Platform Creation and Testing

From TinyOS Wiki

A new platform wiring debug tool with only two code files.

We have a fairly simple goal -- to harness GPIOs so we can control them and see what they are inputting on a MSP430 based platform. Seeing means displaying, so we will use the three LEDs found on many MSP430 platforms to display a 3 bit binary number corresponding to pin0 through Pin7 that we wish to test. This module is for MSP430 testing purposes so will never exist as Hardware Independent HIL modules. This code helps with testing whether a new printed circuit board has traces going to the right MSP430 IO ports.

With a HAL, (hardware adaptation layer), configuration and code file plus a HPL, (hardware presentation layer), configuration and code file we would have 4 files. Since we reuse the HplMsp430GeneralIO interface provided by HplMsp430GeneralIOC.nc with new names and add no new functions, we can do without configuration or code file, (beyond all the core tinyOS modules), for this exercise. TestMSP430platfIOC.nc, the configuration file, maps connections from any port names predefined in HplMsp430GeneralIOC.nc to our desired group of eight ports named Pin0 - Pin7. This test is done on a new platform, defined by a new directory added to the tinyos source code tree: for example /opt/tinyos-2.x/tos/platforms/ecosens1. Some necessary parts for a new platform variant were copied from Telosb: PlatformC.nc MoteClockC.nc MoteClockP.nc PlatformP.nc. They set up the clocks and initialize the MSP430. Also needed in this new platform is hardware.h, which was copied from telosb and modified to change names to leave off the humidity sensor, and rename some GeneralIO signal names.

Creating a new platform

There is plenty of ground work done for us, so the first steps of this process are copying -- This example code can run on an unmodified telosb hardware. Some details you will see are specific to an Ecosensory platform, but they don't stop you from learning by making the IO tester program below on any telosb derived platform. So copy from telosa and telosb and update .platform file and remember support/make/*.target -- see lesson 10. To use the tutorial as is, the name of the platform can be ecosens1p, p for prototype, but it will be better to use your own name and get the hang of it more. When you are done copying your new dir should look like this:

```
john@ecolab:/opt/tinyos-2.x-contrib/ecosensory/tos/platforms/ecosens1$ ls
ActiveMessageC.nc   DemoSensorStreamC.nc  MoteClockC.nc  PlatformC.nc
PlatformSerialC.nc  UserButtonP.nc  chips                  Ecosens1SerialP.nc
MoteClockP.nc       platform.h      SwitchC.nc             UserSwitchC.nc
CVS                 hardware.h      MotePlatformC.nc       PlatformLedsC.nc
SwitchToggleC.nc    UserSwitchP.nc  DemoSensorC.nc         HplUserButtonC.nc
Msp430Timer32khzMapC.nc  platform_message.h    UserButtonC.nc   VoltageC.nc
DemoSensorNowC.nc        HplUserSwitchC.nc     NotifyBit.nc     PlatformP.nc
UserButton.h             VoltageStreamC.nc
```

To simplify we will learn about the bottom, close to the hardware, HPL layer of tinyOS modules first in the test program below.

GPIO Tester program

The configuration file called TestNewPlatformIOC.nc, declares standard telosb tinyos parts MainC, userButtonC, LedsC, TimerMilliC() and then the program module TestNewPlatformIOP.nc, and a low level Hpl component:

```
components HplMsp430GeneralIOC;
TestMSP430platfIOP.Pin0 ->  HplMsp430GeneralIOC.Port60;
TestMSP430platfIOP.Pin1 ->  HplMsp430GeneralIOC.Port61;
```

The arrows in a configuration file point in the direction of more specific hardware, or the root of a definition, whether pointing left or right. The above arrows connect references in the program sequence file to the hardware presentation layer file specific definition, so this file can have differences depending on platform. If two versions exist, they will be in different platform or sensorboard dirs that also use the MSP430 chip. This snippet from TestMSP430platfIOC.nc shows reusing interface HplMsp430GeneralIO with new names.

```
module TestMSP430platfIOP {
  uses {
    interface Boot;
    interface Get<state_t>;
    interface Notify<state_t>;
    interface Leds;
    interface Timer<TMilli>;
    interface HplMsp430GeneralIO as Pin0;
    interface HplMsp430GeneralIO as Pin1;
           etc....
```

Next, declaring the HPL for the MSP430.

```
implementation {
  components HplMsp430GeneralIOC;
  TestMSP430platfIOP.Pin0 ->  HplMsp430GeneralIOC.Port60;
  TestMSP430platfIOP.Pin1 ->  HplMsp430GeneralIOC.Port61;
         etc....
```

The above wiring connections are made from the names to specific MSP430 port pins, that number many more than 8. We are just declaring a way to use 8 because we have a goal of displaying with three bits! We have a widespread beginner display in the three LEDs, able to generate some excitement in a newbie, so we use them. A future tutorial that repeats this program for another platform would be a nice learning tool.

Notice the names for our 8 IOs (Pin0, Pin1...) can be anything in this file, they are local, and can't be used anywhere else. To use this program as a hardware testing tool, you can redefine the right sides of the above list to suit, using names from HplMsp430GeneralIOC.nc , and put them on a piece of paper with room to the side to write notes, then load the program on the new platform hardware and apply current limited 2.8 or 2.9volts if your batteries are at 3.0 Volts, and see changes in LED output as you select Pin0 or Pin5 or Pin7 by pressing the user button to count through the LED displayed numbers. Telosb hardware has some pull down resistance on the inputs of GPIOs -- I didn't need to pull inputs down to get a LO, and the input voltages measured between .001 and .05 Volts. The sequences and flashing times used can be changed in TestNewPlatformIOP.nc if you want a different "user interface"...

Here is the core of the TestNewPlatformIOP.nc program, showing use of the new names for MSP430 port pins:

```
event void Timer.fired() {
  periodhalf = !periodhalf; //every other time period get a pin value.
  if (counter == 0x0)  { pinHi = call Pin0.get();
```

You could get fancier displaying the pin number or the pin value read. What is more important for understanding tinyOS is that Pin0.get() is a renaming of GeneralIO.get, one of the built in interfaces, and that it has been mapped to a specific port using names from HplMsp430GeneralIOC.nc. In HplMsp430GeneralIOC.nc

we see lines like:

```
ADC0 = P60;
```

```
#ifdef __msp430_have_port6
  Port60 = P60;
```

Port60 has two names. Both work fine -- Port60 or ADC0. P60 is a local name not usable anywhere else.

By John Griessen www.ecosensory.com copyright 2007

Related Documentation

- TEP 2: Hardware Abstraction Architecture (http://tinyos.net/tinyos-2.x/doc/html/tep2.html)
- TEP 107: TinyOS 2.x Boot Sequence (http://tinyos.net/tinyos-2.x/doc/html/tep107.html)
- TEP 109: Sensors and Sensor Boards (http://tinyos.net/tinyos-2.x/doc/html/tep109.html)
- TEP 114: SIDs: Source and Sink Independent Drivers (http://tinyos.net/tinyos-2.x/doc/html/tep114.html)
- TEP 117: Low-level I/O (http://tinyos.net/tinyos-2.x/doc/html/tep117.html)
- nesC reference manual (pdf file) (http://nescc.sourceforge.net/papers/nesc-ref.pdf)
- TinyOS Programming Guide (pdf file) (http://www.tinyos.net/tinyos-2.x/doc/pdf/tinyos-programming.pdf)

< **Previous Lesson** | **Top** >
