olsonet Communications

Installation and quickstart

for

MSP430-based boards

November 2014

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 requisite third party software. All that software is available (free of charge) from other (official) sites under their respective licenses. Needless to say, all those licenses hold retaining in full the rights of the respective 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 refer to the same thing. Here is a brief explanation of what each of the four packages is about:

> **PICOS** contains the code of the PicOS operating system, libraries, documents, and sample applications (praxes) organized into separate directories (projects). Theoretically, this package would suffice (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).
>
> **VUEE** brings in the set of libraries providing the requisite add-ons to SMURPH/SIDE to create a virtual environment for emulated execution of PicOS praxes. It also contains a GUI to VUEE models (the udaemon script). Note that VUEE is useless without SMURPH/SIDE.
>
> **SIDE** contains an independent simulation/emulation package for networks and reactive systems. 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, compilation, uploading (into physical nodes), debugging, and virtual execution (as VUEE models).

If you are acquiring the packages via GIT, you may want to have a quick look at a document (available from the REPO) explaining how to do it (the acquisition requires a user name and password to access our repository). Otherwise, you have probably received a single tar'red and gzip'ped archive, which unpacks into four separate directories: PICOS, SIDE, VUEE, and PIP. They can be unpacked into any location in your home directory hierarchy. It makes sense if all three directories occur in the same place (as subdirectories of the same directory). If you are using Windows, install Cygwin (see below) before unpacking the software.

Hardware

You need:

A PC or laptop capable of running Windows (XP is fine, and the PC doesn't have to be high end) or Linux (say a recent version of Ubuntu).

A JTAG programmer. We have used quite extensively these devices:

> Tiny USB JTAG (from OLIMEX)

MSP-FET430UIF (from Texas Instruments)

EZ430-RF (aka RF2500, also from Texas Instruments)

All the above programmers work under Windows as well as Linux. The last one, RF2500) is primarily used with EZ430-CHRONOS. It is possible that other JTAG programmers will work, too (especially with mspdebug under Linux – see below), but I cannot vouch for them.

A USB to serial dongle, preferably TTL232R3V3 from FTDI (it works both under Windows and Linux with no problems). You need it to communicate with boards running PicOS (via UART). 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) was especially designed for that dongle. If you want to try other solutions, please keep in mind that only two pins on the connector (besides ground) are needed for serial communication (pin 4 = RX and pin 5 = TX). Those pins are connected to the microcontroller pins and require 3V logic. No handshakes (CTS/RTS) are needed. 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).

On all recent Windows versions (with the possible exception of XP), the FDDI dongle needs no driver installation (Windows manages to locate the drivers on its own). If needed, the drivers are available from our REPO (or from the FTDI download area).

The requisite cables: (for the programmer), i.e., a USB cable, a JTAG cable, and so on.

Windows

Note that these days things are a little bit more friendly under Linux (e.g., Ubuntu) than under Windows. On 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. Select the recommended setting “For all users”. To avoid problems with missing items, make sure you have installed *everything*. In the “Select packages” window, click on the looped arrows in the topmost line (the one that says “All”) until the text to the right of it reads “Install”. This selects “Install” for “All” packages.

Rebasing

As of version 1.7.13 of Cygwin, this issue should not be longer present thanks to the so-called *autorebasing* done at the installation. I strongly recommend to use the newest available version of Cygwin, unless you have an extremely good excuse for using something different.

If, following the installation, things begin to crash on you in a weird sort of way (failing forks, aborting programs), try this:

1. Shut down all Cygwin activities, X-server, Cygwin windows.
2. Start the DOS command prompt.
3. Execute this command in the DOS window:

   ```
   C:\cygwin\bin\ash.exe
   ```

   to start a statically linked Cygwin shell.
4. Execute (in that shell) this command:

   ```
   /bin/rebaseall
   ```

Then start Cygwin again and see whether things aren't better.

The C compiler

Set up MSPGCC. The simplest way is to fetch ready binaries from the official site, or, from our REPO. That is a ZIP archive. Make sure that it unzips into C:\, i.e., the top-level directory of the package is C:\mspgcc, with its subdirectories bin, lib, libexec, msp430, and share.

Alternatively, you can fetch the full source package from the REPO and compile it according to the enclosed instructions. You may want to do that, if the binary package doesn't work for you for whatever reason. But note that I haven't tried to compile the package for Windows (I only did it under Linux), so you are on your own.

Following the installation (this applies to both cases), make sure that the path to MSPGCC executables is known to Windows (and to Cygwin, but it is sufficient to make it known just to Windows). For that:

1. Right-click My Computer and click Properties.
2. In the System Properties window (in the left pane on Windows 7) click Advanced (system settings).
3. In the Advanced section, click the Environment Variables button.
4. In the Environment Variables window, highlight the Path variable in the Systems Variable section and click the Edit button. Add C:\mspgcc\bin to the Path.

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 path is there. If not, you can always add it by hand (by editing .bash_profile).

The Doc subdirectory of PICOS includes mspgcc.pdf which is a (somewhat aged, but still good) manual of MSPGCC (written in 2003 by Steve Underwood). Glance through chapters 8 and 10. Chapter 10 explains how to use gdb with MSP430 via gdbproxy, which may be useful for debugging. These days gdbproxy is a bit outdated and it doesn't work with the new firmware for the popular programmers (so we do not cover it in this document). Under Linux, gdb can be used with mspdebug. With PIP, most of useful activities, including invocations of flash loaders, have been automated, so you need not worry about it right away.

Tcl/Tk

The package relies heavily on Tcl/Tk for the execution of its numerous scripts, including the GUI-oriented ones, especially pip, piter, and udaemon (of VUEE). Some of those scripts require version 8.5 (or higher) of Tcl/Tk which, until recently, wasn't directly distributed with Cygwin or Linux. Starting with version 1.7.13 of Cygwin and 11.10 of Ubuntu, Tcl/Tk version 8.5 comes with the standard distribution, which means that you do not have to provide an external (independent) installation of Tcl/Tk.

If for whatever (good) reason you are forced to use an older version of Cygwin, you will have to install Tcl 8.5 (or Tcl 8.6, which should also be fine) for Windows. For that you may go to ActiveState's download area and select the respective (recent) version fitting your system, or download the last one we have been using from the REPO. Let it go to the default place, which will make it easy for the deploy script (see below) to find. Note that whatever version of Tcl comes with Cygwin, you will need it as well: the Cygwin version is for compatibility with Cygwin, the other (external) one is needed for some useful features that are missing in 8.4.

Also note that even if your Cygwin (native) Tcl/Tk is 8.5 or higher, you can still use the ActiveState Tcl/Tk as an option. It will give you a slightly different (perhaps more classy) look of the windows. Also, instead of UNIX-style naming for serial devices (/dev/ttySx), with the ActiveState version of Tcl/Tk, the scripts will be using Windows-style names (COMx).

Having installed Tcl 8.5, you will have to restart Cygwin, so it can pick the new component of the PATH environment variable from its updated Windows version. Do this as a habit whenever you install anything under Windows that Cygwin programs may depend on.

FET programmers

Fetch "windows tools" from the REPO. If you are using MSP-FET430UIF, install the CDC drivers. If you are using JTAG-TINY from OLIMEX, read the README file in the OLIMEX directory and follow the instructions. In a nutshell, you have to make sure that the system will recognize your JTAG programmer.

It also makes sense to install some flash programmers. Under Windows, it is a good idea to start with the Elprotronics FET-Pro430-Lite (which you will also find in "windows tools"). This is basically the default flash loader configured into PIP under Windows. Feel free to also install TI SmartRFProgrammer (available from the REPO) and/or the OLIMEX flash loader (which comes with "windows tools"). With a bit of creative tweaking, the OLIMEX loader can be made to work with PIP. Of course, you can always use any external (correctly installed) flash loader manually, i.e., independently of the SDK.

Note that the installation of TI SmartRFProgrammer will bring in the drivers for RF2500 needed to program EZ CHRONOS (which can be accomplished from FET-Pro430-Lite).

For manual programming, note that some flash loaders require the image file to have the suffix .hex (or perhaps something else), while the suffix of the (Intel) HEX file generated by mkmk/PIP is .a43. In such cases, you will have to rename/copy this file as needed.

FTDI USB dongles

On Windows XP, you may have to explicitly install a driver for TTL232R3V3 (the newer systems can manage without help). You can get one from FTDI's web site, but a driver is also available in the REPO. Plug in the TTL232R3V3 dongle into a USB socket on your computer (there is no need to attach the board to the other end). When (and if) the system prompts you for drivers, run the installation program of the FTDI driver that suits your system.

The actual installation

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 prepare 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 many other scripts used by the setup) requires the standard (native) version of Tcl, which comes with the full Cygwin installation. The script determines the version number of the native Tcl and, if it is 8.5 or higher, will assume that native Tcl/Tk is to be used for all scripts (see Section Tcl/Tk above). If you want to override that decision, and you have installed Tcl/Tk from ActiveState, execute:

```
./deploy -l
```

instead. When called this way, deploy will try to use the external installation of Tcl/Tk (8.5 or 8.6) for those scripts that require version 8.5, specifically udaemon, piter, pip, and its friends.

A quick-start exercise (without PIP)

Note: this exercise involves command-line tools. If you find its complication intimidating, feel free to skip it and go directly to PIP's documentation. PIP provides natural clickable shortcuts for all the typical actions amounting to program development under PicOS.

1. Move to directory PICOS/Apps/RFPing which contains a simple praxis testing RF communication between pairs of nodes.
2. Execute this command in that directory:

```
mkmk WARSAW
```

The argument identifies the board for which the praxis should be compiled. It corresponds to a directory which you will find in PICOS/PicOS/MSP430/BOARDS. That directory contains the description of EMSPCC11 (which is also known as the "Warsaw" board).

3. The script will create a Makefile in the praxis directory. Now, execute make to compile the praxis into a program that can be loaded into the device.

   Having performed the above steps, you should see in the praxis directory (among other things) these two files: Image and Image.a43. They are two versions (ELF and HEX) of the same code that can be uploaded into a microcontroller.

4. Make sure the target (Warsaw) board is connected to the programmer and powered on. For this exercise, we assume MSP-FET430UIF programmed through Elprotronic FET-Pro430-Lite.

5. Launch FET-Pro430-Lite, select MSP430F1611 as the "microcontroller type", and open the file Image.a43. Click the AUTO PROG button. if everything is OK, the image should be flashed into the microcontroller.

The flash loader is GUI-based, so it is easy to explore its options and resolve some minor problems that may occur along the way. You may be asked whether to upgrade the firmware in the FET programmer. There is no need to do it as long as everything works fine. With version 3 of TI MSP430.DLL, the flash loaders insist on having in the FET programmers the exact firmware version that they expect, but in some cases what they call "upgrade" may be in fact a "downgrade".

If you want to use OLIMEX JTAG-TINY to upload the program image, then you should launch the OLIMEX loader instead. Again, select the proper CPU model (MSP430F1611), and USB for the port. You have to rename (or copy) the image file to something like Image.hex, because this rigid file type is expected by the OLIMEX loader. The rest you will be able to figure out from the loader's window.

To communicate with the board over UART, you will have to connect the board to your PC via the TTL232R3V3 dongle. When connected, the dongle appears as a COM port. You can use the terminal emulator available from the REPO to connect to that port. This program can be executed directly (it is not installed). Set the baud rate to 9600, 8 data bits, no parity, 1 stop bit, and no handshake.

Before you invoke the terminal emulator, you have to make sure that the dongle's COM port number is not greater than 10 (the emulator doesn't see COM numbers higher than 10). Run the Device Manager, find the device, and, if its port number is higher than 10, change it. Windows will likely object to this action telling you that the target port number is in use, but (unless you have reasons to believe that the system is right) you can safely ignore the warning and force the change.

The terminal emulator offers you a number of options. Select the COM port number of the dongle and select the baud rate of 9600. In the "Transmit" area, check the CR=CR+LF box (this isn't absolutely necessary). You may also want to change the default font (the "Set font" button in the "Settings" area) to something fixed (like Courier). Finally, hit the "Connect" button to activate the connection.

An alternative, Tcl-based, GUI/command-line terminal emulator is available as piter (it has been copied by deploy to your bin directory from PICOS/Scripts). This is a Tcl script which you invoke, e.g., this way (for a command line version):

```
piter -p port -s rate
```


where port is the COM[1] number and rate is the baud rate, e.g.:

```
piter -p 4 -s 9600
```

Note that the script accepts arbitrary COM numbers (unlike terminal).

When you call piter without arguments, it will open a GUI window. You will have to select the serial device, the rate, and the communication mode (use Direct protocol and make sure the Bin box is not checked). A detailed description of piter can be found in Serial.pdf, in PICOS/Docs.

If you now reset the board (by switching it off and on, or from the loader), you should see in the terminal window the menu of commands of the RFPing praxis. Note that if you do not enter a command within 10 seconds, the praxis will assume that the UART is not connected, and it will commence automatic packet transmission and reception.

Linux

You should be on a reasonably recent Ubuntu.

The C compiler

Set up MSPGCC. The simplest way is to fetch the precompiled version from the REPO. Become root and and cd to /. Then unzip and untar the archive, e.g.,

```
zcat 131118_mspgcc_bin_linux.tar.gz | tar -xvf -
```

The package will unwrap into /usr/local/msp430. Also, an ldconfig entry for the MSP430 (shared) library will be stored in /etc. Make sure that /usr/local/msp430/bin is in your PATH and execute ldconfig to make the shared library visible. You are basically set.

Note: if the mspgcc compiler fails complaining about missing libmpc.so.2, create a link to libmpc.so.2 from libmpc.so.3. On Ubuntu (14.04+) the library is located in /usr/lib/i386-linux-gnu. Do this (as root):

```
cd /usr/lib/i386-linux-gnu
ln -sf libmpc.so.3 libmpc.so.2
```

If you don't have the library at all, install it (the package name is libmpc3).

Alternatively, you can compile from sources. For that, you have to fetch the source pack from our REPO and read the instructions. Note that the package includes the MSP430 DLL by TI (in its Linux version) as well as mspdebug, which is the preferred flash loader under Linux. All those components come ready in the precompiled version.

Tcl/Tk

Make sure that Tcl and Tk are included in your system (as the respective packages). Recent Ubuntu comes with Tcl/Tk version 8.5 (or maybe higher, if it is even more recent), which is fine for all our needs. You just have to make sure that both Ubuntu packages: tcl8.5 and tk8.5 are installed. You can also install an external version of Tcl/Tk from Active State and override the native package with -l specified as the argument to deploy (the same as for Windows).

FET programmers

With the MSP430 DLL by TI, MSP-FET430UIF as well as RF2500 work out of the box (with mspdebug). Depending on the firmware version present in MSP-FET430UIF, the operation of upgrading its firmware (carried out by mspdebug) may require some manual attention. A

[1] With the native version of Tcl/Tk, the device names recognized by piter are UNIX-like, e.g., /dev/ttyS2 corresponds to COM3. Note the shift in numbering: COM ports are numbered from 1 (COM0 is not a legal device name under Windows), so /dev/ttyS0 is mapped to COM1. You can ask piter to scan for responding devices, so you can easily learn which UART-like devices can be potentially connected to.

simple solution is to do it under Windows, say from TI SmartRFProgrammer, and then the "upgrade" by mspdebug under Linux will be smooth and semi-automatic.

JTAG-TINY from OLIMEX supposedly works with mspdebug as well. I couldn't test it, because I only have an old version of JTAG-TINY (version 1) which normally should also work, but I accidentally erased firmware on it some time ago. Although I have managed to put some firmware into it (using the OLIMEX programmer), and it does work with the OLIMEX programmer under Windows, it is not correctly recognized as version 1 by mspdebug. I will order a new device from OLIMEX and check this later.

The bottom line is that all FET programmers that we have ever used should work fine with mspdebug under Linux.

FTDI USB dongles

The UART dongle (TTL232R3V3) requires no special attention. When you plug it in, a device should pop up in /dev, whose name will be most likely ttyUSB0. You can use the same piter script as under Cygwin for a terminal emulator. The port argument can be either a number or a full device name, e.g., these calls:

```
piter -p 0 -s 9600
piter -p /dev/ttyUSB0 9600
```

are equivalent. The GUI version of piter works similar to the Windows version.

A quick-start exercise (without PIP)

This is basically the same exercise (involving command-line tools) that we did for Windows. This time we show how to use mspdebug in the simple command-line mode, including debugging by gdb. PIP automates most of these actions, but it is educational to see what exactly is involved.

Perform steps 1-3 from the Windows exercise to produce the two loadable image files Image and Image.a43., then:

1. Make sure the target (Warsaw) board is connected to the programmer and powered on. For this exercise, we assume MSP-FET430UIF programmed through mspdebug.[2]
2. Execute this command:

   ```
   mspdebug --allow-fw-update tilib "prog Image"
   ```

   The first argument of mspdebug allows the program to update the firmware on MSP-FET430UIF, which it probably will want to do the first time around. The second argument selects the TI MSP430 library as the "driver" for the FET programmer. This is the recommended way. For old firmware (if you are reluctant to replace it), you may try uif instead of tilib. The last argument is the command to be executed by mspdebug, which means "program the specified image file into the target device". Note that mspdebug deals with ELF files rather than their HEX versions.

It is possible to use mspdebug as a "bridge" for gdb to connect to the target device and debug the program. In a separate terminal window (not necessarily in the praxis directory), execute:

```
mspdebug tilib "gdb 2000"
```

The number following gdb is the socket port for gdb connection. Then, in the praxis directory, execute:

[2] If mspdebug complains about not being able to access the USB programmer device, make sure that the device permissions are right. This may involve writing udev rules or fiddling with groups. A quick and dirty hack solving the problem is to make mspdebug "suid root", i.e., sudo bash, chown root:root mspdebug, chmod +s mspdebug. Although this flies in the face of security, it is OK, I guess, especially if your Linux runs in Virtual Box.

```
msp430-gdb Image
```

Note that mkmk has created in the praxis directory two files: gdb.ini and .gdbinit with identical contents. One of these files is read by msp430-gdb (depending on whether you are on Windows or on Linux) and, in particular, identifies the (socket) port over which gdb, when called in this directory, will talk to a bridge (proxy). If everything is OK, gdb should display this piece of text:

```
0x00004000 in _reset_vector__ ()
```

before presenting its prompt. Now you can erase the board's code flash with this command:

```
monitor erase all
```

and then load the program into it:

```
load Image
```

which may take a few moments. When done type:

```
monitor reset
continue
```

to run the program. These basic steps are explained in chapter 10 of the MSPGCC manual by Steve Underwood.

Note that the communication between gdb and mspdebug involves a TCP socket. Thus, it is possible to have the two parties run on different machines connected via the Internet. For this, you will have to edit gdb.ini/.gdbinit replacing localhost with the address of the machine running the bridge.

SIDE + VUEE

SIDE comes with extensive documentation. SIDE/MANUAL/manual.pdf contains a (reference) manual of the present version, which I have (obsessively) tried to keep up to date trading the accuracy for style and general friendliness of the document. SIDE/MANUAL/BOOK/book.pdf is the image of an old book, which is much more friendly than the manual, but considerably outdated (in particular, is knows nothing about PicOS or VUEE).

Note that deploy has set up SIDE and VUEE to work with PICOS.

Try this exercise now:[3]

Go to PICOS/Apps/RFPing_u. That directory contains a slightly sterilized variant of the RFPing praxis, which can be compiled by MSPGCC as well as for VUEE. You can try compiling it the standard way, i.e.,

```
mkmk WARSAW
make
```

and then for VUEE:

```
picomp
```

The two compilations do not interfere. For example, you can load the Image file into the board and check if it works. Then, you can execute the same praxis virtually under SIDE:

```
./side data1.xml
```

Having issued the above command, execute this in a separate terminal window:

```
udaemon
```

[3] Again, this is a command-line exercise. You can accomplish the same feat from PIP with more ease, but, probably, with less understanding of the mechanisms involved.

This should open a Tk (wish) window providing an interface to the virtual network run by SIDE. Enter 0 into the “Node number” field and click “Connect”. This will open a UART window for Node 0. Do the same for Node 1 (the simple network described in data1.xml) consists of two nodes. To enter a UART input for a node, type it in the bottom area of the window and hit the “Return” key. When you enter the commands s and then r, the node will start sending its own packets and listening for packets from other nodes. When you do this for both nodes, they will begin exchanging packets.