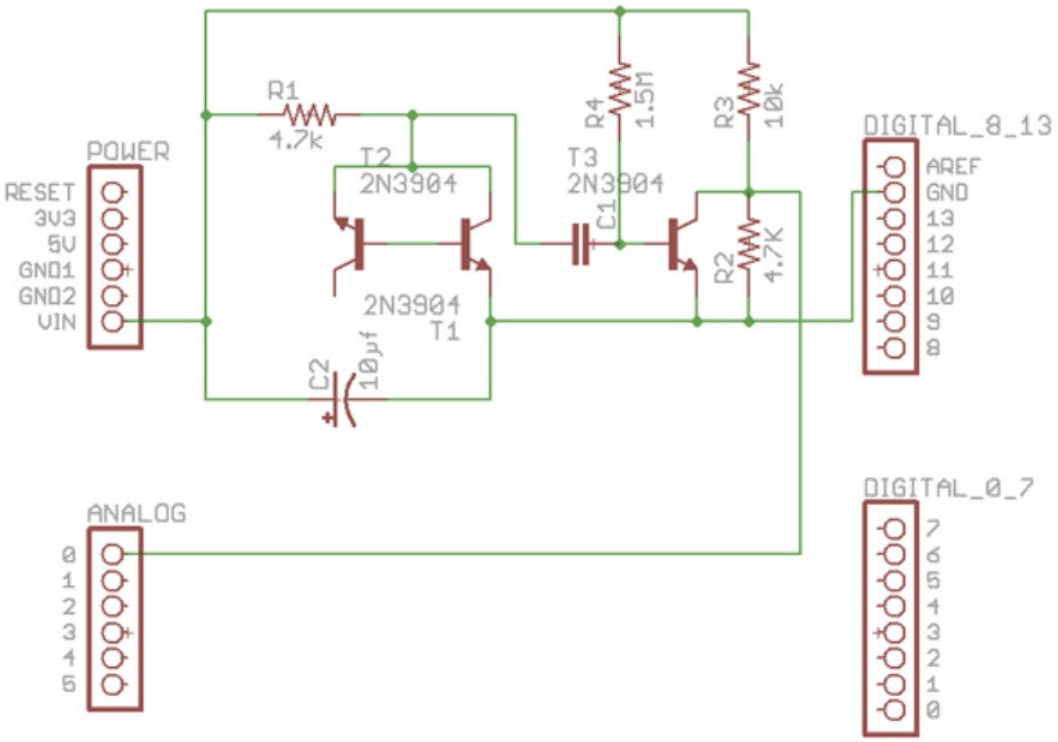
TRNG

This generator uses well known and analysed avalanche noise.
The Design calibrates itself. Thus, if the signal changes, the Arduino micro-controller adapts. It fixed the problem.

The Circuit



How it Works

The two transistors with their bases touching create "avalanche noise in a reverse-biased PN junction." This noise is amplified by the third transistor and sent across a voltage divider to Arduino. On the oscilloscope, the signal looks very fuzzy. In software, that randomness is converted into a stream of 1s and 0s.

The code works like this: for the first 10 seconds, record the signal from the circuit on the analog input pin. Find the median. Then for each subsequent reading of the input pin, see if it is above or below the median. If it is above, output a 1, if it is below, output a 0. Filtering can be applied to the output stream of 1s and 0s to reduce bias.

Part List

Component	Quantity
Arduino	1
2N3904 Transistor	3
4.7k Resistor	2
10k Resistor	1
1.5M Resistor	1
0.1µf Capacitor	1
10µf Capacitor	1
Breadboard	1
12v DC Adapter	1

The Code

```
#define BINS_SIZE 256
#define CALIBRATION_SIZE 50000
#define NO_BIAS_REMOVAL 0
#define EXCLUSIVE_OR 1
#define VON_NEUMANN 2
#define ASCII_BYTE 0
#define BINARY 1
#define ASCII_BOOL 2

/***  Configure the RNG **************/
int bias_removal = NO_BIAS_REMOVAL;
int output_format = BINARY;
int baud_rate = 19200;

/************************************/

unsigned int bins[BINS_SIZE];
int adc_pin = 0;
int led_pin = 13;
boolean initializing = true;
unsigned int calibration_counter = 0;
void setup() {

  pinMode(led_pin, OUTPUT);
  Serial.begin(baud_rate);
  for (int i = 0; i < BINS_SIZE; i++) {
    bins[i] = 0;
  }

}

void loop() {
  byte threshold;
  int adc_value = analogRead(adc_pin);
```

```
  byte adc_byte = adc_value >> 2;
  if (calibration_counter >= CALIBRATION_SIZE) {
    threshold = findThreshold();
    initializing = false;

  }

  if (initializing) {

    calibrate(adc_byte);
    calibration_counter++;

  } else {

    processInput(adc_byte, threshold);

  }

}

void processInput(byte adc_byte, byte threshold) {

  boolean input_bool;
  input_bool = (adc_byte < threshold) ? 1 : 0;

  switch (bias_removal) {

    case VON_NEUMANN:
      vonNeumann(input_bool);
      break;

    case EXCLUSIVE_OR:
      exclusiveOr(input_bool);
      break;

    case NO_BIAS_REMOVAL:
      buildByte(input_bool);
      break;

  }

}

void exclusiveOr(byte input) {

  static boolean flip_flop = 0;
  flip_flop = !flip_flop;
  buildByte(flip_flop ^ input);

}

void vonNeumann(byte input) {

  static int count = 1;
  static boolean previous = 0;
  static boolean flip_flop = 0;
  flip_flop = !flip_flop;

  if (flip_flop) {

    if (input == 1 && previous == 0) {
```

```
      buildByte(0);
    } else if (input == 0 && previous == 1) {

      buildByte(1);

    }

  }

  previous = input;

}

void buildByte(boolean input) {

  static int byte_counter = 0;
  static byte out = 0;
  if (input == 1) {

    out = (out << 1) | 0x01;

  } else {

    out = (out << 1);

  }

  byte_counter++;
  byte_counter %= 8;
  if (byte_counter == 0) {

    if (output_format == ASCII_BYTE) Serial.println(out, DEC);
    if (output_format == BINARY) Serial.print(out, BIN);
    out = 0;

  }

  if (output_format == ASCII_BOOL) Serial.print(input, DEC);

}

void calibrate(byte adc_byte) {

  bins[adc_byte]++;
  printStatus();

}

unsigned int findThreshold() {

  unsigned long half;
  unsigned long total = 0;
  int i;
  for (i = 0; i < BINS_SIZE; i++) {

    total += bins[i];

  }

  half = total >> 1;

  total = 0;

  for (i = 0; i < BINS_SIZE; i++) {
```

```
      total += bins[i];
      if (total > half) {
        break;
      }
    }
    return i;
}
//Blinks an LED after each 10th of the calibration completes
void printStatus() {
    unsigned int increment = CALIBRATION_SIZE / 10;
    static unsigned int num_increments = 0; //progress units so far
    unsigned int threshold;
    threshold = (num_increments + 1) * increment;
    if (calibration_counter > threshold) {
      num_increments++;
      //Serial.print("*");
      blinkLed();
    }
}
void blinkLed() {
    digitalWrite(led_pin, HIGH);
    delay(30);
    digitalWrite(led_pin, LOW);
}
```