PHYS 319 Guide to the HEBB 42 computers

Logging in

Log in with your PHAS account id. If you don't have one, go to Henn 205 to sign up. Once logged in, your home directory will be your PHAS home directory, so that if you log in from one of the machines in Henn 205, all your files will be accessible.

Getting started with the Assembler

All of the software tools you will need are already installed. Plug in your Launchpad board.

Code for the examples in the lab activities can be found under /usr/local/share/P319/Examples
You will need to copy the files you want from here to your own directory to use them, as they are 'read-only'.

You'll need a terminal window. Choose Applications->System Tools->Terminal

In the terminal try:

Create a directory for your assembly code:
```
mkdir 430ASM
```

Change into it:
```
cd 430ASM
```

Copy the example assembly code:
```
cp /usr/local/share/P319/Examples/blink.asm .
```

Run the assembler:
```
naken430asm blink.asm -o blink.hex
```

Start mspdebug to load the code into the board:
```
mspdebug rf2500
```

Download it:
```
prog blink.hex
```

Quit mspdebug:
CTRL-D

To work on your own programs, you will need to use a text editor. If you are familiar with emacs or vi, they are available. If you don't know either of those, I would recommend gedit. Try:
```
cp blink.asm myblink.asm
gedit myblink.asm &
```

Programming in C
First, make sure the compiler and other tools are in your path. Edit your profile:
```
nano ~/.profile
```

at the bottom of the file add:
```
export "PATH=$PATH:/opt/msp430_gcc/bin"
```
You'll need to open a new terminal for this to take effect.

Copy the example program:
```
cp -r /usr/local/share/P319/Exanples/cblink .
```

Change into the new directory:
```
cd cblink
```

Build the program:
```
make
```

Start mspdebug:
```
mspdebug rf2500
```

Download it into your board:
```
prog main.elf
```
Quit out of mspdebug:
CTRL-D

Debugging from the command line [not needed immediately, for debugging programs later]

To run the debugger, compile and load programs as above (you need a `-g -fomit-frame-pointer` on the compilation command line, but the makefile we supply should have that). Don't quit out of mspdebug after downloading your program, instead, type:
```
gdb
```

which puts mspdebug in a mode where it is awaiting debugger commands from another program. Now, you'll need a second terminal window (right-click the icon on the dock and choose 'New Terminal.'
In the second terminal, get into the same directory as your source code, and then:
```
msp430-elf-gdb main.elf
```

which tells the debugger 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
```

If all is well, you can now 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>
```

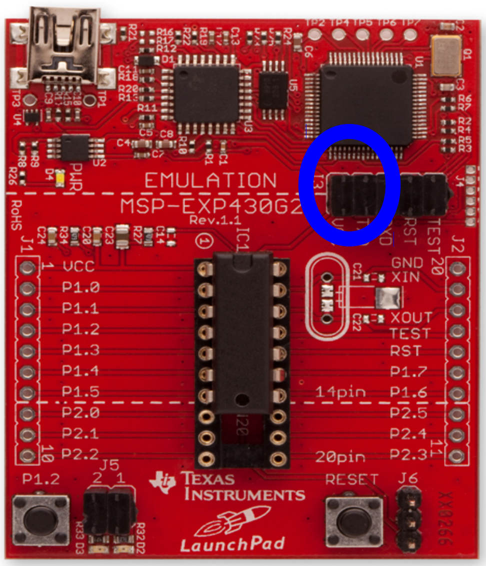
`print a` shows you the value of the variable a.
`list` will show you a few lines of source code just ahead of the current position
`info reg` will show you all the registers.
`condition 1 i == 3` sets a condition on breakpoint 1, if i is not 3, it will continue (note that this is very slow, so if you're 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 the serial port example in weeks 5-6 we provide a demo program to demonstrate how to use serial communications between the Launchpad and a program running on the host computer.

We have two versions of the Launchpad board floating around and there are two versions of our demo program. One version of the program will work on either version of the board (but is more complicated). The simpler version of the program will only work on the more recent version of the board. Have a look at your board. Underneath the words "Emulation" and "MSP-EXP430G2" there should be either Rev. 1.4 or Rev. 1.5. If your board says Rev. 1.5, then:
a) `cp -r /usr/local/share/P319/Examples/temperature_demo4 .`
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) `cp -r /usr/local/share/P319/Examples/temperature_demo3 .`
b) Ensure that all the jumpers are oriented vertically.

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



```
cd temperature_demo3
make
mspdebug rf2500
prog main.elf
CTRL-D
python2 python-serial-plot.py
```
(then press the button on the launchpad board)

You actually can use the debugger and have the python program reading in temperatures at the same time.

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.