PHYS 319 Programming and Debugging with the MSP430 for Windows

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. The Launchpad is supported by Texas Instruments primarily on Windows, but their support for the Launchpad we are using has not kept up with recent versions. Using this board with Windows 8-10 on a modern computer presents some difficulties. In particular, communications with programs running on the Launchpad board over the serial interface (which we will need for labs 5 & 6) do not work reliably.

You have a choice of three options to proceed:

1) You can try using Windows anyway. If you have an older computer with USB 2 ports and Windows 7, this will probably work fine. Newer computers with only USB 3 ports will **probably** encounter serious difficulties in the weeks 5 & 6 mini-project. If you choose to proceed with Windows but have trouble with the serial communications, you can fall back to one of the other options.
2) You can use the lab computers. The lab computers have Linux installed and are known to work with the Launchpad boards. The downsides of this option are that the lab computers are a) old, b) slow, and c) not available when you're not in the lab.
3) We have prepared a Linux operating system image that you can write to a USB flash drive. You can then boot your computer from this USB flash drive and use the same software as we have installed on the lab computers. This will not harm the software on your computer. When you remove the flash drive and reboot, your computer will be just as it was before. With a modern flash drive, this option is actually very attractive, and is recommended.

Instructions for setting up Option 3 follow, with instructions for Option 1 afterwards. Instructions for Option 2 are in a separate document.

Setting up a USB flash drive [Option 3]

1) Buy a decent USB flash drive. Really, spend ~$20 and get a USB 3.0 drive with a read speed greater than 100 MB/s. Write speed is less important. It doesn't need to be huge (though larger drives tend to be faster). 8 GB is plenty big, but you probably want a 16 or 32 GB drive for the increased speed. If the packaging doesn't specify the read speed, keep looking. London Drugs has a pretty good selection. A decent flash drive on a USB 3 port runs nearly as fast as an installed OS. A cheap USB 2 drive will feel like molasses.

2) Download our OS image from:
http://www.phas.ubc.ca/~kotlicki/Physics_319/ubuntu-mate-p319.iso

3) Download and install unetbootin from https://unetbootin.github.io/

4) plug in your USB stick. Ensure there is nothing on it that you want to keep, as we'll overwrite the entire stick.

5) Start up unetbootin. Select Diskimage, ISO, and navigate to the iso image your downloaded in step 2).
IMPORTANT: enter 4000 in the field: "Space used to preserve files across reboots)." Without this, nothing you do will ever get saved.
Then, select the USB stick as the target drive (make sure you select the correct drive!), and hit OK.

6) Wait till its done. It might take a while.

7) Test the USB stick: Reboot your computer. There is a slim chance that it will boot from the USB stick immediately. Probably it won't, and you will probably need to tell the BIOS setup program to boot from the USB stick. Reboot again, and press F1 or F2 or F8 or F10 or F12 (which one varies from computer to computer) to get into the BIOS setup program. You'll need to dig around the options to find something like "Boot Order." Set the USB stick so it is first. Then exit the BIOS setup, making sure to Save Changes. Now the computer should boot from the USB stick into Linux. The username and password are both phys319.

On some computers, the BIOS will reset the boot order after the USB stick is removed. If your computer doesn't boot from the USB stick the next time you insert it, you may need to repeat the above steps to put the USB stick as the highest boot priority each time the USB stick is used.

Once booted from the USB stick, no software installation should be necessary and you can follow the "Using the Lab Computers" document (everything except the Logging In section).

The os will probably prompt you at some point to install updates. Don't do it! It will eat up the persistent storage on the flash drive.

Installing Linux on your own computer [Not recommended]
It is possible to install Linux onto your computer from the USB flash drive, but this is outside the scope of these instructions and carries some risk of loss of data. If you do attempt this, following instructions from the web on resizing your windows partition, one quirk with the image we created is that the installer will fail unless it is started with root permissions. So to do an install from the flash drive you created, boot it with the 'live session' option, then from a terminal window run: 'sudo ubiquity' from a terminal window to start the installer.

Setting up for MSP430 on Windows [Option 1]

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.

We strongly recommend using a browser other than a Microsoft browser for these steps. Use Firefox or Chrome.

Installing and using an Assembler (Needed for Labs 1+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.

1) Download the .zip file with the assembler from:
http://www.mikekohn.net/micro/naken430asm_msp430_assembler.php
For now: Download the package and unzip it.

Grab:https://www.phas.ubc.ca/~kotlicki/Physics_319/blink.asm
and ensure that you can assemble it without errors.

To use the assembler:
Move your program to the directory where the assembler was unzipped, then get a command window,

```
naken430asm.exe -I .\include\ -o <name>.hex <name>.asm
```
(where <name> is the name of your program)

2) To load programs into the MSP430 on Windows, we'll use a utility provided by TI, MSP430Flasher. Download it from:
http://processors.wiki.ti.com/index.php/MSP430_Flasher_-_Command_Line_Programmer

TI will force you to register at their site and make you promise that you won't export their code to Cuba before allowing the download. Execute the .exe installer. By default the flasher will install into `\ti\MSP430Flasher_1.3.7`
The installer will prompt you to get USB FET drivers. You'll need them, but don't follow the link for now. It takes you to a page that is somewhat confusing.

Copy two files: MSP430FLASHER.EXE and MSP430.dll from the install directory into your assembler directory so you can do all your assembly programming and flashing from that one directory.

You will need a driver to talk to the serial port on the Launchpad. Download and install EZ430-UART.zip from:
https://github.com/energia/Energia/raw/gh-pages/files/EZ430-UART.zip
Unzip it, and run DPInst.exe or DPInst64.exe (depending on whether your windows version is 32 or 64 bits – probably 64 for a recent computer).

You won't be able to test downloading programs without a Launchpad, but to use it, from the command line type:
```
MSP430Flasher.exe -n MSP430g2xx3 -w <name>.hex -v -g -z [VCC]
```
You can look at the MSP430_Flasher web page to see what all the options do.

Sometimes the EZ430-UART driver installation doesn't 'take.' After you plug in the Launchpad, it should appear as a serial port in the windows device manager. If the port appears to have a problem, open the Properties, and Update Driver. Then point to the .inf file that came in the EZ430-UART.zip download.

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). We'll be using an open-source compiler, gcc.

1) First, download the compiler suite itself. Download the windows installer package from:
http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSPGCC/latest/index_FDS.html
Please make sure to install version MSP430-GCC_4_02_00_00 (you'll need to register with TI and declare that you won't export the software to any hostile states). Run the installer – you may need to 'let it run anyway' if windows complains that the program is unknown.

When installing, let it install into the default folder (C:\ti).
2) Next, you'll want a utility called 'make' that automates compiling programs. Get the “Complete package, except sources” setup file from:
http://gnuwin32.sourceforge.net/packages/make.htm
and run the installer.

3) The next tool we'll need is called mspdebug, it acts as a bridge between the debugger and the microcontroller board (there is another program to do this job supplied along with the compiler, but it doesn't work well).
Visit:
http://www.cybercircuits.co.nz/cgi-bin/downloads/index.cgi?ID=20
and get version 0.25. Unzip it into its own directory.

4) To be able to use these tools we want to set up the windows path so that they can be called easily. Right-click on My Computer (or “This PC”) then under Computer name, domain and workgroup settings (or SYSTEM Properties), click 'Change Settings.' On the Advanced tab, click Environment Variables, then edit Path. You want to add the paths to the compiler and make tools installed above to the Path variable. I added the following to my path:
;c:\ti\msp430_gcc\bin
;c:\Program Files (x86)\GnuWin32\bin
;c:\users\michal\mspdebug\
(all on one long line)

Open a new Command Prompt window and try:
```
msp430-elf-gcc -v
make -v
mspdebug --help
```
In each case the programs should give you some version information. If any of those give you something like: “'make' is not recognized as an internal or external command...” then you haven't set up the Path properly.

5) There's one last thing we need to do, and that is to tell windows which driver it should use when programming and debugging the board. Visit:
http://zadig.akeo.ie/
and download Zadig for Windows Vista or later. No need to install it, just run it from where it gets saved.
- Make sure your Launchpad board is plugged in, and that no software is accessing it.
- Then under options, select “List All Devices.”
- In the drop down menu, select “MSP430 Debug-interface (Interface 1)”
- For the driver, use arrow buttons to select libusb-win32
- Double check that you have the last two settings correct!
- Press the large “Install Driver” or “Replace Driver” button, and wait till it completes.

After doing this, you will no longer be able to talk to the board with MSP430Flasher (or with Code Composer Studio), unless you undo this. To **undo** it, go into the device manager, expand the libusb devices item and select the msp430 debug interface. Right click and go to properties. On the driver tab, select uninstall, and ensure the “Delete the driver software for this device” box is ticked, then proceed. Then unplug/replug the device, and it should be back to normal.

This driver replacement will be effective for only this one board. If you swap boards at some point, you'll need to redo these steps.

Using the Compiler:

1) Download https://www.phas.ubc.ca/~kotlicki/Physics_319/cblink.zip
2) Extract the files from the zip archive.
3) In a command prompt window, navigate (cd) into the created directory.
4) `make` should compile the program. If all is well it should just show you two lines of commands that do the work. Anything that looks like an error indicates a problem.
5) Type:
   `mspdebug rf2500`
   In mspdebug, type:
   `prog main.elf`
   `^Z`
   which will load your program onto the board, and quit out of mspdebug.
   (`^Z` is Control-Z)

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.

This is where people with modern windows computers (with only USB 3 ports) are likely to run in to trouble. If you want, you can give it a go, but the chances of this serial connection working reliably under windows are pretty slim. It may work for a while and then stop. If you want to give this a try the instructions are:

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.

Components needed are: python, numpy, matplotlib, pyserial and pygtk

You may already have a python distribution installed from a previous course. If so, you may be able to use the python you already have. It will need to be a 2.x 32-bit version. If you already have a python distribution, you probably already have Numpy, and matplotlib, if not, you'll need to add them. You probably won't already have pyserial and pygtk – and you will need these as well.

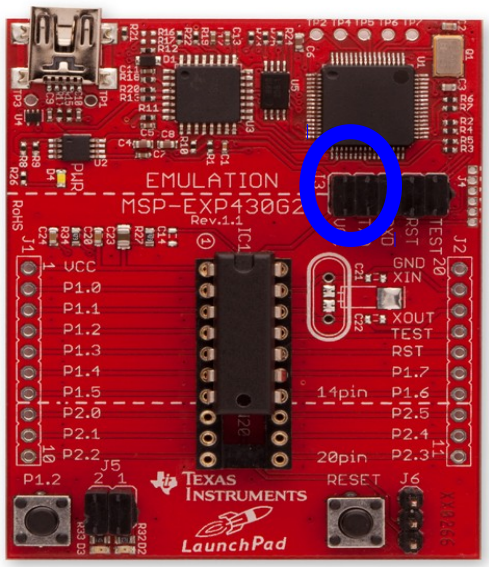
Assuming you don't already have python installed:
Visit:
https://www.python.org/downloads/release/python-2713/
and download and install the Windows x86 (32bit) MSI installer. Don't get the 64bit version even if you have a 64bit version of windows. Some of the libraries we'll need aren't well supported with 64bit python.

Then:
http://sourceforge.net/projects/numpy/files/NumPy/1.10.2/
and download the numpy-1.10.2-superpack-python2.7.exe
Then:
https://github.com/matplotlib/matplotlib/downloads/
and download matplotlib-1.2.0win32-py2.7.exe

Then:
https://sourceforge.net/projects/pyserial/files/pyserial/2.7/
and download pyserial-2.7.win32.exe

And finally:
http://ftp.gnome.org/pub/GNOME/binaries/win32/pygtk/2.24/
and get pygtk-all-in-one-2.24.2.win32-py2.7.msi

Now: 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 ver
complicated). The simpler version of the program will only work o
board. Have a look at your board. Underneath the words "Emulatio
should be either Rev. 1.4 or Rev. 1.5. If your board says Rev. 1.5, t
a) Download
http://www.phas.ubc.ca/~michal/phys319/temperature_demo4.zip
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.zip
b) Ensure that all the jumpers are oriented vertically.

Unzip the demo program you downloaded.

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 Windows this might be COM3 or COM5. But you should be able to confirm which port it is in the ports item of the device manager.

Then you can start the python program - you can double click on it

If anything hasn't gone properly in your python installation, double clicking on the program won't give you much information. Your best bet is to open a 'python command window' (from the start menu) and start it with: `python 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.