https://www.raspberry.ma/

ISSUE 28 - NOV 2014

Get printed copies at themagpi.com

The MagPi™

A Magazine for Raspberry Pi Users

FUZE BASIC
VoIP Server
Arduberry
Using Git
BitScope

Introducing OpenCV

Win £450 worth of prizes in our FUZE BASIC programming competition

ISSN 2051-9990

BY NC SA

Raspberry Pi is a trademark of The Raspberry Pi Foundation.
This magazine was created using a Raspberry Pi computer.

The MagPi

http://www.themagpi.com

This month's Issue is packed with hardware and programming articles. We are pleased to present the first article in an OpenCV (open source computer vision) image recognition software series by Derek Campbell. The robot that Derek used to test the software configuration is shown on this month's cover.

Expanding the I/O possibilities of the Raspberry Pi is often a first step of electronics projects. This time, Dougie Lawson presents a review of the Arduberry board from Dexter Industries. This little board provides an ideal microcontroller interface for more complicated electronics projects. This month's hardware articles are rounded off by Karl-Ludwig Butte's third BitScope article, which includes examples of preamplifier circuits and associated test and measurement.

The Raspberry Pi provides the opportunity to run many different software applications. Voice over IP (VoIP) allows telephone calls to be carried over an internet connection. Walberto Abad continues his mini-series by describing how to setup an Asterisk VoIP server.

The second application article this month continues the discussion of git (distributed version control system). Git was originally produced for Linux kernel development, but is now a mainstay of many different development projects and has been adopted by several schools too. Alec Clews leads us through his second tutorial on the subject.

This month's programming article demonstrates how to build an arcade game using FUZE BASIC. Jon Silvera includes instructions, code and images to build a horizontally scrolling game.

We are on the look out for more articles at all levels and on all subjects. If you are interested in submitting an article, please get in touch with us by emailing articles@themagpi.com.

If you have any other comments, you can find us on Twitter (@TheMagP1) and Facebook (http://www.facebook.com/MagPiMagazine) too.

Chief Editor of The MagPi

The MagPi Team

Ash Stone - Chief Editor / Administration
Ian McAlpine - Layout / Proof Reading
W.H. Bell - Issue Editor / Administration / Layout
Bryan Butler - Page Design / Graphics
Matt Judge - Website
Nick Hitch - Administration
Colin Deady - Layout / Proof Reading
Dougie Lawson - Testing
Nick Liversidge - Proof Reading
Martin Wolstencroft - Layout / Proof Reading
Mark Pearson - Layout
David Bannon - Testing / Proof Reading
Shelton Caruthers - Proof Reading
Rita Smith - Proof Reading
Claire Price - Proof Reading

Contents

4 **OPENCV**
Computer Vision on the Raspberry Pi

10 **ARDUBERRY**
Unite the Raspberry Pi and Arduino

14 **BITSCOPE**
Part 3: Electronic measurement with the Oscilloscope add-on board

20 **VOICE OVER IP**
Part 2: Connecting to the telephone network

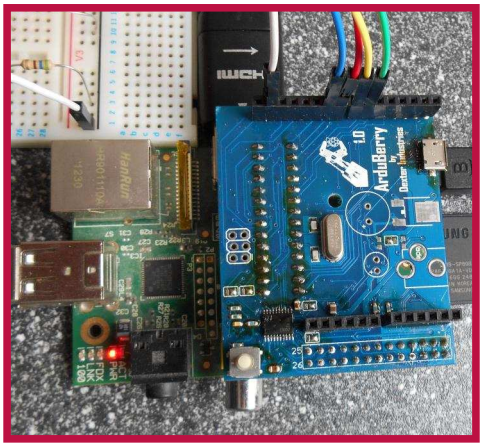
26 **VERSION CONTROL**
Part 2: What happens when you make document changes

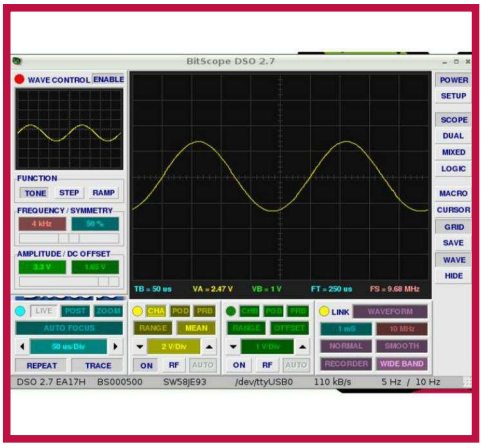
32 **FUZE BASIC**
Part 4: Font scaling plus we add the final touches to our game

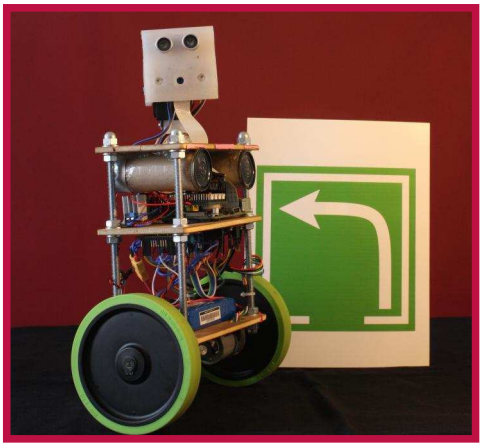
39 **COMPETITION**
Win one of three Raspberry Pi FUZE kits worth a total of £450 in our programming competition

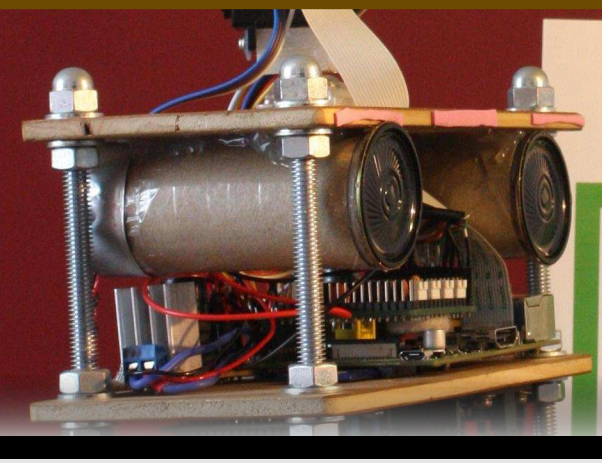
41 **THIS MONTH'S EVENTS GUIDE**
Mechelen Belgium, Berlin Germany, Cheltenham UK, Hull UK, Warwick USA

42 **HAVE YOUR SAY**
Send us your feedback and article ideas









Computer Vision on the Raspberry Pi

Derek Campbell

Guest Writer

SKILL LEVEL : INTERMEDIATE

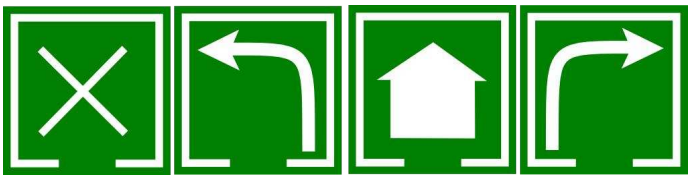
Meet Piter, my avatar robot project. He stands on two wheels and has a Raspberry Pi for brains. Piter has a Pi camera and hardware for making light and sound. His head has servos so that he can look around, and he is able to tell how far away he is from obstacles in his path.

This article describes how his camera helps him navigate. You don't need a robot to try out the ideas. All you will need is a Raspberry Pi and camera (a USB webcam will also work).

I'm trying to promote interest in robotics at my local cub scout pack. Cubs learn lots of techniques for use when they are out and about hiking. One of them is called 'tracking'. When one group of cubs needs to follow a trail left by another group, they look for signs left behind by the lead group to tell them where to go. The lead group leaves behind marks -- knotted grass or small groups of stones -- that tell the following group which way they went.

Piter does tracking

To simplify image recognition for the robot (spotting knotted grass is a bit beyond simple computer vision) we created some symbols that mimic the ones the cubs would use in the field, but that the robot could find and follow. The symbols are printed in green and look like this:



Robot tracking symbols
Turn Back, Turn Left, Home, Turn Right

How Piter sees

Piter follows the symbols by taking the following approach:

First of all he looks for a patch of green colour in his field of view. He drives towards the largest matching patch until he is close enough to recognise the symbol. Once the symbol is recognised the robot follows the instruction and then looks for the next patch. He repeats this until he reaches the symbol telling him he has reached his goal.

To do this, Piter uses the onboard Raspberry Pi and camera and processes the real time images using an awesome open source computer vision library called OpenCV.

OpenCV runs on all major operating systems and provides all the tools needed to locate and

decode the symbols. I could have sent the camera images to a host computer and done the OpenCV processing there, returning the results to the Raspberry Pi. However, that would mean the robot would not be truly autonomous.

Therefore I chose to implement the camera processing in the Raspberry Pi on board the robot. This did mean that I had to think a bit more about performance. In the end I achieved a processing rate for live images of about 2 to 3 frames per second. Not enough to play tennis, but easily enough to find and follow cub scout tracking signs.

Let's get started...

Installing OpenCV has a couple of steps, and instructions for doing so are included at my robot web site https://github.com/Guzunty/Pi/wiki/PiTeR-and-OpenCV.

To allow OpenCV to read images from the Raspberry Pi camera, we also need a utility called uv4l (Userspace Video for Linux). Instructions for this are also found on the web site.

Having installed these two, you can try out the programs. You may find it best to clone the whole Guzunty/Pi section of the link below, then explore the tree to study the programs. I used Python for the programs. The first demo shows how to capture images to make them available for further processing. The Git link is https://github.com/Guzunty/Pi/tree/master/src/gz_piter/MagPi

Program 1 - imageCap.py

```
import cv2
cap = cv2.VideoCapture(-1)
if (not cap.isOpened()):
  print("Cannot open camera")
else:
  while True:
    success, frame = cap.read()
    cv2.imshow("Captured:", frame)
    if (cv2.waitKey(1) == 27):
      break
cap.release()
cv2.destroyAllWindows()
```

Code walkthrough

To read an image from the Raspberry Pi camera we must first open the camera inside OpenCV.

```
import cv2
cap = cv2.VideoCapture(-1)
```

Next, we want to read a frame from the camera.

```
success, frame = cap.read()
```

It's that easy! Now, let's show the image in a window to make sure we really got a picture.

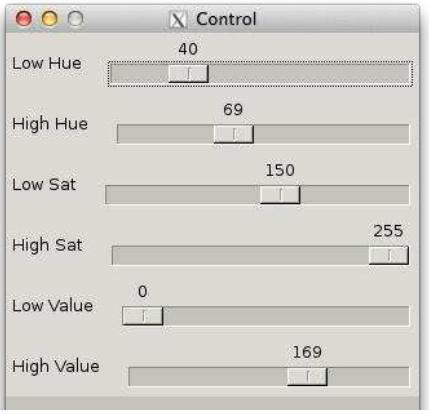
```
cv2.imshow(frame)
```

So now we have a picture. Each frame is a single bitmap in the stream of video data coming from the Pi camera. Within that frame we need to locate the symbol, so we hunt for the colour.

Program 2 - colour.py

The first step is to look for a patch of a particular colour. How do we do that? Fortunately OpenCV has a built in function to do just that. It's called `inRange()`.

The program `colour.py` displays six sliders to control the range of the colour values: Hue, Saturation and Value. We use these values rather than the more familiar Red, Green and Blue because they are more suited to setting up a colour range. Each colour value has a high and low which mark the ends of the range that will be accepted as the part of the image we're looking for using the `inRange()` function.



```
imgThresholded = cv2.inRange(imgHSV, (lowH,
lowS, lowV), (highH, highS, highV))
```

Here, low'x' and high'x' are two tri-valued colour variables that represent the ends of a range of colours between which OpenCV will accept as being part of the patch. Why do we need a range?

Well, a symbol in the real world has all sorts of things affecting how it looks. There may be shadows or reflections falling across it, and different lighting will make the actual colour received at the camera differ slightly from the ideal. Accordingly, we use `inRange()` to accept a range of colours so that we ignore these effects.

What about the returned value, `imgThresholded` (aka 'mask')? This variable contains another image, but unlike the original frame it is black and white. The white parts represent the parts of the image that OpenCV thinks are part of our colour patch and the black parts are thought NOT to be. We will use this mask later on.

Choosing the correct colour range is critical for successfully finding the patch. If the range is too narrow then the mask will not show all of the patch or might even be all black. If the range is too wide then the mask will include parts of the scene that are not part of the patch or might even be all white.

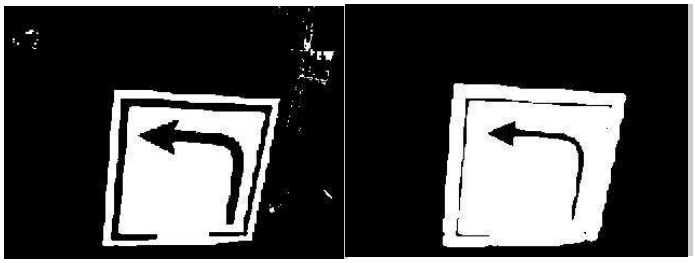
Once we have used this program for the lighting situation in which we are going to operate, the mask image will look like the top right quadrant of the figure on the left. We make a note of the high and low colour values and encode them into the actual runtime program as `lowColour` and `highColour`.

Cleaning up the mask

When using the `cv2.inRange()` function, the resulting mask can be very grainy unless you have very good lighting. OpenCV has two functions that can help with this: `erode()` and `dilate()`.

The `erode()` function shrinks the mask by ignoring patches that are smaller than a passed reference shape. The `dilate()` function does the opposite. It expands the mask so that islands in the mask fuse together. A passed shape is used the way a shaped paint tool is used in a painting program like Gimp.

We call `erode()` first to remove isolated spots in the mask, and afterwards call `dilate()` to fuse the remaining pieces into a nice solid mask.



Left: before erode() *Right: after dilate()*

Using the image to navigate

Ok, so now we have a bitmap which shows the part of the scene where the symbol is found. This is nice, but it's not very helpful if we are going to drive the robot closer to the symbol in the real world. It would be better if we knew the coordinates of the centre of the patch. That way, if the x value is in the left hand side of the image we can steer left to centre it.

OpenCV has just the thing. To find the centre of

the patch, we use a function called `findContours()`. You know what contour lines on a map are, right? They are lines that join points that are at the same height above sea level. These OpenCV contours represent a path through the bits in the mask image which have the same colour value.

Program 3 - tracking.py

In `tracking.py` we pass in the mask we found using `inRange()` and we get back a set of contours found in the 'mask' (aka. `imgThresholded` in program 2).

```
contours, hierarchy = cv2.findContours(mask,
RETR_TREE, CHAIN_APPROX_SIMPLE)
```

Why do we get more than one contour? Well, if we didn't get our colour range perfect, there will be some holes or islands in the mask. OpenCV can't know which contour is the actual part of the image we want, so it returns all the contours it finds.

But which one do we need? No worries - Since we tuned our colour range to find our patch, we can assume that the contour with the largest area houses our patch.

```
# Find the contour with the greatest area area
= 0.0 contour = None for candidateContour in
contours:
candidateArea =
 cv2.contourArea(candidateContour)
if candidateArea > area:
  area = candidateArea
  contour = candidateContour
```

After executing this loop, the contour variable holds the contour which surrounds our patch in the robot's field of view. Now we can find the co-ordinates of the centre:

```
m = cv2.moments(c)
centroid =
 (m['m10']/m['m00'],m['m01']/m['m00'])
```

We call it a centroid as opposed to a centre because the contour is almost certainly not a regular shape.

Armed with our centroid we can use the x value to steer the robot left and right until we get close enough to the symbol to identify it. We can tell when we are close enough when the the patch gets to a certain size:

```
x, y, w, h = cv2.boundingRect(contour)
```

This will give us the size of the patch as it appears to the robot, so now we know when to stop driving forward.

Which symbol did we find?

Next, we need to decide what symbol we are stopped in front of. To do this we use OpenCV's feature detection and object recognition support.

First of all, we need a grayscale image. To do this I split out the red channel from the original frame. Because the symbols are printed in green, the red channel will show the symbol in the strongest contrast (a bit like what happens with old fashioned 3D glasses).

Sample image in red and green channels

We also do some auto exposure on the image to make it as clear as possible for the recognition process.

```
image =
 cv2.equalizeHist(cv2.split(frame)[RED])
```

Next, we crop out the part of the image that contains the patch (x, y, h and w come from the bounding rectangle we got from the contour).

```
image = image[y:y+h, x:x+w]
```

Now, we'll use an OpenCV SURF feature detector to detect key points in the image. Key points are parts of the image which contain

corners and other significant features. We extract them using the function `detectAndCompute()`.

```
keyPoints, descriptors =
 detector.detectAndCompute(image, None)
```

The descriptors are data structures which tell more about the key points, such as the orientation of a corner or edge. We don't need to know anything about this information because OpenCV also provides tools for matching key point descriptors in different images. This is how we decide which symbol we are looking at. What we do is to provide the program with a perfect image of each symbol. We run the feature detector on the sample images and compute their key points too.

```
symbol = cv2.imread(fileName)
symbolKeypoints, symbolDescriptors =
 detector.detectAndCompute(symbol, None)
```

Since the ideal images don't change, we only need to compute these key points and descriptors once.

Now, armed with descriptors for our ideal images we run an OpenCV matcher on the two sets of key points.

```
FLANN_INDEX_KDTREE = 0

index_params = dict(algorithm =
 FLANN_INDEX_KDTREE, trees = 5)

search_params = dict(checks = 50)

matcher = cv2.FlannBasedMatcher(
 index_params, search_params)

matches = matcher.knnMatch( symbolDescriptors,
 descriptors, 2)
```

We rank the matches by how closely they match between the images and only take the best ones.

```
good_matches = []
for match in matches:
  if match[0].distance < match[1].distance *
0.7:
    good_matches.append(match)
```

The matches we have left define how closely the ideal image resembles the actual piece of the robot's view. We repeat the match process for each candidate symbol. The more matches we get, the more likely the symbol we're looking at is the one in the ideal image.

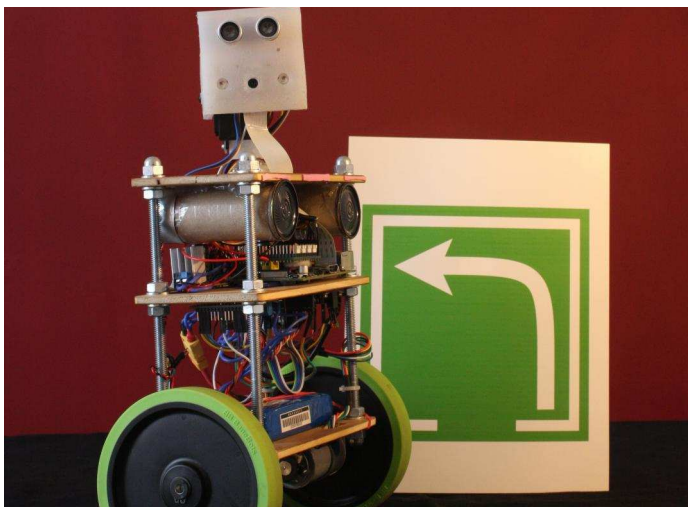
Matches between symbol and real scene

Next steps

Once we know which image matches, we can implement the code which makes the robot respond to the symbol, but that is a topic for another article.

All the code for Piter is available at https://github.com/Guzunty/Pi/tree/master/src/gz_piter. Remember, you don't even need a robot to try it out.

You can also get involved at the Raspberry Pi forums where the author will be hosting a thread for PiTeR in the Automation, sensing and Robotics category.



Wyliodrin

Just drag & drop blocks to create your applications, using Visual Programming

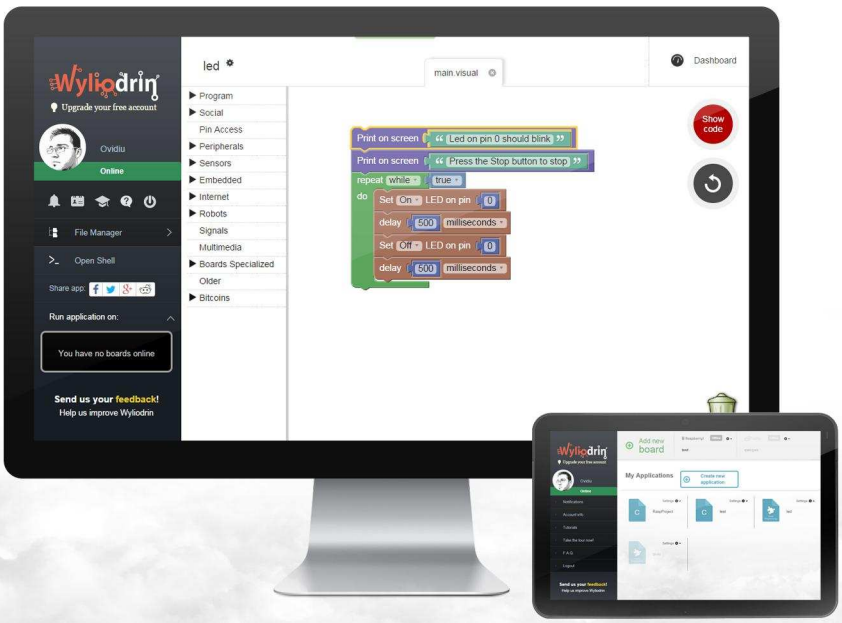
Program and monitor your Pi from anywhere in the Internet

Use our graphs to display your sensors' data

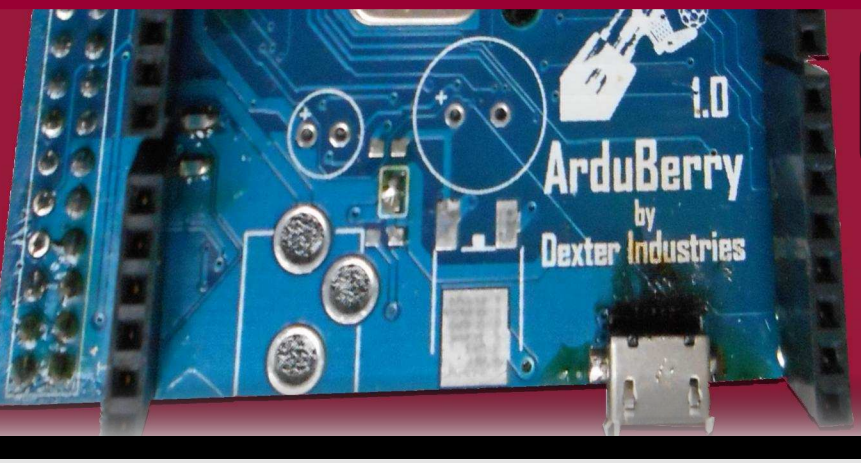
Use a browser from any device to program and monitor your Raspberry Pi

For **advanced** users **Wyliodrin** has **multiple** programming languages and **shell** access

Java · python · C++ · ARDUINO · php · Perl · node.js



www.wyliodrin.com



Introducing the Arduberry

Dougie Lawson

MagPi Writer

SKILL LEVEL : BEGINNER

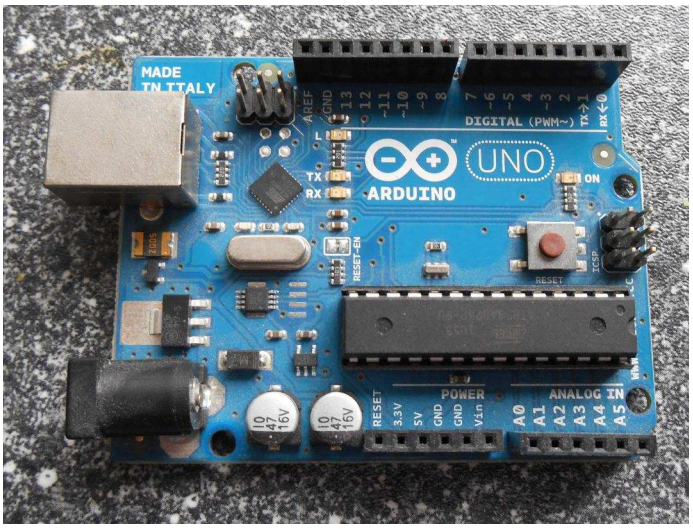
Before I start I will give you two crucial definitions;

1) Micro-controller - an autonomous, single processor computer system that does not have an operating system, has general purpose I/O (GPIO), programmable memory (flash & RAM) and some read-only memory. It will normally run a single custom made, dedicated program for a single purpose.

2) Micro computer - a single or multiple processor system that runs an operating system. It may have external GPIO and it usually has a keyboard and screen. It will normally run a multi-programming, multi-tasking system and a variety of general purpose application programs, plus some locally written programs. The crucial difference is that a micro computer appears to be running more than one task at a time while a micro-controller runs one task only.

Since its introduction in 2005, the Arduino has become a very popular micro-controller used by hobbyists. It is cheap, it is easy to program, it is versatile and well supported. The single task application program running on an Arduino is typically called a "sketch". Arduino has a standard layout of GPIO pins so there is a massive collection of add-on hardware available. The additional hardware that fits an Arduino is typically called a "shield".

Compare that with the Raspberry Pi which is a very popular micro computer used for education, by hobbyists and a whole variety of other folks for a multitude of tasks. They are from the same stable, built by their developers for similar reasons.



The Raspberry Pi and the Arduino may be suited to different projects but they work well together. It is quite common for the Raspberry Pi with the Arduino development IDE to be used to develop sketches that can be uploaded to an Arduino. The starting point is:

```
sudo apt-get install arduino
```

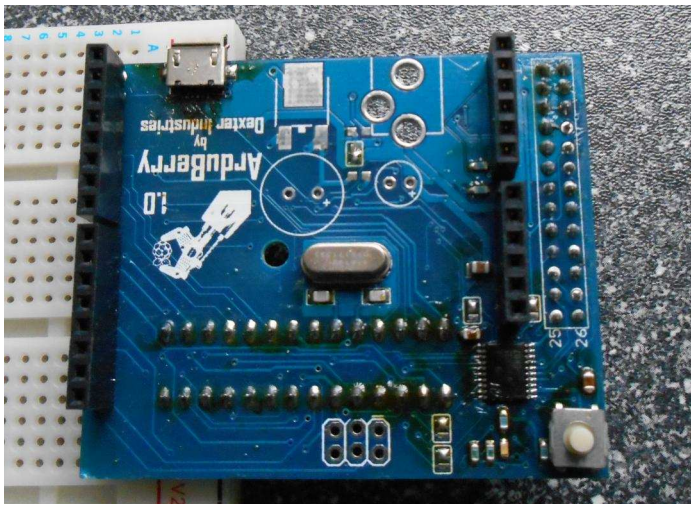
When that is complete. connect your Arduino with a USB-A to USB-B cable, start the GUI and write your first application.

Here is a variation on the normal blink program that will light four LEDs in sequence:

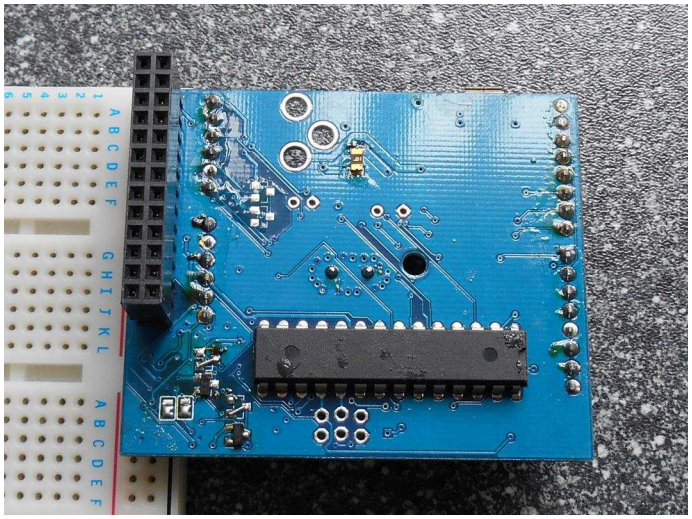
```
/* blinkSeq.ino
  A variation on "Hello World" for Arduberry
 (C) 2014 Dougie Lawson,
 Creative Commons v4.0 BY-NC-SA
 Wire four LEDs with 560 ohm
 resistors to pins 6, 7, 8 & 9         */

void setup() {
  for (int i=6; i<=9; i++) {
    pinMode(i, OUTPUT);
  }
}
void loop() {
  for (int i=6; i<=9; i++) {
    digitalWrite(i,LOW);
    delay(999);
    digitalWrite(i,HIGH);
    delay(999);
  }
}
```

Wire four LEDs, each with a 560 ohm current limiting resistor between pin 6, pin 7, pin 8 and pin 9 and GND on your Arduino Uno. Upload the sketch and the LEDs will blink.

So we have the basics of using Raspbian with the GUI to program an Arduino. The only thing that is cumbersome is the wiring and using up a USB socket on the Raspberry Pi (or in our USB hub). Wouldn't it be good if we could get an Arduino (primarily as a development tool) that sat on the Raspberry Pi GPIO pins (in the way that a shield fits on an Arduino Uno)? That is exactly what the Arduberry from Dexter Industries does.



From the top the Arduberry looks like an Arduino Uno; there is the standard layout of sockets that are used for stacking Arduino shields. Note: this Arduberry is missing a few optional components - there is no 5V power supply, voltage regulator, capacitors or header pins for the ICSP connector.



From the bottom you see that there is a standard 26-pin GPIO connector and the heart of the Arduino, the Atmel Atmega328P chip, down there. The Arduberry will sit on a Raspberry Pi model B or model B+. When it is mounted the Arduberry gets connections to the Raspberry Pi 5V, 3V3, GND, UART TX & RX, SPI SCLK, MOSI, MISO & CE0, I²C SDA & SCL pins. So you MUST shutdown your Raspberry Pi and pull the power before mounting an Arduberry on the GPIO pins. With a Raspberry Pi model B+ you MUST ensure that just the first 26 pins are connected and are lined up correctly. Failure to power down or line up the pins could destroy either your Raspberry Pi or your Arduberry or both.

We have now got the Arduberry mounted and we have rebooted Raspbian. There is still some work needed to get the Arduberry running. On the command line, enter:

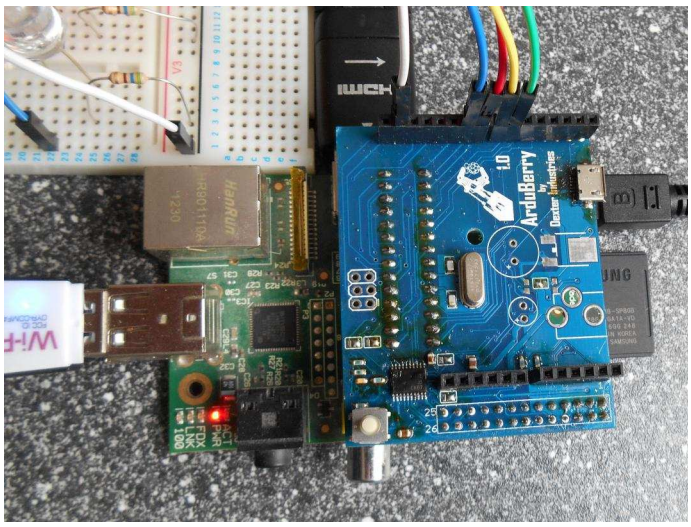
```
cd /tmp
git clone \
  https://github.com/DexterInd/ArduBerry.git
cd Arduberry
sudo chmod +x install.sh
sudo ./install.sh
```

Reply Y to the "Do you want to continue [Y/N]" prompt.

That `install.sh` script will install all the pieces you need (and have not yet installed), customise them for the Arduberry and when it is done it will reboot your Raspberry Pi.

If your Raspberry Pi is not set-up to use the GUI automatically, you will need to login as user `pi` and use `startx`. When the desktop appears there should be the new Arduino IDE icon (if not look in the start menu). Start the IDE and load the example blink sketch (`File --> Examples --> 01.Basics --> blink`) or type in my version. (Note that the example sketch `blink.ino` uses pin 13 since on many Arduinos there is a factory fitted LED on pin 13. This does not exist on an Arduberry so we need to add an LED with a 560 ohm resistor between pin 13 and GND to run the example `blink.ino` sketch.)

There is a slight quirk with the IDE when it comes to uploading the sketch. With a regular Arduino Uno connected, using a USB connection, we would click the `Upload` icon. With the Arduberry that does not work and generates an obscure error message; "avrdude: stk500_recv(): programmer is not responding". To upload the sketch we need to ensure we are using the right programmer options (`Tools --> Programmer --> Raspberry Pi GPIO`) then use the <CTRL>+<SHIFT>+<U> keys to make it so.



If we swap the LED on pin 9 for a red LED, the LED on pin 8 for a yellow LED and the LED on pin 7 for a green LED, this sketch will run the familiar sequence of traffic lights.

The Arduino programming language is a hybrid between C and C++ so you can use compiler directives. In this sketch if we change it to `undef UK` and `define US` it will change to the United States sequence (from red straight to green, skipping red and amber). Compiler directives are useful for debugging. You can define code that will only be built into the sketch when a true value or number greater than zero value is defined for a DEBUG compiler variable. I use that to litter my code with `Serial.print()` and `Serial.println()` commands. When the sketch works, update it to `#define DEBUG 0` and the debugging code will not be built into the final version.

```
/* TL.ino
  Traffic lights on the Arduino
  (C) 2014 Dougie Lawson,
  Creative Commons v4.0 BY-NC-SA   */
/* Use compiler directives to control */
/* the sequence */
#define UK 1
#define US 2
// RED, RED+AMBER, GREEN, AMBER RED ...
#define COUNTRY UK
// We're going to skip RED+AMBER
// RED, GREEN, AMBER, RED ...
//#undef COUNTRY
//#define COUNTRY US
int redLED = 9;
int yellLED = 8;
int grnLED = 7;

void setup() {
  pinMode(redLED, OUTPUT);
  pinMode(yellLED, OUTPUT);
  pinMode(grnLED, OUTPUT);
 }

void loop() {
  digitalWrite(redLED, HIGH);
  delay(1500);
 #if COUNTRY == UK
  digitalWrite(yellLED, HIGH);
  delay(750);
#endif
  digitalWrite(redLED, LOW);
  digitalWrite(yellLED, LOW);
  digitalWrite(grnLED, HIGH);
  delay(1500);
  digitalWrite(grnLED, LOW);
  digitalWrite(yellLED, HIGH);
  delay(750);
  digitalWrite(yellLED, LOW);
 }
```

One thing I found straight away with the Arduberry is that while the Raspberry Pi is powered up (even if Raspbian has been shut down with a `sudo shutdown -h` command) the Arduberry will continue to run. So I wrote a `null` sketch to effectively leave the Arduberry running dormant:

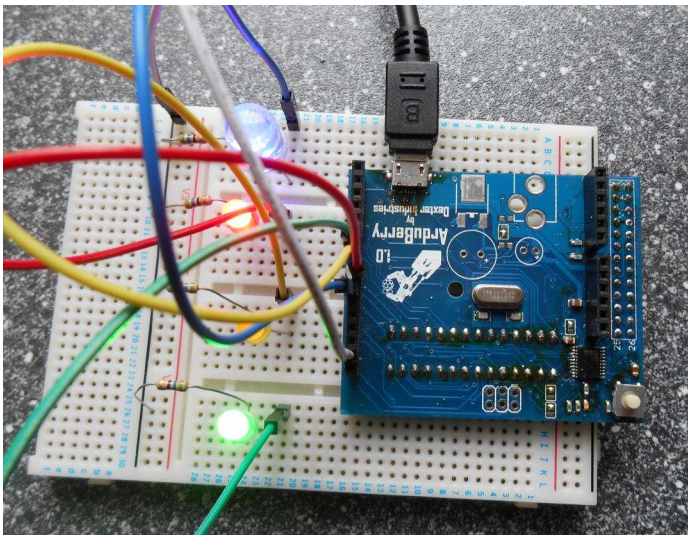
```
/* null.ino
  Null sketch so Arduberry runs dormant   */
void setup() {
}
void loop() {
}
```

With an Arduino Uno it is normal to pull the USB (or the 5V power) to shut it down, but that is less easy with the Arduberry since we have to shutdown Raspbian, pull the power, pull the board off the GPIO pins then reboot our Raspberry Pi. With the `null` sketch the Arduberry is still running the dummy `loop()` function but appears to be doing nothing to the outside world.

Another feature of the Arduberry is that we can upload a sketch while it is stacked on top of the Raspberry Pi, get it debugged and running the way we want then perform an "Arduberry Separation" to have the Arduberry running as a stand-alone system. We upload the sketch with <CTRL>+<SHIFT>+<U> while the Raspberry Pi is running, then shutdown the Raspberry Pi (logout from the GUI and choose the Shutdown option or open a command window and use a `sudo shutdown -h now` command). Pull the power supply from the Raspberry Pi because we do not want to damage either the Raspberry Pi or the Arduberry during separation. The Arduberry has a microUSB connector on the board, so we can borrow the Raspberry Pi's power supply to power our now independent Arduberry.

See https://www.dexterindustries.com/Arduberry/getting-started/powering-arduberry-dc-power-jack/ if you want to add the optional components to add a 5V power supply to the Arduberry.

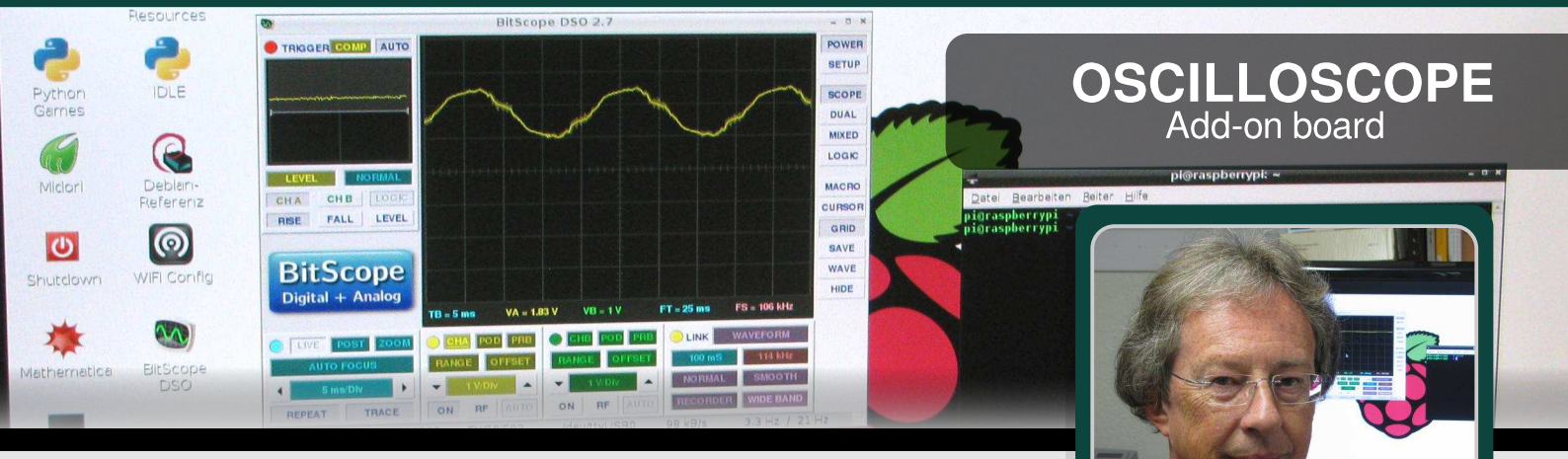
We can also develop code on the Arduberry, get it debugged and running, then upload it to a regular Arduino Uno. The Arduberry becomes a development system rather than a runtime system. The layout of the pins is identical, the programming code we write for the sketch is identical, the size of the file uploaded to the Arduino or Arduberry is identical. That may be an easier way to deploy our Arduino programs into the field where they are going to run, away from the Raspberry Pi.



The only problem I found with the Arduberry during my testing was that the header sockets are not labelled on the silk-screen printing on the Arduberry board. I kept having to refer to a wiring diagram http://forum.arduino.cc/index.php/topic,146315.0.html or stare at my Arduino Uno to be sure I was not wiring things to the wrong places. I am from the school of wire it, check it, check it again, power it up. I do not like wiring (or re-wiring) anything while it is running.

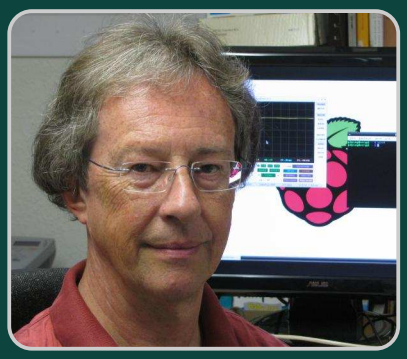
Since starting this article I have done some additional programming using I²C on the Arduberry. I pulled all of my I²C devices (DS1307 RTC, MCP23017 with an 8x8 LED matrix and LSM303DLHC accelometer magnetometer) off my Raspberry Pi and wired them to the Arduberry I²C pins. Programming these devices with C (gcc and Gordon Henderson's wiringPi) on the Raspberry Pi or the C/C++ hybrid and the Arduino IDE is interchangable. That may be a subject for a future article.

For anyone developing applications that use a stand-alone Arduino, or an Arduino connected to a Raspberry Pi, the Arduberry is an excellent piece of kit to add to your collection of small board micro computers and micro-controllers.



OSCILLOSCOPE
Add-on board

Electronic measurement with BitScope - Part 3



Karl-Ludwig Butte
Guest Writer

SKILL LEVEL : INTERMEDIATE

In part 1 and part 2 of this series, we measured DC and AC voltages. Now we are ready to use this knowledge and go bug-hunting in electronic circuits – the sort of things for which oscilloscopes are made.

Before we start...

Did you try to calculate the frequency our clock generator was running in our experiment in part2, Issue 26 of The MagPi? If so then you know that 200 micro seconds is 0.0002 seconds and according to part 1 in Issue 25 we only need to calculate 1 divided by 0.0002 to get our answer: 5000 Hertz (or 5 kHz).

The clock generator frequency

In our first experiment today we will replace the 1k resistor (R2) with a potentiometer (a resistor with adjustable resistance value), as shown in Fig. 1.

For this experiment you need:
- 1x 1k potentiometer or trimmer (Tr1)
- 2x copper wires (single-wire)
- soldering iron and tin-solder
- side cutter, tweezers, cable stripper in addition to the parts of the clock generator described in part 2 in Issue 26.

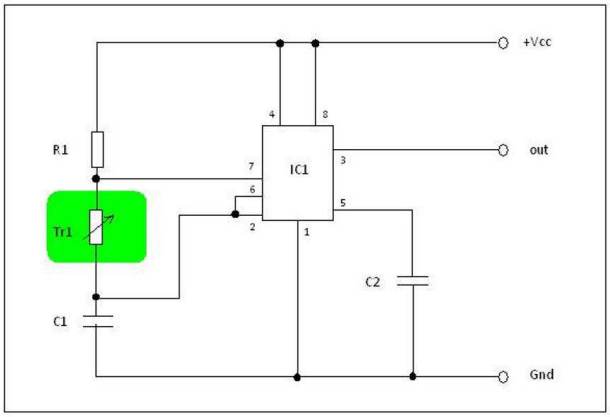


Fig. 1:
The change is marked with a green background and lets us adjust the frequency of the clock generator.

Disconnect all test leads from the board before you implement the change. First we need to prepare the potentiometer because its connection pins do not fit into the tiny holes of the breadboard. Strip about 5mm off the insulation at both ends of the copper wires with a cable stripper. If you do not have such a tool you may take a pair of scissors or a knife but be careful not to cut your fingers!

Because we need to connect two pins of the potentiometer together, one end of one copper wire needs to be stripped about 15 mm. Solder the wires to the pins of the potentiometer as shown in Fig. 2.

Fig. 2:
The 1k potentiometer with the copper wires soldered to its pins.

Fig. 3 shows the breadboard with the changed setup.

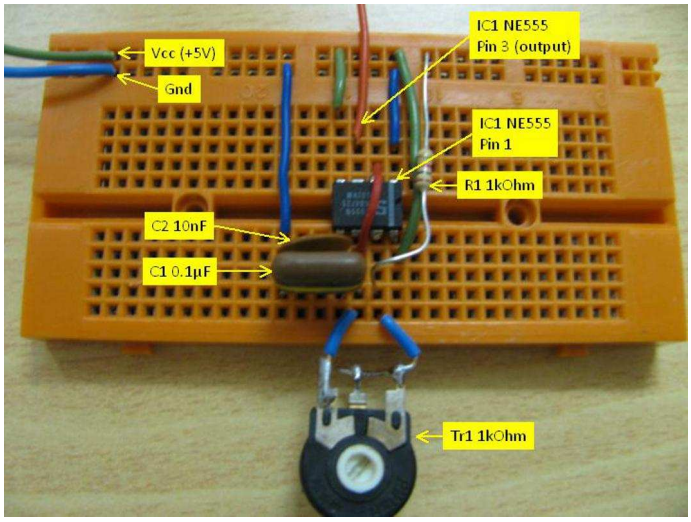


Fig. 3:
The clock generator with potentiometer Tr1 in exchange for resistor R2.

After you have finished the modification, reconnect all test leads as described in part 2 in Issue 26. Set the time base (6) to 50 µsec/Div and the Channel Control Pad for Channel A (7) to 2V/Div. In addition set the trigger controls (4) to MEAN. Now change the value of the trimmer Tr1 by turning the white slider in the middle of Tr1 with a small screwdriver. While doing so, look at the main screen (1) and observe the effect. The higher the frequency the more periods of the square wave you can see on the main screen (1). Now adjust the potentiometer to the lowest frequency and change the time base (6) to 500 µsec/Div. You can still see the square wave, but if you adjust the trimmer Tr1 to the highest frequency there is only a blur on the screen (1). This frequency is simply too much for this time base setting. Slowly decrease the time base setting (6) back to 50 µsec/div and observe how the picture gets clearer with every decrement.

You will have observed that the lower horizontal line of the square wave is much shorter than the upper horizontal line. Therefore changing the value of Tr1 not only changes the frequency but the ratio of the square wave, too. Usually clock generators are set to a 50% ratio, the so called duty cycle, so that the lower and upper horizontal lines of the square wave are equal in length. You need to change R1 and C1, in addition to Tr1, to change the frequency and keep the duty cycle at 50% at the same time. The Internet provides several sites with information about the NE555 timer IC and how to calculate the values for these parts.

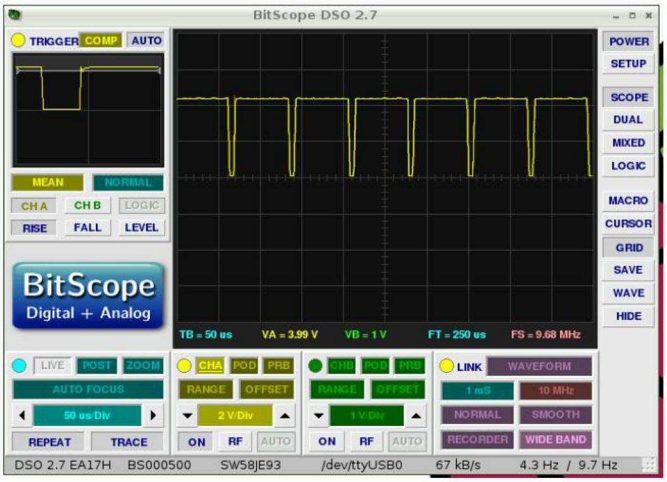


Fig. 4:
Output of the clock generator set to the highest frequency

Let's go bug-hunting

Recently I found a circuit diagram for a small single transistor pre-amplifier (see Fig. 5) and built it on a breadboard (Fig. 6). Please note the polarity of the two electrolytic capacitors. When I tried it out and checked its properties, I was amazed to find out that it clipped the upper curve of a sine wave. But let's see for yourself.

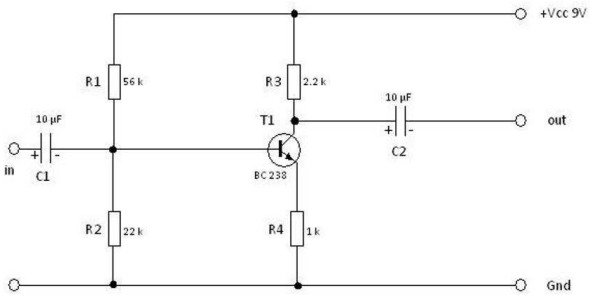


Fig 5:
Circuit diagram of a small one-transistor pre-amplifier.

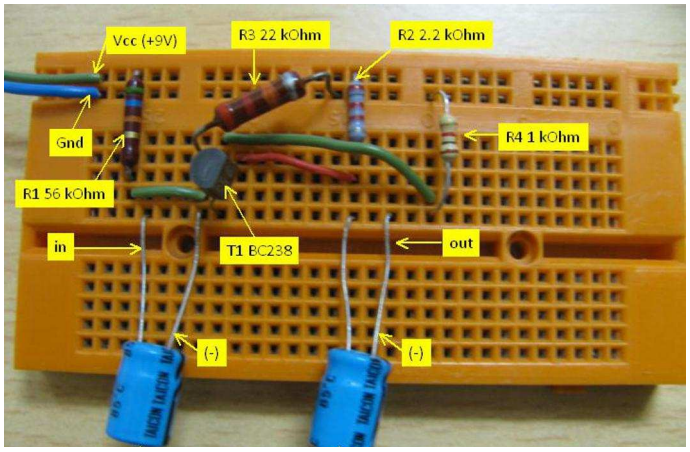


Fig. 6:
Single transistor pre-amplifier built on a breadboard

For this experiment you will need:
- 1x BC238 transistor (T1)
- 1x 56k resistor (R1)
- 1x 2k2 resistor (R2)
- 1x 22k resistor (R3)
- 1x 1k resistor (R4)
- 2x 10 µF/10V electrolytic capacitors (C1, C2)
- 1x 9V battery
- 1x small breadboard

Checking an amplifier with a sine wave is a wonderful idea but where do we get a sine wave? Luckily our BitScope Micro is well prepared for situations like this. It provides us with a full blown waveform generator, which we will use now. When you start up the waveform generator by clicking on the WAVE button on the right side, the WAVE CONTROL panel (Fig. 7) will appear on the left side.

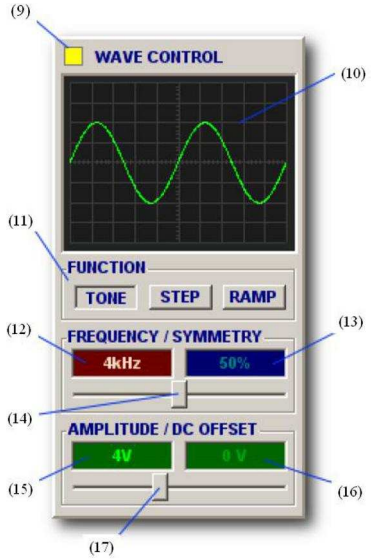


Fig. 7:
The WAVE CONTROL panel for the waveform generator (photo courtesy by BitScope Designs)

When the waveform generator is active, the control indicator (9) lights up and a preview of the generated waveform is displayed (10). With the wave function buttons (11) you can control the generated waveform. TONE generates a sine wave while STEP generates a square wave, like the clock generator we built in Issue 26. Click on these buttons to see the previews and get acquainted with the different waveforms.

If you think I forgot to tell you about the RAMP button, that was intended! If you click on the RAMP button you will instantly see why this waveform is called a ramp and spare me a lengthy description. There are two slider controls (14) and (17) which influence all the necessary parameters of a waveform. Slider control (14) adjusts the frequency (12) and the duty cycle (13) whereas slider control (17) influences the amplitude (15) and the DC-offset (16). For now the default values are OK for our purposes so we can start our first check of the pre-amplifier.

Fig. 8 shows all the necessary connections to and from the pre-amplifier. To help you get everything right I inserted the BitScope pin layout as a reminder and noted the colors of the test leads in brackets.

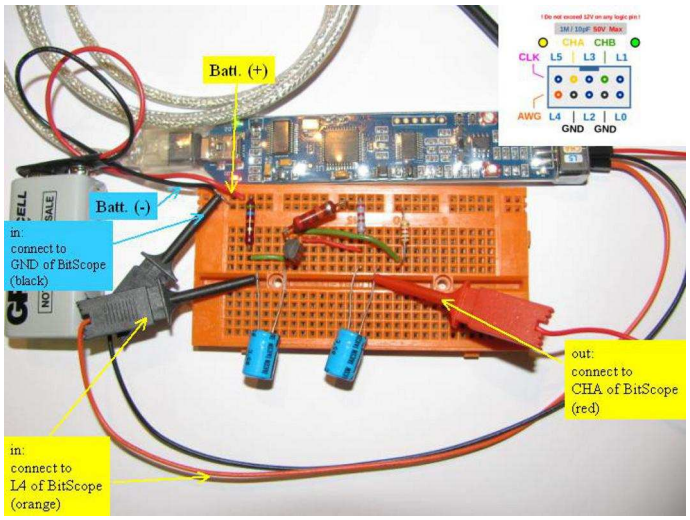


Fig. 8:
Connections to and from the pre-amplifier.

A few adjustments of the BitScope DSO software are still necessary. Please set the time base control (6) to 50 µsec/Div and the Channel Control Pad for Channel A (7) to 2V/Div. In addition, right click the OFFSET indicator of the Channel Control Pad A (7) and select MEAN on the context menu. The OFFSET indicator is located above the 2V/Div display on the right hand side (see Fig. 9). Now look at the result in the main display (1) and compare the waveform with the original in the preview area (10) of the WAVE CONTROL pad. There is definitely something wrong here. The upper curve of the sine wave is clipped!

Fig. 9:
Note the upper curve of the sine wave is clipped.

You may check the input signal by connecting the red test lead to the minus pole of C1. There the sine wave is still complete. Connecting the red test lead to the plus pole of C2 ensures us that this capacitor is innocent because it already gets a clipped signal. This leaves transistor T1 and resistors R3 and R4 as possible candidates for the sine-wave clipping.

The first suspicion is a misadjusted operating point for T1. The operating point is dependent on resistors R3 and R4, so let's try a few alternative values for R3. Perhaps we are lucky and find a value which works. This would spare us the tedious task of calculating all components of the circuit (a so called network analysis). You may try exchanging R3 with other resistors you have available before you read on.

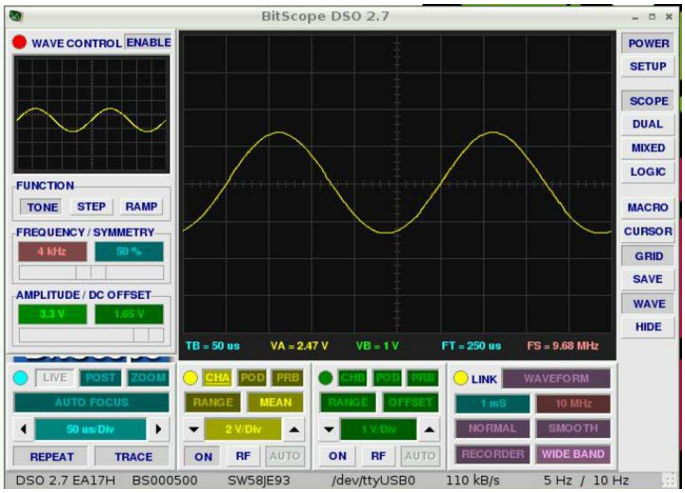


Fig. 10:
Correcting the working point of T1 is the solution.

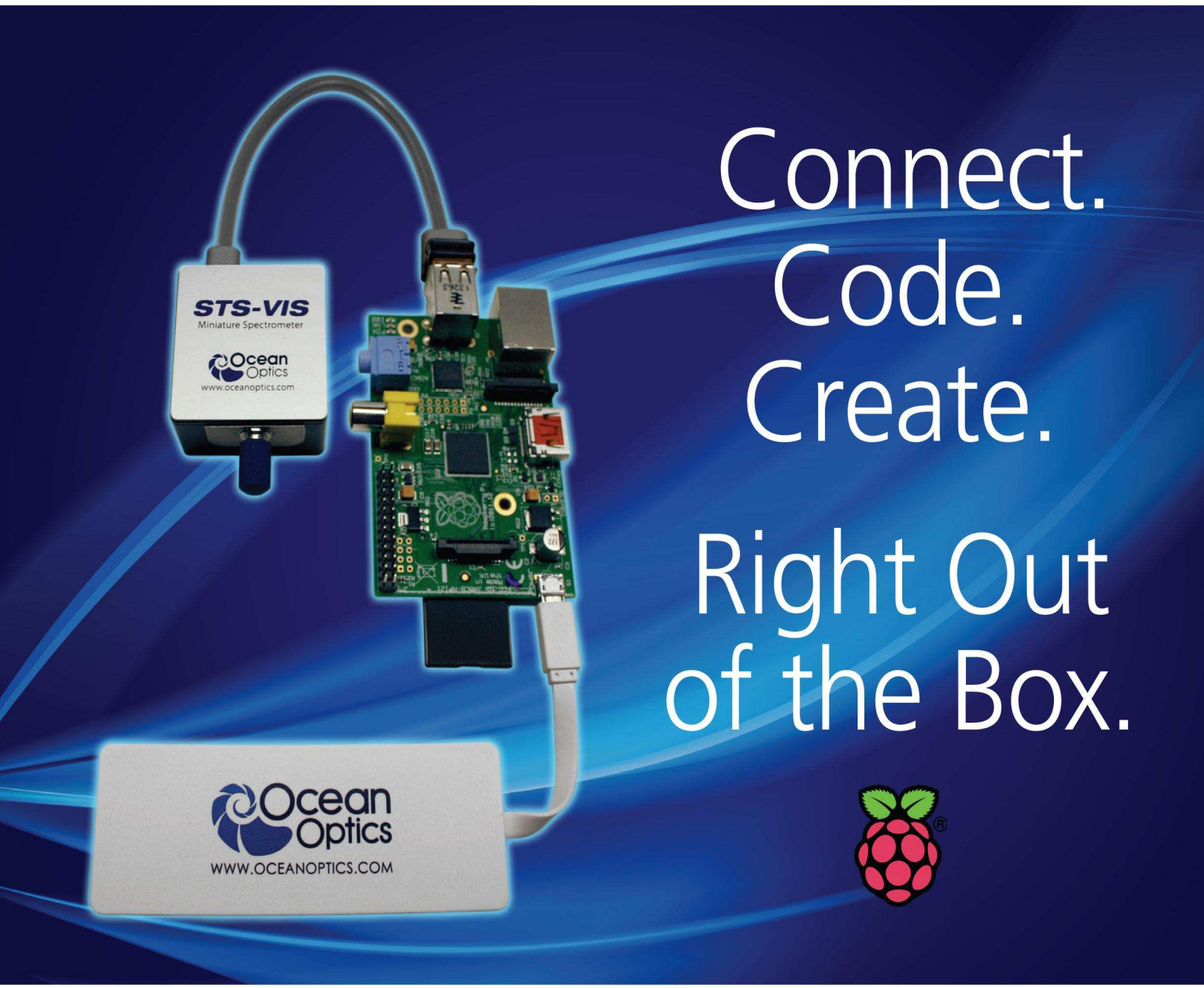
I found a value of 22k to work best, as you can see in Fig. 10. If your screen still shows a little clipping after you changed R3, keep in mind that resistors have some tolerances. The last colored ring on the resistor encodes this tolerance. A golden ring means 5% tolerance and a silver ring stands for 10%.

Conclusion

In this article you have used the BitScope Micro oscilloscope together with its built-in waveform generator to find a miscalculated resistor in a pre-amplifier circuit. The approach we used, measuring a signal in different parts of a circuit to identify a defective or miscalculated component, can be used universally for developing or repairing electronic devices.

In the three parts of this series I could only scratch the tip of the iceberg. I hope you had as much fun as I did. The BitScope Micro has much more to offer but this will have to wait for future issues of The MagPi.

For all those interested in turning their Raspberry Pi into a digital storage oscilloscope with the BitScope Micro add-on board, it is available from BitScope Designs in Australia at http://www.bitscope.com, in Germany at http://www.BUTTE-verlag.de, in the UK at http://shop.pimoroni.com and finally in Switzerland at http://www.pi-shop.ch.



Connect. Code. Create. Right Out of the Box.

Powered by Raspberry Pi, the STS Developers Kit is the best way to integrate spectral sensing into your application.

Measure color more accurately than the human eye, monitor solar UV levels, prototype compact medical self diagnostic tools or monitor crops from a UAV. The STS Developers Kit makes it easy with integrated device drivers and an easy to access API to get going quickly.

Ocean Optics

www.oceanoptics.com | info@oceanoptics.com | **US** +1 727-733-2447 **EUROPE** +31 26-3190500 **ASIA** +86 21-6295-6600

Expand your Pi

Stackable Raspberry Pi expansion boards and accessories

ADC-DAC Pi

2x 12 bit analogue to digital channels and 2x 12 bit digital to analogue channels.

Serial Pi

RS232 serial communication board. Control your Raspberry Pi over RS232 or connect to external serial accessories.

ADC Pi

8 channel analogue to digital converter. I²C address selection allows you to add up to 32 analogue channels to your Raspberry Pi.

1 Wire Pi

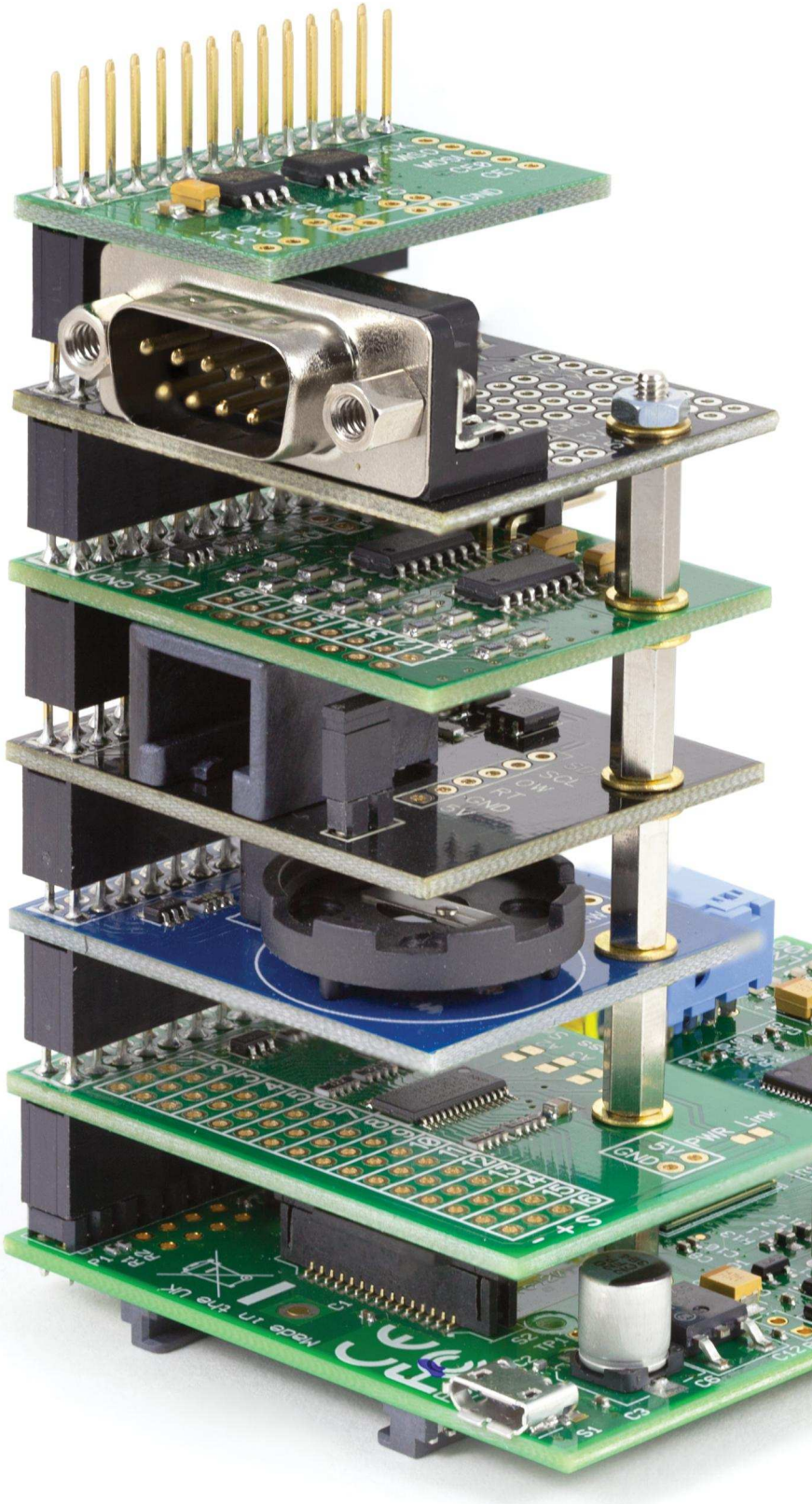
1-Wire® to I²C host interface with ESD protection diode.

RTC Pi

Real-time clock with battery backup and 5V I²C level converter for adding external 5V I²C devices to your Raspberry Pi.

Servo Pi

16-channel, 12-bit PWM controller suitable for driving LEDs and radio control servos.



AB electronics UK

www.abelectronics.co.uk

Using Asterisk to implement a low cost phone system - Part 2

Walberto Abad

Guest Writer

SKILL LEVEL : INTERMEDIATE

Gateway to PSTN

In my previous article, in Issue 26 of The MagPi, we installed Asterisk on a Raspberry Pi Model B/B+ and configured it using FreePBX as a VoIP server with an optional SIP phone or SIP adapter (this article uses the Dlink DPH-150SE). We also installed and configured LinPhone soft phone software, which has versions for iOS, Android, Blackberry, Linux, Windows and OSX.

Now that we have an IP telephone system installed and running, the next step is to connect it to other networks. In this article we will connect to the regular telephone network, otherwise known as PSTN (Public Switched Telephone Network). We will then have a complete telephone system, with the availability of a wired telephone and the features and services of a full IP phone system, at a very low cost.

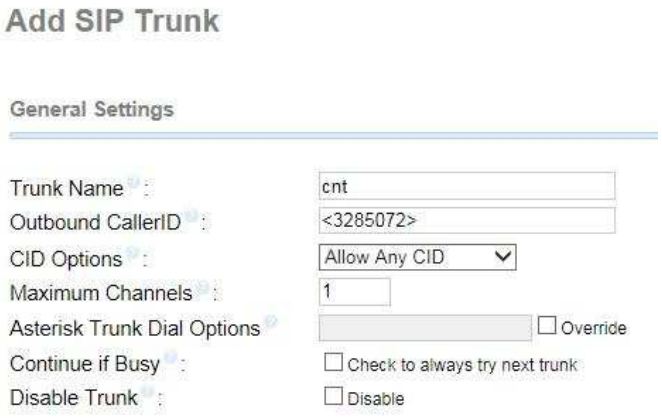
Additional components

The only additional hardware required is a Voice Gateway which supports SIP lines for connection to the PSTN. For this article I used a Cisco SPA-3102 which costs US$60.00, though Voice Gateways from Obihai are also popular.

Installation

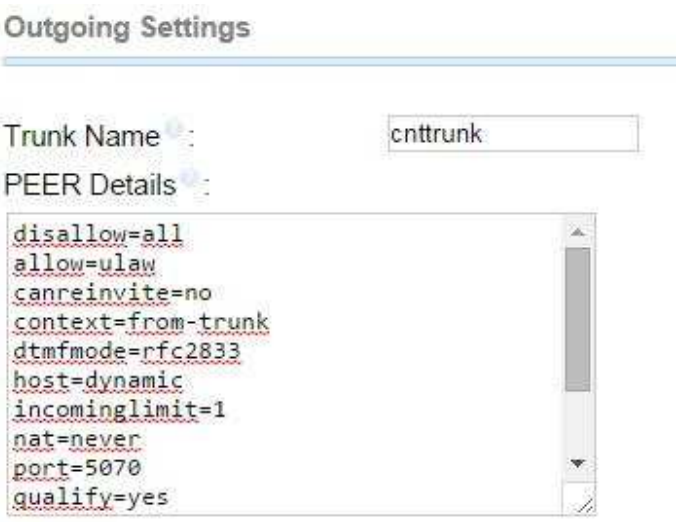
To install the Cisco SPA-3102 you must first configure a Trunk in Asterisk. We will do this using FreePBX. From my previous article, open a web browser and enter `http://raspbx` or your static IP. (For Apple Mac you will enter `http://raspbx.local`). This will open the FreePBX Administration page.

Click on `FreePBX Administration`. The default login is `admin`, with the password `admin`. Click on the `Connectivity` tab and choose the `Trunks` option. Click on `Add SIP Trunk`, then create a trunk for the PSTN telephony supplier.



In the **General Settings** section the main configuration options are:

Trunk Name - Descriptive name for this trunk.
Outbound CallerID - Format: <#######>. You can also use the format "hidden" <#######> to hide the CallerID sent out over Digital lines if supported (i.e. E1/T1/J1/BRI/SIP/IAX).



In the **Outgoing Settings** section the main configuration options are:
Trunk Name - Give the trunk a unique name (e.g. myiaxtel). I have used 'cnttrunk'.
PEER Details - a full list of the required options is shown below.

PEER Details:

```
disallow=all
allow=ulaw
canreinvite=no
context=from-trunk
dtmfmode=rfc2833
host=dynamic
incominglimit=1
nat=never
port=5070
qualify=yes
secret=cnttrunk
type=friend
username=cnttrunk
```

Click on Submit Changes, then click on the red Apply Config button to save your changes.

To add a route for the incoming calls, click on the the Connectivity tab and then on the Inbound Routes option.

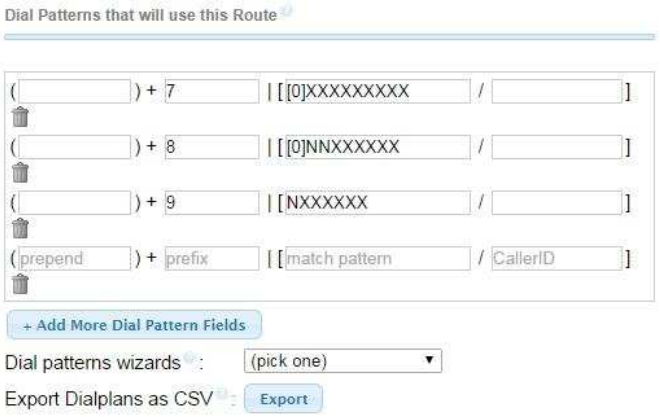
In the **Add Incoming Route** section the main configuration options are:
Description - Provide a meaningful description of what this incoming route is for.
DID Number - Specify the expected DID (Direct Inward Dial) number. You can also use a pattern match (e.g. _2[345]X) to match a range of numbers.

In the **Set Destination** section you specify where the incoming call is routed, in this case to an IVR (Interactive Voice Response).

Click on Submit, then click on the red Apply Config button to save your changes.

To configure outbound calls click on the Connectivity tab and choose the Outbound Routes option.

In the **Route Settings** section, specify the Route Name. This can be used to describe what type of calls this route matches e.g. 'local' or 'long distance'.



A Dial Pattern is a unique set of digits that will select this route and send the call to the designated trunks. If a dialled pattern matches this route, no subsequent routes will be tried. If Time Groups are enabled, subsequent routes will be checked for matches outside of the designated times.

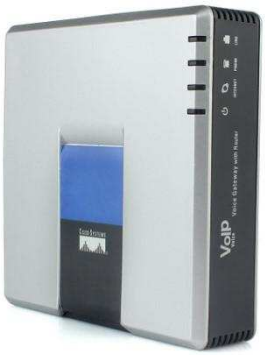
In the **Dial Patterns that will use this Route** section, the dial patterns are configured according to each country or area. We can set restrictions to allow local, regional, national, international or mobile calls.

In the **Trunk Sequence for Matched Routes** section, the trunk sequence controls the order of trunks that will be used when the above Dial Patterns are matched.

Click on `Submit Changes`, then click on the red `Apply Config` button to save your changes.

Configuring the voice gateway

The Cisco SPA-3102 voice gateway is connected to the network via its ethernet port. The setup options are reached using a browser pointing to the default IP address, e.g. `http://192.168.0.1`.

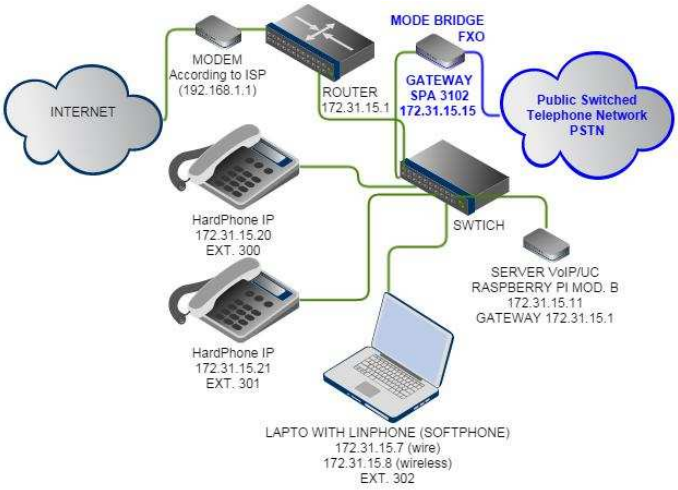


In the **Router** tab choose the `Wan Setup` option.

Change the options to match your system. In my case I used:
`Connection Type` - Static IP
`Static IP` - 172.31.15.15
`Gateway` - 172.31.15.1
`NetMask` - 255.255.255.0

Click on `Submit All Changes`.

An example architecture diagram is shown below.

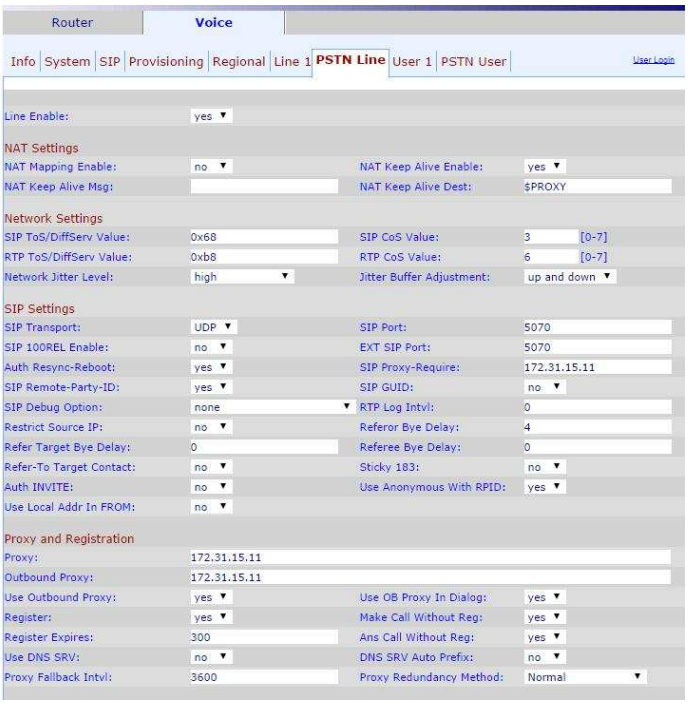


In the **Router** tab choose the Lan Setup option.

Set the Networking Service option to Bridge. Click on Submit All Changes.

In the **Voice** tab choose the SIP option. The main configuration options are:
SIP TCP Port Min - 5060
SIP TCP Port Max - 5080
RTP Packet Size - 0.020

In the **Voice** tab choose the PSTN Line option.



The main configuration options are:
Line Enable - yes
SIP Transport - UDP
SIP Port - 5070
SIP Proxy-Require - 172.31.15.11
Proxy - 172.31.15.11

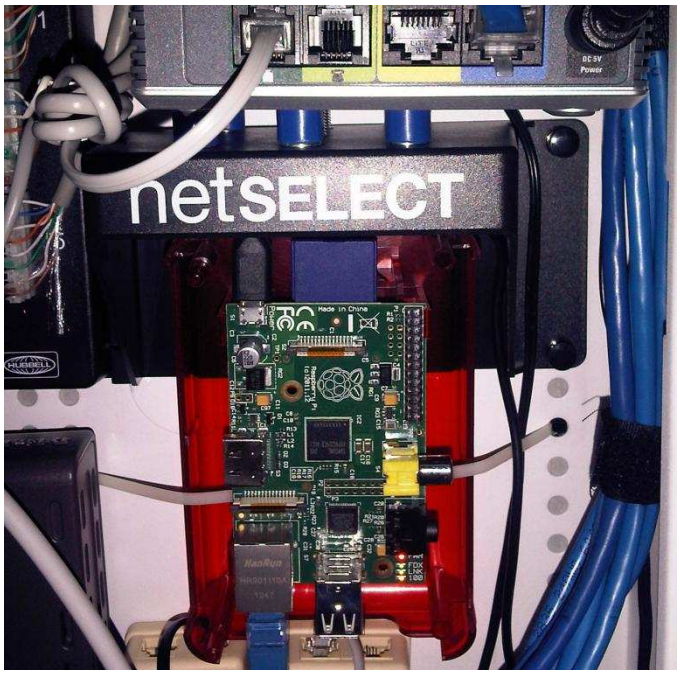
The following configuration options are shown on the right:
Display Name - CNT
User ID - cnttrunk
Password - (As in FreePBX)
Dial Plan 2 - (S0<:3285072)

Finally, set the following option:
PSTN Caller Default DP - 2

Click on Submit All Changes. Other options for Asterisk / FreePBX and the Gateway must be configured according to your requirements.

Conclusion

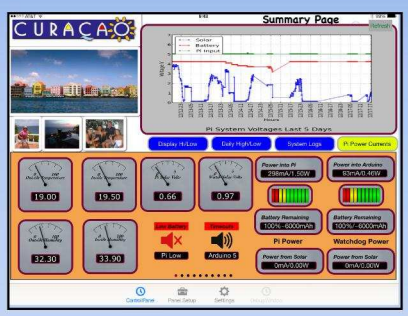
You should now have a fully functioning VoIP server connected to the PSTN and be able to receive and send calls from anywhere. I leave it to your imagination to make further use of this high performance, low cost VoIP server.

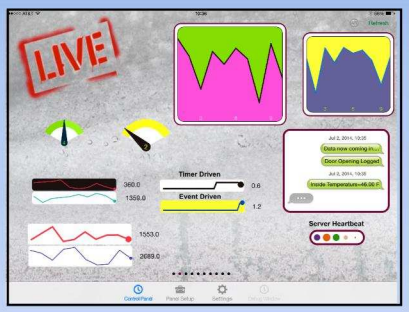


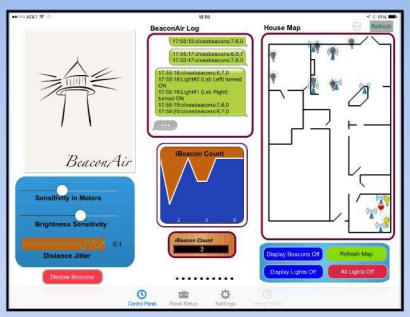
RasPiConnect

Connect your Raspberry Pi to the World!

Build Your Own Contol Panel!







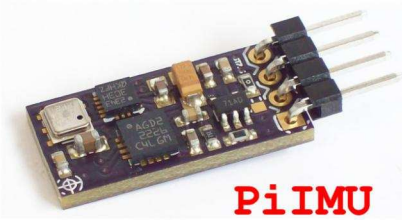
- ***New Live Controls***
- EASY to setup - no syncing required
- Exchange your panels with friends
- Supports multiple Raspberry Pis and multiple Arduinos
- Build your pages on your iPad/iPhone
- Ten pages of control panels
- Supports any computer that supports Python (windows, linux, etc.)

Visit milocreek.com for more info!

Available on the App Store

MiloCreek
Where Innovation Flows

Supports Arduino with in-app purchase

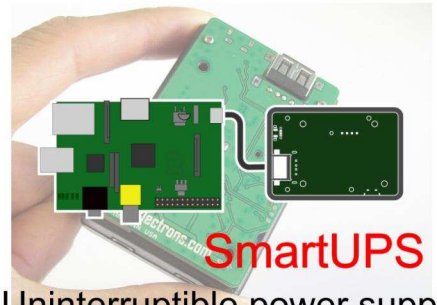


PiIMU

10 DOF Gyro
Compass
Accelerometer &
Altimeter for RPi

a Pan-Tilt for RPi Camera

Scratch Programming for
PiServoController & Pi-Pan



SmartUPS

Uninterruptible power supply
for Raspberry Pi

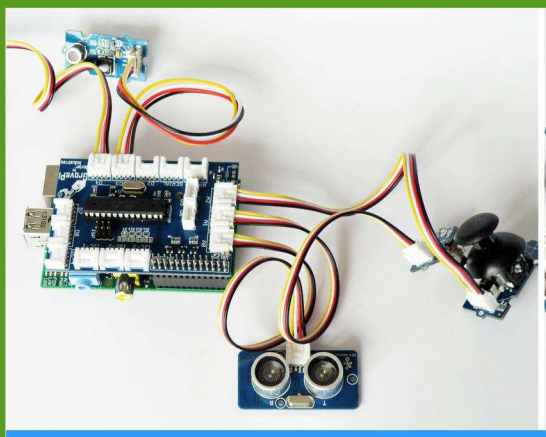
PiConsole

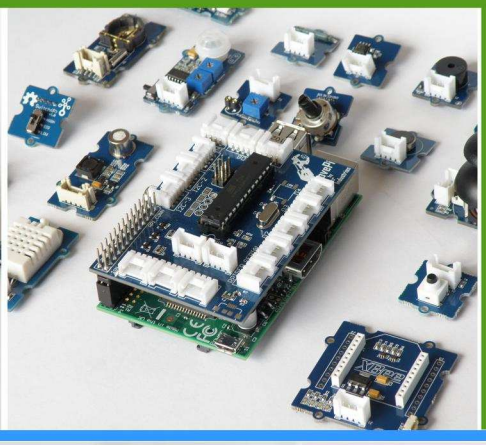
Access RPi Console on
your Android or iPhone!

PiServoController

6 Channel Servo Controller
for Raspberry Pi

OpenElectrons.com



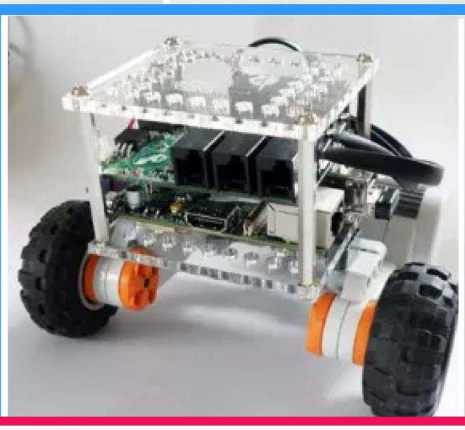


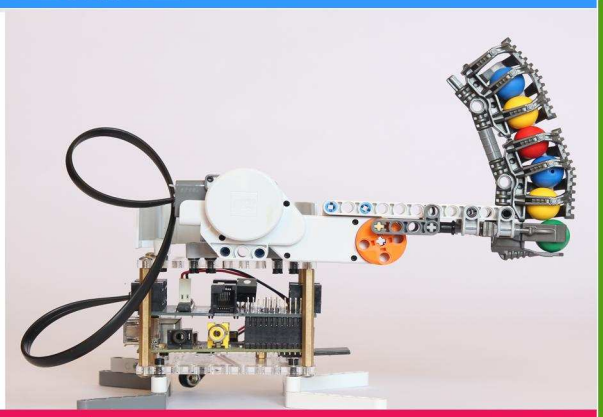
GrovePi

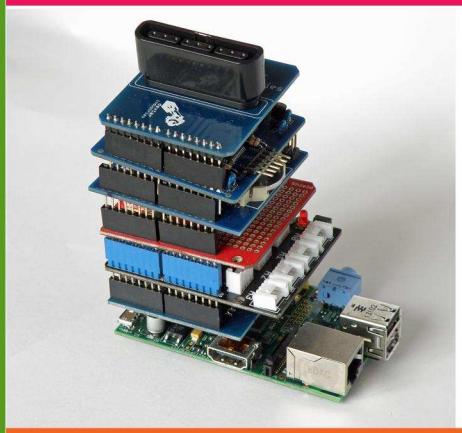
Connect Hundreds of Sensors to your Raspberry Pi

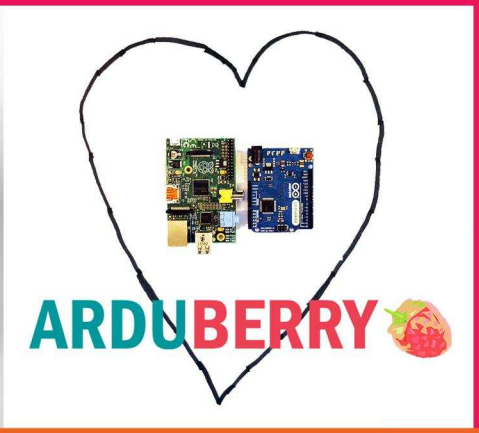
BrickPi

Turn your Raspberry Pi into a LEGO© Robot







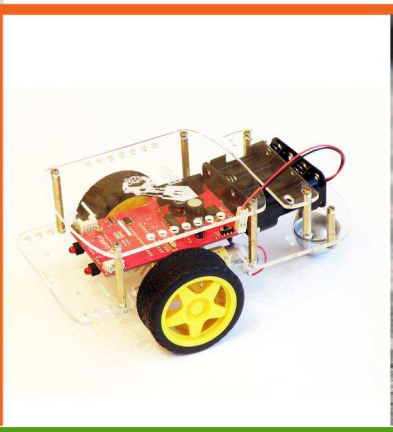


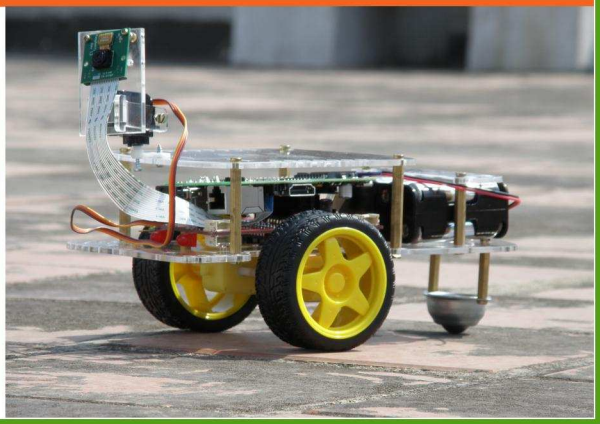
Arduberry

Unite the Raspberry Pi and Arduino

GoPiGo

Turn Your Raspberry Pi into a Robot





dexterindustries.com

MagPi Readers! Use the code “MagPi14” for a 10% discount in our store.

Version control basics using Git - Part 2

Alec Clews

Guest Writer

SKILL LEVEL : BEGINNER

Introduction

In Issue 27 of The MagPi, I introduced you to the Git Version Control software. We installed and configured Git and then downloaded a sample project called `snakes`. We created a repository for the `snakes` project and added all the project files to a Git repository.

This month I will demonstrate the value of running Version Control software by showing what happens when we make changes to our project files. Before we start, make sure you are in the `snakes` folder. Enter:

```
cd ~/snakes
```

Now let's make a change. The first step is to create a work area in which to make the change. In Git (and many other VC tools) this dedicated work area is called a **branch**. When you first create a repo, the default branch that is created is called the **master**. However, it is important to know that there is nothing special about the master branch - it can be treated in exactly the same way as any branches you create yourself.

If you look at the output from the `git status` command you can see that we are currently using the master branch in our working area. Enter:

```
git status
```

What change do I want to make?

When I play the game of snakes the rocks are represented by "Y" which I want to change to "R". The lines I need to change are in the file `game/snake.py` (lines 50 and 52 in my version).

Let's create a branch to work on. Enter:

```
git branch make_rocks_R
```

No message means the command was successful (note that spaces are not allowed in the branch name). Creating a branch means that I have a working area in my project (you can think of it as a sandbox for a mini project) that stops my change from breaking or impacting any other work that is going on in the `snakes` project.

You can get a list of all the branches with the `git branch` command. Enter:

```
git branch
```

Author photo courtesy of Jack Cotton

You will see something similar to:

```
  make_rocks_R
* master
```

The asterisk shows the current branch.

To make the `make_rocks_R` the current branch use the `git checkout` command. Enter:

```
git checkout make_rocks_R
```

You should see the following result:

```
Switched to branch 'make_rocks_R'
```

Now when you enter the `git branch` command it displays:

```
* make_rocks_R
  master
```

In technical terms what has happened is that Git has **checked out** the branch `make_rocks_R` into our **working directory**. The working directory contains that set of files from the specific branch that we are currently working on. Any changes we now make are isolated in the branch and will not impact anything else.

At this point you may want to play snakes for a couple of minutes, so that you will be able to see the difference later. Use the cursor keys to control the snake, press <Spacebar> to restart and press <Ctrl>+<C> to exit. Enter:

```
python game/snake.py
```

Changing the file

Edit the file `game/snake.py` using your favourite text editor. In the version of snakes I have there are two changes to make - a comment on line 50 and the actual code on line 52.

Save the changes and test the game by playing it again. The rocks should now look like “R” instead of “Y”.

Showing the diff

So let us see what has changed. Git can provide a nice listing. The simplest way is by using the command `git diff`. Enter:

```
git diff
```

You should see a report similar to this:

```
diff --git a/game/snake.py b/game/snake.py
index cef8d07..7e65efe 100755
---  a/game/snake.py
+++  b/game/snake.py
@@ -47,9 +47,9 @@ def add_block(scr,
width, height):
             empty = False

         if empty:
-          # if it is, replace it with a
"Y" and return
+          # if it is, replace it with a
"R" and return

-          scr.addch(y, x, ord("Y"),
curses.color_pair(2))
+          scr.addch(y, x, ord("R"),
curses.color_pair(2))
           return

 def snake(scr):
```

This report can be a little confusing the first time you see it. However, if you look carefully you can see lines marked with **+** and **-**. These are the lines that have been changed. If we had made changes to more than one file then each set of file differences would be listed. This type of information is often referred to as a **diff report** or **diff output**.

You can get a more user friendly display of these differences by using a graphical compare tool. First install the `kdiff3` program by entering:

```
sudo apt-get install kdiff3-qt
```

Now, instead of using `git diff` to get a text report of the differences in your change you can run `git difftool` to scroll through a side by side list. The `difftool` command supports several different GUI style tools to present

the differences. Setting them up is left as an exercise.

Committing the change

Now that we have a change and we have tested it and have verified it using the `difftool` command, it is time to add the change to our version control history.

This is a two stage process, in a similar way to our first commit:

1. Add the changes to the index.
2. Commit the change to the repo, along with a useful comment.

The first part is simple as only one file has changed. Enter:

```
git add game/snake.py
```

You should then verify that the addition was successful by running a `git status` command.

This time when we commit we want to add a more complete report (called a **commit message**). But first let us make sure that our editor is set up in Git. As an example we will set up Leafpad as the editor. Enter:

```
git config --global core.editor "/usr/bin
/leafpad"
```

Note: Leafpad is a GUI editor and you will need to run LXDE (`startx`) for it to work. If you are not using LXDE, or prefer a different editor, then use the appropriate program name e.g. `/usr/bin/vim`.

Now let's make the commit. This time the command, `git commit`, is a little simpler but something a little more spectacular will happen. Your editor will pop into life in front of you with information ready for you to write a commit message.

You now have two choices:

1. Exit the editor without saving any changes to the commit message. The commit is aborted and no changes occur in the repo (but the index still contains the change).

2. Enter some text, save it and exit the editor. The commit is completed and all changes are recorded in the repo.

A word about commit messages: The commit message consists of two parts. Line 1 is the header and should be followed by a blank line. The header is displayed in short log messages. After the blank line comes the message body which contains the details. A detailed set of suggestions can be read at http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html.

The following is an example of a commit message that might be used for the change we have just made:

```
Changed  Rocks Y > R

1. Changed all references to rocks from
the char "Y" to "R"
a. In a comment
b. In a single line of code
2. Tested
```

Enter the `git commit` command and use the above commit message:

```
git commit
```

You should get output similar to the following:

```
[make_rocks_R ce3ed3f] Changed Rocks Y > R
 1 file changed, 2 insertions(+), 2
deletions(-)
```

Showing the history

Notice how, in the picture on the next page, the arrow points from the child commit to the parent commit. This is an important convention. Enter:

```
git log
```

You will see something like:

```
commit ce3ed3fbb350688a10eaa793dc2142b14b8
a2b74
Author: Pi <acdsip61-pi@yahoo.com>
Date:   Sun Sep 28 12:40:53 2014 +1000

    Changed Rocks Y > R

    1. Changed all references to rocks
from the char "Y" to "R"
    a. In a comment
    b. In a single line of code
    2. Tested

commit 74893eddd81140341b267e7c9c1fa26fadf
e86de
Author: Pi <acdsip61-pi@yahoo.com>
Date:   Sun Sep 28 12:40:52 2014 +1000

    Initial   Commit
```

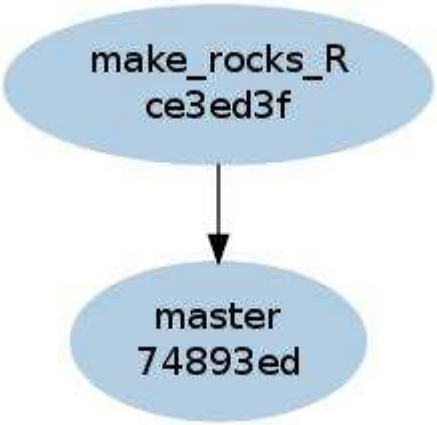


Figure 1: A simple picture of the current repo history

For more details you might want to look at http://git-scm.com/book/en/Git-Basics-Recording-Changes-to-the-Repository.

Branches

We now have two branches - `master` and `make_rocks_R`. Let's make another change on a new branch and then look at the history. Make sure that you are using the master branch. Enter:

```
git checkout master
```

You will see the output:

```
Switched to branch 'master'
```

Now let's examine the file `game/snake.py` again. This time I have noticed that when setting up colours (with the method call `curses.color_pair()`) the original programmer used a literal constant. It is good practice to use more meaningful symbolic names (like `curses.COLOR_RED` instead of the literal value 1).

We are going to make two changes. The text `curses.color_pair(2)` will be changed to `curses.color_pair(curses.COLOR_GREEN)` and the text `curses.color_pair(1)` will be changed to `curses.color_pair(curses.COLOR_RED)`. Documentation on the Curses library is available at http://docs.python.org/howto/curses.html. Enter:

```
git branch use_curses_symbols
git checkout use_curses_symbols
```

You will now see the output:

```
Switched to branch 'use_curses_symbols'
```

With the above commands I created a new branch (from `master`, not from `make_rocks_R`) called `use_curses_symbols` and checked it out.

Edit the file `game/snake.py` using your favourite text editor. In the version of snakes I have there are two code changes to make - one on line 52 and the other on line 148. Make the changes as described above. Save the changes and test the game by playing it again.

If I run the command `git diff` I can see the following report:

```
diff --git a/game/snake.py b/game/snake.py
index cef8d07..ec8ee6e 100755
---  a/game/snake.py
+++  b/game/snake.py
@@ -49,7 +49,7 @@ def add_block(scr,
width, height):
       if empty:
         # if it is, replace it with a "Y"
and return

-        scr.addch(y, x, ord("Y"),
```

```
curses.color_pair(2))
+        scr.addch(y, x, ord("Y"),
curses.color_pair(curses.COLOR_GREEN))
         return

 def snake(scr):
@@ -145,7 +145,7 @@ def snake(scr):

       # replace the character with a "O"

-      scr.addch(y, x, ord("O"),
curses.color_pair(1))
+      scr.addch(y, x, ord("O"),
curses.color_pair(curses.COLOR_RED))

       # update  the  screen
```

Now we can add and commit our changes. Enter:

```
git add game/snake.py
git commit -m "Use curses lib symbolic na
mes in color_pair() method calls"
```

You should see the following output:

```
[use_curses_symbols d7d093a] Use curses
lib symbolic names in color_pair() method
calls
 1 file changed, 2 insertions(+), 2
deletions(-)
```

Now if we run the `git log` command we only see two commits:

```
commit 1f2962b4e0714de810b97a35ca87290c484
42d10
Author: Pi <acdsip61-pi@yahoo.com>
Date:   Sun Sep 28 12:40:55 2014 +1000

    Use curses lib symbolic names in
color_pair() method calls

commit 74893eddd81140341b267e7c9c1fa26fadf
e86de
Author: Pi <acdsip61-pi@yahoo.com>
Date:   Sun Sep 28 12:40:52 2014 +1000

    Initial   Commit
```

What happened to our other commit where we changed the character for our rocks? The answer is that it is on another branch – it is not part of the history of our current workspace.

Add the option `--all` to see all the commits across all the branches. Enter:

```
git log --all
```

You will see something like the following:

```
commit 1f2962b4e0714de810b97a35ca87290c484
42d10
Author: Pi <acdsip61-pi@yahoo.com>
Date:   Sun Sep 28 12:40:55 2014 +1000

    Use curses lib symbolic names in
color_pair() method calls

commit ce3ed3fbb350688a10eaa793dc2142b14b8
a2b74
Author: Pi <acdsip61-pi@yahoo.com>
Date:   Sun Sep 28 12:40:53 2014 +1000

    Changed Rocks Y > R

    1. Changed all references to rocks
from the char "Y" to "R"
    a. In a comment
    b. In a single line of code
    2. Tested

commit 74893eddd81140341b267e7c9c1fa26fadf
e86de
Author: Pi <acdsip61-pi@yahoo.com>
Date:   Sun Sep 28 12:40:52 2014 +1000

    Initial   Commit
```

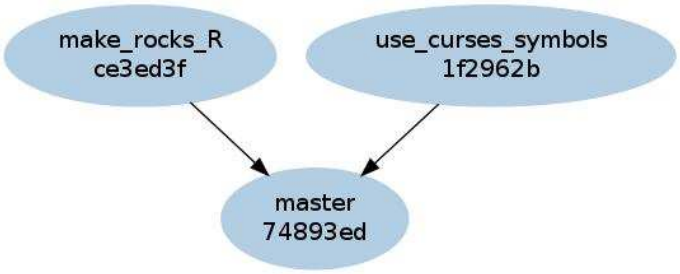


Figure 2: The current repo history with three branches and one commit on each branch.

As you can see `git` commands take extra parameters to change the way they work.

A useful way to see the above history using quite a complex log command is shown below (enter it all as one continuous line:

```
git log --graph --pretty=format:'%Cred%h%
Creset -%C(yellow)%d%Creset %s %Cgreen(%c
r) %C(bold blue)<%an>%Creset' --abbrev-co
mmit --date=relative --all
```

It is quite hard work to type this in. Fortunately Git has an alias feature which allows us to simplify commands. Enter the following command, again as one continuous line:

```
git config --global alias.lg "log --graph
--pretty=format:'%Cred%h%Creset -%C(yello
w)%d%Creset %s %Cgreen(%cr) %C(bold blue)
<%an>%Creset' --abbrev-commit --date=rela
tive --all"
```

Now all you need to do in future is enter the command `git lg` as `lg` has become an alias for the much longer version of `log` shown above. More information about aliases is available at https://git.wiki.kernel.org/index.php/Aliases.

As you have installed the `gitk` program (from part 1 of this article) you can also use this to display this log information in a graphical program. Enter:

```
gitk --all&
```

All the various reports from `git log` and `gitk` refer to our branches by name. In addition, there is a HEAD revision label. This is a reference to the last commit we made on a branch, so every branch has a HEAD. Generally the HEAD refers to the last commit on the current default branch.

Commit IDs

At the end of part 1 I promised I would explain the commit field (commit ID), as shown in the output of the `git log` command. The commit ID is an important concept that deserves its own section.

In many VCS tools it is enough to give each new commit a revision number such as 1, 2, 3 and so on. We can also identify branches by using dotted numbers. For example, 3.2.5 would be the 5th revision of the 2nd branch from version 3.

However in Git we are not sharing a single repo database and there has to be a way of keeping all the possible commits on a distributed project unique. Git solves this problem by using a SHA-1 string. SHA-1 (Secure Hash Algorithm) is a computer algorithm that, when presented with a string of bits (1's and 0's), will present a different 40 character result even when two strings are different in any way, even just one bit.

You can see this effect by running the following experiment. Enter:

```
echo 'Hello World' | git hash-object --st
din
```

The result will be:
`557db03de997c86a4a028e1ebd3a1ceb225be238`

Now enter:

```
echo 'Hello World!' | git hash-object --s
tdin
```

This time the result is:
`980a0d5f19a64b4b30a87d4206aade58726b60e3`

This is exactly what Git does for each commit, only it uses the contents of the committed files (plus the ID of the commit parents) to calculate the new SHA-1 commit ID. If two commits from two different repos have the same ID they are the same commits and we consider them identical.

Using the Git graphical tools

Most of the examples so far have used the command line interface. However, Git does come with two GUI interfaces – `gitk` and `git gui`. You have already seen that `gitk` is useful for looking at the history of changes in a repository. `git gui` can be used to perform operations such as add, commit, checkout etc.

Let's replicate our previous examples using the standard git GUI tools. Create a directory called `~/snakes2` and unpack the `game.tar.gz` file into it. Enter the following command and explore:

```
git gui
```

Part 4: Font scaling plus adding the final touches to our game

Jon Silvera

Guest Writer

SKILL LEVEL : BEGINNER

So far in this series we have covered sprites, collisions and controls. This month we will tidy things up a bit by adding a start screen, scoring, lives, a boss and an ending.

New FUZE BASIC editor

However before we get started I would like to make a suggestion. At FUZE we are busy working on a new version of the editor. You can download this immediately to your Raspbian desktop. Visit http://www.fuze.co.uk/resources-2 then click on the "GET FUZE BASIC" tab for details on downloading. The "work in progress" version has a more standard editor and as such is easier to save, load and edit programs. However if you prefer to stick to the original then you can still run FUZE BASIC from the RUN <F3> menu.

From the beginning...

First, you will need to load several new sprites so please visit http://www.fuze.co.uk/resources-2 and then click on the "Tutorials" tab and download the following sprites; `themagpi.bmp`, `fblogo.bmp`, `blackberry.bmp` and `ship.bmp`. You need to download and save these sprites to the `MagPi` folder where your program is saved.

Unfortunately you have a lot of work on your hands this month as this is the final and finished program. The problem is that so much has changed it really needs typing in again from the beginning. Also it would take too much space to add everything individually so we are going to go through the whole program section by section. The downside is that this means the game will not work until we have finished entering the complete listing.

Once again please bring up the FUZE BASIC environment using the icon on the Desktop. If you are using the older version, go to the Editor with <F2> and enter:

```
// MagPi Game
PROC SetupMain
  CYCLE
    PROC SetupGame
    PROC attract
    WHILE lives > 0 CYCLE
      PROC CheckControls
      PROC DrawShip
      PROC DrawEnemy
      PROC DrawBullet
      PROC BOSS
      IF WIN THEN PROC wellDone
      PROC updateScreen
      UPDATE
    REPEAT
    PROC gameOver
  REPEAT
END
```

Remember, you will only get errors if you try and RUN <F3> this so just be patient for now.

First we call the main setup routine to load all the sprites and variables. These items are required each time the game is executed for the first time but not when the game restarts within the program. This way we can keep a high score, have lives and levels.

Next is the main loop followed by SetupGame and attract procedures. The first resets the main variables to start each new game while attract displays the start-up screen (known as attract mode in the old days).

Next we check to see if we have any lives left and then check for keys pressed, draw everything and then check to see if it is 'boss time' yet. All going well, and as long as there are lives left, then it will REPEAT back to WHILE lives > 0. If we are out of lives then the gameOver procedure is called and we start again.

Now add the following after the END statement:

```
DEF PROC updateScreen
  CLS2
  INK = Yellow
  fontScale(3, 3)
  hvTab (0, 0)
  PRINT "Score "; score
  INK = Red
  hvTab (tWidth / 2 - 7, 0)
  PRINT "Hi Score "; hiScore
  INK = Yellow
  hvTab (tWidth - 7, 0)
  PRINT "Level "; Level
  FOR num = 0 TO lives - 1 CYCLE
    plotImage (shipPic,0 + getImageW (shipPic)
* num, -10)
  REPEAT
  COLOUR = Silver
  RECT (0, 62, gWidth, 4, 1)
ENDPROC
```

This procedure prepares and displays various pieces of information like the player's score, the hi-score and the number of lives remaining.

CLS2, as previously covered, clears the buffer screen. fontScale(3, 3) sets the current font size to 3 times normal size, both horizontally and vertically. hvTab positions the text cursor at (x, y). The FOR loop draws the number of lives as player ships along the bottom and the RECT command draws a thin silver rectangle along the bottom of the screen.

For the remainder, just add each new section of code to the end of the existing program. Enter:

```
DEF PROC BOSS
  IF Enemy(EnemyMax,1) <= 0 OR Enemy(EnemyMax,
3) = 0 THEN Warning = 1
  IF Warning THEN
    WarningCount = WarningCount + 1
    IF WarningCount <= 150 THEN
      INK = RND(15) + 1
      fontScale(3, 5)
      hvTab (tWidth / 2 - LEN (Warning$) / 2,
tHeight / 4) * 3
      PRINT Warning$
      UPDATE
    ELSE
      Warning = 0
      bossActive = 1
    ENDIF
  ENDIF
  IF bossActive THEN
    bossX = bossX - 2 * Level
    IF bossX < -getSpriteW(boss) THEN PROC ga
meOver
    bossAng = bossAng + 3
    bossX = bossX + bossXX * COS(bossAng)
    bossY = bossY + bossYY * SIN(-bossAng)
    plotSprite(boss, bossX, bossY, 0)
    IF bossAng >= 360 then bossAng = 0
  ENDIF
ENDPROC
```

The boss springs into action only if the Warning has been activated, and this is only activated if the very last enemy has either left the screen to the left or has been destroyed. The "Warning" is then displayed, and when finished the boss itself appears. The boss follows a simple circular pattern but its speed across the screen increases due to the statement bossX = bossX - 2 * Level.

Once again don't bother trying to RUN <F3> this yet as it just does not have enough to go on but here is a preview to whet your appetite.

Now add the following code:

```
DEF PROC DrawBullet
  IF Shot(1) > gWidth THEN
    hideSprite (Shot(0))
    Shot(3) = 0
    Fire = 0
  ENDIF
  IF Shot(3) THEN
    Shot(1) = Shot(1) + 8
    plotSprite (Shot(0), Shot(1), Shot(2), 0)
    Hit = spriteCollidePP (Shot(0), 2)
    IF Hit > 0 AND Hit <= 64 THEN
      score = score + Enemy(Hit - 1,5) * Level
      EnemyCount = EnemyCount + 1
      Enemy(Hit - 1, 3) = 0
      hideSprite (Hit)
      hideSprite (Shot(0))
      Shot(3) = 0
      Fire = 0
    ENDIF
    IF Hit = 68 THEN
      score = score + 500
      bossHit = bossHit - 1
      hideSprite (Hit)
      hideSprite (Shot(0))
      Shot(3) = 0
      Fire = 0
      IF bossHit <= 0 THEN
        WIN = 1
        bossActive = 0
        Level=Level + 1
        PROC SetupGame
      ENDIF
    ENDIF
  ENDIF
ENDPROC
```

You are only allowed a single bullet on the screen at a time. First we check if it is still on the screen and if not then we hide it then reset the `Fire` variable so another bullet can be shot. The bullet moves along at 8 pixels at a time and is checked to see if it has collided with anything.

If it hits a rock then the rock is removed, the `score` is increased and the bullet is removed so another can be fired. If it comes into contact with the boss then the `bossHit` count decreases. If the `bossHit` counter gets to 0 then the game starts again with an increased level and the score carried over. The `WIN` variable activates the “Congratulations” screen later.

The next three procedures are responsible for working out bullet positions, checking various collisions and plotting rocks and the player’s ship on the screen. Enter:

```
DEF PROC Bullet
  Fire = 1
  Shot(1) = ShipX + getSpriteW (Ship) + 8
  Shot(2) = ShipY + getSpriteH (Ship) / 2 - 10
  Shot(3) = 1
ENDPROC

DEF PROC DrawEnemy
  FOR eID = 0 TO EnemyMax CYCLE
    IF Enemy(eID, 3) THEN
      Enemy(eID, 1) = Enemy(eID, 1) - Enemy(eI
D, 6)
      EY = Enemy(eID, 2) + COS (Enemy(eID, 1))
* Enemy(eID, 4) * 10
      IF Enemy(eID, 1) > -getSpriteW(Rock(eID)
) * 2 AND Enemy(eID, 1) <= gWidth THEN plotSpr
ite (Enemy(eID, 0), Enemy(eID, 1), EY, 0)
      IF Enemy(eID, 1) <= -getSpriteW(Rock(eID
)) * 2 THEN
        hideSprite(Rock(eID))
        Enemy(eID, 3) = 0
      ENDIF
    ENDIF
  REPEAT
ENDPROC

DEF PROC DrawShip
  plotSprite (Ship, ShipX, ShipY, ShipID)
  Crash = spriteCollidePP (Ship, 2)
  IF Crash > 0 AND Crash <= 64 THEN
    lives = lives - 1
    Enemy(Crash - 1, 3) = 0
    hideSprite (Crash)
    ShipX = 0
    ShipY = gHeight / 2
  ENDIF
  IF Crash = 68 THEN
    lives = lives - 1
    ShipX = 0
    ShipY = gHeight / 2
  ENDIF
ENDPROC
```

Note: The highlighted code is one single line. What it does is to figure out if a rock is visible on the screen by checking if it is too far left or too far right. It only plots the sprite if it is in the viewable area.

The `DrawShip` procedure plots the player's ship and checks to see if it has collided with either the rocks (sprite IDs 0 to 64) or the boss (sprite ID 68). If it crashes into a rock the player's ship is returned to its original position, `lives` is reduced by 1 and any rocks we may have crashed into are removed with the `hideSprite` command.

The player controls procedure is very straightforward and we have covered most of it before. Enter:

```
DEF PROC CheckControls
  ShipID = 1
  UpKey = scanKeyboard (scanUp)
  DownKey = scanKeyboard (scanDown)
  LeftKey = scanKeyboard (scanLeft)
  RightKey = scanKeyboard (scanRight)
  SpaceKey = scanKeyboard (scanSpace)
  IF SpaceKey AND NOT Fire THEN PROC Bullet
  IF UpKey AND ShipY <= gHeight - 100 THEN
    ShipY = ShipY + 8
    ShipID = 2
  ENDIF
  IF DownKey AND ShipY >= 64 THEN
    ShipY = ShipY - 8
    ShipID = 0
  ENDIF
  IF LeftKey AND ShipX >= 0 THEN ShipX = ShipX
- 8
  IF RightKey AND ShipX <= gWidth / 2 THEN Shi
pX = ShipX + 4
ENDPROC
```

Various keys are checked to see if they are being pressed and actions take place accordingly. Notice the `RightKey` will only work if the player is not too far forward and it moves at half the number of pixels (4) than all the other directions (8). Moving slowly forward gives an extra sense of realism.

Here are some smaller procedures. Enter:

```
DEF PROC killEverything
  FOR num = 0 TO EnemyMax CYCLE
    hideSprite (Rock(num))
  REPEAT
  hideSprite (Shot(0))
  hideSprite (Ship)
  IF bossActive THEN hideSprite (boss)
  bossActive = 0
  CLS2
ENDPROC

DEF PROC wellDone
  WAIT(1)
  PROC killEverything
  FOR delay = 0 TO 300 CYCLE
    INK = RND(15) + 1
    fontScale(5, 5)
    hvTab (tWidth / 2 - LEN (Congrats$) / 2,
tHeight / 2)
    PRINT Congrats$
    UPDATE
  REPEAT
  WIN = 0
ENDPROC

DEF PROC gameOver
  PROC killEverything
  text$ = "GAME OVER"
  fontScale (4, 4)
  FOR num = 0 TO 100 CYCLE
    hvTab (tWidth / 2 - LEN (text$) / 2, tHeig
ht / 2)
    INK = RND (15) + 1
    PRINT text$
    UPDATE
  REPEAT
  IF score > hiScore THEN hiScore = score
  clearKeyboard
  WIN = 0
  lives = 0
ENDPROC
```

`killEverything` is used to reset and hide all the active sprites. It is called before anything major happens like killing the boss and losing all your lives.

wellDone is only called when you have successfully killed the boss. It simply displays “Congratulations” in the middle of the screen.

gameOver displays “GAME OVER” in the middle of the screen and resets a few ‘trigger’ variables. For example lives is set to 0, which controls the main program loop.

Attracting players

Every half decent game must have an attract screen. It just provides something to look at when the game is not running. Enter:

```
DEF PROC attract
  CLS
  t1X = gWidth/2 - getSpriteW (title1) / 2
  t2X = gWidth/2 - getSpriteW (title2) / 2
  t1Y = gHeight - getSpriteH (title1) * 1.5
  t2Y = getSpriteH (title2)
  angle = 0
  WHILE NOT scanKeyboard (scanSpace) CYCLE
    t1XX = t1X + 250 * COS (angle)
    t1YY = t1Y + 40 * SIN (angle)
    t2XX = t2X - 80 * COS (-angle)
    t2YY = t2Y - 10 * SIN (-angle)
    plotSprite (title1, t1XX, t1YY, 0)
    plotSprite (title2, t2XX, t2YY, 0)
    INK = RND (15) + 1
    fontScale (3, 3)
    hvTab (tWidth / 2 - LEN (Press$) / 2, tHei
ght / 2)
    PRINT Press$
    UPDATE
    angle = angle + 2
  REPEAT
  hideSprite (title1)
  hideSprite (title2)
  WAIT (0.5)
  CLS
  clearKeyboard
ENDPROC
```

There are several cosine and sine calculations here. The angle variable is increased to provide 360 degree movement. The X and Y positions of both images are calculated by taking a point of origin (t1X) then adding and multiplying a radius (250) by the cosine of the current angle. The point of origin (t1X) never changes as it is only the radius that is operated on. This is repeated on t1Y, t2X and t2Y and then the two images are drawn as sprites.

“Press the Space Bar” is displayed in the middle of the screen using hvTab and PRINT Press$. The whole thing is enclosed in a WHILE loop that checks for the <Space bar> to be pressed. Once the <Space bar> is pressed the loop ends, the sprites are removed and the procedure returns.

There are two setup procedures - SetupMain and SetupGame. The SetupMain procedure initialises everything that is required the very first time you run the game; things like text messages, all of the sprite graphics and major variables. Enter:

```
DEF PROC SetupMain
  HGR
  hiScore = 0
  WIN = 0
  updateMode = 0
  Warning$ = "Warning, Huge Fruit Approaches!"
  Press$ = "Press the Space Bar"
  Congrats$ = "CONGRATULATIONS!"
  Ship = newSprite (3)
  loadSprite ("Player1.bmp", Ship, 0)
  loadSprite ("Player2.bmp", Ship, 1)
  loadSprite ("Player3.bmp", Ship, 2)
  setSpriteTrans (Ship, 255, 0, 255)
  EnemyMax = 63
  DIM Enemy(EnemyMax, 6)
  DIM Rock(EnemyMax)
  FOR num = 0 TO EnemyMax CYCLE
    Rock(num) = newSprite (1)
    loadSprite ("BigRock.bmp", Rock(num), 0)
    setSpriteTrans (Rock(num), 255, 0, 255)
  REPEAT
  DIM Shot(3)
  Shot(0) = newSprite (1)
  loadSprite ("Bullet.bmp", Shot(0), 0)
  setSpriteTrans (Shot(0), 255, 0, 255)
```

```
  title1 = newSprite (1)
  loadSprite ("themagpi.bmp", title1, 0)
  setSpriteTrans (title1, 255, 0, 255)
  title2 = newSprite (1)
  loadSprite ("fblogo.bmp", title2, 0)
  setSpriteTrans (title2, 255, 0, 255)
  shipPic = loadImage ("ship.bmp")
  boss = newSprite (1)
  loadSprite ("blackberry.bmp", boss, 0)
  setSpriteTrans (boss, 255, 0, 255)
ENDPROC
```

We have covered this before but as a reminder the `newSprite(#)` command sets up a variable to store a sprite container ID. The # signifies how many sprites go into the container. For example, `Ship = newSprite (3)` defines the sprite `Ship` and gives it three slots for graphics. This is then referenced by `plotSprite (Ship, X, Y, #)` and is how we change the `Ship` graphic depending on if it is moving up, down or just staying still.

`loadSprite` loads a sprite graphic into a container slot specified by the number at the end of the command.

`setSpriteTrans` forces one colour to be transparent, meaning it will never be displayed. This is important as even black would be displayed over a white background. I generally use bright pink as the background colour for sprites as it is not often used for anything else.

The great thing about simple sprite controls like the above is that it is so easy to change the graphics by just changing a file name. You should probably use similar sizes though and remember to set the transparent colour.

Almost there...

Ok, this is it... the final furlong. The last procedure is the game setup. This configures all the variables as required each time a level is played. The first part checks to see if it is a new game or a new level. Everything thereafter configures game variables to start a new level. Enter:

```
DEF PROC SetupGame
  IF NOT WIN THEN
    lives = 3
    Level = 1
    score = 0
  ELSE
    lives = lives + 1
    score = score + 10000
  ENDIF
  ShipX = 0
  ShipY = gHeight / 2
  ShipID = 0
  bossX = gWidth + getSpriteW (boss)
  bossXX = 10
  bossY = gHeight / 2 + getSpriteH (boss) / 2
  bossYY = 10
  bossAng = 0
  bossHit = 10
  WIN = 0
  eID = 0
  EnemyID = 0
  EnemyX = 0
  EnemyY = 0
  EnemyActive = 1
  EnemyVariation = 0
  EnemyScore = 0
  EnemySpeed = 0
  EnemyCount = 0
  RESTORE
  UNTIL EnemyCount > EnemyMax CYCLE
    READ EnemyX
    READ EnemyY
    READ EnemyVariation
    READ EnemyScore
    READ EnemySpeed
    EnemyScore = EnemyScore * EnemySpeed
    DATA 1280, 100, 3, 50, 3
    DATA 1280, 500, -3, 50, 3
    DATA 4000, 366, 4, 50, 4
    DATA 4000, 230, -4, 50, 4
    DATA 7500, 100, 6, 50, 5
    DATA 7500, 500, -6, 50, 5
    DATA 11500, 400, 5, 50, 6
    DATA 11500, 300, -5, 50, 6
    FOR num = 0 TO 7 CYCLE
      Enemy(EnemyCount + num, 0) = Rock(Enemy
Count + num)
      Enemy(EnemyCount + num, 1) = EnemyX + nu
m * getSpriteW (Rock(0))
      Enemy(EnemyCount + num, 2) = EnemyY
      Enemy(EnemyCount + num, 3) = EnemyActive
      Enemy(EnemyCount + num, 4) = EnemyVaria
tion
      Enemy(EnemyCount + num, 5) = EnemyScore
      Enemy(EnemyCount + num, 6) = EnemySpeed
* Level
    REPEAT
    EnemyCount = EnemyCount + 8
  REPEAT
```

```
  Fire = 0
  bossActive = 0
  Warning = 0
  WarningCount = 0
ENDPROC
```

The DATA section resets all the enemy rocks to their default position. The boss is reset and then the procedure returns to the main loop.

Time to play

Once you have typed in all of the listing, you can attempt to RUN it by pressing <F3>.

Now, the chances of this working perfectly first time are not high. Errors usually creep in so expect to do some debugging. Generally most errors are typing mistakes, incorrect spelling and/or wrong capitalisation. You have to be very thorough.

Once you get it up and running, play the game. I have managed to get to the level five boss, can you?

The great thing about a listing like this is that you get to experiment, which is exactly what your next steps should be. Most of the variable names make sense so you should be able to change most things.

However, if you are up to the challenge then why not add some sound, explosions, more animation and perhaps different enemies and bosses for later levels... and let's not forget the Power Up! The MagPi game is a decent little game start but there is plenty of scope to improve, change and experiment.

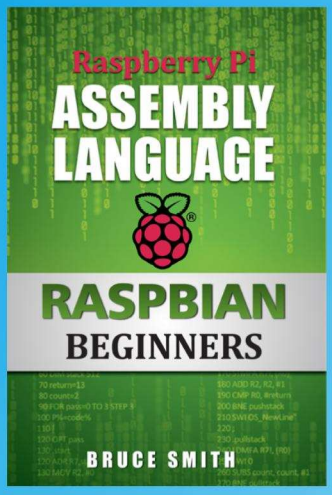
This just leaves me to say, it has been an absolute privilege to work on this project and I have thoroughly enjoyed doing it. I hope you have enjoyed it too and at the very least gained a better understanding of BASIC and now appreciate just how great a language it can be.

Why not use your new FUZE BASIC skills and enter our programming competition where there are £450 of prizes to be won. Full details are on the opposite page.

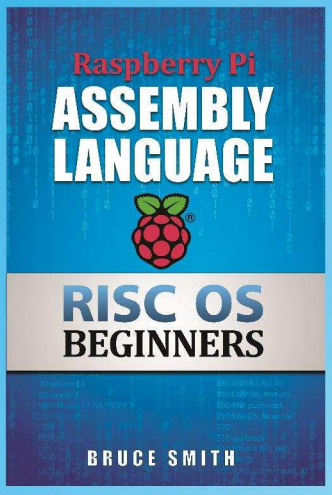
My name is Jon Silvera. Please feel free to contact me at contact@fuze.co.uk. I am happy to try and help if you have any questions and I would also love to see any projects you develop with FUZE BASIC.

BOOKS FOR THE SERIOUS PROGRAMMER

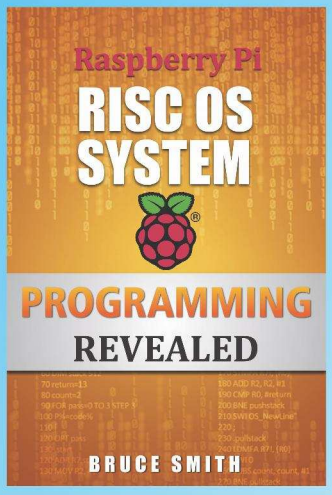
Full details at www.brucesmith.info Print and eBooks available.



With nothing other than Raspbian installed on your Raspberry Pi, this book shows you how to create your own machine code programs using ARM assembly language. With lucid descriptions, Bruce Smith keeps things simple and includes plenty of program examples you can try for yourself using the GCC Compiler.



Learn how to use the inbuilt BBC BASIC Assembler to create and generate machine code. Includes examples that show how to integrate the flexibility of BASIC into your assembler. Later chapters introduce the GCC Compiler and demonstrate how to use it's features to create stand alone machine code.



This 320-page book takes the lid off RISC OS. Aimed at those wishing to learn how to program RISC OS directly but are struggling with the Programmers Reference Manuals or simply don't know where to start. This book will teach you everything you need to know to get the most from RISC OS.

About Our Books

BSB books are ideal for the novice. Starting from first principles they lead you comfortably on your way to becoming an accomplished programmer. All books are fully supported at www.brucesmith.info with additional information, downloads, links, hints and tips.

Amazon 5-Star Reviews

All three books featured here have received 5-star reviews from readers on Amazon:

Excellent little book.

This is an excellent book which is a must-buy for anyone who wants to learn how to program their Raspberry Pi.

This is a great book for budding programmers.

Great book. An easy, step by step intro to assembly language.

Bruce has a very readable style of writing. The information is well presented.

"This is a great book. Bruce Smith writes great books."

FUZE BASIC COMPETITION

To conclude our FUZE BASIC programming series, the folks at FUZE are generously running a FUZE BASIC programming competition, with an incredible **£450** (US$720) of prizes!

First prize is the amazing FUZE T2-R kit, worth £230. Not only does this have everything from the FUZE T2-B kit, you also get a Raspberry Pi. To maximise your enjoyment of the Raspberry Pi, it also includes an OWI programmable robotic arm kit!

Second prize is the superb FUZE T2-B kit, worth £130. Again, this contains everything you need including a solderless breadboard, various electronic components, SD card with FUZE BASIC, printed Programmer's Reference Guide and much more! You just add your Raspberry Pi.

Third prize is the excellent FUZE T2-C kit for your Raspberry Pi. Worth £90, this kit contains the FUZE case, keyboard, integrated USB hub, FUZE I/O board and power supply.

Details of the prizes can be found at http://www.fuze.co.uk/products.

In addition to the FUZE BASIC programming series that appeared in Issues 25, 26, 27 and 28 of The MagPi, you can also download the FUZE BASIC Programmer's Reference Guide from http://www.fuze.co.uk/resources-2/.

Terms and conditions

• The type of entry is your choice. For example, you could program a game, a tech demo, an artificial intelligence project or anything, as long as it demonstrates some thought, creativity and ingenuity.
• The program must be written using **FUZE BASIC**.
• The closing date for the competition is **Sunday 7th December 2014**. The winners will be announced in Issue 30 of The MagPi.
• To submit your entry / entries, send a zip file for each entry via email to contact@fuze.co.uk by the closing date.
• All prizes are non-refundable and non-transferrable.
• No cash alternative will be offered for any prizes.
• The judge's decision is final.
• Entries will not be accepted from employees of FUZE Technologies Ltd or any company involved in the competition / prize draw and their associated, affiliated or subsidiary companies, their families, agents, or anyone connected with the competition / prize draw.
• Entrants agree that their name, artwork and representations of their submission may be used without remuneration for marketing purposes.
• Entrants agree to grant FUZE Technologies a perpetual license to freely distribute their submissions without remuneration or contract. The copyright of the submission remains with the author.

FUZE®

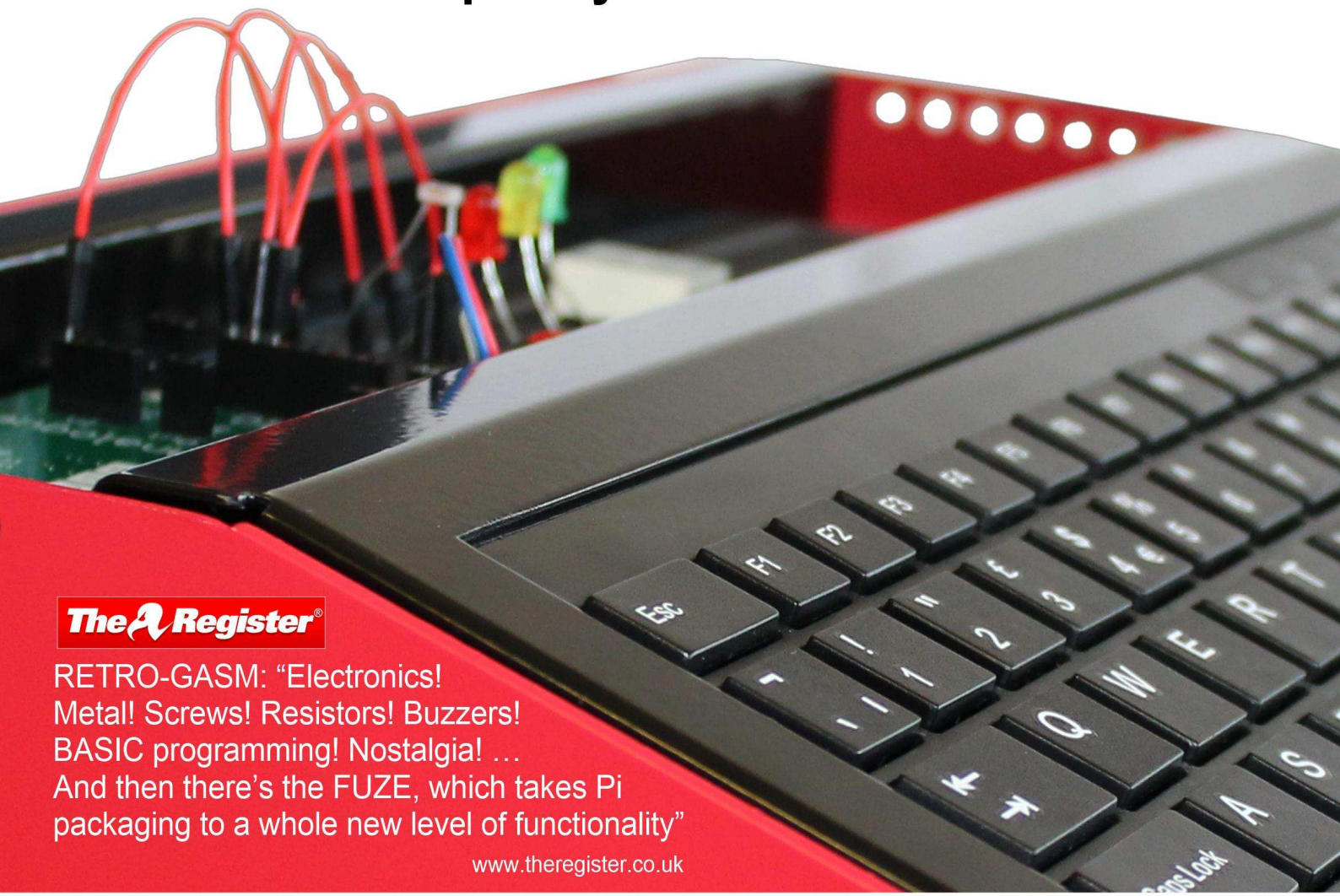
"The FUZE is what the Raspberry Pi was designed for"

THE NATIONAL MUSEUM OF COMPUTING

FUZE Type II

The Ultimate Case

for Raspberry Pi B & the new B+



The Register®

RETRO-GASM: "Electronics! Metal! Screws! Resistors! Buzzers! BASIC programming! Nostalgia! … And then there's the FUZE, which takes Pi packaging to a whole new level of functionality"

www.theregister.co.uk

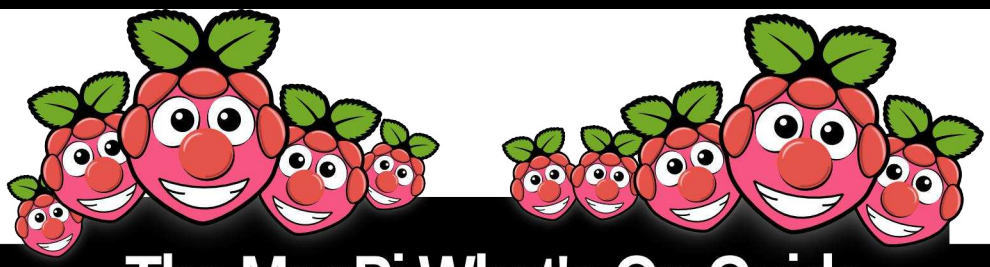
Prices start from £89.99
See FUZE website for details

- **Protect your Pi from physical & static damage**
- **UK keyboard* & 4 Extra USB ports**
- **FUZE I/O Board with GPIO pass-thru**
- **Clearly labeled input output ports**
- **2 Amp power supply and on/off switch!**
- **Adds analogue ports, 4 in & 1 out**

FUZE Technology Ltd - www.fuze.co.uk
+44 (0) 1844 239 432 - contact@fuze.co.uk

* USA & German keyboard layouts are also available

©2014 FUZE & the FUZE logo are trademarks of FUZE Technologies Ltd. Raspberry Pi and the Raspberry Logo are trademarks of the Raspberry Pi Foundation and are used with permission. All rights reserved.



The MagPi What's On Guide

Want to keep up to date with all things Raspberry Pi in your area?

Then this section of The MagPi is for you! We aim to list Raspberry Jam events in your area, providing you with a Raspberry Pi calendar for the month ahead.

Are you in charge of running a Raspberry Pi event? Want to publicise it?
Email us at: editor@themagpi.com

Raspberry Jam Belgium

When: Thursday 13th November 2014, 7.00pm to 10.00pm CET
Where: Oh! Mechelen, Kanunnik De Deckerstraat 20 A, Mechelen 2800 BE, Belgium

The first Raspberry Jam in the region. Both Dutch and non-Dutch speakers are welcome. An introduction to the Raspberry Pi for all. http://www.eventbrite.com/e/13561393493

Hull Raspberry Jam

When: Saturday 22nd November 2014, 10.30am to 3.30pm
Where: The Kingswood Academy, Wawne Road, Bransholme, Hull, HU7 4WR, UK

Supported by the DfE, Raspberry Pi Foundation and RM Education. The event will have workshops, talks and demonstrations. http://www.eventbrite.co.uk/e/13501929635

Raspberry Jam Berlin

When: Saturday 22nd November 2014, 12.00pm to 5.00pm CET
Where: Technische Universität Berlin, 4th Floor, Room FH 403, Fraunhoferstraße 33-36, 10587 Berlin

Raspberry Jam Berlin event is a user group of enthusiasts looking to learn or teach what they know. For full event details visit http://raspberryjamberlin.bytingidea.com

Cotswold Raspberry Jam

When: Saturday 22nd November 2014, 1.00pm to 4.00pm
Where: 95 Promenade, Cheltenham, Gloucestershire, GL50 1HZ. UK

A Raspberry Pi computing event for Gloucestershire and beyond. Deliberately family friendly, with a portion of tickets reserved for children. http://www.eventbrite.co.uk/e/13815828515

Warwick (USA) Raspberry Jam

When: Monday 24th November 2014, 7.00pm to 8.30pm
Where: Warwick Public Library, 600 Sandy Lane, Warwick, RI 02889, USA

Join our monthly Raspberry Jam, an event to learn about the Raspberry Pi, show off your projects and meet some fellow Pi users. Event details at http://goo.gl/JRmJa1

PRINT EDITION AVAILABLE WORLDWIDE

The MagPi is available for FREE from http://www.themagpi.com, from The MagPi iOS and Android apps and also from the Pi Store. However, because so many readers have asked us to produce printed copies of the magazine, we are pleased to announce that printed copies are now regularly available for purchase at the following Raspberry Pi retailers...

Americas

adafruit INDUSTRIES

EMEA

MODMYPI

The Pi Hut

AUTHENTIC SWAG EDUCATING SINCE '12

Pi Supply

AsiaPac

AusPi Technologies Australia

Have Your Say...

The MagPi is produced by the Raspberry Pi community, for the Raspberry Pi community. Each month we aim to educate and entertain you with exciting projects for every skill level. We are always looking for new ideas, opinions and feedback, to help us continue to produce the kind of magazine you want to read.

Please send your feedback to editor@themagpi.com, or post to our Facebook page at http://www.facebook.com/MagPiMagazine, or send a tweet to @TheMagP1. Please send your article ideas to articles@themagpi.com. We look forward to reading your comments.

The MagPi is a trademark of The MagPi Ltd. Raspberry Pi is a trademark of the Raspberry Pi Foundation. The MagPi magazine is collaboratively produced by an independent group of Raspberry Pi owners, and is not affiliated in any way with the Raspberry Pi Foundation. It is prohibited to commercially produce this magazine without authorization from The MagPi Ltd. Printing for non commercial purposes is agreeable under the Creative Commons license below. The MagPi does not accept ownership or responsibility for the content or opinions expressed in any of the articles included in this issue. All articles are checked and tested before the release deadline is met but some faults may remain. The reader is responsible for all consequences, both to software and hardware, following the implementation of any of the advice or code printed. The MagPi does not claim to own any copyright licenses and all content of the articles are submitted with the responsibility lying with that of the article writer. This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License. To view a copy of this license, visit:

http://creativecommons.org/licenses/by-nc-sa/3.0/

Alternatively, send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.