USB ARDUINO LINK – Part 2

Last month I wrote about how to set up your Raspberry Pi so that it could communicate with an Arduino via a serial USB cable using Nanpy. In this mode the Raspberry Pi is the master and the Arduino its slave input/output board, providing protection and extra facilities for the master. I listed simple Python digital input and output programs using a button switch and an LED to test the system. I hope you managed to get it working. I am now going into more detail about driving a Liquid Crystal Display and the 6 analogue pins (A0 – A5) on the Arduino, which can read voltages between 0 and 5 volts.

**Using a Liquid Crystal Display (LCD)**

If you have access to a 5 volt 16x2 HD4470 Compatible LCD (currently about £7 from oomlout.co.uk, who have full circuit information on-line, and from other suppliers).

<http://oomlout.com/parts/LCDD-01-guide.pdf>

It is very easy to control via the built in Nanpy LCD library. When running Nanpy, pins 0 &1 of the Arduino are used for communication with the Raspberry Pi; so I connected my LCD to pins 2 to 7.

**LCD pin Name Arduino pin**

1 GND GND

2 Vdd 5v 5v

3 Contrast 10K ohm potentiometer wiper, other pins to 5v & GND

4 RS 7 # Not 12

5 R/W GND

6 Enable 6 # Not 11

11 Data 4 5

12 Data 5 4

13 Data 6 3

14 Data 7 2

15 Light +ve 5v

16 Light –ve GND

The following program demonstates the operation by counting up and down:

#!/usr/bin/env python

# Basic LCD use via nanpy

from time import sleep

from nanpy import Arduino

from nanpy import serial\_manager

serial\_manager.connect('/dev/ttyACM0')

from nanpy import Lcd

lcd = Lcd([7,6,5,4,3,2], [16,2]) #LCD set-up

print "### Starting ###\n"

# Heading - top row

lcd.printString("LCD Demo - Counting", 0,0)

sleep(1)

for i in range(0,21):

position = 4

if i < 10:

position = position + 1

# Clear 2nd row

lcd.printString(" ", 0,1)

lcd.printString(i, position, 1)

i2 = 20 - i

position2 = 9

if i2 < 10:

position2 = position2 + 1

lcd.printString(i2, position2, 1)

sleep(0.7)

print "\n### FINISHED ###"

**Reading analogue values**

The Arduino has 6 analogue ports for reading voltages between 0 and 5 volts with 10-bit resolution giving values from 0 to 1023. The second circuit from the left of the photograph, between breadboard columns 30 and 40, shows the simplest analogue demonstration circuit. It uses a 10K ohm potentiometer with the outer pins connected to GND and 5 volts. The central wiper pin is connected via the orange wire to pin A1 on the Arduino. This pin is also called pin 15. (A0 is 14, A1 is 15 …. A5 is 19.)

The following code demonstrates how to read analogue values in Python.

#!/usr/bin/env python

# Read values from Analogue pin A1

from nanpy import Arduino

from nanpy import serial\_manager

serial\_manager.connect('/dev/ttyACM0')

from time import sleep

pot = 15 # Pot on A1 - Analog input

print "Turn the pot - 10 bit ADC input"

for i in range(0,40):

val = Arduino.analogRead(pot)

print val

sleep(0.3)

As you turn the spindle on the potentiometer the values change within the range 0 to 1023.

You can connect many different components/devices to the analogue pins to read temperatures, and measure distances etc.

Once you start building more ambitious projects you will find that you can quickly run out of pins. As you will have seen, a liquid crystal display uses up 6 of your digital pins. Imagine that you were building a model lift. You need digital pins to control the motor, an LCD and several LEDS. You also need buttons to call the lift on each floor and others in the cage. One trick is to use a single analogue pin to monitor the 5 switches on the panel in the lift cage.

The circuit is shown in the photograph in breadboard columns 40 to 61. It has six resistors in series and five button switches connecting the resistor junctions to GND. The circuit is connected to Arduino pin A1 via the yellow wire from the junction between the right most resistors.

The circuit is shown below:

Pin A1 (15)

|

+---3K3---+---1K0---+---680R---+---330R---+---2K2------5v

| | | | |

Alarm Close 0 1 2 Switch name

| | | | |

[772] [490+-] [322] [134] [0] Analogue Value

| | | | |

GND--------+---------+----------+----------+

Here the 2K2 ohm resistor acts as a pull up resistor to 5 volts. With no button pressed the reading from the analogue pin is 1023.If the button for floor 2 is pressed the junction is shorted to GND and the reading changes to 0, just like a digital switch. As each button is pressed in turn, from right to left, the resistance to GND increases (0, 330R, 1K0, 2K0, 5K3 Ohms) and the voltage at the main junction changes. Unfortunately, resistors with the same nominal value can vary significantly but still be within their quoted tolerance. It is not really worth trying to calculate the analogue value for the other switches as each board will be slightly different. It is much easier to re-run the last program a few times with the yellow wire connected to pin A1, press the buttons in turn and record the values for each button. My values are shown in the square brackets and listed in the program.

There is usually at least one ‘difficult’ button whose value oscillates between two adjacent values. My reading for the ‘Close Door’ button was either 490 or 489. This problem is easily overcome by testing for a range between upper and lower limits rather than a specific value. There are very large gaps between the button values leaving plenty of room for the ‘fudge factor’.

For example: (value > 485 and value < 495) rather than (value == 490).

Here is the basic code to read the switches:

#!/usr/bin/env python

# Read values 5 switch/resistor array on pin A1 (15)

# Lift cage control panel simulation

from time import sleep

from nanpy import Arduino

from nanpy import serial\_manager

serial\_manager.connect('/dev/ttyACM0')

pot = 15 # Pot on A1 - ADC input

# You may want to change these values to match your board

# or increase fudge value

floor2 = 0

floor1 = 134

floor0 = 322

close = 490

alarm = 772

fudge = 5

print "Press the buttons"

print "\nCTRL-C to stop program\n"

old\_val = -1

val = Arduino.analogRead(pot)

while True:

try: # Trapping CTRL-C

# Has val changed > ADC 'wobble' ?

if abs(old\_val - val) > 3:

old\_val = val

if val < fudge:

print"Floor 2"

if val > floor1 - fudge and val < floor1 + fudge:

print"Floor 1"

if val > floor0 - fudge and val < floor0 + fudge:

print"Floor 0"

if val > close - fudge and val < close + fudge:

print"Close doors"

if val > alarm - fudge and val < alarm + fudge:

print"Alarm"

val = Arduino.analogRead(pot)

except KeyboardInterrupt:

print "\nProgram interrupted by user. \n"

break

print "\*\*\* Finished \*\*\*"

This is an ‘endless loop’ program – lifts run all the time. Stopping it with CTRL-C normally results in a mess of red error messages. Notice how this has been trapped resulting in a ‘clean’ termination.

**Things to try**

Extend the multiple buttons circuit with an extra button and resistor. Then change the program to allow for the extra floor.

If you enjoy soldering you could transfer the five buttons and resistors onto a piece strip board – see photograph. Arrange four of the buttons in a diamond pattern to indicate’ Up’, ‘Down’, ‘Left’ and ‘Right’. Place the fifth button either in the centre of a large diamond or at the side of a small diamond for ‘OK’, ‘Stop’ or’ Fire’. The illustration shows, “one I made earlier”. The copper strips are vertical on the back. Only one track was cut on the underside. The cut track is under the ‘Up’ button’s bottom, right corner and marked with the magenta blob. This will useful for the following exercises but the ‘in-line’ version will work as well.

Set up three or more LEDs with resistors (330R or more – we do not need them to be too bright) in a row on digital pins.

1. Light the leftmost LED and make the light appear to move from left to right and back again by pressing two buttons - Right & Left. Use Ctrl-C to stop the program.
2. Control the brightness of an LED by two buttons – Up & Down. Use another button to stop the program.
3. Combine the first two ideas so that you can set the brightness of all the LEDs with 4 buttons. Use the other button to stop the program. Hint: flash the LED that is currently selected while you adjust its brightness. NB: the PWM pins are marked with ‘~’: 2, 5, 6, 9, 10 & 11.

Modify the LCD counting program and control the two sets of numbers with the buttons rather than a counted loop. Use the one button pair to increase/decrease the left hand number on the display and another pair to control the right number independently. Stop the program if either number reaches 0.

If you do not have an LCD you could write a similar program but output the two numbers as columns on your monitor – one line each time a button is pressed. Can you keep the tens and units lined up vertically? Stop program with CRTL-C, but trap the error messages.

There is currently a forum on using the Arduino with Nanpy at:

<http://www.raspberrypi.org/phpBB3/viewtopic.php?p=368522#p368522>

You can use it to ask for help from other users.

You can contact me with email via [arduinolink@outlook.com](mailto:arduinolink@outlook.com) with feedback, suggestions and questions. I enjoy experimenting with hacking projects and robots and hope to encourage others to take up this great hobby (especially in Leicestershire, where I taught computing for 23 years. How about a Raspberry Jam? Lancashire is doing so much more!).

Thanks again to Andrea Stagi for producing Nanpy.