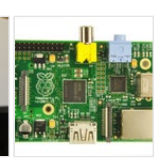
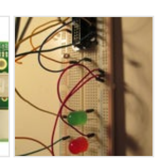
A Raspberry Pi-based Truly Random Number Generator

By I Ate The Pi (/member/I+Ate+The+Pi/) in Circuits (/circuits/) > Raspberry Pi (/circuits/raspberry-pi/projects/)

(/member/I+Ate+The+Pi/)
By **I Ate The Pi (/member/I+Ate+The+Pi/)**
Follow

More by the author: (/member/I+A





Random numbers are essential for all kinds of things, especially cryptography. Computers, however, can only produce pseudorandom numbers, which can be "guessed" by using sophisticated software. Truly random numbers are hard to come by. Luckily, with a few wires and a Ras Pi, one can create a lot of random numbers very quickly.

For this project you will need:

1x Raspberry Pi

3x Breadboard wires

And, for the optional LED output section:

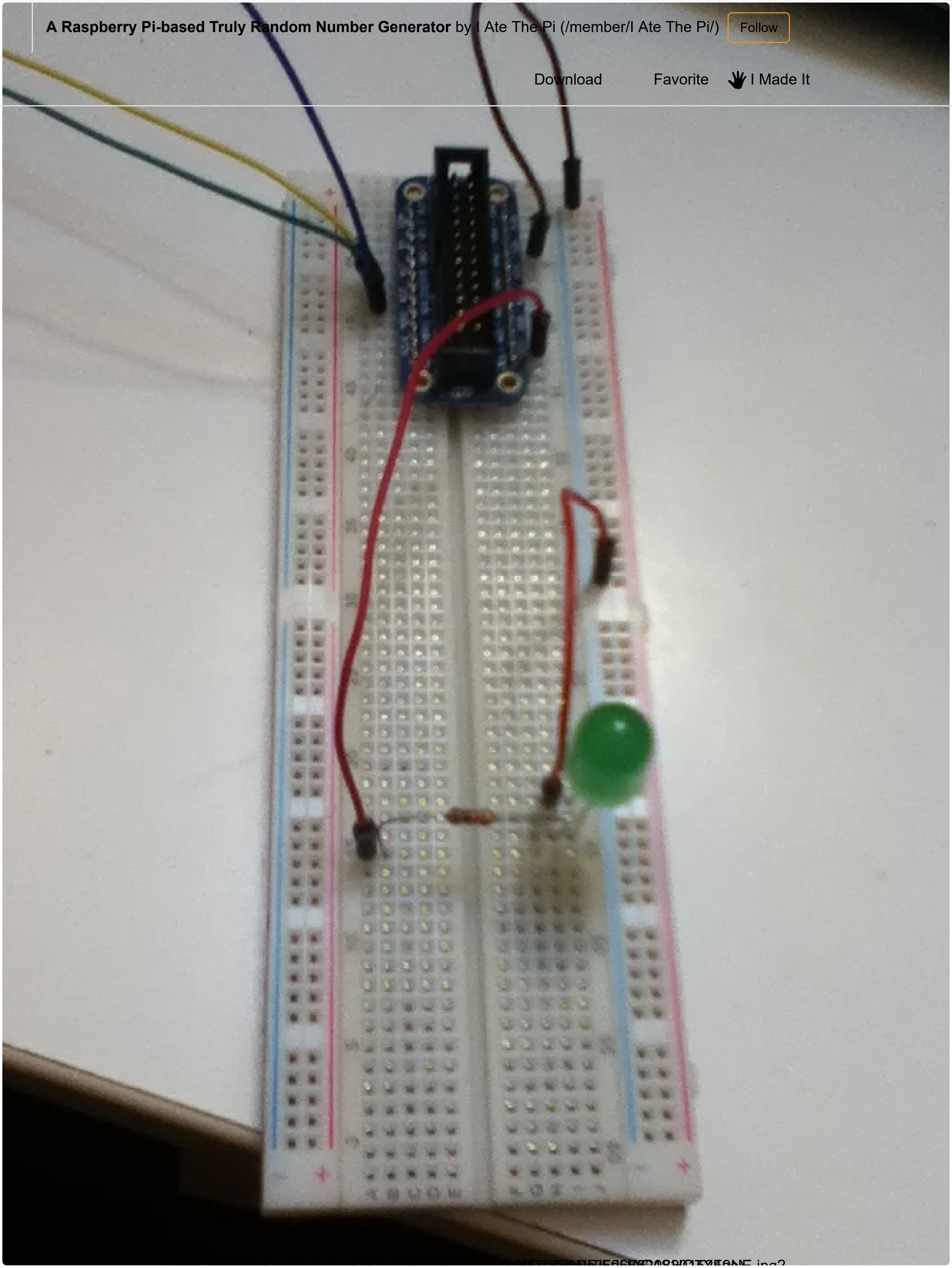
1x LED

1x current-limiting resistor (for the LED)

3x Breadboard wires

Add Tip Ask Question Comment Download

Step 1: Wiring



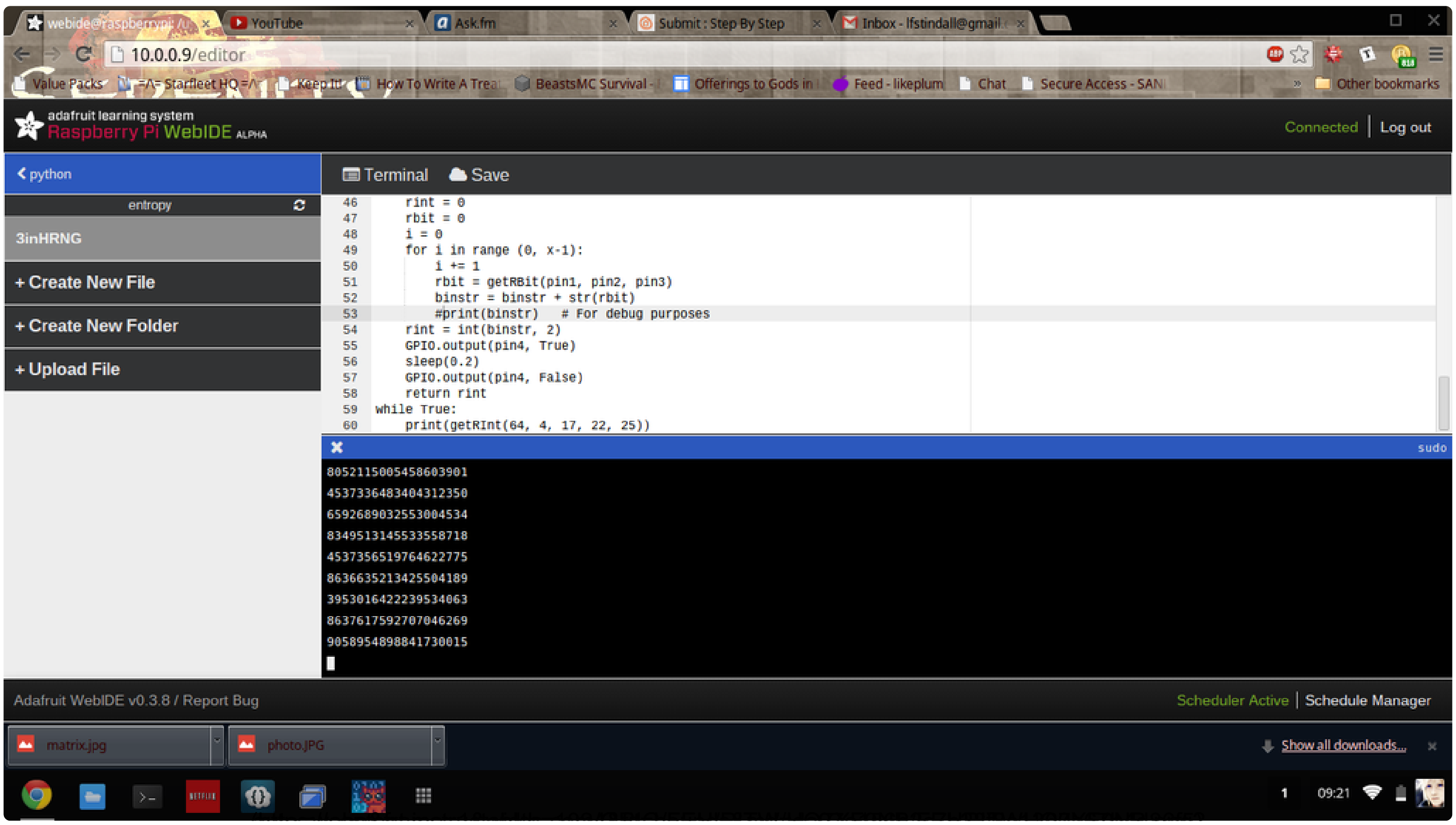
This is the easiest wiring project you've ever done.

For the RNG inputs, connect breadboard wires to GPIO 4, 17, and 22. If that's all you want, you're done. skip to the coding.

For the LED output, connect a resistor and an LED in series (with the resistor on the positive pin of the LED), then connect the Pi's ground to the ground rail on the breadboard. Connect the other end of the LED to ground and the other end of the resistor to GPIO 25.

Add Tip Ask Question Comment Download

Step 2: Code



This code has 6 configurable parameters: Length of the random numbers to output (in bits), the three input pins, the output pin, and the Time to Sleep (tts). A shorter TTS speeds up the generator but reduces entropy. TTS defaults to 0.01 seconds.

#!/usr/bin/env python

#Uses floating inputs on GPIO4, GPIO17, and GPIO22 to generate truly random numbers

#Outputs to GPIO 25 when a new number is done and sends the number to STDOUT

import RPi.GPIO as GPIO

```
import sys
from time import sleep

GPIO.setmode(GPIO.BCM)

def getRBit(pin1, pin2, pin3, tts):   #gets a random set of bits, XORs them, and outputs one random bit
   bit1 = 0
   bit2 = 0
   bit3 = 0
   bitv = 0
   GPIO.setup(pin1, GPIO.IN)
   GPIO.setup(pin2, GPIO.IN)
   GPIO.setup(pin3, GPIO.IN)
   sleep(tts) #Sleep so the CPU can mess around and change the EMF environment
   bit1 = GPIO.input(pin1)
   if bit1:
      bit1 = 1
   else:
      bit1 = 0
   sleep(tts) #Sleep so the CPU can mess around and change the EMF environment
   bit2 = GPIO.input(pin2)
   if bit2:
      bit2 = 1
   else:
      bit2 = 0
   sleep(tts) #Sleep so the CPU can mess around and change the EMF environment
   bit3 = GPIO.input(pin3)
   if bit3:
      bit3 = 1
   else:
      bit3 = 0
   #Now do some XOR logic
   bitv = bit1 ^ bit2
   out = bitv ^ bit3
```

```
def getRInt(x, pin1, pin2, pin3, pin4, tts=0.01): #get an x-bit number by looping through a string
a bunch. Pin4 is LEDout.
    GPIO.setup(pin4, GPIO.OUT)
    binstr = "" #Set up to be converted to binary
    rint = 0
    rbit = 0
    i = 0
    for i in range (0, x-1):
        i += 1
        rbit = getRBit(pin1, pin2, pin3, tts)
        binstr = binstr + str(rbit)
        #print(binstr)   # For debug purposes
    rint = int(binstr, 2)
    GPIO.output(pin4, True)
    sleep(0.2)
    GPIO.output(pin4, False)
    return rint
while True:
    print(getRInt(64, 4, 17, 22, 25, 0.01)) #bits, in1, in2, in3, out, tts
```

Add Tip Ask Question Comment Download

Step 3: Uses and Notes

I suggest using this generator for encryption as the numbers that it generates are highly entropic and pretty much unguessable, barring a bruteforce attack. Using these numbers to seed, for example, the PHP PRNG is a great way to make its output unguessable.
A small note: it may take a VERY long time (~ several minutes) for the generator to make numbers with lots of bits (above 1k). Instead of doing that, I suggest seeding a pseudorandom generator with this truly random output. It's still unguessable.

Add Tip Ask Question Comment Download

Be the First to Share

Did you make this project? Share it with us!

I Made It!

Recommendations

(/NOCTURNAL-SOLAR-LIGHT-BULB-V20/)

NOCTURNAL SOLAR LIGHT BULB V2.0 (/NOCTURNAL-SOLAR-LIGHT-BULB-V20/) by opengreenenergy

69 5.5K

(/Arduino-Powered-Painting-Robot/)

Arduino Powered Painting Robot (/Arduino-Powered-Painting-Robot/) by Technovation (/member/Technovation/) in

88 9.3K

Add Tip

Ask Question

Post Comment

We have a **be nice** policy.
Please be positive and constructive.

Add Images Post

8 Discussions

(/member/JDF7/) JDF7 (/member/JDF7/) 3 years ago
Reply Upvote

I just built a similar program in Node.js with node-rpio. I had it output random numbers to a 8x8 neopixel grid. To do this I hooked up all the GPIO ports to floating wires, and would loop several times each loop recording 8 pseudo randomly chosen wires into one octal. I would then turn it all into a string like this: "123-5-34,0-67-98,255-89-43, ... \n" As you can see, each pixel has a set of 3 numbers to make up RGB. This is then sent over serial to the Arduino which displays the picture. What i found was stunning. These numbers are in fact random, but not from random events. The wires are picking up radio waves! I could clearly see a consistent wave pattern on the pixel grid, and it drove me crazy trying to figure out why. To test the wave interference theory, i simply held up my phone to the wires. As soon as i got within 3cm the program mysteriously crashed with 100% consistency. I then bundled all the wires together with tinfoil. This caused the display to go completely dark, until i moved my hand within a meter. The closer my hand the more visible the wave pattern. It's also worth mentioning i did not connect ground to anything, so no noise was coming from there. Here's the source code: github.com/triforcey/neo-pixel

1 reply

(/member/keithkim/) keithkim (/member/keithkim/) 6 years ago on Step 2
Reply Upvote

Very interesting. Thanks for your instruction.

(/member/scruss/) scruss (/member/scruss/) 7 years ago on Introduction
Reply Upvote

How well does this RNG fare against the FIPS tests included in rngtest? It should pass most of them if you wish to consider it for crypto usage. If it's genuinely really fast, run it against the diehard suite.

Reading floating inputs is strongly influenced by how clean your power supply is, and what other RF noise in the neighbourhood. I'd be very surprised if these weren't just sampling 50 Hz/60 Hz ripple. Even the Arduino folks — who have built-in AtoD converters on their boards, unlike the Raspberry Pi's digital inputs — no longer recommend reading a floating input as a random seed for anything other than toy applications.

Your circuit is easily tampered with (join or ground the wires; you'll get a sweet stream of zeroes) and your code has no way of detecting if the input values are biased and stopping the output. Producing a good source of random bits is *hard*; even IBM got it wrong for years with RANDU (https://en.wikipedia.org/wiki/RANDU), and Intel had to jump through hoops to make RdRand (http://spectrum.ieee.org/computing/hardware/behind-intels-new-randomnumber-generator) useful.

So, while this is a good first effort, the real work comes in verifying and hardening the system. You might find out that the Raspberry Pi's RNG built into the SOC is not so bad after all …

(/member/planetscape/) planetscape (/member/planetscape/) 7 years ago on Introduction

Reply Upvote

If you want a hardware RNG, The RPi has one already, no wiring required:

http://scruss.com/blog/2013/06/07/well-that-was-unexpected-the-raspberry-pis-hardware-random-number-generator/ (http://scruss.com/blog/2013/06/07/well-that-was-unexpected-the-raspberry-pis-hardware-random-number-generator/)

1 reply

(/member/kelseymh/) kelseymh (/member/kelseymh/) 7 years ago on Introduction

Reply Upvote

It sounds like you're trying to rely on the "random" noise present on the ground line to generate what you call "truly random" numbers. Is that true? If so, you should be aware that the randomness of those numbers is strongly coupled to how well filtered your power supply is.

1 reply

Post Comment