olsonet Communications

Pawel Gburzynski

Installation and quickstart

May 2021

Preamble

Each of the four Olsonet software packages mentioned in this note: PICOS, SIDE, VUEE, and PIP comes with documentation. In addition to the documents that you will discover inside 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/REPO/contents.html

points to the collection of the requisite 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. Needless to say, all those licenses hold retaining in full the rights of their authors.

For some rather 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 is:

> **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 (e.g., the MSPGCC compiler toolchain for MSP430).
>
> **VUEE** brings in the set of libraries providing add-ons to SMURPH/SIDE to create a virtual environment for the emulated execution of PicOS praxes. It also contains a GUI to VUEE models (the udaemon script). VUEE is useless without SMURPH/SIDE and without PICOS.
>
> **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** 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 virtual execution (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. Elvis needs a separate installation because it requires root access (on UNIX systems).

Hardware

You need:

A PC or laptop capable of running Windows or Ubuntu Linux. The programs have been tested on Windows 10 + Cygwin and Ubuntu 20.04. These days, we no longer bother to test 32-bit systems, so I cannot vouch for them. My plan is to abandon the Cygwin environment and switch to Linux/Ubuntu/Fedora (which can be comfortably run in a virtual machine on Windows). This will simplify things even more. Although we still use Windows 10 + Cygwin, the Linux setup is friendlier and cleaner.

The remaining hardware items are needed if you want to work with real devices. They are not needed, if you only want to compile and run 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 above programmers work under Windows as well as Linux. Note that the XDS100 programmer that comes "integrated" with LAUNCHXL-CC1350 can be easily 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. 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.

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 a number of places, including the manufacturer: http://www.ftdichip.com/. The connector on our workhorse 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 down-regulated to 3V). The dongle also works with CC1350, including LAUNCHXL-CC1350 detached from the XDS110 debugger. When connecting the dongle by 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: (for the programmer), i.e., a USB cable, a JTAG cable, and so on.

MSPGCC (for MSP430/CC430)

These software components are required for programming real devices (as opposed to their VUEE models) based on MSP430/CC430 microcontrollers. In addition to our platform, you need the MSPGCC compiler. That compiler (together with the associated development tools) was launched ages ago as a GNU open-source project (based on GCC) and made available for Linux and Windows systems. In 2013 the project was kindly "adopted" by Texas Instruments which basically means that, while remaining an open-source project, it is no longer supported (or even influenced) by the community. While formally, the most recent official version of the compiler can be used with our platform, we strongly prefer to use instead the last community version (with a few tweaks). This is because the TI compiler generates much longer code (by about 30%). No big surprise: TI has to boast their commercial tools as being superior to the freebies that they so generously give away.

This is why, for MSP430/CC430, we cultivate the last community version of the gcc compiler and make sure that it runs on all the modern systems. It has never failed us, and we see no

reasons to fix what ain't broke. The last (vanilla) community version of MSPGCC (in a binary edition for Windows) is available from https://sourceforge.net/projects/mspgcc/. The TI version (Windows + Linux), including the sources, can be obtained here: http://www.ti.com/tool/msp430-gcc-opensource. In our REPO, we provide ready binaries for Windows (Cygwin) and Ubuntu which have been thoroughly tested with our platform.

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

In our REPO, we provide the last version tested with our platform, but (with a very high likelihood) a newer version (if available) will be more than fine. When installed manually (from the above link) the toolchain provides all the components needed by our platform.

Depending on your Linux flavor, the tools also come as packages, and this should probably be the preferred way of installing them. 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 may want to try openocd which can be obtained from this GIT repo:

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

This is a debugger interface (with support for XDS110) that will allow you to connect to CC1350 devices from (a multiarch version) of gdb (H/T Nick Boers).

Ubuntu/Fedora

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

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

Set up the toolchains. For MSP430/CC430, install MSPGCC. Fetch the prepackaged compiler from our REPO and unpack it, e.g.,

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

This will unpack into a directory. Move to that directory, become root, and execute:

```
./install.sh
```

The installation script stores the package in /usr/local/msp430 and creates this file: /etc/ld.so.conf.d/mspgcc.conf. This command:

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

completely uninstalls the compiler.

Following the installation, make sure to add /usr/local/msp430/bin to your shell PATH, so the compiler can be easily found by the platform.

For CC1350, install the ARM toolchain (see above). For the non-packaged version, the most natural way to do that is to unzip and untar the tarball file into /usr/local and make a link from /usr/local/arm to the directory created this way. Then add /usr/local/arm/bin to your shell PATH. The packaged versions install into more prominent standard places.

Install the UNIFLASH tool from Texas Instruments (regardless of the system). It is an authoritative flash loader for all devices manufactured by TI, so you can always use it, if only as a last resort. You may also want to install MSP430-FLASHER which is a TI command-line utility for flashing MSP430/CC430 devices.

If on Windows, also install SmartRF Flash Programmer. Its handy feature (for CC1350) is a command-line flasher for CC1350 named srfprog.exe to be found in the bin directory of the installation.

If your Linux system runs in a VM VirtualBox, you will have to transfer the USB devices (programmers, UART dongles) to its jurisdiction. This can be done manually, but it makes better sense to do it through USB device filters, so the delegation happens automatically as soon as the device shows up. For example, when the XDS100 (for CC1350) needs to have its firmware replaced (say by UNIFLASH), and it will definitely need that on first use, the USB device will change its parameters (and name) half-way through the procedure causing it to fail (because the VM will lose the device from its sight). It seems that setting a crude filter, e.g., using the string “Texas Instruments” as the device qualifier does the trick. The filter can be removed (or made subtler) after the programmer has been reprogrammed.

Figure 1: Connecting EMSPC11 to the PC.

Step 2:

Make sure that you have a directory named bin or BIN in your home directory and that it is mentioned in your PATH. Unpack the software (see **Preamble**), 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 written in Tcl.

Install the modified elvis editor that comes with PIP. On Cygwin, just execute

```
./instelvis
```

in PIP's directory. Under Linux, you should first check 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.

The MSP430/CC430 setup comes with MSPDebug flash loader/debugger (included in our version of the toolchain). This allows you to immediately flash target devices with the programmers listed above.

For CC1350, you have to install the UNIFLASH tool from TI.

An exercise (MSP430):

You are basically set. To check whether things work, do this exercise. We assume that:

1. The platform has been installed as described in Steps 1 and 2 above.
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 EMSPCC11 (aka the Warsaw board).
3. You have one of the supported flash programmers, e.g., MSP-FET430UIF (from Texas Instruments).
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.

Make sure that you (as a Linux user) can control the USB devices. This boils down to making yourself a member of the respective group (typically dialout – edit /etc/group and add the user Id at the end of the group entry). Re-login/reboot if necessary.

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 the picture above. 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.

Execute this command:

```
pip
```

in a terminal window. You may type:

```
pip -D
```

instead (especially if doing it for the first time), to see the debug messages produced by pip, which may help you see and diagnose problems. This command invokes a script, which has been put into your personal bin (or BIN) directory by deploy, implementing the platform's GUI SDK.

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--

Connect to the node's UART (optional). 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 window 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.

Flash the node. First configure the loader. Select “Loaders” in the “Configuration” menu. In the window that pops up, select the MSPDebug loader (by clicking on the radio button), then make sure that:

FET Device selection is “Auto”

The driver is “tilib”

The “allow firmware update” box is checked

GDB connection port is “none”

Click “Done” to close the window and finalize the loader selection. Note that 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, the box can be unchecked.

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

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 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--
```


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

```
PicOS v5.0/PG201119A-WARSAW, (C) Olsonet Communications, 2002-2020
Leftover RAM: 9802 bytes
Commands:
  on
  off
```

This concludes the exercise.

Beware that the flash loader is occasionally capricious and may signal a communication error when invoked for the first time after connecting the programmer to a USB port (error number 35). In such a case, simply try again. Typically, this error occurs only before the first use of the programmer, immediately after plugging it in.

If MSPDebug complains about errors in accessing USB devices (hinting at inadequate permission), the crude but effective solution is to make mspdebug setuid root, i.e., become root, cd to /usr/local/msp430/bin and do:

```
chmod +s mspdebug
```

While such solutions are not recommended for serious production systems, doing this on a virtual machine dedicated to running the platform makes perfect sense.

Windows

You have to 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 (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 it these days, because, e.g., the installation of TeX/LaTeX takes forever), 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 MSPGCC compiler. Fetch the proper binary version from our REPO (it comes as a 7z archive) and unpack it into C:\mspgcc. You can extract the archive into C:\ and subsequently rename the containing folder to mspgcc. You should see there these subfolders: bin, lib, libexec, msp430, and share.

Make sure that C:\mspgcc\bin is appended to the system PATH (in a way that will work both for Windows and for Cygwin). On Windows 10, go to “Control Panel” → “System and Security” → “System” → “Advanced system settings” → “Environment variables”. Select “Path” in the lower pane and click “Edit”. Then click “New” and add C:\mspgcc\bin\, e.g., at the end of 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 addition to PATH is there.

The rest, including the exercise is exactly as for Linux, assuming that you have invoked X11 under Cygwin and are entering commands from an xterm. Please read the introduction to PIP (the link can be found in the REPO).

As an option, you may prefer to use under Windows the native version of Tcl/Tk from ActiveState (native, in the sense that it doesn't rely on Cygwin's X11), which gives a somewhat nicer appearance of Tk windows. For that, install the most recent “community edition” (free) version of Tcl/Tk from here: http://www.activestate.com/activetcl/downloads.

Then, to make sure that the platform will use the ActiveState Tc/Tk version whenever it can, you have to add links to the respective binaries to you personal BIN directory, like this:

```
cd ~/BIN
ln -sf /cygdrive/c/ActiveTcl/bin/tclsh.exe tclsh86
ln -sf /cygdrive/c/ActiveTcl/bin/wish.exe wish86
```

The source paths may differ in your setup, depending on your installation of the ActiveState package, but the names under which the programs are made visible from BIN must be as shown.

Then, when you provide the -l argument to deploy (this is el, not capital aye), i.e.,

```
./deploy -l
```

then PIP and a few other GUI scripts will opt to use the ActiveState Tcl/Tk for their execution environment. Note that this doesn't make the platform Cygwin-independent, or even X11-independent.

For CC1350, and if you are under Cygwin, you will be able to use a handy command-line programmer that comes with SmartRF Flash Programmer. The package consists of two flash programmers: GUI + command line. The first one can be used independently of PIP, the other can be configured with PIP, so it can be invoked from PIP's Execute menu. For that, within a CC1350 project, click Configuration → Loaders … and fill in the “Command line” section. First browse to the executable file of the command-line loader (the file is somewhat confusingly named srfprog.exe and can be found in the bin directory of the installation), then enter this string into the Arguments field:

```
-t lsidx(0) -e pif -p -f %f.hex
```

The sequence %f will be replaced by the root file name of the flashable image when the loader is invoked. You can learn about the arguments by running srfprog.exe from a DOS command prompt.

Note that you can also configure a GUI-based (non-command-line) loader such that it is invoked from PIP's Execute menu; however, it may not be easy to pass it the location of the file to be flashed into the device. You may want to do this with UNIFLASH on Ubuntu which will give you a foolproof (if somewhat imperfectly integrated) flash programmer for CC1350. The executable is confusingly located in file node-webkit/nw in the installation directory of UNIFLASH. Select the path to that file in “Command line” section of Configuration → Loaders … and clear the Arguments field. When the loader pops up for the first time, you will have to manually navigate to the file to be flashed in, but on subsequent occasions the file will be accessible as a suggestion.

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 create the model. Click Configuration → VUEE and make sure that the “Run with udaemon” box is checked. Then click Execute → VUEE. On Windows, 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 the main pane of the PIP window) and a small udaemon (model interface) window will pop up. 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 “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 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 from the UART.