices (typically dialout). Edit /etc/group (as root) and add your user Id at the end of the group entry. Re-login/reboot if necessary.

CC1350 LAUNCHLX:

Connect the device to the PC with a USB cable. If you are on the VM appliance, make sure that the USB device has been captured by the VM (see above).

Run this command:

pip

in a terminal window. You may type:

pip -D

instead to see the debug messages produced by pip, which may help you see and diagnose problems. This command invokes the PIP script [pip], which has been put into your personal bin (or BIN)[1] directory by deploy.

Select Open project from the File menu, then choose EXAMPLES→INTRO→LEDS, by double-clicking on the respective directories in the file browser. When you double-click on the last directory, i.e., LEDS, the browser window will show no more directories (you are within the LEDS project directory). Then click OK in the window. You have entered the LEDS project. It amounts to a very simple program that blinks a LED on the target board.

Select Arch+Board from the Configuration menu. In the window that pops up make sure that the Lib mode box is unchecked, the Arch selection is CC13XX, and the Board selection is CC1350_LAUNCHXL. Click Done when finished.

Click Build (make) in the Build menu. You should see various compilation messages in the main display pane of the PIP window. If the compilation is successful, which it should be, you will see at the end the list of sizes of the various sections of the image produced by the compiler terminated by the string:

--DONE--

Click Configuration→Loaders … to have a look at the list of flash loaders configured with the project. For now, the only loader that is going to work is UNIFLASH. Its executable is named /usr/local/ti/uniflash/uniFlashGUI.sh and it should show up in one of the entries (in Path to the program with empty Arguments). Make sure that the Use button of that entry is clicked (the entry is selected).

Click Execute→Upload image … which will invoke UNIFLASH. The loader should find the Launchpad in a few seconds and announce this in the upper portion of its window. Click Start in that area. The program will show you a widget for selecting the flash image. Navigate to the

[1] It is BIN in the appliance.

project directory and select the file named Image.out. Click Open and then Load Image. UNIFLASH may (or may not) want to re-flash the XDS100 programmer onboard the Launchpad (it probably will on the first try). It is critical for this step that the recognition of USB devices connected to the VM is smooth. The action will be long and messy, but it will be much shorter and smoother on subsequent calls. You will know when UNIFLASH is done.

The device is now flashed with the project's image, and it has been reset and started. You should see the green LED (one of the two LEDs connected to the CC1350) blink. This is what the application is expected to do.

For a bit more fun, connect to the node's UART. Select Start piter from the Execute menu. This opens a terminal emulator. Click the Connect button in its window and select the right device from the topmost menu in the popup dialog that the button has triggered. Easier said than done. The Launchpad is seen (through XDS100) as two serial devices, probably named like /dev/ttyACM0 and /dev/ttyACM1. One of them is the UART, the other is the programmer, but you cannot tell which is which just by looking at their names.[2] Just try them. Make sure that Speed is 9600 and Prot is Direct (these should be the defaults). Then click Proceed or Save & Proceed. If you have hit the right device, then when you enter a piece of text into the bottom input field of the terminal emulator you should see a response: Illegal command.

What commands are legal? Once you are connected to the right UART device, reset the board by pressing the switch close to the USB connector. When the Launchpad resets, it produces a text like this:

```
PicOS v5.4/PG210804A-CC1350_LAUNCHXL, (C) Olsonet Communications, 2002-2021
Leftover RAM: 19340 bytes
Commands:
  on
  off
```

on its UART. This lets you see what the legal commands are. Try them.

MSP430

Plug the programmer to the PC and connect its other end to the node. Also connect the node's UART to the PC, as shown in Figure 2. When you do that, you should see two new devices: /dev/ttyUSB0 (the UART) and /dev/ttyACM0 (the programmer). The numbers may differ if you happen to have other serial devices connected to the PC.

[2] My bet would be that the one with the lower number is the UART.

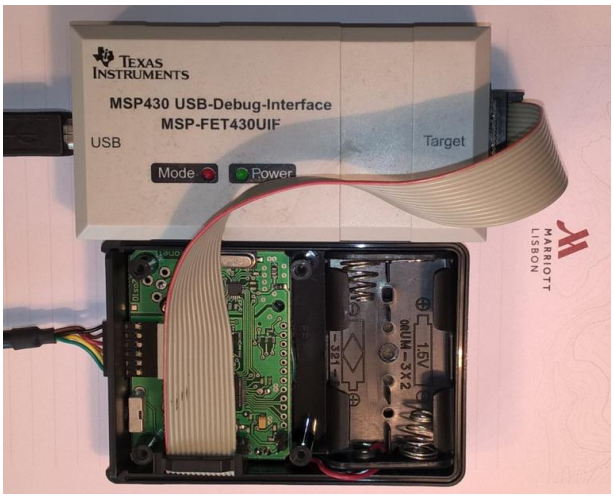

Figure 2. Connecting EMSPC11 to a PC.

If you are doing this from the appliance (a virtual machine, in general) you must make sure that the USB devices have been claimed by the VM, as explained above.

Run this command:

pip

in a terminal window. You may type:

pip -D

instead to see the debug messages produced by pip, which may help you see and diagnose problems. This command invokes the PIP script [pip], which has been put into your personal bin (or BIN)[3] directory by deploy.

Select Open project from the File menu, then choose EXAMPLES→INTRO→LEDS, by double-clicking on the respective directories in the file browser. When you double-click on the last directory, i.e., LEDS, the browser window will show no more directories (you are within the LEDS project directory). Then click OK in the window. You have entered the LEDS project. It amounts to a very simple program that blinks a LED.

Select Arch+Board from the Configuration menu. In the window that pops up make sure that the Lib mode box is unchecked, the Arch selection is MSP430, and the Board selection agrees with your device (it should be WARSAW for EMSPC11). Click Done when finished.

Click Build (make) in the Build menu. You should see various compilation messages in the main display pane of the PIP window. If the compilation is successful, which it should be, you will see at the end the list of sizes of the various sections of the image produced by the compiler terminated by the string:

--DONE--

[3] It is BIN in the appliance.

Connect to the node's UART. For that, select Start piter from the Execute menu. This opens a terminal emulator window. Click the Connect button in that window and select the right device from the topmost menu in the popup dialog that the button has triggered. In the simple case when there is a single ttyUSB device, it should automatically appear at the top. Make sure that Speed is 9600 and Prot is Direct (these should be the defaults). Then click Proceed or Save & Proceed. The terminal emulator is now connected to the device's UART.

Power on the device (I mean EMSPC11). If there are no batteries, this means pushing the switch towards the dongle socket.

By default, the list of flash loaders that you see when you click Configuration→Loaders … should include MSPDebug (this is the case for the VM appliance). Make sure that the Use button at MSPDebug (the top entry) is checked.

Flash the device by clicking Execute→Upload image …. Most likely you will need a firmware update on the programmer when you use it for the first time. In subsequent scenarios, following a successful firmware update, things should proceed faster. Note that the firmware update procedure can be capricious, so you may have to try it several times. Generally, once we get pass this stage, communication with the programmer becomes reasonably smooth.

The programmer writes messages to the PIP's console window. The flashing procedure consists of erasing the previous contents of the node's flash memory and writing the new image over it. At the end of the sequence of messages indicating successful programming, you should see something like:

```
Programming...
Writing 4096 bytes at 4000 [section: .text]...
Writing 4096 bytes at 5000 [section: .text]...
Writing 4096 bytes at 6000 [section: .text]...
Writing 2272 bytes at 7000 [section: .text]...
Writing  870 bytes at 78e0 [section: .rodata]...
Writing   16 bytes at 7c46 [section: .data]...
Writing   32 bytes at ffe0 [section: .vectors]...
Done, 15478 bytes total
MSP430_Run
MSP430_Close
--DONE--
```

Then the programmer resets the device, and the program starts. One of the LEDs should start blinking and the terminal window should show lines looking like this:

```
PicOS v5.4/PG210804A-WARSAW (C) Olsonet Communications, 2002-2021
Leftover RAM: 9802 bytes
Commands:
  on
  off
```

Try the two commands from the terminal emulator.

VUEE

To check whether VUEE/SIDE has been installed correctly, try running the same LEDS program under the emulator. Under PIP, while still within the LEDS project, choose Build→VUEE. After a while, the compiler will have created the model. Click

Configuration→VUEE and make sure that the Run with udaemon box is checked. Then click Execute→VUEE. On Windows/Cygwin, this may trigger a firewall warning, because the model needs to open a network socket to communicate with its GUI. Just let the program do it.

The model will start (you will see some messages in PIP's console) and a small window (labeled "udaemon") will pop up as an interface to the model. The trivial network in our example consists of a single node, so there isn't a lot you can do with it. Enter 0 into the Node number field (this is the node number), select LEDS in the menu to the right (initially showing UART) and click Connect. You will see the three LEDs of the node with the green LED blinking, as it would in the real device.

Note that you can also connect to the node's virtual UART. For that, select UART (ascii) in the menu and click Connect again. You will see the output produced by the node. You can switch the blinking on and off by entering commands into the UART.