USING THE MSP430F5529

There may be a few people who's projects won't be feasible with MSP430G2553, either because there aren't enough IO pins, not enough RAM, not enough timers, etc. For those people, we do have another board available, the MSP430F5529 Launchpad. The 'F5529 is a much newer and more capable chip than the 'G2553. It has more flash, more ram, more timers, more GPIO pins, and a number of other significant enhancements. It also has one significant drawback: it is a surface mount device that is soldered directly onto the board. That means that it cannot be easily replaced, so if the chip itself is damaged, the entire board needs to be replaced. Please be very careful in connecting this board to other devices: connect input pins through resistors, and make sure to ground yourself before touching the board or other circuitry connected to it.

There are also many more options in terms of development tools for working with the 'F5529. Some of the options are: Code Composer Studio (which should work on all platforms), The Cloud edition of Code Composer Studio, and the same gcc tool-chain we've been using with the 'G2553.

We'll provide instructions here for using gcc, but you are welcome to try CCS or CCS-cloud if you wish.

Building programs with gcc is essentially the same as it was with the 'G2553. In the Makefile you need to set the device to be msp430f5529 so that the compiler knows where to put the program in memory, and which include file to use.

Driver Install

If you're using windows, you might need a driver for the board. Try plugging in the board to see if windows complains that a driver is needed. Double check in the device manager, under Ports (COM & LPT). The board should show up as two COM ports: one debug interface, and one Application UART. If they are both there you shouldn't need a driver). If you do need the driver, you can get it from:
http://energia.nu/files/ezFET-Lite.zip
If the driver is needed, you might need to disable driver signing enforcement to install it. If windows complains that the driver isn't signed, then you can follow the instructions at:
https://learn.sparkfun.com/tutorials/disabling-driver-signature-on-windows-8
to disable driver signing enforcement. (Hopefully none of this is necessary)

Demo Program:

There is a demo program at: http://www.phas.ubc.ca/~michal/P319/F5529-serial.tar.gz
(or http://www.phas.ubc.ca/~michal/P319/F5529-serial.zip for windows users).

1) This program demonstrates serial communications with the 'F5529. To use it, unpack the demo directory. Edit the Makefile so that INSTALL_DIR points to the directory where the compiler was installed (for linux and mac, installed on your computer, this should be ~/ti/msp430_gcc, for windows try c:/ti/msp430_gcc for the lab computers and our linux image , try /opt/msp430_gcc).

2) To flash the program onto the board, and debug, the situation is a little complicated.
mspdebug can still be used, though it needs a library from TI to talk to the board. for linux and mac it will be simpler to use the bridge program that came with the compiler.
So, for linux and mac, try:
```
gdb_agent_console ~/ti/msp430_gcc/msp430.dat
```
or, if you're using the lab computers or our os image:

```
gdb_agent_console /opt/msp430_gcc/msp430.dat
```
in one terminal. Then in another terminal start up the debugger:
```
msp430-elf-gdb main.elf
target remote :55000
load
```
Then you can debug and run the program as before.

While you can do essentially the same thing in windows, the windows version of gdb_agent_console stinks. There is no good way to interrupt a running program on the board. Fortunately, the windows version of mspdebug we used came with the TI library needed to talk to the board. So on windows try this:
In one terminal, change into the directory where mspdebug was unpacked, and then:
```
mspdebug tilib
```
If it tells you it needs to change the firmware on the board, start it again with:
```
mspdebug tilib –allow-fw-update
```
Let it finish the update and then start it again.
Then in mspdebug: `gdb`
Then in another terminal, start up the debugger:
```
msp430-elf-gdb main.elf
target remote :2000
mon erase
load
```
Then you can debug and run the program as before.

Serial Terminal:
The demo program receives characters over the serial port, toggles the red LED every time a character is received, and returns the character back over the port. You need a serial terminal emulator on the host computer to test this out. When you installed pyserial, it came with miniterm which should do the trick.

On linux and mac, just typing `miniterm.py` should start it. Then you'll need to select the correct port (select the Application UART). On windows, miniterm is buried. Try:
```
c:\python27\Lib\site-packages\serial\tools\miniterm.py
```

When the emulator is running, slowly type characters. The LED on the board should blink, and the characters you type should appear on the terminal emulator screen.

If it doesn't work, try quitting out of miniterm (CTRL-]) then unplugging and replugging the board before trying again.