NetRexx on the Raspberry π

24th International Rexx Language Symposium

Raleigh/Durham, NC

René Vincent Jansen, May 8th, 2013

What is it?

A $35 ARM SBC

- Single board computer
- ARM instruction set architecture processor
- Max 1Ghz clock, 512 MB RAM
- Runs e.g. Linux
- Runs NetRexx just fine

BYKMM

- Bring your own keyboard, monitor and mouse
- I mostly use it headless over SSH, no tube or keyboard
- Model B has Ethernet and 2 USB; Model A ($25) only 1 USB
- It has a HDMI and Optical audio output of very high quality. This is wasted on my character mode mindset
- But: memory split is adjustable, use all that MB

GPIO

- Difference with your PC: in addition to everything your big computer can, it can interact with outside electronic hardware components through a GPIO interface
- General Purpose I/O: This is a set of programmable pins on the system board, for you to use
- You'll never have so much fun with a cheap computer
- Eminently programmable in NetRexx

The ARM and USB chip

The Pi is built around a Broadcom SOC that might have been part of your smartphone

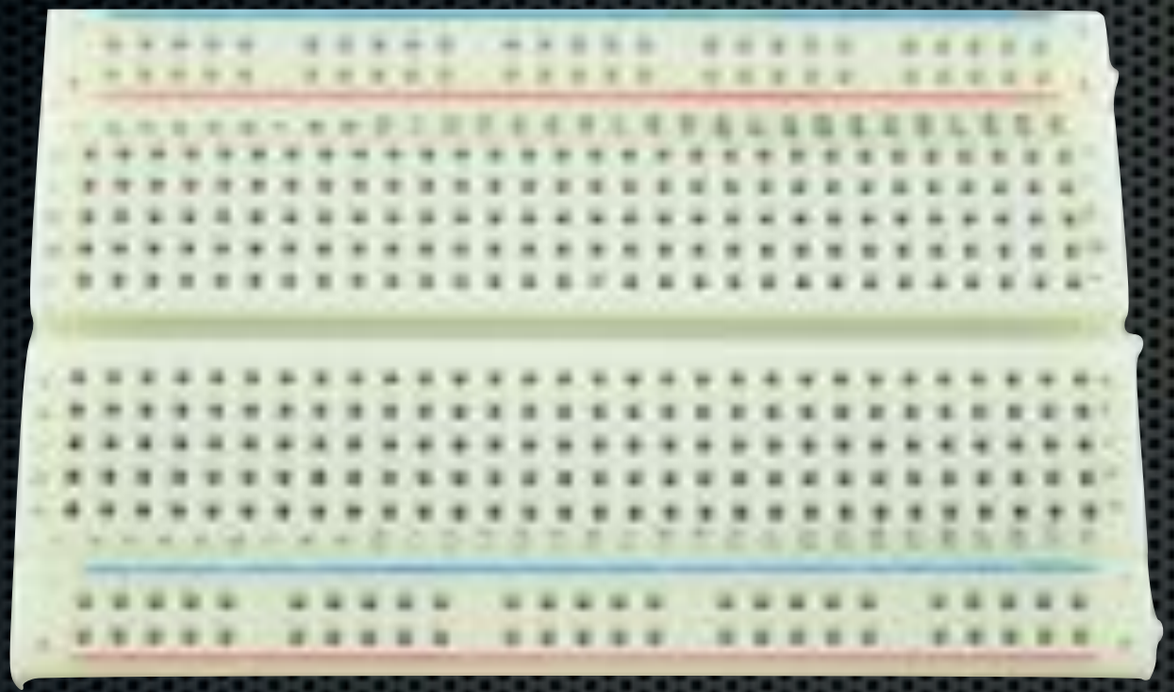
GPIO Pins

The entry points into embedded development

GPIO Layout

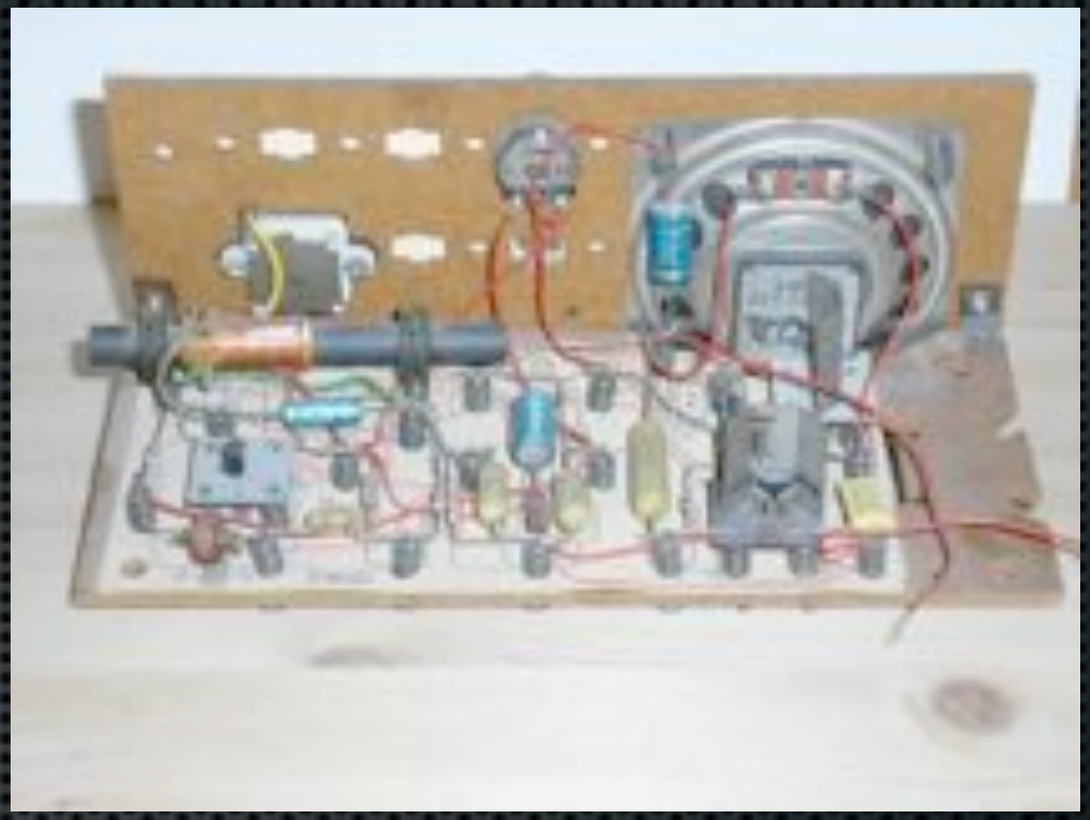
Use a 'T-cobbler' for easy connection to a breadboard

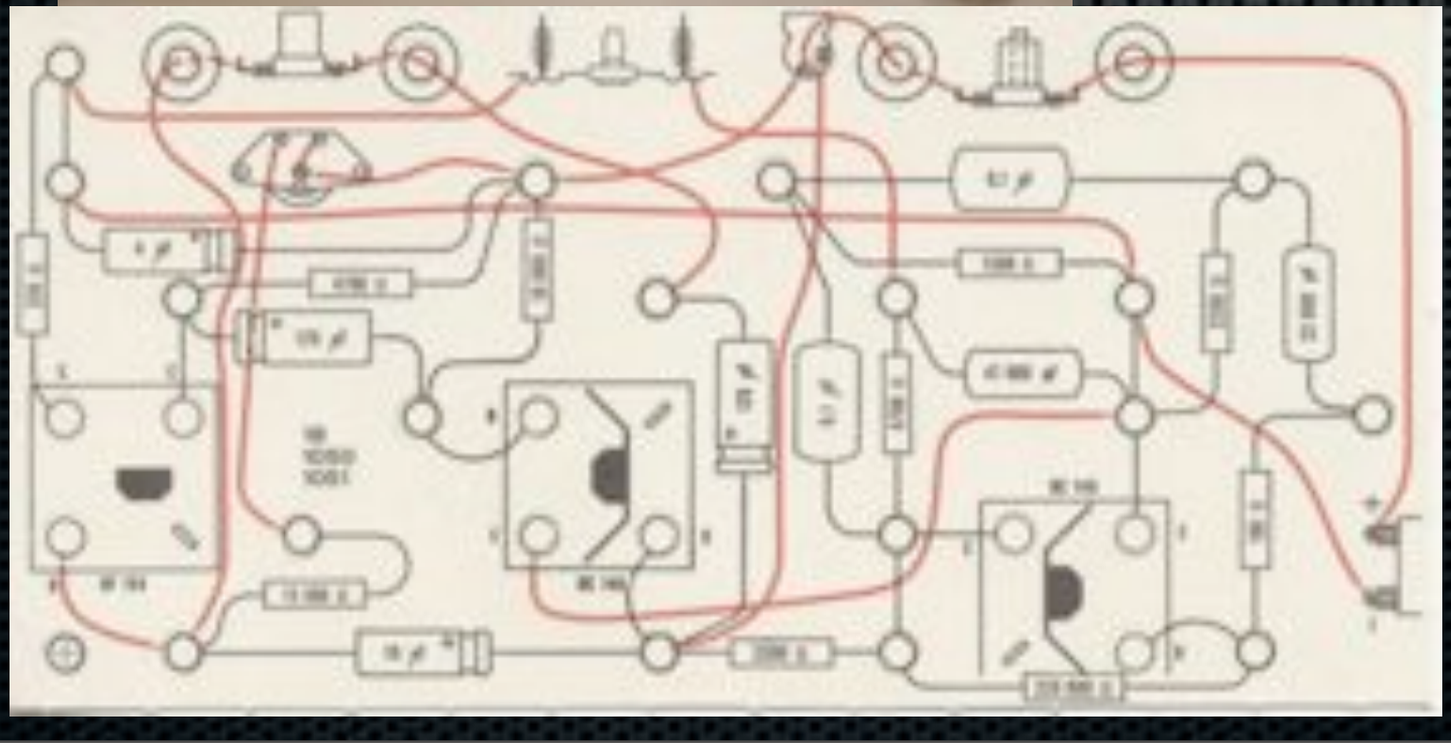
What is a breadboard



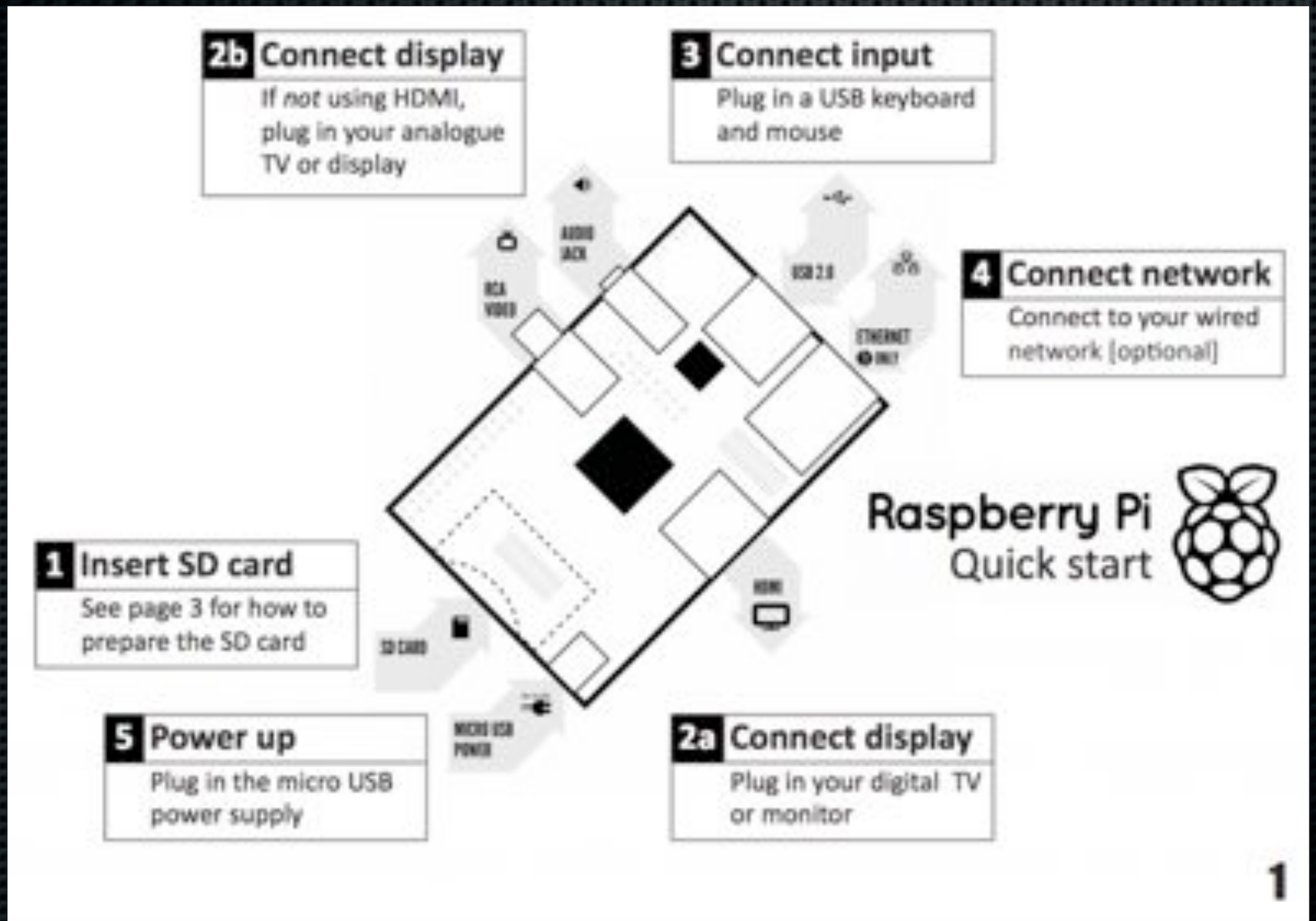
A convenient, solderless way to set up prototype circuits

The last time I did electronic circuits (1972)





Setting it up



Setting up the Pi for NetRexx

- Write the ssd card, connect to network and switch on
- nmap -A 10.0.0.0/24 -p22
- edit .ssh/known_hosts if necessary (on the machine you use for ssh)
- ssh to it (pi@10.0.0.X) (or pi@192.168.0.X)
- sudo raspi-config

Write the ssd card

- You'll need of of these if your computer does not have a slot for it
- The one in your camera will probably work
- Best instruction at: http://elinux.org/RPi_Easy_SD_Card_Setup

raspi-config

- expand fs
- memory split (16 for headless systems)
- enable ssh
- locale (choose one only or it takes very long)
- reboot

Install some essentials

- sudo apt-get update
- sudo apt-get install git
- sudo apt-get install emacs
- sudo apt-get nmap
- sudo apt-get install sysstat
- sudo apt-get install subversion

Which Java to install

L'embaras du choix

Which Java VM

- Oracle: (at the moment at) http://jdk8.java.net/fxarmpreview/index.html
- There are others, but not as fast - install these for ideological/sentimental/research reasons
- This preview does not have Swing, only JavaFx

Install the JVM

- download java from https://jdk8.java.net/fxarmpreview/index.html
- scp jdk-8-ea-b36e-linux-arm-hflt-29_nov_2012.tar.gz pi@10.0.0.78:
- tar xvfz jdk-8-ea-b36e-linux-arm-hflt-29_nov_2012.tar.gz
- sudo mkdir -p /opt/java
- sudo mv jdk1.8.0/ /opt/java

Install Mosquitto/MQTT

- wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
- sudo apt-key add mosquitto-repo.gpg.key
- cd /etc/apt/sources.list.d/
- sudo wget http://repo.mosquitto.org/debian/mosquitto-repo.list
- sudo apt-get update
- sudo apt-get install mosquito
- sudo apt-get install mosquitto-clients

What is MQTT

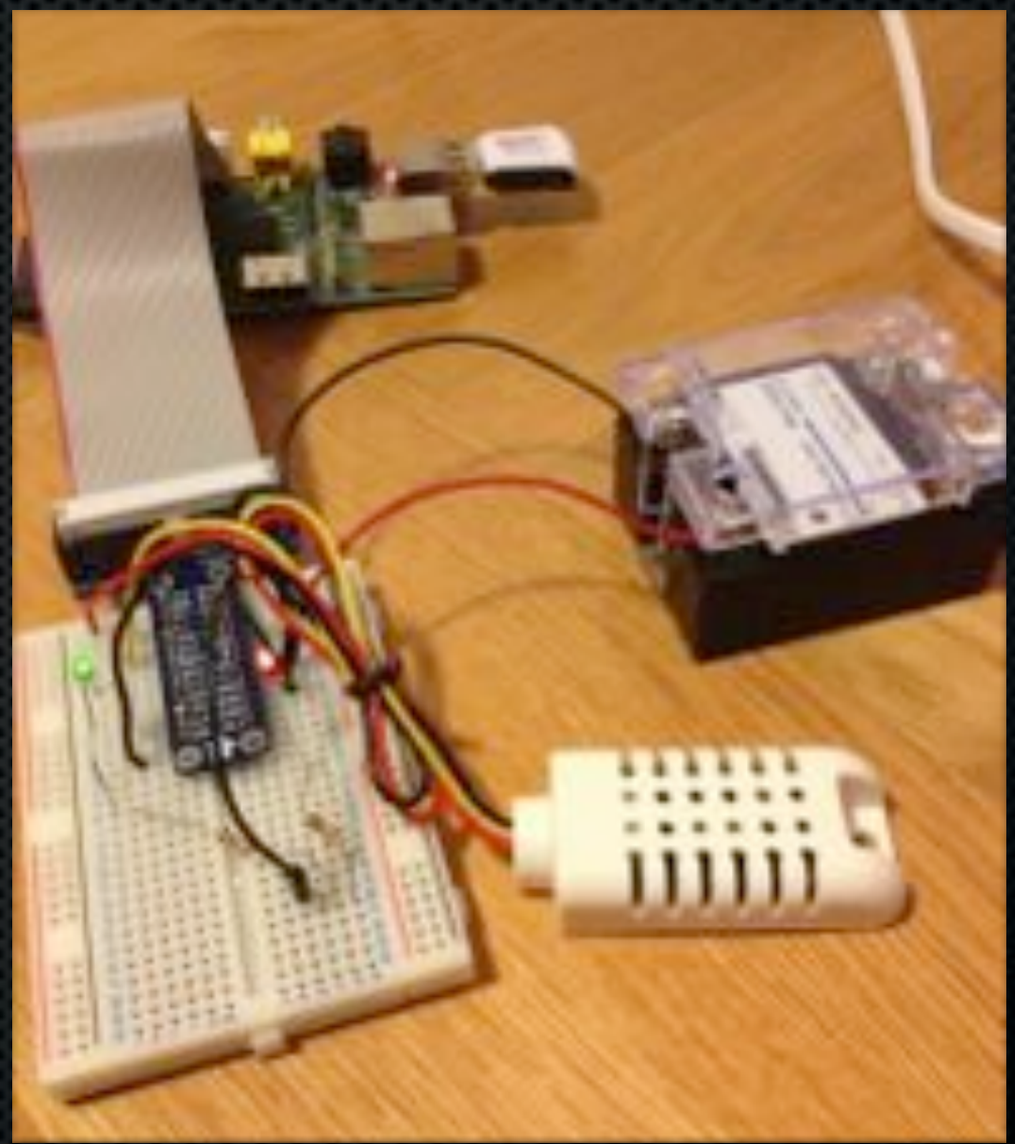
- It is a low-overhead messaging protocol
- Saves bytes and energy compared to http/ssl
- Robust (knows how to handle outages - even has a *last-will-and-testament* facility)
- Ready for 'the internet of things'
- Excellently programmable in NetRexx through PAHO client

Install WiringPI utilities

- git clone git://git.drogon.net/wiringPi
- cd wiringPi
- ./build
  - test it:
  - gpio -v
  - gpio readall

Switching Mains

this optical solid state relay is entirely safe, for you and for the raspberry



USB to TTL

(Only connect the red lead if there is no other power supply connected)

The red lead should be connected to 5V,

- The black lead to GND,
- The white lead to TXD.
- The green lead to RXD.

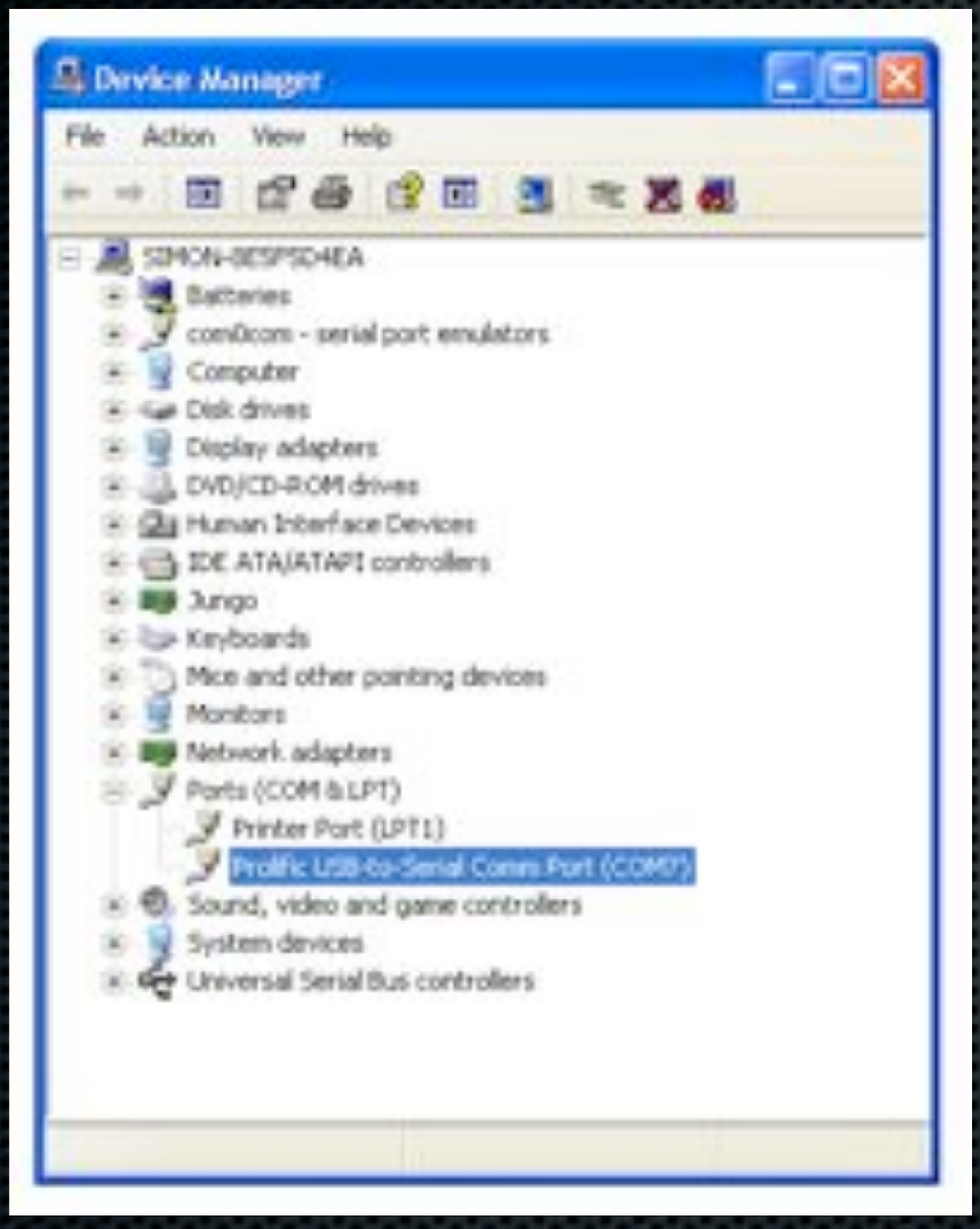
Download the driver

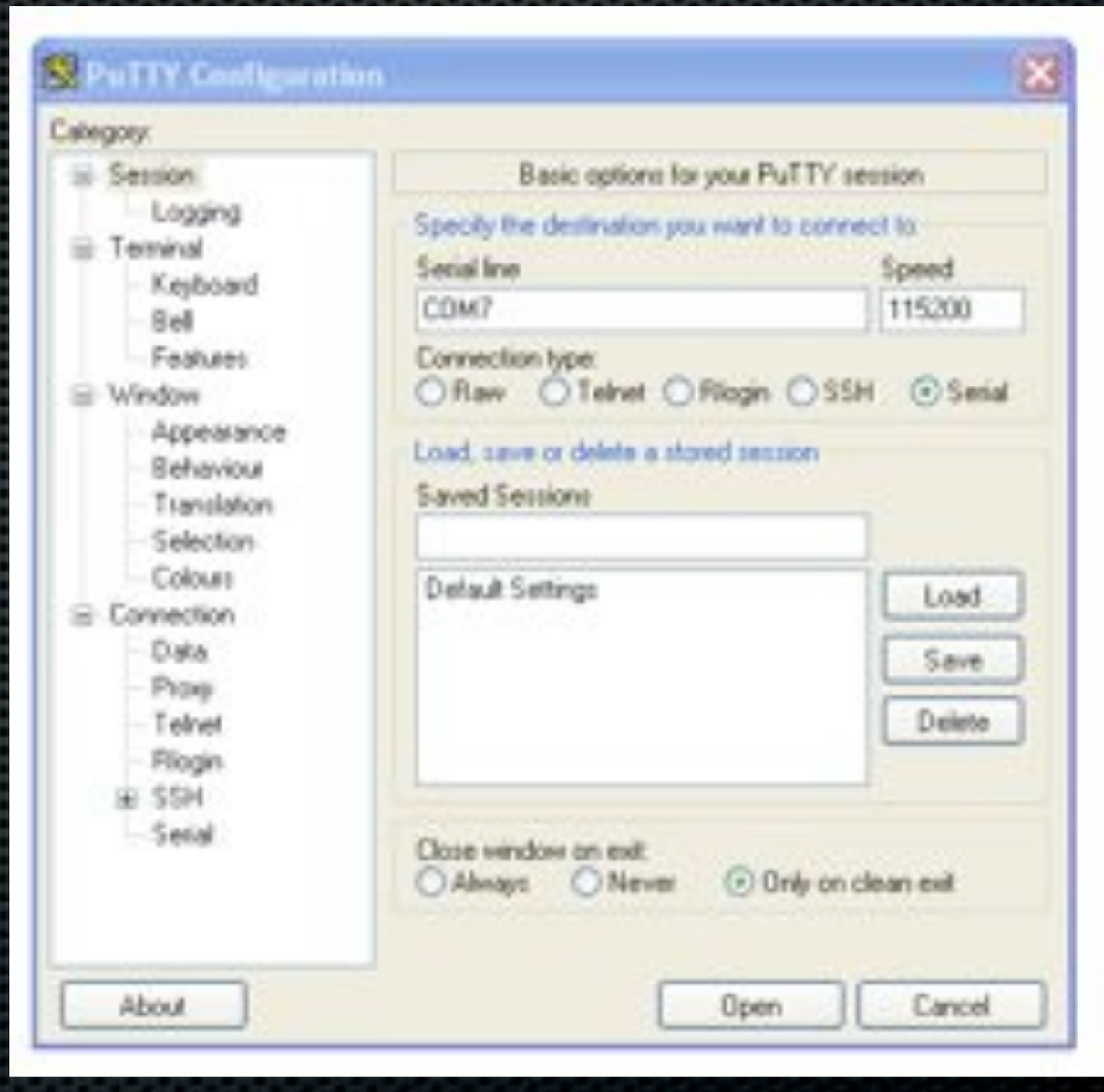
- Windows: http://www.prolific.com.tw/US/ShowProduct.aspx?p_id=225&pcid=41
- MacOSX Lion: http://changux.co/osx-installer-to-pl2303-serial-usb-on-osx-lio

Connect

- Mac: screen /dev/cu.PL2303-00001004 115200
  - the actual device name can be different, use tab completion to get to it
- Linux: sudo screen /dev/ttyUSB0 115200

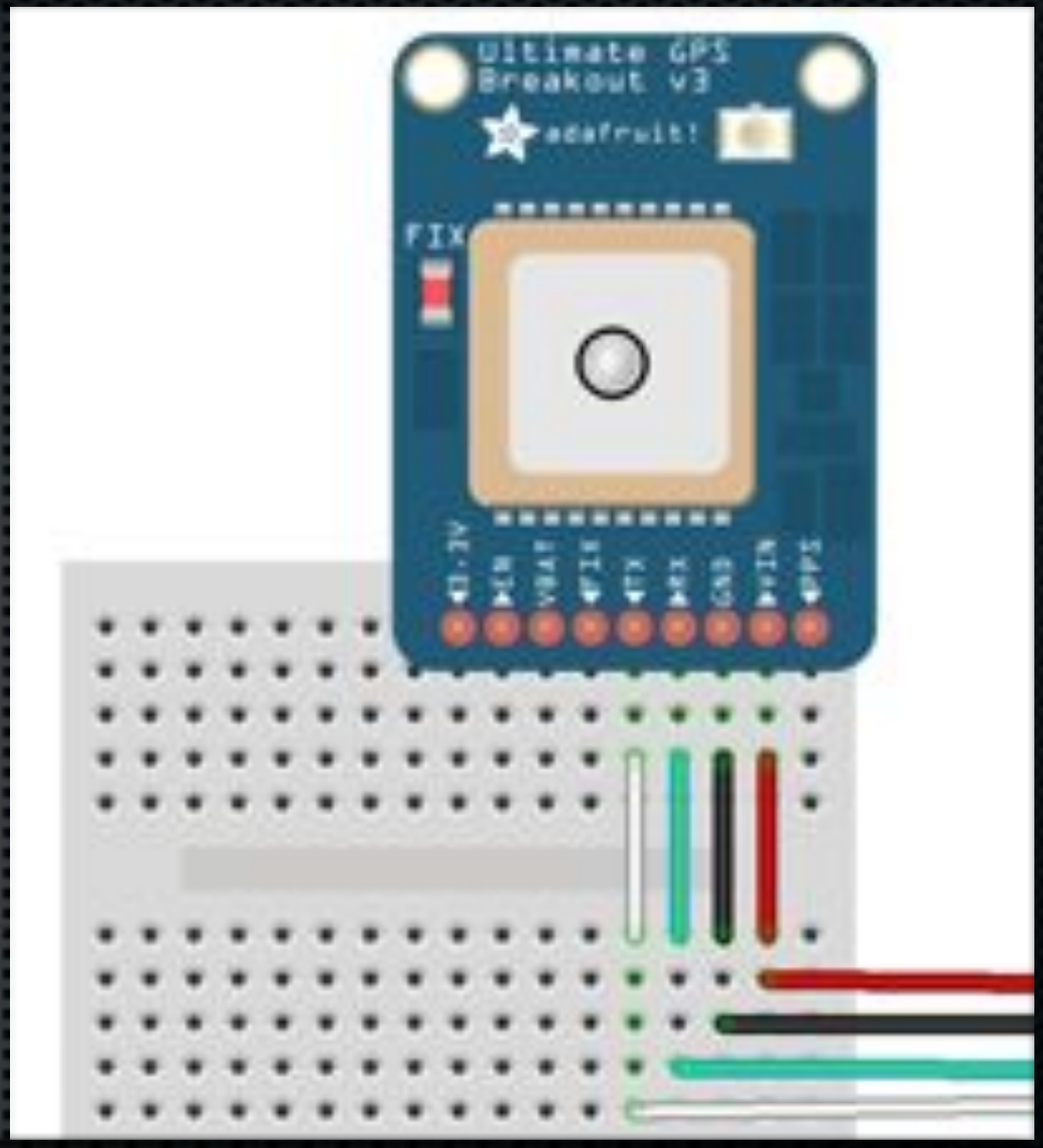
Connect on Windows





GPS Receiver

- This also communicates using SPI
- Easy installation with USB-TTL cable, USB driver and *gpsd* daemon program



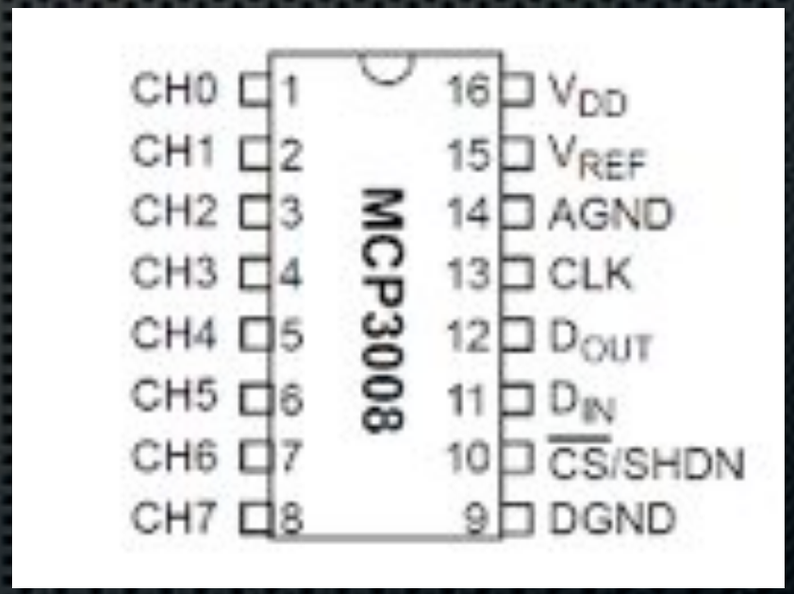
MCP3008

- The Raspberry Pi has no analog inputs
- This is easily remedied with an MCP3008
- 8-channel AD converter
- 0-3.3V becomes a 10bit value (0-1023)
- Communicates using SPI protocol

MCP3008 Connections

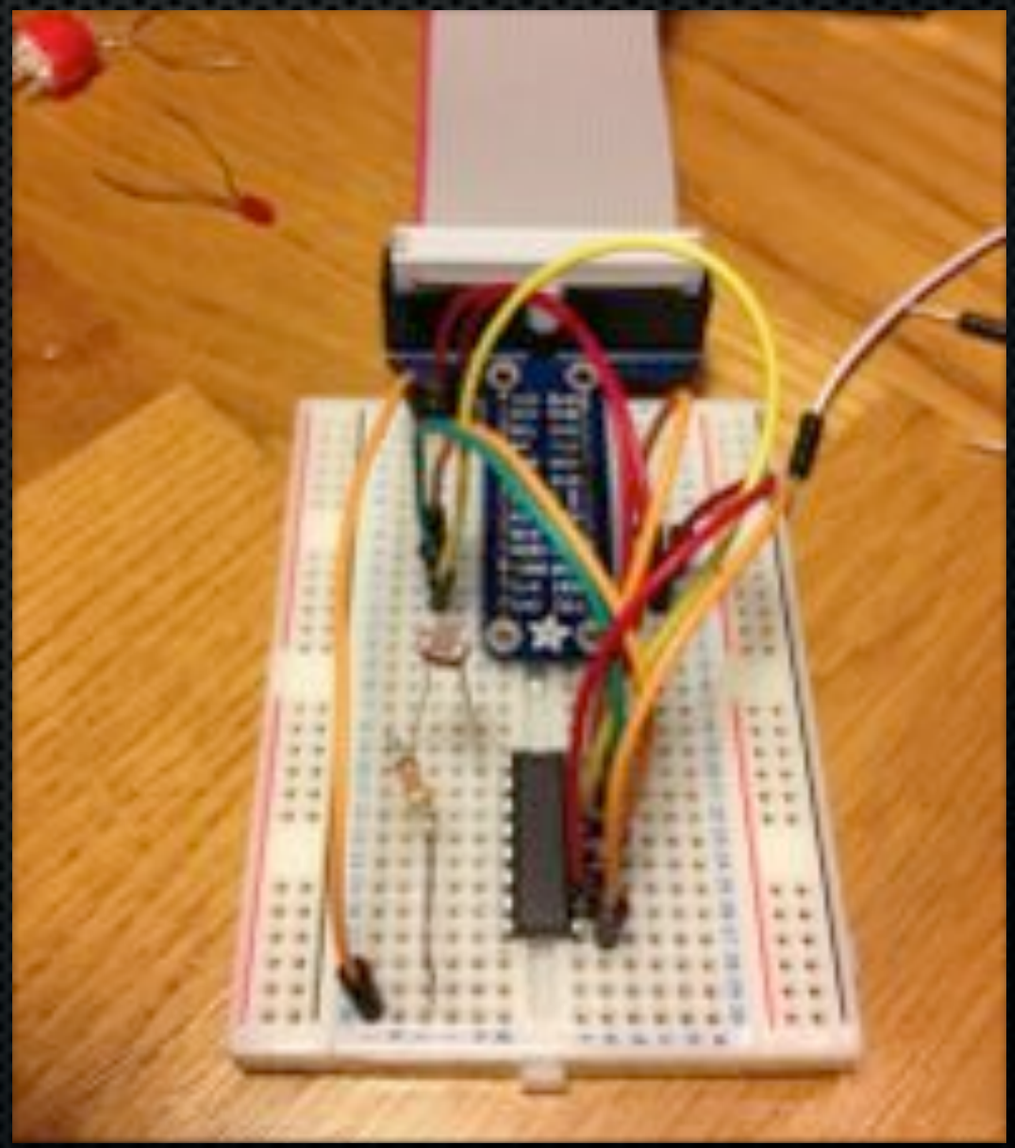
The connections from the cobbler to the MCP3008 are as follows:

- MCP3008 VDD -> 3.3V (red)
- MCP3008 VREF -> 3.3V (red)
- MCP3008 AGND -> GND (orange)
- MCP3008 CLK -> SCLK (yellow)
- MCP3008 DOUT -> MISO (green)
- MCP3008 DIN -> MOSI (yellow)
- MCP3008 CS -> CE0 (red)
- MCP3008 DGND -> GND (orange)



Analog-to-digital

- Here an LDR (Light Dependent Resistor) is attached to channel 0 of the MCP3008
- This continuously (200kps) samples its resistance
- Use an SPI program to read the channel(s)



Enable SPI driver

- In Debian Wheezy, the spi and i2c modules are currently disabled (blacklisted)
- Edit: /etc/modprobe.d/raspi-blacklist.conf
- # blacklist spi and i2c by default (many users don't need them)

  #blacklist spi-bcm2708

  blacklist i2c-bcm2708

SPI Devices

- This gives you:
  - $ ls -al /dev/spi*

```
crw------T 1 root root 153, 0 Jan  1  1970 /dev/spidev0.0
crw------T 1 root root 153, 1 Jan  1  1970 /dev/spidev0.1
```

  -

What is SPI

- Serial Peripheral Interface Bus
- A Motorola protocol for low-level serials comms
- Full Duplex
- Need to write in order to read
- De facto standard, not formalized

How to speak SPI

- Options in order of (NetRexx) preference:
  - Embedded Java driver
  - JNI interface to WiringPI library
  - Exec the WiringPI library
  - Exec the Python py-spidev calls via Python
  - Exec bit banged Python code

Tried these in reverse order

- To have something that works, anyway
- Python is not hard, has good PI support and py-spidev is easily installed using sudo apt-get install
- But we would not be caught depending on Python
- WiringPi is a standard solution with easy call interface

Bit Banged Python code

- Without the SPI device support in the hardware (some very early Raspberry Pi revision), the code can bit-bang the GPIO ports
- I am not sure I want to understand how this works, although I tried it (running, not understanding) and it does work
- Apparently the JVM performance tests were inspired by the desire to bit-bang as fast as possible

SPI from Python

- The Raspberry Pi primarily supports SPI through Python
- One can exec a Python command and read the output

Exec a wiring-pi call, C glue

```c
#include <stdio.h>
#include <stdint.h>
#include <wiringPi.h>

// read SPI data from MCP3008 chip, 8 possible adc's (0 thru 7)
int readadc(adcnum)
{
  uint8_t buff[3];
  int adc;
  if ((adcnum > 7) || (adcnum < 0))
    return -1;
  buff[0] = 1;
  buff[1] = (8+adcnum)<<4;
  buff[2] = 0;
  wiringPiSPIDataRW(0, buff, 3);
  adc = ((buff[1]&3) << 8) + buff[2];
  return adc;
}
```

```c
int main(int argc, char *argv[])
{
  int chan;
  uint32_t x1;

  if (wiringPiSPISetup (0, 1000000) < 0)
    return -1 ;

  if (argc>1)
    chan = atoi(argv[1]);
  else
    return 99;

  x1 = readadc(chan);

  printf("%d\n", x1) ;
  return 0 ;
}
```

To make a JNI lib out of this

```c
#include <stdio.h>
#include <stdint.h>
#include <wiringPi.h>
#include "jnispi.h"

// read SPI data from MCP3008 chip, 8 possible adc's (0 thru 7)
JNIEXPORT jint JNICALL Java_jnispi_readadc(JNIEnv *env, jobject obj, jint adcnum)
{
  uint8_t buff[3];
  int adc;
  if ((adcnum > 7) || (adcnum < 0))
    return -1;

  buff[0] = 1;
  buff[1] = (8+adcnum)<<4;
  buff[2] = 0;
  wiringPiSPIDataRW(0, buff, 3);
  adc = ((buff[1]&3) << 8) + buff[2];
  return adc;
}

JNIEXPORT jint JNICALL Java_jnispi_init(JNIEnv *env, jobject obj)
{
  if (wiringPiSPISetup (0, 1000000) < 0)
    return -1 ;
}
```

jnispi.h is generated

- use javah jnispi > jnispi.h
- this generates the required native method signatures from the jnispi.class file

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class jnispi */

#ifndef _Included_jnispi
#define _Included_jnispi
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     jnispi
 * Method:    readadc
 * Signature: (I)I
 */
JNIEXPORT jint JNICALL Java_jnispi_readadc
  (JNIEnv *, jobject, jint);

/*
 * Class:     jnispi
 * Method:    init
 * Signature: ()I
 */
JNIEXPORT jint JNICALL Java_jnispi_init
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

To build jnispi.so

```
NRX_COMPILE_COMMAND = java -Dnrx.compiler=ecj org.netrexx.process.NetRexxC
NRX_COMPILE_FLAGS = -comments -sourcedir -time -keepasjava -replace -warnexit0 -compact -nocrossref -utf8

JAVA_COMPILE_COMMAND = java org.eclipse.jdt.internal.compiler.batch.Main
JAVA_COMPILE_FLAGS = -warn:-unusedPrivate -warn:-unusedLocal -Xemacs

.nrx.class:
        ${NRX_COMPILE_COMMAND} $< ${NRX_COMPILE_FLAGS}
        ${JAVA_COMPILE_COMMAND} ${JAVA_COMPILE_FLAGS} $*.java

NRX_SRC          := ${wildcard *.nrx}
NRX_OBJS         := ${NRX_SRC:.nrx=.class}
JAVA_SRC         := ${wildcard *.java}
JAVA_OBJS        := ${JAVA_SRC:.java=.class}

CC      = gcc
INCLUDE = -I/usr/local/include -I/opt/java/jdk1.8.0/include/ -I/opt/java/jdk1.8.0/include/linux
CFLAGS  = ${DEBUG} ${INCLUDE} -Winline -pipe

LDFLAGS = -L/usr/local/lib
LDLIBS    = -lwiringPi -lpthread -lm


.SUFFIXES: .nrx .nry .njp .class .skel .xsl .java .pdf .so

#
# target all compiles the netrexx and java code
#
all::   OutputLineEvent.class OutputEventListener.class ${NRX_OBJS} ${JAVA_OBJS}
        gcc ${INCLUDE} ${LDFLAGS} ${LDLIBS} -o libapi.so -shared jniapi.c
```

To use it

```
class jnispi

  method jnispi()
    System.loadLibrary("spi") -- loads "libspi.so" on Linux

  method readadc(chan=int) native returns int
  method init() native returns int

  method main(args=String[]) static
    t = jnispi()
    i = t.init()
    say t.readadc(0)
```

Embedded Java

- does not currently work due to class file byte code release differences between JavaME and the Java 8 prerelease
- So this stopped one level before the ideal pure java solution, but a working JNI lib is no bad result
- The current solution has as an advantage that it (the JNI lib) does not introduce a dependency on a single JVM supplier (embedded Java does, there is only one)

Demo

- it worked ...