



Installation and quickstart

for

MSP430-based boards



Preamble

Each of the three Olsonet software packages mentioned in this note: PicOS, SMURPH, and VUEE, comes with documentation. The quality of that documentation varies. For example, SMURPH/SIDE has been documented quite extensively, while PicOS and VUEE (which have been evolving rapidly, especially in the area of drivers for diverse hardware interfaced to the system) leave quite a bit to be desired. 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 academic papers available from the Olsonet's web site <http://www.olsonet.com>.

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. Let it not 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.

The three packages come as a single `tar`'red and `gzip`'ped archive, which unpacks into three separate directories: `PICOS`, `SIDE`, and `VUEE`. 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** (2000K 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 recommended, but not absolutely required. Parallel ports are not easy to come by these days.

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

[Tiny USB JTAG](#) (from Olimex). This one works under Windows/Cygwin.

[TI MSP-FET430UIF](#) (from Texas Instruments). This one works under Windows/Cygwin as well as Linux.

It is quite possible that other JTAG programmers will work, too, but we cannot vouch for them. Quite likely, there is a way to make the Tiny USB JTAG work under Linux, but we 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. They may be more convenient with a modern laptop where a parallel port is usually absent. The recommended parallel port programmer, [MSP430-JTAG](#), is cheap and 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 not running Vista (I have no idea about Windows 7), use [MSP430-JTAG](#). This also applies to Linux.
- If you insist on Vista (or Windows 7), use [Tiny USB JTAG](#).



- With Linux, you can use [MSP430-JTAG](#), if you have a true parallel port, or [TI MSP-FET430UIF](#).

A USB to serial dongle, preferably [TTL232R3V3](#) from [FTDI](#) (works under Windows and Linux with no problems). It can be purchased at a number of places, including the [manufacturer](#). 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 (4 – RX and 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

Windows XP (or 2000) is preferred. Parallel port programming doesn't work on Vista, 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 run it as administrator. Select the recommended settings, i.e., “For all users” and “Unix/binary” text file type. 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.

Download mspgcc from http://sourceforge.net/project/showfiles.php?group_id=42303 (the mspgcc-win32 package is all you need). Last time I looked there, the version was 20081230). That version will work with all CPU models, except for the CC430 family (because of a bug in integer multiplication/division). The mspgcc4 package (to be found at <http://mspgcc4.sourceforge.net/>) is better, but its installation is a bit more cumbersome. We will shortly try to prepare a more convenient self-installing version and make it available from our site.

In directory `C:\mspgcc\docs\` (assuming the default installation), you will find the manual by Steve Underwood (file **mspgcc-manual.pdf**). Glance through chapters 8 and 10. Especially, chapter 10 explains how to use **gdb** with MSP430 via **gdbproxy**, which will be useful for debugging. Needless to say, it makes sense to become familiar with the general set of features of **gdb** (there is no shortage of documentation on the Network).

Install Tcl 8.5 for Windows from <http://www.activestate.com>. Let it go into 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). You will need them both: the Cygwin version for compatibility with Cygwin, the other for some useful features that are missing in 8.4.

Now it is time to unpack the software (see **Preamble**). Having accomplished that, go to directory **PICOS/Scripts** and unzip **UTILS.zip**.

Install the drivers for TTL232R3V3. You can get them from FTDI's web site, but they are also available in the unzipped **UTILS** directory. 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 – this is normal.



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 **Tcl**, which comes with the full **Cygwin** installation.

A quick-start exercise

If you want to use the parallel port for programming, you are now ready for a quick-start exercise. Do this:

1. Move to directory **PICOS/Apps/RFPing**, which contains a simple praxis testing RF communication between a pair 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**. Directory contains a 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 **gdb** and **gdbproxy** and it works as follows:

1. Connect the **MSP430-JTAG** to the **JTAG** port on the board and to the parallel port on your computer. Make sure that the board is powered on.
2. Open a **Cygwin** window (it can be an **X** window) and execute in it this command:

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

If this step fails, i.e., the program complains that it cannot talk to the device or 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 your parallel port setting. In the latter case, you may try different **BIOS** settings for the port.

3. In **PICOS/Apps/RFPing**, 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 **gdb** (depending on whether you are on **Windows** or on **Linux**) and, in particular, identifies the (socket) port over



which **gdb** will talk to **gdbproxy**. If everything is OK, **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.

4. Type:

```
monitor reset
continue
```

to run the program. These basic steps are described in chapter 10 of the manual that comes with **mspgcc**.

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 **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 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 this way:

```
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. It will echo all lines received from the UART to the screen and send all lines entered from the keyboard to the UART.

If you want to use the Tiny JTAG (USB) programmer, you will have to install the drivers, which can be found in **UTILS/MSP430-JTAG-TINY-1032-drivers/Drivers**. The installation is straightforward: when you plug in the programmer and the system prompts you for drivers, point it to that directory. Following this step, go to directory **UTILS/MSP430-JTAG-TINY-1032-drivers/DLLs** and move all files (DLLs) from there to **mspgcc/bin** (the directory where **msp430-gdbproxy.exe** is located). Note that two of those DLLs, **HIL.dll** and **MSP430.dll**, replace existing files, which you may want to preserve, just in case. This will not affect **gdbproxy**'s compatibility with the parallel port JTAG.

Here is how to invoke **gdbproxy** with Tiny JTAG:

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

If **gdbproxy** signals problems, try replacing the firmware on the Tiny JTAG this way:

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

Once you get past this step, the rest looks pretty much the same as for MSP430-JTAG over the parallel port.

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**. The same program can be downloaded from <http://www.elprotronic.com/>. You will have to install the drivers from **MSP430-USB-TI-3.0-drivers**.

Linux

We have tried this on Ubuntu compiling everything from sources. 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 (see **The easy way** below), which should be good for any x86-compatible and reasonably recent installation of Ubuntu.



Note that by default Ubuntu installs very few packages, and many basic tools are likely to be missing in a default installation. You need at least Tcl (don't forget about Tk) and **make** to even start thinking about playing with PicOS. Luckily enough, Ubuntu makes it easy to see what is missing and lets you acquire the needed packages as they become necessary.

If you want to do everything by yourself, go to <http://mspgcc.sourceforge.net/> and read the instructions. Here is a shortcut:

Fetch the most recent version of mspgcc via CVS this way:

```
cvs -d:pserver:anonymous@mspgcc.cvs.sourceforge.net:/cvsroot/mspgcc login
cvs -z3 -d:pserver:anonymous@mspgcc.cvs.sourceforge.net:/cvsroot/mspgcc co -P .
```

This will create a subdirectory named **mspgcc**, and, inside it, another subdirectory named **sf**, where the requisite sources will be fetched. The operation may take a while depending on the speed of your Internet connection.

Go to **mspgcc/sf/packaging/** and carefully read the instructions in file **README-MAINTAINER.txt**. Then follow them. This way you will be able to install everything except **gdbproxy**, including a GUI front to **gdb** called **insight**.

Note: the instructions say that you "may" need gcc-3.4 (or older) to compile the stuff. In fact, you DO need gcc-3.4, so install it as a package (it is named **gcc-3.4**). It will not mess up your newest gcc installation, but you will be able to use it as an alternative compiler. Then, at the respective stage of the procedure, type:

```
CC=gcc-3.4 make build
```

instead of

```
make build
```

About gdbproxy

Although the package fetched via CVS does contain **gdbproxy**, that version is generic and not very useful without some serious work. Instead, you may fetch the ready executable from <http://mspgcc.sourceforge.net/> (the download area), specifically these files:

```
msp430-gdbproxy
libMSP430.so
libHIL.so
```

It makes sense to put **msp430-gdbproxy** into **/opt/mspgcc/bin**, and the two libraries into **/opt/mspgcc/lib**. Then add to **/etc/ld.so.conf.d/** a file, e.g., named **mspgcc.conf** with a single line looking like this:

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

and execute

```
/sbin/ldconfig
```

Then, the **gdbproxy/gdb** setup will work the same way as for Windows, at least for MSP430-JTAG across the parallel port. Needless to say, you should make sure that the permissions of the parallel port device (typically **/dev/parport0**) allow it to be accessed by **gdbproxy** (if you invoke it as a regular user).



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 `tttyUSB0`. 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 tttyUSB0 9600
```

are equivalent.

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

The easy way

Download this file:

http://www.olsonet.com/REPO/100402_mspgcc_lin.tgz

This is a precompiled (by us) and complete mspgcc package that unpacks into `/opt/mspgcc`. 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. As a bonus, we include `gdbproxy`. (`/opt/mspgcc/bin` includes `msp430-gdbproxy`). 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.

Using MSP-FET430UIF

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 `tttyUSBx`. You have to make sure that its name is `/dev/tttyUSB0` (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 `tttyUSB`, not necessarily number zero. Here is how you tell `gdbproxy` to talk to it:

```
msp430-gdbproxy msp430 /dev/tttyUSBx
```

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 not-so-friendly (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, but considerably outdated (in particular, it knows nothing about PicOS or VUEE).

Note that `deploy` has set up SIDE and VUEE to work with PICOS. You can try this exercise now:

Go to `PICOS/Apps/VUEE/RFPing`. 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 data.txt
```

In a separate window, `cd` to `VUEE/UDAEMON` and execute:

```
./udaemon
```

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 `data.txt`) 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.

