Setting up UDOO to work with Pd: using [gpio] and [comport]

December 7, 2013

This tutorial explains how to get set up using the Arduino built in to your UDOO with Miller Puckette's audio programming environment, Pure Data. The following is geared towards people that are Linux noobs or that are not used to interfacing with and OS through the command line (as God intended). If you are a true Linux beginner, you might want to check out some of the tutorials on the UDOO website `http://www.udoo.org/tutorials/`. Anyway, for more advanced Linux users, much of this might seem old-hat. On the other hand, if I am unclear, please let me know (dmedine(at)ucsd.edu).

1 Putting Pd on UDOO

First of all, there is a nice tutorial on how to get Pd-extended on the UDOO forums: here (`http://www.udoo.org/forum/viewtopic.php?f=27&t=478&p=3827&hilit=puredata#p3827`). Pd vanilla is the core of the Pd program written and maintained by Miller Puckette. Even that ships with a few 'extra' objects such as [pique], [sigmund], and (notoriously) [pd]. These are external objects written by Miller, but not officially part of Pd itself. Pd-extended is Pd plus dozens and dozens of extras written by third party developers. Many of these are truly excellent pieces of software, but there is also a lot of (in my opinion, anyway) stuff that isn't really needed. It is good to stick with Pd vanilla for development purposes, because it ensures greater compatibility from user to user. Nevertheless, sometimes one needs objects from Pd-extended to do this or that. My approach has been to merely take what I need from Pd-extended by only building those objects that are truly necessary for my own projects to succeed. The third section of this tutorial is just such an example. There, it is explained how to build and install [comport] by Martin Peach so that we can pipe in serial data coming from Arduino into a Pd patch and vice-versa.

You can download pure data from:

`http://msp.ucsd.edu/software.html`

Choose 'Pd version 0.45-4 (source)' and download the tarball. Extract it to whatever folder you wish. Then open up a terminal (ctl-alt-t is the keyboard shortcut) and navigate to the directory that you extracted Pd into:

```
> cd /<path>/pd-0.45-4/src
```

where <path> is whatever folder you extracted Pd into. Then hit

```
> make -f makefile.gnu
```

Go do something else for 4 or 5 minutes. When that is over, cd into pd's bin directory:

```
> cd ../bin
```

and look at it.

```
> ls
```

There should be a bunch of executables including one called 'pd'. To start up Pd, (provided you are in that folder). If not, then something didn't go right with your build.

```
> ./pd
```

2 Setting up serial link between Arduino and Linux

First of all, to program Arduino from Linaro on your UDOO you need the Arduino IDE on your UDOO. As far as I know, the latest version of the Linaro OS for UDOO ships with the Arduino IDE (it did the last time I installed it, anyway). Otherwise download the IDE from the UDOO downloads page, `http://www.udoo.org/downloads/`, and install it on your system. Otherwise you program the Arduino from another computer via the usb cable.

The tricky thing is that you have to link an extra serial port in your /dev directory. The correct way to do this is to type

```
> sudo ln -sf /dev/ttymxc3 /dev/ttyS0
```

on your command line and enter your password. If you want this to happen

automatically every time you start your UDOO, add the line above (without the 'sudo' part) to the file /etc/rc.local. You will have to use su privilege to do this:

```
> sudo gedit /etc/rc.local
```

gedit being my text editor of choice for these things. If you don't have gedit, you can get it (get it?) by typing:

```
> sudo apt-get install gedit
```

Ok. Now when you open up the Arduino IDE, you should have the ability to choose the port /dev/ttyS0 from the pull down menu `Tools->Port`. If not, quit the IDE and try again. If not still, you probably skipped a step before (remember, rc.local only happens on startup, so you have to either reboot or create the virtual port by hand).

Now, you can make a very simple sketch to test everything out.

code:

```
void setup(){
Serial.setup(9600);
}

void loop(){

Serial.println("Hello, world!");
delay(1000);

}
```

This code will send the values that make up the message "Hello, world!" over the port /dev/ttyS0 every second for ever and ever and ever and ever...

To test that everything is hunky-dory, whang on a terminal and do the following:

```
> exec 3<>/dev/ttyS0
```

This informs the Linux that the port /dev/ttyS0 is now file descriptor '3'. Then you can see what's going on in the port by typing

```
> cat <&3
```

This should print out "Hello, world!" the number of times seconds went by since you launched your Arduino sketch.

BTW, you'll want to be careful about closing down the serial connection after you are done using it. Otherwise you will have trouble reprogramming your Arduino across the port.

```
> exec 3>&-
```

The above will close this file descriptor and free the port for programming purposes.

3 Getting [comport]

Now that we can send data between Arduino and Linux, we can route that data into a Pd patch. In order to do this, we need a Pd extern called [comport]. [comport] ships with Pd-extended (as do many, many other third party Pd externs) but you don't have to get the whole Pd-extended package in order to have just [comport].

Step 1 is to download comport `http://puredata.info/downloads/comport`. In order to build it, you will have to make one alteration to the Makefile. You need to tell make where the file m_pd.h is. This file is in Pd's src directory that we visited before when we built pd. So, to do this, under the line in the Makefile that says:

```
CFLAGS = -I"$(PD_INCLUDE)/pd" -Wall -W -g
```

add a line that says this:

```
CFLAGS += -I/<path>/pd-0.45-4/src
```

where, again, <path> is the directory that you put Pd in when you extracted it. On my UDOO, I made a directory called Software in my home folder which is where I put Pd and all my other downloaded software. So, my Makefile for [comport] has this in it:

```
CFLAGS += -I/home/ubuntu/Software/pd-0.45-4/src
```

Save the Makefile and hit:

```
> make
```

It should compile without errors. So now, all that is left is to tell Pd where the binary for the [comport] object lives. To do this, open up Pd (cd into the bin folder and hit ./pd). Navigate to the menu `Edit->preferences->path...`

and add the path that you put your comport-0.2 folder into. The 'New...' button opens up a gui so you can navigate there. Make sure the 'Use standard extensions' box is checked. Then click Apply and hit OK. Now you have [comport] in Pd vanilla.

Presumably your simple Arduino sketch that prints out 'Hello, world!' is still running. If not, start it up again. Once it is verified and loaded open up the patch called `hello_world.pd` in Pd and see that it works. Obviously, one must finesse Pd into receiving the data in a form you prefer. Programming is fun!

At this point, if you are at all familiar with Arduino and Pd, you should have what you need to use your Arduino as a control mechanism for Pd on UDOO. Be careful not to access pins via the OS (ie with gpio object for Pd) whilst running an Arduino sketch that uses the same pins. This will damage your board. Another disclaimer, close down your connection to /dev/ttyS0 before putting a new sketch on your Arduino over the serial port. ttyS0 is doing double duty as a communications relay, and the port through which we upload sketches to the the chip, and we can't do both at the same time. If you have problems, a power cycle never hurts.

4 Getting Data out of Pd, into Arduino

I have also created a Pd patch and associated Arduio sketch called 'basic_serial_io'. The folder is the Arduino sketch which you should be able to open up with the Arduino IDE. The Pd patch is the file that (duh) ends in '.pd'. The sketch is set up to use pin A7 as an ADC input and 53 as a digital output. The patch is pretty self explanatory, so I won't go in to details here. One note, this patch relies on an abstraction that I wrote called 'byte_trans_abs'. This should be in the same folder as all the rest of the files associated with this tutorial. The abstraction takes in n number of bytes and translates them into an integer value. It works by knowing that the method Serial.println in Arduino will terminate a transmission of bytes with the ascii characters 13 then 10. These characters are used as cues to know that the data is done coming in and that it is time to translate the word into an integer. I made these cues arguments tot he abstraction so that one can provide any cue to end a transmission.

5 Building and Using [gpio]

[gpio] is an extern for Pd written by Miller Puckette, originally for the Raspberry Pi. It allows the user to specify a gpio pin and set its mode. So far only digital pins are supported. I modified the Makefile so that it will build easily on UDOO. In order to build it, download the folder and put it in Pd's extra folder:

> mv -f /<path to gpio>/gpio/ /<path to pd>/pd-0.45-4/extra/gpio/

This should move the folder and all its contents into pd's extra folder. cd into that directory and hit:

> make

BTW, you may have noticed that every time you build anything on UDOO you get a warning about a 'clock skew' at the end of the build. This is because your UDOO boots up thinking that it is 0:00 January 1, 1970 GMT. If you want the clock to reflect current GMT, you can force it to:

> sudo ntpdate ntp.ubuntu.com

If ntpdate is not installed, you can get it:

> sudo apt-get install ntpdate

This doesn't really make a difference unless, like me, you are super-OCD about this kind of thing. Anyway, [gpio] should have built and there should be a binary called 'gpio.l_arm' in the directory. You won't need to tell Pd about where this folder is, because Pd already knows about its extra folder and everything in it.

To test it out, simply make a new Pd patch and create the object [gpio]. I wrote a little help patch to show how to get started with digital outs. To see it, do the usual right-click on the object and select the help patch. This is an on going project so it should grow over time as new modes etc are implemented.

One word of warning. It is potentially dangerous to your UDOO to access the pins from both the Linux operating system (which is what [gpio] does) and the Arduino chip itself. Check the UDOO manual for details. Also, be very careful not to plug in any power supply greater than 3.3 volts into any of the pins. This could straight-up fry your processor and ruin your board.