Installation and quickstart

(or as quick as it gets)

for

MSP430-based boards

December 2011

Preamble

Each of the four Olsonet software packages mentioned in this note: PicOS, SMURPH, 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 research 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 (their official) sites under their respective licenses. Needless to say, all those licenses hold retaining the rights of the respective authors.

For some rather obscure historical reasons, SMURPH is called SIDE these days. This is also the name of the directory into which the package unpacks. Don't let it confuse you: SMURH and SIDE mean the same thing. VUEE is a layer of functions bridging PicOS praxes and the virtual hardware provided by SMURPH/SIDE.

If you are acquiring the packages via GIT, you may want to have a quick look at **`Repo.pdf`** 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 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, in fact may be preferred, so the PC doesn't have to be high end) or **Linux** (say a recent version of Ubuntu). A true parallel port (a USB dongle emulating a parallel port is useless for our purpose) is handy, but not absolutely required. Parallel ports are not easy to come by these days.

A JTAG programmer. Parallel-port JTAG programmers are (arguably) a bit less capricious and easier to set up; however, USB programmers can be made to work as well. In particular, we have used quite extensively these two devices:

> Tiny USB JTAG (from Olimex). This one works under Windows/Cygwin.
>
> MSP-FET430UIF (from Texas Instruments). This one works under Windows/Cygwin as well as Linux.

It is possible that other JTAG programmers will work, too, but I cannot vouch for them. Quite likely, there is a way to make the Tiny USB JTAG work under Linux, but I haven't tried too hard.

Generally, USB JTAG programmers are not significantly faster or friendlier than the parallel-port ones, if you have a true parallel port.[1] They may be more convenient with a modern laptop where a parallel port is usually absent. The recommended parallel port programmer, MSP430-JTAG, is very cheap and also very reliable. It works with Windows (at least up to XP) as well as Linux, as long as you have a true parallel port. In summary:

- If you have a true parallel port and are running Linux or ≤ XP, use MSP430-JTAG.

[1] Parallel port interface hasn't been tested for a long while, so I cannot really vouch for it anymore.

- If you insist on Vista (or Windows 7), use MSP-FET430UIF or Tiny USB JTAG.
- With Linux, you can use MSP430-JTAG, if you have a true parallel port, or MSP-FET430UIF.

A USB to serial dongle, preferably TTL232R3V3 from FTDI (works 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 EMSPCC11 (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 are needed for serial communication (pin 4 = RX and pin 5 = TX). Those pins are directly 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 power supply for the board.

The requisite cables: parallel cable, USB cable, and so on.

Windows

Parallel port programming doesn't work on Vista (I haven't checked it on Windows 7 and probably never will), although USB JTAG programmers (in particular, Tiny USB JTAG) work fine.

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.

If, following the installation, things don't work for you in a weird sort of way (failing forks, crashing programs), try this:[2]

1. Shut down all Cygwin activities, X-server, Cygwin windows.
2. Start the DOS command prompt.
3. Execute in the DOS window **`C:\cygwin\bin\ash.exe`** (which starts a statically linked Cygwin shell).
4. Execute (in that shell): **`/bin/rebaseall`**

Then start Cygwin and see if things aren't better.

Set up mspgcc. You have two choices:

1. Fetch the ready set of binaries from our site (see the REPO link above) and unpack it into **`C:`**. Under a Cygwin terminal window do this:

   ```
   cd /cygdrive/c/
   zcat .../mspgcc... | tar -xv -
   ```

   where the argument of **`zcat`** is the path to the archive you have fetched from the REPO.

[2] I don't think this is needed any more. It used to be a problem a few years ago.

2. Fetch the source set from the REPO and compile it according to the instructions in the enclosed README file. You may want to do that, if the binary package doesn't work for you for whatever reason.

Following the installation (this applies to both cases), make sure that the path to **`mspgcc`** executables is known to Windows (and Cygwin, but 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.

The **`Doc`** subdirectory of **`PicOS`** includes **`mspgcc.pdf`** which is a 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. With PIP, most of those things have been automated, so you need not worry about it right away.

Install Tcl 8.5 for Windows. You may go to http://www.activestate.com and select the respective (recent) version, 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 **`deploy`** (see below) to find. Note that Cygwin comes with its own Tcl (version 8.4 as of now). You will need them both: the Cygwin version for compatibility with Cygwin, the other one for some useful features that are missing in 8.4.

Note that 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, which Cygwin programs may depend on.

Now it is time to unpack the software (see **Preamble**). Having accomplished that, go to directory **`PICOS/Scripts`** and unzip **`UTILS.zip`**. Alternatively, you can retrieve the requisite files from the REPO.

If you want to use MSP-FET430UIF or Tiny USB JTAG for the JTAG programmer, you will need the respective drivers, which can be found in **`UTILS`**. These are the subdirectories:

> **`MSP430-USB-TI-3.0-drivers`** (for MSP-FET430UIF) and
> **`MSP430-JTAG-TINY-1.032-drivers`** (for Tiny USB)

When asked for a driver, point the system to the corresponding directory. If you happened to plug the device and the driver search process has failed already, you can replace the driver via the Device Manager indicating the pertinent directory as its source.

On XP or earlier, you will also have to explicitly install a driver for TTL232R3V3 (the newer systems can manage without your help). You can get one from FTDI's web site, but a driver is also available in the **`UTILS`** directory (and 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 the system prompts you for drivers, direct it to subdirectory **TTL232R-drivers** in **UTILS**. You may be prompted twice which is OK.

Make sure that you have a directory named **bin** or **BIN** in your home directory and that it is mentioned in your **PATH**. Go to directory **PICOS** (i.e., one notch up from where you ended up in the previous step) 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 version of Tcl, which comes with the full Cygwin installation (not the Tcl 8.5 that you installed explicitly a while ago).

A quick-start exercise (without PIP)

Try this:

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 the name of 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 of the same uploadable code. One way of loading the program into the board involves **msp430-gdb** and **msp430-gdbproxy** and is described below. Note that the procedure is greatly simplified with PIP. You may go directly to PIP's documentation if you want to skip this step.

4. Connect the programmer to the JTAG port on the board and to the PC. Make sure that the board is powered on.

5. Open a Cygwin window (it can be an X window).

6. If you are using Tiny USB JTAG (from Olimex), execute (in the Cygwin window):

   ```
   usefetdll olimex
   ```

   If your programmer is MSP-FET430UIF (from Texas Instruments), execute:

   ```
   usefetdll tiusb
   ```

   instead. This makes sure that **msp430-gdbproxy** will be using the right DLLs. In the latter case, the programmer (as it has arrived from the factory) will have to be re-flashed with different firmware before it can be used with **msp430-gdbproxy**. To do that, having connected the programmer to the PC, execute:

```
msp430-gdbproxy.exe msp430 --update-usb-fet TIUSB
```

You will see a series of messages indicating progress. At the end of this operation, the proxy will try to establish connection with the target device, if one happens to be attached to the programmer. **Note:** DON'T DO THIS WITH THE TINY JTAG (I just did – to see what would happen – and now it looks like I have damaged the programmer ;-).

7. If your programmer is set, the board connected and powered up, execute this command:

```
msp430-gdbproxy --port=2000 msp430 TIUSB
```

As far as I can see, the 2000 for the socket communication port is the default, so you may skip it. If this step fails, i.e., the program complains that it cannot talk to the device or just exits, it can mean that 1) the board is not powered on or the connection between the board's JTAG port and the programmer is faulty, or 2) there is something wrong with the drivers or DLLs.

8. Having started the proxy, go back to **`PICOS/Apps/RFPing`** and execute:

```
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 **`msp430-gdb`** will talk to **`msp430-gdbproxy`**. If everything is OK, **`msp430-gdb`** should display this piece of text:

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

before presenting its prompt. Then 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. You should see something similar to this:

```
Loading section .text, size 0x5dbc lma 0x4000
Loading section .data, size 0x28 lma 0x9dbc
Loading section .vectors, size 0x20 lma 0xffe0
Start address 0x4000, load size 24068
```

at the end.

9. 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 gdbproxy 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 **`msp430-gdbproxy`**.

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. The program **`Terminal.exe`** in **`UTILS`** is a terminal emulator which you can use to communicate with the board. This program can be executed directly (it is not installed). Before you invoke the terminal emulator, 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, set the baud rate to 9600, 8 data bits, no parity, 1 stop bit, and no handshake. 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.

If you now reset the board (by switching it off and on, or from gdb – by executing **`monitor reset`** followed by **`continue`**), you should see 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 transmission and reception.

A 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 number and ***`rate`*** is the baud rate, e.g.:

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

Note that the script accepts arbitrary COM numbers (unlike **`terminal`**). It will echo all lines received from the UART to the screen and send all lines entered from the keyboard to the UART.

When you call piter without arguments, it will open a GUI window. You will have to decide on the serial device, the rate, and the communication mode (use *Direct* protocol and make sure the *Bin* box is not checked). Note that piter is described in the **`Serial.pdf`** document.

For simple programming over parallel port (without involving gdb) you can use **`MspFet.exe`** from **`UTILS/MSPFET-parallel-port/`**. This program is executable directly (it is not installed), but it needs the files in its directory, so if you want to move it somewhere, you have to move the whole set. The way you use it is that you first "Open" the file to load (this must be the **`.a43`** variant, i.e., **`Image.a43`** in our case), then click "Erase" and "Program".

If you want to use MSP-FET430UIF for the programmer, you can install FET-Pro430-Lite, which you will also find in **`UTILS`** (and in the REPO). The same program can be downloaded from http://www.elprotronic.com/. Remember to install the programmer's driver from **`UTILS/MSP430-USB-TI-3.0-drivers`** before trying the loader. If you have used the programmer with gdbproxy (and replaced the standard firmware), FET-Pro430-Lite will complain about "old" firmware. You can ignore those complaints (it does work fine), or you can replace the firmware as prompted by the program (but you will have to reprogram the device again when switching to gdbproxy).

Linux

I have tried this on Ubuntu compiling mspgcc from sources (you will find them in the REPO). As the installation creates a self-contained subdirectory of **`/opt/`**, the outcome is likely to be relatively indifferent to the system version. Consequently, instead of doing it all by yourself, you can use our pre-compiled binaries (also to be found in the REPO), which should be good for any x86-compatible and reasonably recent installation of Ubuntu.

Become root and **`cd`** to **`/`**. Then **`unzip`** and **`untar`** the file. This will create **`/opt/mspgcc/`** with all its content that you would normally obtain after compiling everything from source. To complete the installation, add to **`/etc/ld.so.conf.d/`** a file, e.g., **`mspgcc.conf`** with a single line looking like this:

```
/opt/mspgcc/lib
```

and execute

```
/sbin/ldconfig
```

to make the libraries needed by **`gdbproxy`** globally visible.

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 ttyUSB0 9600
```

are equivalent. The GUI version of piter works similar to Windows, except that the device names are different.

Now you can go through the **<u>quick-start exercise</u>** described in the Windows section.

<u>Using MSP-FET430UIF</u>

This worked for me on a recent Ubuntu system. Seems a bit flaky, but I have managed to replicate it on several slightly different Ubuntu systems.

First, you have to reload firmware into the programmer. For that, just plug it into a USB slot. The device will show up in **`/dev`** as **`ttyUSBx`**. You have to make sure that its name is **`/dev/ttyUSB0`** (device number zero), so if you have any other devices mapped into this class, unplug them first. Then execute:

```
msp430-gdbproxy msp430 --update-usb-fet TIUSB
```

and wait until done.

From that point on, you will be able to run the device as any **`ttyUSB`**, not necessarily number zero. Here is how you tell **`gdbproxy`** to talk to it:

```
msp430-gdbproxy msp430 /dev/ttyUSBx
```

where the second argument identifies the device into which the programmer has been mapped (you can look into **syslog** if in doubt).

SMURPH/SIDE + VUEE

SMURPH/SIDE comes with extensive documentation. **SIDE/MANUAL/manual.pdf** contains the a (reference) manual of the present version, which has been kept up to date. **SIDE/MANUAL/BOOK/book.pdf** is the image of an old book, which is 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:

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 praxis virtually under SIDE:

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

In a separate window execute:

```
udaemon
```

Note that the **udaemon** script is put into your personal **bin** (or **BIN**) directory by **deploy**. This should open a **Tk** (**wish**) window providing a rudimentary interface to the virtual network run by SIDE. Enter 0 into the “Node Id” 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 immobile 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, you will see them exchange packets.