PHYS 319 Programming and Debugging with the MSP430 for Mac OS X

This is a summary of how to get started writing assembly and C programs for the MSP430 Launchpad that we will be using in Phys 319. There are separate Windows and Linux versions of this document.

We will need several components: an assembler, a program to load our assembled programs onto the MSP430, a C compiler, and a debugger. There are a number of options available for several of these components. For the first couple of weeks we only need the assembler and loader, so we'll deal with those first. For the third and fourth weeks we need a C compiler and debugger, and for the following two weeks, we'll use some other software on the host computer to talk to the Launchpad via its serial port.

Installing and using an Assembler (Needed for Lab 2)
There are a number of assemblers available for the MSP430 line. To keep things simple and the same for everyone, we're going to use a small, bare-bones assembler.

The default install of OS X is missing some important things that you will need to do useful work on a Mac. You will want to become acquainted with the terminal window: in the finder, head to Applications/Utilities/Terminal, or just ask spotlight for terminal.

We're going to install a package manager called MacPorts. This will allow easy access to a huge variety of useful software to you. There is a comprehensive guide to installing MacPorts at:
http://guide.macports.org/
but don't worry, you don't have to read the whole thing. There are other, similar package managers: eg fink and brew. If you use fink or brew already, you can probably do everything you need with them rather than MacPorts.

1) Macports section 2.1: Install Xcode. Follow the instructions in the guide for your version of OS X. For some versions of OS X, you may need to register as an Apple Developer to get access to the downloads at the Apple Developer website. Be very careful to get the correct version of Xcode and enable the correct package options as specified.

2) Macports section 2.2: Install the MacPorts package. Either use the direct downloads in section 2.2.1, or follow the link the GitHub to find the package for your version of OS X.

3) Next, we'll use MacPorts to install mspdebug. In a terminal window, type:

```
sudo port install mspdebug
```

4) Now download the assembler:
Download the .tar.gz for MacOSX from (don't worry if your MacOSX is newer than 10.6) :
http://www.mikekohn.net/micro/naken430asm_msp430_assembler.php

At a command line, uncompress the source code with:
`tar -zxvf ~/Downloads/naken430asm-2011-10-30-macosx10.6-x86_64.tar.gz`
move into the source code directory:
`cd naken430asm-2011-10-30`
run the configure script:
`./configure`

build the assembler:
```
make
```
install it:
```
sudo make install
```

5) One last thing: we need a driver for the Launchpad. TI supplies one for Mac, but apparently it's broken. There is a working driver at: http://energia.nu/files/MSP430LPCDC-1.0.3b-Signed.zip. Unzip the file to get a .pkg file that you can install from the finder.

To use the assembler:
```
naken430asm -o <name>.hex <name>.asm
```
(where <name> is the name of your program)

Grab: http://phas.ubc.ca/~michal/phys319/blink.asm
and ensure that you can assemble it without errors.

Then to load it into the Launchpad (this won't work unless you have a Launchpad board connected):
```
mspdebug rf2500
prog <name>.hex
^D
```

^D is CTRL-D. rf2500 is the name of the protocol used to talk to the Launchpad.

Installing a C compiler and debugger (Needed for Lab 3)

For labs after the first two weeks, we'll need a C compiler, and a debugger will likely be useful.
TI provides a feature-rich integrated development environment (IDE) called Code-Composer Studio (CCS). There is a Mac version of CCS, but it doesn't support our Launchpad so we'll use gcc/gdb/mspdebug.

Download the osx-install version of msp430-gcc at:
http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSPGCC/latest/index_FDS.html
(you'll need to register with TI and declare that you won't export the software to any hostile states).
Unzip it to get the app and then install from the finder.
When installing, let it install into the default folder ($HOME/ti).

Finally, in your home directory edit your profile to add the compiler to your path:
```
nano ~/.bash_profile
```
There will be a line near the bottom that starts with:
export PATH=

Just before the final quotation, add:
```
:$HOME/ti/msp430_gcc/bin
```

Mine looks like:
```
export PATH="/opt/local/bin:/opt/local/sbin:$PATH:$HOME/ti/msp430_gcc/bin"
```
You'll need to open a new terminal for this to take effect.

Using the Compiler:

You will have the best experience if you put your program into its own directory, along with a Makefile that automates some of the building.

Grab: http://www.phas.ubc.ca/~michal/phys319/cblink.tar.gz

uncompress it:
```
tar -zxvf cblink.tar.gz
```
move into the directory:
```
cd cblink
```
compile the source:
```
make
```
load it into the Launchpad:
```
mspdebug rf2500
prog main.elf
^D
```

Using the debugger [not needed immediately, for debugging programs later]

You need to ensure that your program is compiled with a `-g` option (the supplied Makefiles do this) so that the executable files contain information allowing the debugger to know which instructions came from which source code lines, and where variables reside in memory. Debugging is complicated by compiler optimizations, so you should check to make sure that there is no -O option (-Os, -O2 etc) in the compilation command.

Now, load your program to be debugged into the Launchpad with mspdebug:
```
mspdebug rf2500
prog main.elf
```
but now, instead of quitting mspdebug, start gdb:
```
gdb
```

mspdebug is now in a mode where it is awaiting debugger commands from another program. Open a second terminal window, and type:
```
msp430-elf-gdb main.elf
```

which starts the debugger and tells it which program you want to debug. This program must be identical to the one downloaded with mspdebug! Now tell gdb that the program is running on a remote target, accessible on port 2000:
```
target remote :2000
```

Now you can use gdb commands to execute the program and examine variables as they run. You can find lots of documentation on gdb on the web, but some useful things to try are:

```
break <line>
```
sets a breakpoint at line number `<line>`.

```
c
```
'continue' until a breakpoint is hit. This will let the program execute until the breakpoint you just set.

You can only set two different breakpoints. You can see which breakpoints are set with
`info break`
and remove a breakpoint with:
`clear <line>`

`print a` shows you the value of the variable a.
`list` will show you a few lines of source code just ahead of the current position
`info reg` will show you all the registers.
`condition 1 i == 3` sets a condition on breakpoint 1, if i is not 3, it will continue (note that this is very slow, so if you're waiting for i to get to 5000, be prepared to wait a long time.
`monitor reset` resets the program on the Launchpad to start over. Anything after `monitor` is assumed to be a raw mspdebug command.
`load prog.elf` will load the program from prog.elf into the msp430 (same as the prog command entered directly into mspdebug).

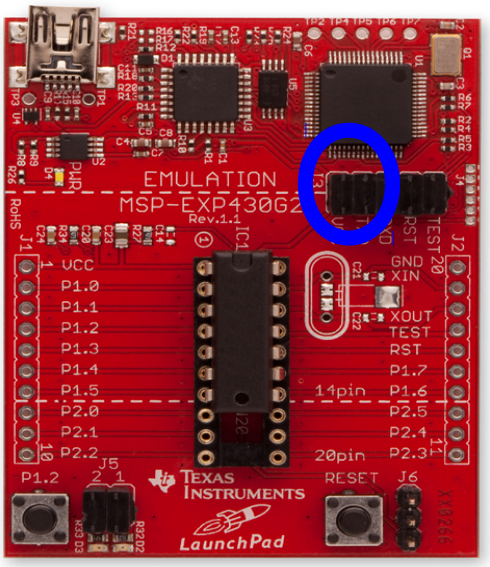
Serial Port Example (For lab 5).

Our next example is to set up a program that talks to the MSP430 while it is running. The Launchpad board presents a USB-Serial interface to the host computer, so that we can talk to the MSP430 as though it were connected by an old-fashioned serial port.

We will do this with a program in python. The python program uses a graphical user interface called gtk to draw windows on the screen, and a plotting library called matplotlib to make graphs. We will need to install these components.

```
sudo port install xorg-server py27-serial py27-pygtk py27-numpy
sudo port install py27-matplotlib +gtk2
```

You must log out and log in from your computer (or reboot).



Now: we have two versions of the Launchpad board floating around a demo program. One version of the program will work on either versio complicated). The simpler version of the program will only work on t board. Have a look at your board. Underneath the words “Emulation” should be either Rev. 1.4 or Rev. 1.5. If your board says Rev. 1.5, then
a) Download
http://www.phas.ubc.ca/~michal/phys319/temperature_demo4.tar.gz
b) Ensure that the two left-most jumpers (circled in the photo) are oriented horizontally (opposite to all the other jumpers).

If your board says Rev. 1.4 (or earlier), then
a) Download
http://www.phas.ubc.ca/~michal/phys319/temperature_demo3.tar.gz
b) Ensure that all the jumpers are oriented vertically.

Build the program in main.c and flash into the Launchpad.

You will need to edit the python program to contain the correct

serial port name.

On Mac it is something like /dev/tty.uart-37FF41CE96132B33 where that long hexadecimal string is the serial number of your board. Do `ls /dev/tty.uart*` to find it.

Then you can start the python program (you can double click on it, or start it from the command line with `python2.7 python-serial-plot.py`). If that doesn't work and you have a previous installation of python2.7 from another course, you may need to be more specific, something like:
`/opt/local/bin/python2.7 python-serial-plot.py`
After it has started, push button s2 on the Launchpad, and temperature measurements will be delivered from the Launchpad to the python program and plotted.

If you don't see data coming onto the plot immediately after pushing the button, check that the jumpers circled in the image are oriented correctly for the program version you're using.