Universidad de Valladolid

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNIACIÓN

PROYECTO:

GENERACIÓN DE NÚMEROS ALEATORIOS FÍSICOS EN DISPOSITIVOS ELECTRÓNICOS

Un proyecto realizado por Mario González de la Fuente

Supervisado por:
Juan Carlos García Escartín

Valladolid, Septiembre 2016

Índice general

Índice de figuras

Resumen

Los números aleatorios se han convertido en un elemento muy importante dentro de la informática actual.
El problema que presentan estos es que no son tan sencillos de conseguir como en un primer momento pudiera parecer dado que al final muchos de estos pueden llegar a ser predecibles.
El objetivo del proyecto es dar una visión principal de lo que son los números aleatorios para posteriormente comentar las diferentes aplicaciones y formas de conseguirlos, por último, construiremos un generador de números aleatorios utilizando diferentes componentes electrónicos y su posterior análisis para comprobar que nuestros números generados son realmente aleatorios.
Dentro del proyecto, donde nos vamos a centrar es la parte de la fabricación de generadores de números aleatorios. Vamos a adquirir una serie de componentes que veremos en el apéndice B y vamos a utilizarlos para fabricar nuestro generador. Una vez lo hayamos hecho, utilizaremos un ordenador para tomar muestras de la señal generada y las introduciremos en diferentes programas para poder comprobar que los números generados son verdaderamente aleatorios.
Los fenómenos físicos en los que nos vamos a basar principalmente serán el ruido térmico y el efecto Zener. Por un lado, el ruido térmico está muy presente en las resistencias mientras que el efecto Zener se encuentra en diodos Zener por lo que estos dos tipos de componentes serán muy importantes en los generadores diseñados.

Abstract

Random numbers have become a very important element in modern computing.
Their problem is that they are not as easy to get as first could look, as many of these can become predictable.
The aim of the project is to give a main vision of what the random numbers are to later discuss the various applications and forms and, finally, build a random number generator using different electronic components for further processing and see that our numbers generated are truly random.
Within the project, where we are going to focus on is the part of the manufacture of random number generators. We will acquire a number of components that we will see on Appendix B and we will use them to make our generator. Once we do thar, we will use a computer to take samples from the generated signal and we will introduce them to different programs to verify that the generated numbers are truly random.
Physical phenomena in which we will be based primarily on are thermal noise and the Zener effect. On one hand, the thermal noise is very present in the resistors while the Zener effect is in Zener diode so these two types of components will be very important in the generators designed.

Palabras clave

Números aleatorios, generación de números aleatorios, ruido térmico, efecto Zener.

Capítulo 1

Introducción

1. Motivación

Habiendo finalizado todas las asignaturas del grado en ingeniería de tecnologías específicas de telecomunicación, mención sistemas electrónicos, me encuentro haciendo las prácticas en la empresa Audiotec S.A. y con la duda de si comenzar mi trabajo de fin de grado o hacerlo el siguiente curso.
Finalmente me decido a hacerlo durante el verano para poder dedicar el nuevo curso a nuevos proyectos en mi vida y, por lo tanto, empiezo a buscar el posible tema sobre el que puedo escribir. Comienzo buscando por los que ofrecen los profesores que me dan clase y encuentro uno que me llama la atención sobre generación de números aleatorios, y empiezo a indagar sobre el tema aprendiendo más y más y entendiendo lo necesarios que son en nuestra vida hoy en día, tan ligada a los ordenadores, los cuales tienen dificultades para generarlos por si sólos y es por esto que contacto con Juan Carlos García Escartín para comenzarlo cuanto antes.
Es verdad que al principio me encontraba algo perdido con el tema y que, hasta que he empezado a diseñar los generadores han tenido que pasar varios días formándome en los elementos que componen el proyecto pero, actualmente, puedo decir que he alcanzado gran soltura en el mismo siendo, capaz de detectar los posibles fallos cuando los tengo mucho más rápido que antes.
A decir verdad, el hecho de estar haciendo el TFG sobre un tema que utiliza el ruido, el cual se suele considerar un problema en el mundo de la electrónica, me ha servido para ver que no todo en esta vida es bueno o malo sino depende del punto de vista con el que lo veas. También, este TFG me ha permitido perfeccionar mis conocimientos en el mundo de la electrónica, el cual me apasiona y me ha permitido manejarme en el tanto en el ámbito digital (al programar el dispositivo de captura de datos) como en el analógico (al diseñar el generador).

2. Objetivos del proyecto

Los generadores de números aleatorios, como vamos a ver a lo largo del proyecto, se han convertido en un elemento muy importante en el mundo informático hoy en día, y es por eso que deben diseñarse de forma que cumplan una serie de especificaciones en función de la utilidad que se le vaya a dar.
A lo largo del proyecto trataremos de resolver los siguientes problemas:

- Dar una definición adecuada de lo que es un número aleatorio.
- Mostrar la utilidad de los números aleatorios.
- Explicar las formas de obtener números aleatorios, poniendo énfasis en ejemplos concretos.

- Enseñar el método que vamos a usar para evaluar la aleatoriedad de un generador en concreto.
- Diseñar nuestro propio generador de números aleatorios y evaluarlo.
- Obtener una serie de conclusiones que nos permita tener claro cual es un método adecuado para diseñar un generador de forma que, al evaluarlo, cumpla los test necesarios.

3. Estructura del proyecto

Este proyecto se va a estructurar en 11 capítulos, de forma que cada uno se pueda consultar de manera independiente en función de la información que se busque.
Los capítulos que lo componen son los siguientes:

Capítulo 1: Introducción
En este capítulo se presenta el proyecto a realizar con su motivación para realizarlo y sus objetivos.

Capítulo 2: Números aleatorios
En este capítulo vamos a dar unas pequeñas nociones sobre lo que son los número aleatorios, de forma que, cuando lo hallamos leído, tendremos claro lo que son, cómo se pueden obtener y qué utilidad tienen.

Capítulo 3: Ruido
En este capítulo hablaremos del ruido y de los diferentes tipos de ruido que pueden existir.

Capítulo 4: Ruido en diodos Zener
Veremos los diferentes tipos de ruido en diodos Zener de forma que obtengamos números aleatorios a partir de ellos.

Capítulo 5: Dispositivos para la captura de números aleatorios
Explicaremos las diferentes placas de desarrollo utilizadas, cada una con sus ventajas y con sus respectivas limitaciones.

Capítulo 6: Método para el análisis de los datos tomados
Veremos el programa Dieharder, herramienta de Linux que nos permitirá analizar los datos producidos por los generadores. Veremos los diferentes test que realiza el programa y comprobaremos su funcionamiento con los generadores internos de Arduino y Linux.

Capítulo 7: Diseño del generador usando resistencias
Veremos una propuesta de generador diseñado por nosotros basado en el ruido térmico de las resistencias.

Capítulo 8: Diseño del generador usando diodos Zener
Veremos una propuesta de generador diseñado por nosotros basado en

las variaciones aleatorias de corriente en los diodos Zener en ruptura.

Capítulo 9: Conclusiones
En este capítulo se describen las conclusiones obtenidas tras la realización del proyecto.

Apéndice A: Lista de materiales
En el apéndice A, haremos un estudio del coste generado por los materiales del trabajo de fin de grado.

Apéndice B: Códigos del proyecto
En el apéndice B, veremos los códigos necesarios para programar todos los componentes utilizados por el proyecto.

Bibliografía: Veremos las diferentes referencias utilizadas para documentar todo el proyecto.

Capítulo 2

Números aleatorios

Los números aleatorios son un recurso esencial en la ciencia, la tecnología y muchos aspectos de la vida cotidiana. Se requiere aleatoriedad en diferentes grados en aplicaciones como la criptografía, la simulación, la coordinación de las redes de ordenadores o loterías. Algunas aplicaciones requieren una pequeña cantidad de números aleatorios y siguen utilizando métodos manuales y mecánicos para generar aleatoriedad, como lanzar una moneda al aire, lanzando un dado, haciendo girar una ruleta o un sacando una pelota de una máquina de lotería. Aquí, vamos a referirnos a la generación de números aleatorios para los ordenadores.

1. Qué son los números aleatorios

Es aquel obtenido al azar, es decir, todo conjunto de números que tengan la misma probabilidad de ser elegidos (en la mayoría de los casos) y que la elección de uno no dependa de la elección del otro (que no haya correlación). El ejemplo clásico más utilizado para generarlos es el lanzamiento repetitivo de una moneda o dado ideal no trucado.
Una secuencia de números aleatorios generados a partir de una fuente física o método matemático debe poseer algunas propiedades:[**6**]

Secuencias no correlacionadas: Esto significa que, en una sucesión de números aleatorios, una subsecuencia de números aleatorios no debe estar relacionada con ninguna otra.

Independencia estadística y equiprobabilidad: Lo cual implica que la probabilidad de que un número específico aparezca en la sucesión debe ser la misma para cada uno de los elementos del conjunto de números aleatorios; además, la aparición de un número dentro de la sucesión no implica ni excluye la aparición de cualquier otro.

Periodo máximo: Los generadores de números generalmente son cíclicos, por lo cual es deseable que cada uno de los elementos del conjunto aparezca exactamente una vez en la secuencia antes de que empiecen a repetirse (ciclo completo). El periodo y las propiedades de la secuencia no deben depender del valor inicial.

Uniformidad: Los números aleatorios pueden ser modelados mediante la función de densidad de probabilidad de la variable aleatoria uniforme. Esto es, si R es la fuente generadora de la sucesión de números aleatorios, entonces

$$R \sim U(r, a = 0, b = 1)$$

donde $R = \frac{X}{m}$.

1.1. Qué son los números pseudo-aleatorios. En informática, es importante distinguir entre los números generados mediante algoritmos que siguen las estadísticas de las distribuciones aleatorias y números aleatorios generados a partir de los acontecimientos físicos impredecibles. La generación de números aleatorios directamente desde un ordenador parece una idea particularmente atractiva. Los métodos que producen números aleatorios a partir de un algoritmo determinista son llamados generadores de números pseudo-aleatorios (o PRNGs). Si bien es evidente que cualquier secuencia generada algorítmicamente no puede ser verdaderamente aleatoria, para muchas aplicaciones esta aparente aleatoriedad es suficiente.
El principal problema de los números pseudo-aleatorios es que no son impredecibles del todo y esto supone un inconveniente dado que estos números se suelen utilizar en cifrado y es por eso que surge la necesidad de generar números realmente aleatorios.

2. Historia de los números aleatorios por ordenador

Podemos afirmar que el origen formal de la generación de números aleatorios por ordenador comienza en la década de los años 40 con el nacimiento del método de Montecarlo, que explicaremos más adelante, y von Neumann, Metropolis, Ulam y Lehmer pueden ser nombrados como los pioneros en este campo. John von Neumann aparentemente conjeturó el potencial de los ordenadores para tratar problemas estocásticos en 1945 [1]. Durante los cuarenta, la simulación de procesos estocásticos permaneció restringida al proyecto secreto del Departamento de Defensa de Estados Unidos. La publicación de *The Monte Carlo method* por N. Metropolis y Stanislaw M. Ulam en 1949 denota el inicio de la historia oficial del método. Dos años más tarde, D.H. Lehmer propuso el generador lineal de congruencia, el cual, con pequeñas modificaciones propuestas por Thomson y Rotenberg, ha llegado a convertirse en el método para la generación de números aleatorios más utilizado en la actualidad. Aunque, originalmente, el método de Montecarlo fue implementado usando ruletas y dados en los problemas de difusión de los neutrones, su auge y creciente uso se debe a que posteriormente se empezaron a emplear números aleatorios generados por ordenador.
Antes de la llegada de los ordenadores, los números aleatorios eran generados por dispositivos físicos. En 1939, Kendall y Babington-Smith publicaron 100.000 dígitos aleatorios obtenidos con un disco giratorio iluminado con una lámpara flash. En 1955, la Rand Corporation publicó un millón de dígitos producidos controlando una fuente de pulsos de frecuencia aleatoria. Estos se encuentran disponibles en cintas magnéticas de la Rand [**6**].

3. Qué utilidad tienen los números aleatorios

Los generadores de números aleatorios son ampliamente utilizados en campos tales como el modelado por ordenador, estadística, diseño experimental,

[1]Un proceso estocástico es un concepto matemático que sirve para caracterizar una sucesión de variables aleatorias (estocásticas) que evolucionan en función de otra variable, generalmente el tiempo.

etc.
Destaca su uso en el llamado método de Montecarlo, con múltiples utilidades, por ejemplo para hallar áreas/volúmenes encerradas en una gráfica y cuyas integrales son muy difíciles de hallar o irresolubles. Mediante la generación de puntos basados en estos números, podemos hacer una buena aproximación de la superficie/volumen total, encerrándolo en un cuadrado/cubo, aunque no lo suficientemente buena. Este método consiste en los siguiente:

- En primer lugar, el software coloca el modelo dentro de un volumen conocido (por ejemplo, dentro de un cubo de $1m^3$ de volumen).
- A continuación, genera un punto aleatorio del interior del volumen conocido, y registra si el punto "ha caído"dentro o fuera del modelo. Esto se repite un gran número de veces (miles o millones), consiguiendo un registro muy grande de cuántos puntos han quedado dentro y cuántos fuera.
- Como la probabilidad de que caiga dentro es proporcional al volumen del modelo, la proporción de puntos que han caído dentro con respecto al total de puntos generados es la misma proporción de volumen que ocupa el modelo dentro del cubo de $1m^3$.

Un campo donde resulta imprescindible es en la programación de juegos, donde a menudo se necesita disponer de series elegidas al azar. Por ejemplo, para crear nubes con patrones diferentes según escenarios. Esto es aún más necesario en aquellos juegos donde el azar es primordial (como juegos donde el azar está implícito en la propia dinámica del juego, por ejemplo, juegos de cartas, que necesitan ser barajadas) o incluso una cuestión que garantice la fiabilidad para dotar al juego de imparcialidad, como en los casos donde en esos juegos se realizan apuestas económicas y que suelen recurrir al algoritmo Fisher-Yates. Este algoritmo consiste en ir eligiendo elementos de un vector y eliminándolos del mismo para colocarlos en un nuevo vector vacío, este proceso continuará hasta que el vector inicial quede sin elementos.
Otro campo donde resultan muy útiles los números aleatorios es la criptografía. Está claro que la seguridad de la información es importante para el comercio electrónico. Los servicios de seguridad están basados en mecanismos de criptografía, los cuales, a su vez, hacen uso de números aleatorios. Estos son utilizados para:

- Mecanismos de autentificación (usando una clave generada con números aleatorios) para proteger un mensaje.
- Mecanismo de confidencialidad, utilizando claves secretas o claves de sesión (obtenidas con números aleatorios) para proteger datos durante un intercambio de información.
- Mecanismo de firma digital requieren claves privadas, los cuales son generados con números aleatorios.

Otro escenario de uso es el sistema GPS, este sistema, desarrollado por el Departamento de Defensa de Estados Unidos, tiene como principal característica que utiliza señales de satélites codificadas, las cuales pueden ser procesadas por un receptor GPS, entonces el receptor es capaz de determinar su posición.

Estas señales son códigos pseudo-aleatorios que se generan en los satélites y varían de acuerdo a la fase de la comunicación [**6**].

4. Formas de obtener números aleatorios

Los generadores de números verdaderamente aleatorios miden algunos procesos físicos impredecibles o, al menos, difíciles de predecir y utilizar los resultados para crear una secuencia de números aleatorios. O bien se basan en los valores impredecibles a los que se puede acceder desde el software en el ordenador o crean la secuencia en un dispositivo de propósito especial que lo alimenta en el sistema operativo. El proceso de recolección de datos impredecibles generalmente se llama recolección de entropía. Algunas de las fuentes de entropía estándar a las que el sistema operativo puede tener acceso incluyen los datos de la tarjeta de sonido, los tiempos de acceso al disco, el tiempo de las interrupciones o los datos de interacción del usuario, como el movimiento del ratón o pulsaciones de teclas. Hay generadores físicos de números verdaderamente aleatorios (los llamaremos TRNG a partir de ahora en referencia al nombre en inglés *Trully Random Number Generator*) basados en principios diferentes, tales como los sistemas caóticos, ruido térmico en los circuitos electrónicos, osciladores de libre funcionamiento, o parámetros biométricos como algunos ejemplos. Los generadores de números aleatorios cuánticos son un caso particular de TRNGs físicos en las que los datos son el resultado de un suceso cuántico. A diferencia de otros sistemas físicos donde la incertidumbre es el resultado de un conocimiento incompleto del sistema, la aleatoriedad real es una parte esencial de la mecánica cuántica como la conocemos.
A primera vista, parece que los TRNGs son más deseables que los métodos deterministas, sin embargo, hay inconvenientes que han impedido su adopción más amplia como pueden ser:

- Tasa de generación limitada. La formación suele producir números aleatorios a un ritmo mucho menor que los métodos de software. En muchos casos, hay una limitación fundamental en la tasa de cambio de la señal muestreada. Algunos de los problemas de los TRNGs son parámetros físicos. Si el sistema se muestrea a una velocidad alta, no hay tiempo suficiente para que el sistema cambie y los números aleatorios no serán del todo independientes.
- Es complicado dar argumentos convincentes de que los datos tomados son del todo aleatorios.
- Añadir un dispositivo externo suele presentar un inconveniente.
- Los fallos son difíciles de detectar. Si un generador de números aleatorios de hardware falla, puede ser difícil darse cuenta de ello. Oficialmente se recomienda la introducción de un test de arranque, una prueba de fracaso total y una prueba en línea para comprobar errores durante la operación.

En los próximos capítulos veremos los métodos físicos de los que disponemos para obtener números aleatorios. [**19, 17, 16**]

Capítulo 3

Ruido

El fenómeno físico que vamos a utilizar para obtener números aleatorios de las resistencias es lo que denominamos ruido. El ruido es toda aquella señal anómala o perturbación en un sistema de transmisión que provoca que la información no llegue con claridad o se distorsione. Este tipo de señales, que no contienen información, están compuestas por una mezcla aleatoria de longitudes de onda. En sistemas de comunicación se le asigna el término ruido a la falta de información en una señal.
En general el ruido eléctrico se define como cualquier energía eléctrica no deseada presente en la pasa-banda útil de un circuito de comunicaciones o, en términos físicos, como toda fluctuación aleatoria de una magnitud eléctrica, (ya sea corriente, tensión, etc.), que tienda a enmascarar la señal de interés. Otra manera de definir al ruido es como "señales eléctricas indeseables que penetran en los equipos, o perturbaciones naturales que degradan el comportamiento deseado de un componente electrónico. En la toma de datos digital, los errores que son causados por ruidos se manifiestan como bits adicionales o faltantes.
Para que algo perturbe a una señal debe tener:

1. Una energía indeseada de la misma naturaleza que la señal perturbada.
2. Se debe encontrar presente en la misma banda de frecuencia que la señal.

El ruido en un canal de comunicación es aditivo, esto quiere decir que habrá una variación en la amplitud de la señal transmitida, lo que proporciona un error, esto podrá evitarse al limitar la amplitud, pero dado que el ruido es una señal aleatoria, su amplitud y su fase también son aleatorias. El ruido se representa como un vector que se suma a la señal, dando como resultado una portadora con una amplitud y una fase variable, lo que dificulta la transmisión de la información dado que esta se transmite en la fase, en la amplitud y en la frecuencia de la portadora [**14**].
A continuación, veremos, divididos en secciones, los diferentes tipos de ruido que pueden existir.

1. Ruido externo o interferencias

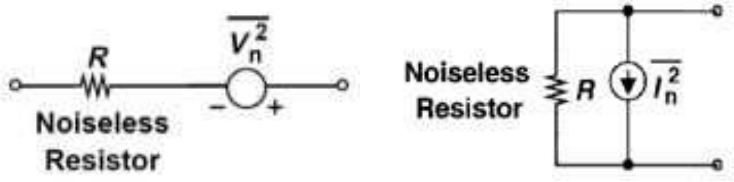
Es el ruido producido por el medio de transmisión, es decir corresponde al que se genera en un punto del sistema como consecuencia de acoplamiento eléctrico o magnético con otro punto del propio sistema, o con otros sistemas naturales (tormentas, etc.) o construidos por el hombre (motores, equipos, etc.). El ruido de interferencia puede ser periódico, intermitente, o aleatorio. Normalmente, se reduce minimizando el acoplo eléctrico o electromagnético,

bien a través de blindajes, o bien, con la re-orientación adecuada de los diferentes componentes y conexiones. Este ruido incluye:

- Ruido artificial.
- Ruido atmosférico.
  Se produce por la estática que se encuentra dentro de la atmósfera terrestre, la cual está cargada de estática que se manifiesta habitualmente en forma de relámpagos, centellas, rayos, etc. La respuesta de estos ruidos no es plana, sino creciente desde frecuencias bajas hasta los 20 MHz y decreciente de allí en adelante, con valores por encima de los 30 MHz.
- Ruido espacial.

2. Ruido térmico

Este ruido está asociado con el movimiento browniano de electrones dentro de un conductor.
El movimiento aleatorio de los electrones en un conductor produce fluctuaciones en la tensión medidas en sus extremos.
El espectro es proporcional a la temperatura absoluta, esto quiere decir que cuanta menor sea la temperatura, menor será el ruido térmico.
Se puede describir como una fuente de tensión o de corriente cuya polaridad será arbitraria.
Este tipo de ruido no aparecerá ni en condensadores ni en inductancias.



(a) Con fuente de tensión. (b) Con fuente de corriente.

FIGURA 1. Caracterización del ruido térmico [**12**].

Este ruido está caracterizado por la siguiente ecuación:

$$\overline{V_n{}^2} = 4kTR\Delta f = 4kTR(V^2/Hz)$$

$$\overline{I_n{}^2} = \frac{4kT}{R}\Delta f = \frac{4kT}{R}(A^2/Hz)$$

Donde $k = 1{,}38 \cdot 10^{-23}\frac{J}{K}$, T es la temperatura absoluta en grados Kelvin y Δf es el ancho de banda del canal en Hz. El ruido térmico se puede reducir enfriando la fuente de ruido y este mismo principio se está aplicando en algunos receptores de radio utilizando enfriadores criogénicos, para mejorar la sensibilidad del receptor [**12**].

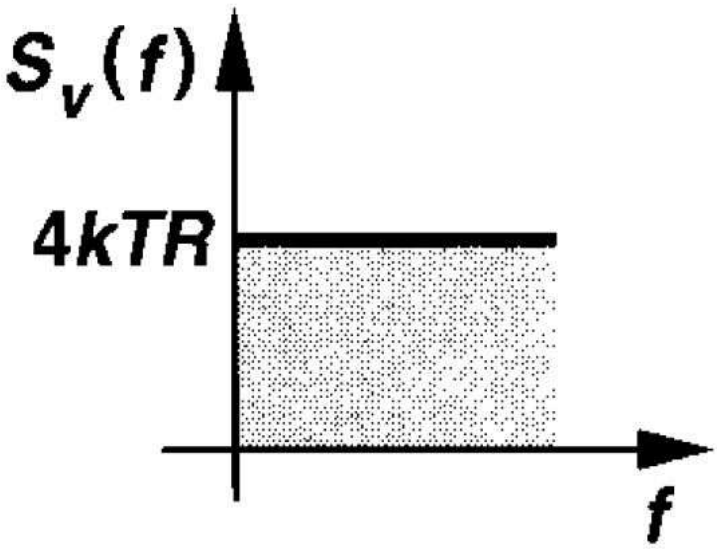


FIGURA 2. Espectro del ruido térmico [**12**].

3. Ruido flicker (1/f)

Se debe a los dopantes que capturan y liberan portadores de forma aleatoria. La constante de tiempo asociada con el proceso da lugar a una señal de ruido con energía concentrada en bajas frecuencias.
En los MOSFETs, los estados de energía en la interfaz entre el silicio y el óxido capturan y liberan portadores de forma aleatoria y a diferentes velocidades, dando lugar a un ruido en la corriente de drenador.
La potencia media del ruido flicker no se puede predecir fácilmente, depende de la tecnología y viene determinada por la siguiente ecuación.

$$\overline{V_{n,1/f}}^2 = \frac{K}{C_{ox}WL}\frac{1}{f}$$

$$\overline{I_{n,1/f}}^2 = \frac{K}{C_{ox}WL}\frac{1}{f}{g_m}^2$$

Con $K \approx 10^{-25}V^2f$, C_{ox} es la capacidad del óxido W es el ancho del canal, L es el largo del canal y g_m es la transconductancia del transistor [**12**].
El método que tenemos para minimizar este fenómeno es aumentando el área (WL) o trabajando a frecuencias más altas.

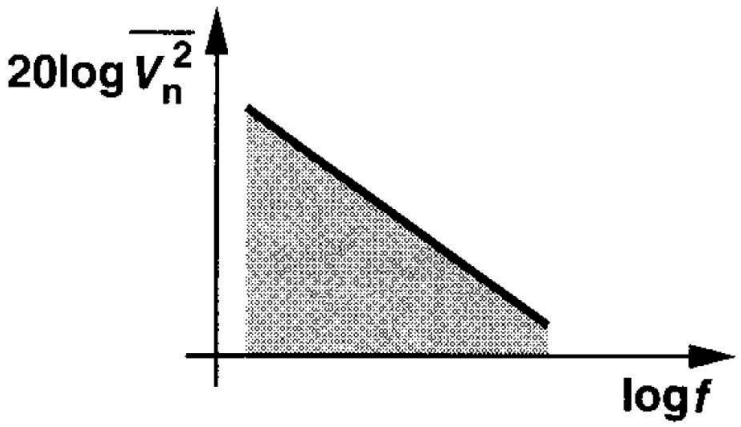


FIGURA 3. Espectro del ruido flicker [**12**].

4. Ruido shot

Consiste en fluctuaciones aleatorias de la corriente eléctrica a través de un conductor causadas por el hecho de que la corriente se transporta en cargas discretas (electrones). Se origina por el movimiento de los electrones o de otras partículas cargadas a través de una unión. Esto no sólo ocurre en las uniones p-n, sino en cualquier conductor, incluso en las situaciones en que la carga no esté bien localizada. En la óptica cuántica, el ruido shot se origina por las fluctuaciones de los fotones detectados, de nuevo como consecuencia de la discretización (en este caso, de la discretización de la energía en el campo electromagnético). El ruido shot es una parte importante del ruido cuántico. El ruido shot resulta medible tanto en mediciones del orden de unos pocos fotones mediante fotomultiplicadores como en mediciones de intensidades luminosas más fuertes realizadas a través de fotodiodos cuando se emplean osciloscopios de alta resolución temporal. En tanto que la fotocorriente es proporcional a la intensidad luminosa (número de fotones), las fluctuaciones del campo electromagnético están normalmente presentes en la intensidad de corriente eléctrica medida.

5. Ruido de tiempo de tránsito

Cualquier modificación a una corriente de portadores conforme pasa desde la entrada hasta la salida de un dispositivo (tal como el emisor al colector de un transistor). Produce una variación aleatoria irregular calificada como ruido de tránsito.

6. Otros tipos de ruido

- **Ruido de transición.**
  Causado por los desfases que aparecen entre las tensiones y las corrientes en el interior de los dispositivos debidos al tiempo que los portadores de carga tardan en atravesarlos, desfases que se incrementan con el aumento de la frecuencia, lo que provoca una disminución de la impedancia de entrada del dispositivo, un incremento de la realimentación

positiva del ruido y en definitiva una potencia adicional de ruido que depende cuadráticamente de la frecuencia y que se hace rápidamente dominante sobre los demás tipos de ruido.

- **Ruido de impulsos o agujas.**
  Es el principal causante de errores en la comunicación de datos, se reconoce por un "click" durante las comunicaciones de voz. Este provoca un error en ráfaga donde el impulso puede variar de 1 o 2 bits a decenas o centenas de estos dependiendo del índice de transferencia de información. La principal fuente de este ruido es la variación de voltajes en líneas adyacentes y falsos contactos entre otros.

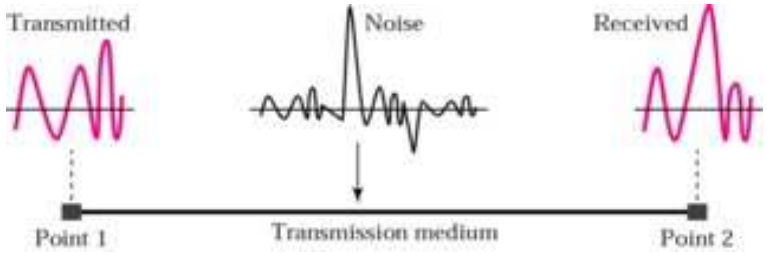


FIGURA 4. Ejemplo de ruido de impulsos.

- **Ruido de amplitud.**
  Este tipo de ruido comprende un cambio repentino en los niveles de potencia causado principalmente por amplificadores defectuosos, contactos sucios con resistencias variables o por labores de mantenimiento.
- **Ruido de intermodulación.** Este tipo de ruido se presenta por la intermodulación de dos tipos de líneas independientes, que pueden caer en un tipo de banda de frecuencias que difiere de ambas entradas, así mismo puede caer dentro de una banda de una tercera señal, usualmente aparece cuando el sistema de transmisión es no lineal, lo que provocará la aparición de nuevas frecuencias. Las nuevas frecuencias se suman o restan con las originales dando lugar a componentes de frecuencias que antes no existían y que distorsionan la verdadera señal.
- **Ruido rosa.**
  Es una señal o un proceso con un espectro de frecuencias de tal manera que su densidad espectral de potencia es proporcional a su reciproco de frecuencia, su energía por frecuencia disminuye en 3 dB por octava, lo que hace que cada banda tenga la misma energía. Se usa mucho en pruebas de mediciones acústicas.
- **Ruido en los canales telefónicos (diafonía o cruce aparente).**
  Se ocasiona por las interferencias que producen otros pares de hilos telefónicos próximos (conocida como cruce de líneas o *crosstalk*). Es un fenómeno mediante el cual una señal que transita se induce en otro que discurre paralelo, perturbándolo.
- **Ruido en los canales telefónicos (eco).**
  Es una señal de las mismas características que la original pero atenuada y retardada respecto a ella. El efecto nocivo del eco afecta tanto a las

conversaciones telefónicas como a la transmisión de datos y es mayor cuanto menos "atenuada" y más "retardada" llega la señal del eco.

Existen otros tipos de ruido que veremos en el siguiente capítulo ya que están íntimamente relacionados con los diodos Zener y he creído conveniente incluirlos ahí [**14, 18**].

Capítulo 4

Ruido en diodos Zener

El voltaje de polarización inversa máxima que se puede aplicar a un diodo p-n está limitada por su tensión de ruptura. Esta es la tensión aplicada que debemos aplicar para alcanzar una región de ruptura que definiremos más adelante. La tensión de ruptura se caracteriza por el rápido aumento de la corriente bajo polarización inversa.
El voltaje de ruptura es un parámetro clave de dispositivos de potencia. La tensión de ruptura de los dispositivos lógicos es igualmente importante ya que uno normalmente reduce las dimensiones del dispositivo sin reducir los voltajes aplicados, lo que aumenta el campo eléctrico interno.
Dos mecanismos pueden causar ruptura: la multiplicación por avalancha y el efecto túnel o efecto Zener.
El voltaje de ruptura resultante es inversamente proporcional a la densidad de dopaje si se ignora la débil dependencia del dopaje con el campo eléctrico en ruptura.
A continuación, veremos los efectos que producen esta tensión de ruptura y describiremos su relación con la generación de números aleatorios.

1. Ruptura por efecto avalancha

La ruptura por avalancha está causada por el impacto de pares electrón-hueco. Si bien, es muy pequeño, existe un flujo de corriente en condiciones de polarización inversa. El campo eléctrico en esta región de vaciamiento puede ser muy alta. Los pares electrón-hueco que entran en la región de vaciamiento se someten a una tremenda aceleración y, como estos portadores acelerados chocan con los átomos, pueden extraer electrones de ellos, creando pares electrón-hueco adicionales y, por lo tanto, generar corriente. Como estos portadores secundarios son barridos dentro de la región de vaciamiento, ellos también se aceleran y el proceso se repite una y otra vez. Este efecto es similar al de una avalancha y de ahí viene su nombre. La eficacia del efecto de avalancha se caracteriza por un factor llamado de multiplicación M que depende de la tensión inversa y que se determina con la ecuación $M = \frac{1}{1-\left|\frac{V}{V_{br}}\right|^n}$, donde n está en el rango 2-6, V es el voltaje aplicado (inverso) y V_{br} es el voltaje de ruptura.
La ruptura por avalancha se produce en uniones p-n ligeramente dopadas donde la región de vaciamiento es grande. La densidad del dopado controlará la tensión de ruptura.
Este efecto depende de la temperatura con un coeficiente positivo, es decir, a medida que aumenta la temperatura, aumenta la tensión de ruptura.

2. Ruptura por efecto Zener

La ruptura por efecto Zener sucede en uniones p-n fuertemente dopadas, lo cual hace que la región de vaciamiento sea extremadamente delgada. De hecho, es tan delgada que los portadores no pueden acelerar lo suficiente como para causar ionización por impacto. Sin embargo, la barrera de potencial es tan fina que es posible atravesarla por el denominado efecto túnel haciendo que la corriente fluya. El análisis de este efecto es idéntico al de un túnel en una unión de metal-semiconductor.
La corriente que produce este efecto se obtiene a partir del producto de la carga de los portadores, la velocidad y de la densidad de portadores. La velocidad es igual a la velocidad de Richardson, que es la velocidad con la que, en promedio, los portadores se acercan a la región de vaciamiento, mientras que la densidad de portadores es igual a la densidad de electrones disponibles multiplicada por la probabilidad de que se produzca este efecto.
Este efecto tiene un coeficiente de temperatura esencialmente negativo, el voltaje de ruptura de estos diodos disminuye con la temperatura.

3. Conclusiones

Diodo Zener y diodo de avalancha son términos que, a menudo, se utilizan indistintamente, siendo el primero mucho más común. Ambos se refieren a la ruptura de un diodo bajo polarización inversa. En concreto, cuando un diodo se polariza en inversa, circula muy poca corriente y el diodo se comporta aproximadamente como circuito abierto. A medida que aumenta la tensión inversa, sin embargo, se alcanza un punto en el que hay un aumento espectacular en la corriente.
En un diodo Zener, uno o ambos mecanismos de ruptura pueden estar presentes. El efecto que sea predominante dependerá del nivel de dopado. Con dopajes bajos, predominará el efecto avalancha con mayores niveles de tensión de ruptura, mientras que, con dopajes bajos, predominará el efecto Zener con una tensión de ruptura más baja. Con un cierto nivel de dopaje y con una tensión de unos 6V para el silicio, ambos efectos estarán presentes con coeficientes de temperatura que se pueden cancelar entre sí. Es por esto que se pueden hacer diodos Zener con coeficientes de temperatura bastante bajos.
Ni el efecto Zener ni el efecto avalancha son efectos destructivos para los dispositivos, sin embargo, el calor generado por las enormes cantidades de corriente que fluyen puede causar daños, por lo cual la corriente que circula debe ser limitada, además de incluir algún método de disipación de calor.
El nivel de corriente exacto generado por estos efectos es difícil de predecir y es por eso que se usan estos dispositivos para generar números aleatorios [**7, 15**].

Capítulo 5

Dispositivos para la captura de números aleatorios

Para la captura de los números aleatorios hemos utilizado 3 placas de desarrollo diferentes que son:

- Arduino Nano.
- Fubarino SD.
- Arduino Mega.

De todos ellos veremos sus características y veremos cual ha sido la mejor opción.

1. Arduino Nano

El Arduino Nano tiene las siguientes especificaciones:

Microcontrolador ATmega328
Voltaje de operación (nivel lógico) 5 V.
Tensión de entrada (recomendada) 7-12 V.
Tensión de entrada (límite) 6-20 V.
Pines de entrada/salida digital 14 (de los cuales 6 proporcionan salida PWM).
Pines de entrada analógica 8.
Corriente continua por cada pin de entrada/salida 40 mA.
Memoria flash 32 KB de los cuales 2 kB son usados por el bootloader.
SRAM 2 kB.
EEPROM 1 kB.
Frecuencia del reloj 16 MHz.

La principal ventaja de este dispositivo es su bajo coste, sus pines de entrada analógica son de 10 bits de resolución, lo cual permite detectar variaciones de tensión de unos 5 mV. Esto será un aspecto clave dado que lo utilizaremos para la captura de voltajes aleatorios que utilizaremos para generar nuestros números [**2**].

2. Fubarino SD

Para poder aumentar la velocidad de lectura y hacer más cómoda la toma de datos, decidimos usar una placa Fubarino SD. Sus principales ventajas están en que permite una mayor velocidad al tener una frecuencia de reloj más alta y en que incorpora una ranura para tarjetas micro SD, lo cual permite almacenar los datos en ella sin que tenga que intervenir el ordenador. El Fubarino SD tiene las siguientes especificaciones:

Microcontrolador PIC32MX795F512H
Voltaje de operación (nivel lógico) 3.3 V.

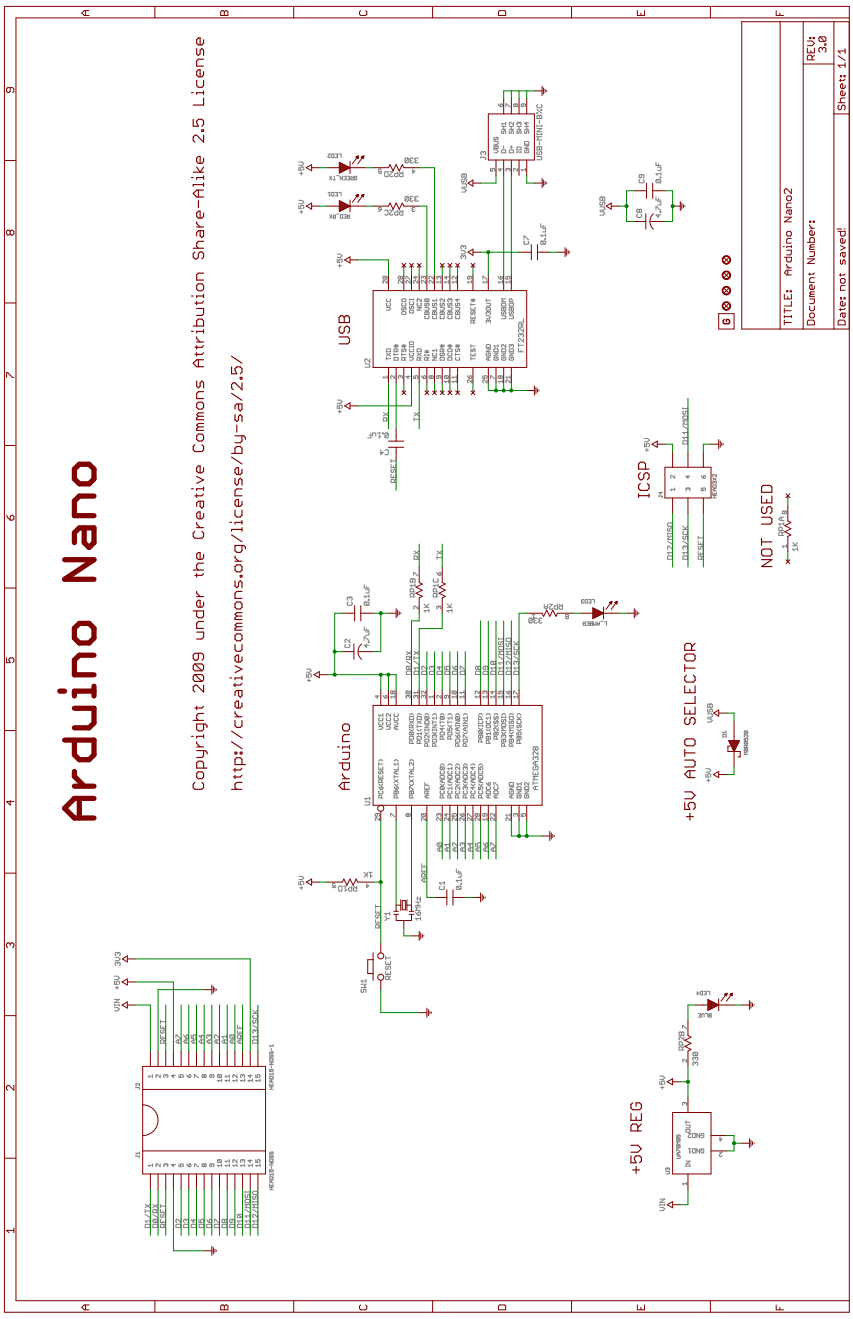


FIGURA 5. Arduino Nano [**2**].

Pines de entrada/salida digital 45 (muchos de los cuales son tolerantes a 5 V).
Pines de entrada analógica 15.
Memoria flash 512 kB.
RAM 128 kB.
Frecuencia del reloj 80 MHz.

Para la captura de los datos, este es el dispositivo más cómodo, la principal desventaja es que la operación de escritura en la tarjeta SD es lenta y al final puede ser mejor enviar los datos directamente al ordenador y almacenarlos en él directamente [**5**].

3. Arduino Mega

El Arduino Mega tiene las siguientes especificaciones:

Microcontrolador ATmega2560
Voltaje de operación (nivel lógico) 5 V.
Tensión de entrada (recomendada) 7-12 V.
Tensión de entrada (límite) 6-20 V.
Pines de entrada/salida digital 54 (de los cuales 15 proporcionan salida PWM).
Pines de entrada analógica 16.
Corriente continua por cada pin de entrada/salida 20 mA.
Corriente continua por cada pin de 3.3 V 50 mA.
Memoria flash 256 KB de los cuales 8 kB son usados por el bootloader.
SRAM 8 kB.
EEPROM 4 kB.
Frecuencia del reloj 16 MHz.

Este dispositivo es adecuado cuando necesitemos un elevado nivel de corriente. En el diseño de un generador utilizando diodos Zener, la corriente generada es impredecible y puede llegar a ser muy elevada. Si tenemos la corriente limitada por la que pueda suministrar la placa, podríamos no obtener la aleatoriedad que buscamos.
Este extra de corriente se consigue con la alimentación a través de una fuente de alimentación conectada a un enchufe. Con ella, podremos conseguir hasta 1 A [**1**].

4. Conclusiones sobre los microcontroladores usados

Como hemos podido ver, no tenemos un método definitivo para la captura de los datos, si bien, el Fubarino será el más adecuado siempre que estemos seguros de que va a ser capaz de proporcionar la corriente necesaria pues tiene un reloj más rápido que los otros dos y ejecutará las intrucciones más rápido. Este aspecto lo consideraremos clave pues la lectura a través de un puerto analógico es lenta (unos 110 μs) y tendremos que hacer un montón de lecturas para cada generador (del orden de 10^8). Dentro del generador diseñado, buscaremos la forma de utilizar la lectura digital ya que esta es mucho más rapida (unos 10 μs), pero para ello necesitaremos adaptar la señal a un formato binario.

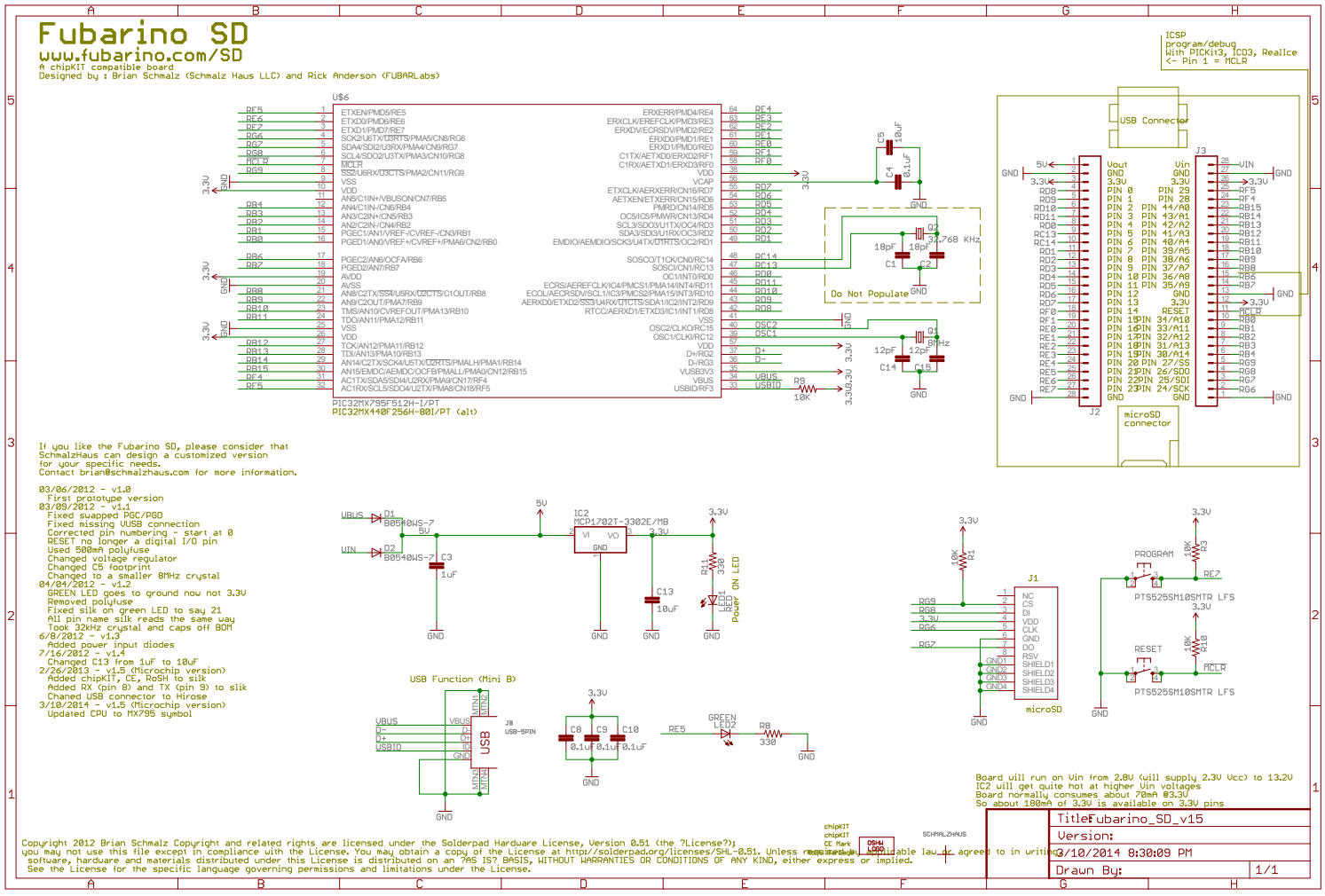


FIGURA 6. Fubarino SD [5].

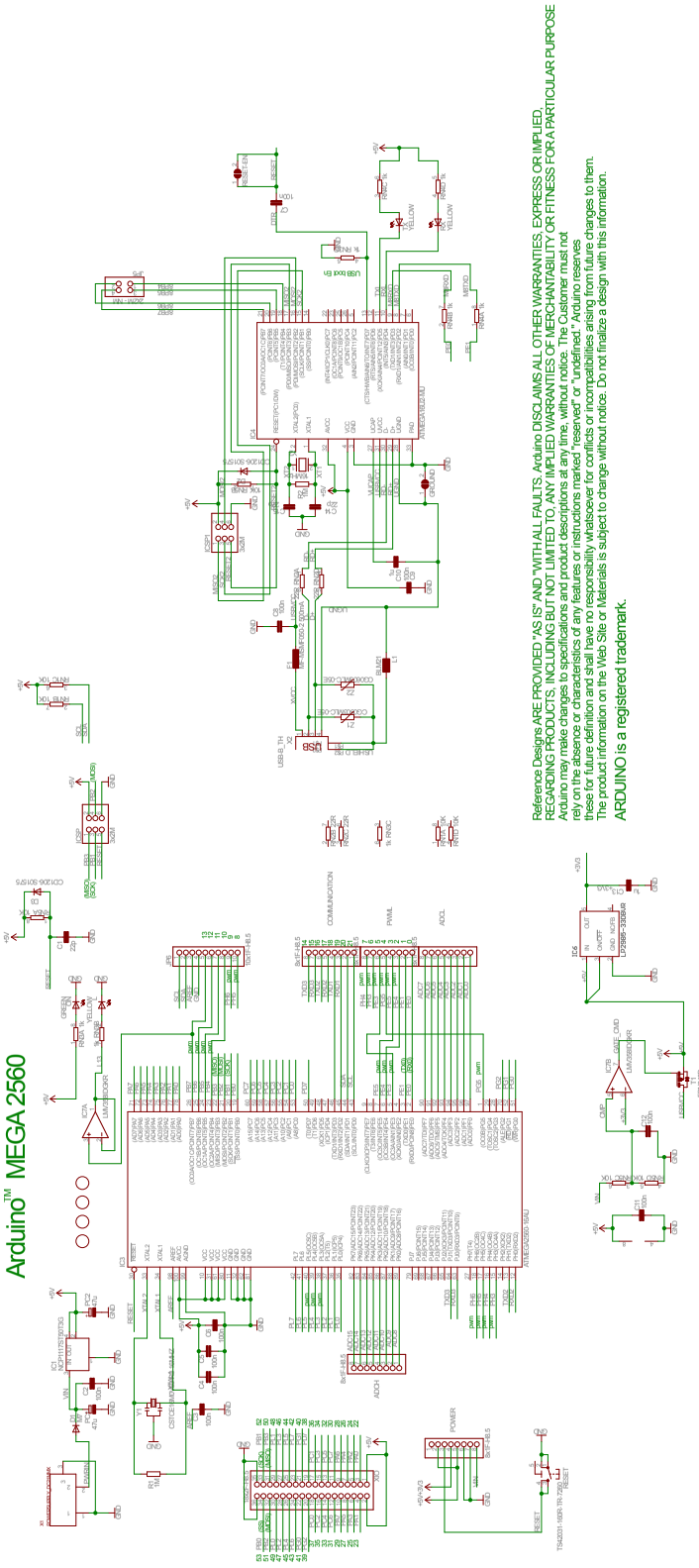


FIGURA 7. Arduino Mega [1].

5. Comunicación con el PC

El método utilizado para programar los microcontroladores y para ver los resultados obtenidos visualmente ha sido el programa Arduino IDE descargado de la página web `https://www.arduino.cc/en/Main/Software`. Este nos va a permitir programar en un lenguaje similar a C nuestros microcontroladores y, así mismo, ver y dibujar los datos recibidos por el puerto serie. Esto último es un aspecto importante dado que, a la hora de diseñar nuestro generador, podremos comprobar a simple vista si la variación de la tensión de nuestro generador es suficientemente elevada. Para el caso del Fubarino SD se ha tenido que instalar el paquete ChipKit que contiene todos los archivos necesarios (código fuente de la API, compilador, definiciones de las placas y las herramientas de programación necesarias para subir el programa al microcontrolador) para la programación con este tipo de placas que antes solo se podían programar con MPIDE. La forma de instalar este paquete en nuestro Arduino IDE viene descrito en la página `http://chipkit.net/wiki/index.php?title=ChipKIT_core`.

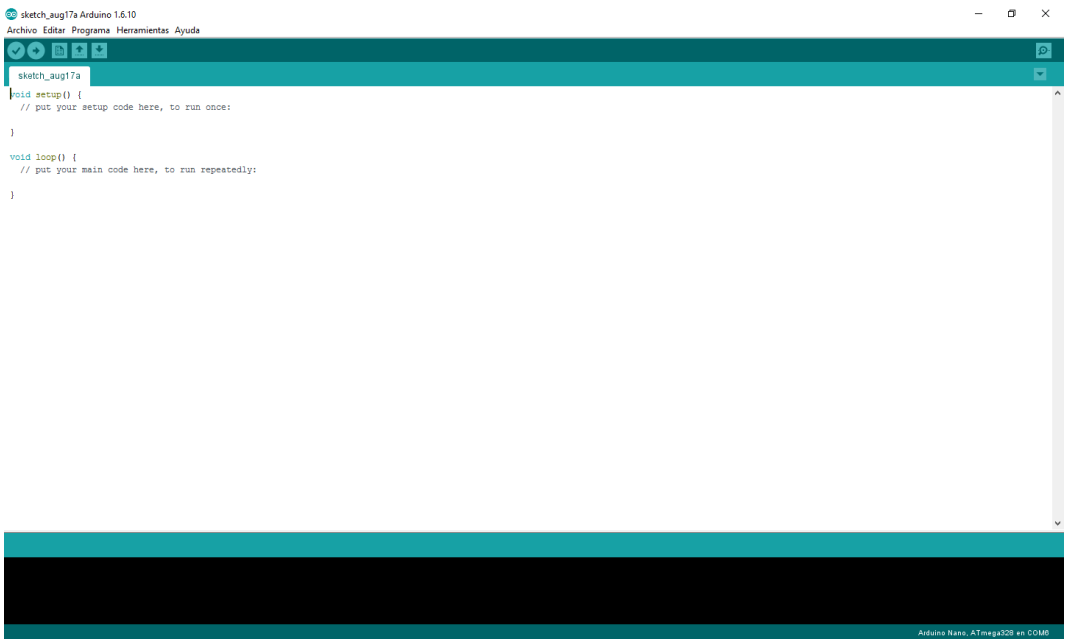


FIGURA 8. Imagen de Arduino IDE.

Otro método utilizado para la captura y el análisis de los datos que se transmiten al ordenador será el programa Matlab. Con el vamos a poder guardar las muestras en un vector y hacer cálculos con ellas tales como la media o un histograma para comprobar visualmente en que rangos tenemos los valores de tensión capturados.

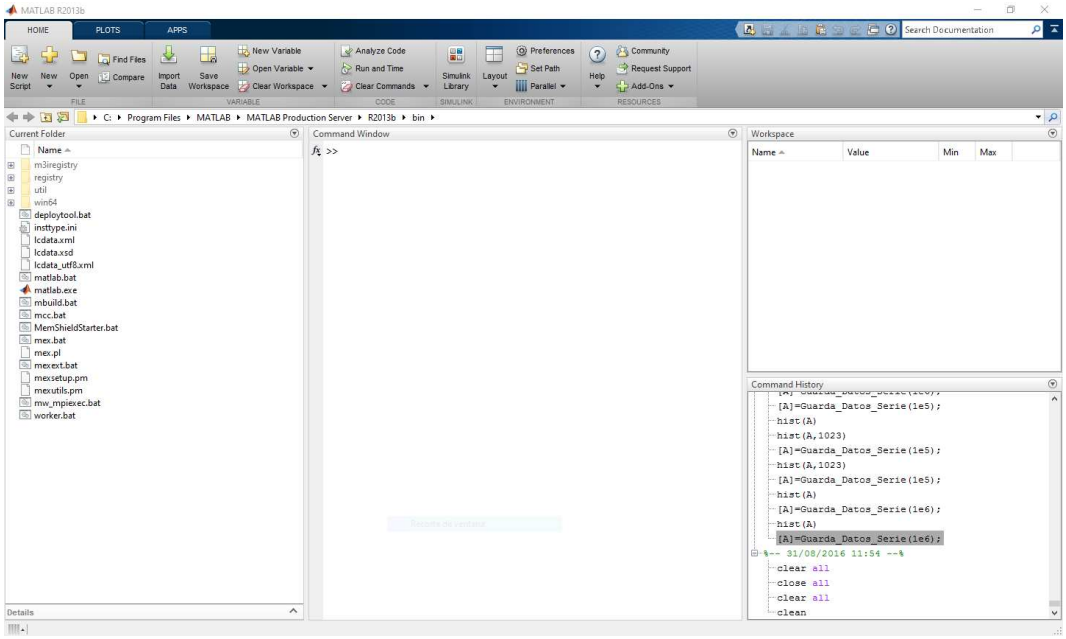


FIGURA 9. Imagen de Matlab 2013b.

El último método que vamos a utilizar es el terminal de Linux Ubuntu 16.04 LTS. En el utilizaremos el comando cat, que permite concatenar archivos e imprimir por pantalla el contenido de los mismo. Con esto y con el comando redirección podremos guardar los datos transmitidos al ordenador en un archivo.
Generalmente usaremos el siguiente comando: *cat /dev/ttyUSB0 >> nombreArchivo.txt* donde */dev/ttyUSB0* es el puerto USB correspondiente y *nombreArchivo.txt* es el nombre que queramos dar al archivo donde será guardado, el comando > está dos veces para que no reemplace el contenido del archivo por el que ya había sino que vaya escribiendo a continuación. Para saber el puerto USB en el que está conectado el dispositivo (puede cambiar) podemos comprobarlo en Arduino IDE en el menú herramientas y, a continuación, en puerto.

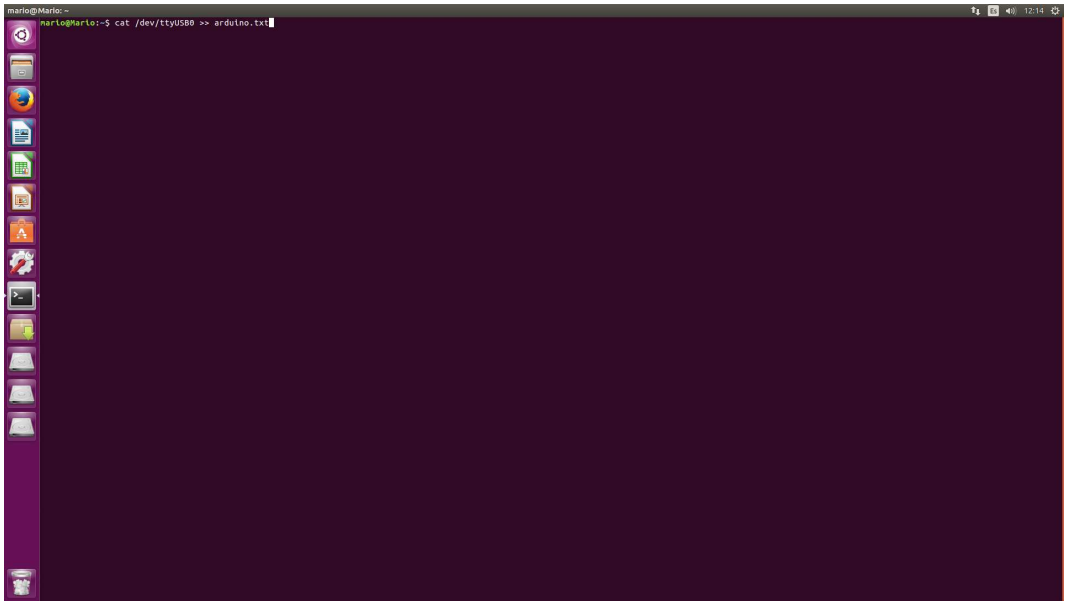


FIGURA 10. Imagen del terminal de Ubuntu.

Capítulo 6

Método para el análisis de los datos tomados

Dieharder es una batería de pruebas para generadores de números aleatorios. El objetivo general de los tests es comprobar que las secuencias generadas cumplen con las propiedades que se esperan de una secuencia aleatoria.
Otro objetivo del programa es proporcionar alguna indicación de por qué un generador no pasa la prueba cuando dicha información se pueda extraer durante el proceso de prueba.
La mayoría de las pruebas en Dieharder devuelven un valor de p, que debe ser uniforme en [0,1) si el archivo de entrada contiene bits aleatorios verdaderamente independientes. Esos valores de p se obtienen como p = F (X), donde F es la función de distribución de la muestra de la variable aleatoria X, a menudo normalizada. Pero eso supone que F es sólo una aproximación asintótica, para lo cual el ajuste será peor en las colas. Por lo tanto uno no debe guiarse mucho con los valores ocasionales de p que se encuentren cerca de 0 o de 1.

1. Diferentes test dentro de Dieharder

A continuación vamos a describir la mayor parte de los test que realiza Dieharder:

1.1. Batería de test de aleatoriedad de Marsaglia. La descripción de estos test es la obtenida en [**8**].

La prueba de separación de cumpleaños: Elije m cumpleaños en un año de n días y enumera las separaciones entre los cumpleaños. Si j es el número de valores que ocurren más de una vez en la lista, entonces j es asintóticamente una distribución Poisson con media $m^{\frac{3}{4n}}$.
La experiencia muestra que n debe ser bastante grande, digamos $n \geq 2^{18}$ para poder comparar los resultados de la distribución de Poisson con la media. Esta prueba utiliza $n = 2^{24}$ y $M = 2^9$, por lo que la distribución subyacente de j se toma como Poisson con $\alpha = \frac{2^{27}}{2^{26}} = 2$. Se toma una muestra de 500 j, y un χ-cuadrado [1] de prueba de ajuste para ofrecer un valor de p. La primera prueba utiliza los bits 1-24 (contando desde la izquierda) de los números enteros en el archivo especificado. A continuación, el archivo se cierra y se vuelve a abrir. A continuación, los bits 2-25 se utilizan para proporcionar los cumpleaños, a continuación, 3-26 y así sucesivamente para los bits 9-32.
Cada conjunto de bits proporciona un valor de p, y los 9 valores de p proporcionan una muestra para el test.

[1]Una prueba de χ-cuadrado es una prueba de hipótesis que compara la distribución observada de los datos con una distribución esperada de los datos.

La prueba de 5 permutaciones que se superponen: Busca una secuencia de un millón de números enteros aleatorios de 32 bits. Cada conjunto de cinco enteros consecutivos puede estar en una de las 120 colocaciones diferentes para las 5! posibles ordenaciones de cinco números. Por lo tanto, los números quinto, sexto y séptimo para cada uno proporcionan un estado. A medida que se observan muchos miles de transiciones de estado, los recuentos acumulativos se hacen del número de ocurrencias de cada estado. A continuación, la forma cuadrática en la matriz de covarianza de 120 x 120 proporciona una prueba equivalente a la prueba de razón de probabilidad de que los 120 recuentos de células proceden de la distribución normal (asintótica) especificada con la matriz especificada de 120 x 120 de covarianza (con rango 99). Esta versión utiliza 1.000.000 de enteros, dos veces.

La prueba de rango binario de matrices de 31 x 31: Coge los 31 bits más a la izquierda de 31 números enteros aleatorios a partir de la secuencia de prueba y se utiliza para formar una matriz binaria de 31 x 31 sobre el rango 0,1. El rango se determina. Ese rango puede ser de 0 a 31, pero los rangos <28 son raros, y sus recuentos se ponen en común con los de rango 28. Los rangos se buscan para 40.000 matrices aleatorias y se hace una prueba de χ-cuadrado en el recuento de las filas 31, 30 , 29 y $\leq$ 28.

La prueba de rango binario de matrices de 32 x 32: Se forma una matriz binaria aleatoria de 32 x 32, cada fila será un número entero aleatorio de 32 bits. Se determina el rango. Ese rango puede ser de 0 a 32, filas de menos de 29 son raras, y sus recuentos se combinaron con los de rango 29. Los rangos se encontraron para 40.000 matrices aleatorias y una prueba de χ-cuadrado se lleva a cabo en el recuento de las filas 32, 31, 30 y $\leq$ 29.

La prueba de rango binario de matrices de 6 x 8: A partir de cada uno de seis enteros de 32 bits aleatorios del generador bajo prueba, se elige un byte especificado, y los seis bytes resultantes forman una matriz binaria de 6 x 8 cuyo rango está determinado. Ese rango puede ser de 0 a 6, pero los que tengan rango 0, 1, 2 ó 3 son raros; sus recuentos se combinaron con los de rango 4. Los rangos son encontrados para 100.000 matrices aleatorias, y se realiza una prueba de χ-cuadrado en el recuento de filas 6, 5 y $\leq$ 4.

La prueba de secuencia de bits: El archivo bajo prueba es visto como una secuencia de bits a los que llamaremos b1, b2, ... Se considera un alfabeto con dos "letras", 0 y 1, y genera una secuencia de bits como una sucesión de palabras de 20 letras superpuestas. Así, la primera palabra es b1b2 ... b20, la segunda será b2b3 ... b21, y así sucesivamente. La prueba de secuencia de bits cuenta el número de palabras de 20 letras (20 bits) de una serie de superposición 2^{21} palabras de 20 letras. Hay 2^{20} posibles palabras de 20 letras. Para una secuencia verdaderamente aleatoria de 2^{21} + 19 bits, el número de palabras que faltan j debe estar (muy cerca) distribuido normalmente con una media de 141909 y sigma 428. Por lo tanto (j - 141909) / 428 debería ser una variable

aleatoria normal estándar que conduce a un valor p uniforme [0,1). El ensayo se repite veinte veces.

Las pruebas OPSO, OQSO y ADN: OPSO significa superposición de pares de escasa densidad de ocupación. La prueba OPSO considera palabras de 2 letras de un alfabeto de 1024 letras. Cada letra está determinada por 10 bits de un entero de 32 bits en la secuencia a ensayar. OPSO genera 221 palabras (por superposición) de 2 letras y cuenta el número de palabras que faltan, es decir, las palabras de 2 letras que no aparecen en la secuencia completa. Ese conteo debe estar muy cerca de una distribución normal con media 141909 y sigma 290. Por lo tanto $\frac{palabrasperdidas-141909}{290}$ debe ser una variable normal estándar.

La prueba OPSO toma 32 bits a la vez desde el archivo de prueba y utiliza un conjunto designado de diez bits consecutivos. A continuación, se reinicia el archivo para los siguientes 10 bits designados, y así sucesivamente.

OQSO significa superposición cuádruple de escasa ocupación. La prueba OQSO es similar, excepto que considera las palabras de 4 letras de un alfabeto de 32 letras, cada letra determinada por una cadena designada de 5 bits consecutivos a partir del archivo de prueba, que se asume que contendrá elementos enteros aleatorios de 32 bits. El número medio de palabras que faltan en una secuencia de 2^{21} palabras de cuatro letras, es de nuevo 141909, con sigma = 295. La media se basa en cálculos teóricos, sigma viene de una extensa simulación.

La prueba de ADN considera un alfabeto de 4 letras C, G, A, T, determinado por dos bits designados en la secuencia de números enteros aleatorios que se está probando. Se considera palabras de 10 letras, de modo que, como en OPSO y OQSO, hay 2^{20} palabras posibles y, el número medio de palabras que faltan de una cadena de 2^{21} palabras de 10 letras es 141909. La desviación estándar sigma = 339 se determinó como para OQSO por simulación.

La prueba del recuento de 1s en una cadena de bytes: Dado el archivo que se quiere probar y considerándolo como un flujo de bytes (cuatro por cada entero de 32 bits). Cada byte puede contener de 0 a ocho 1s, con probabilidades $\frac{1}{256}$, $\frac{8}{256}$, $\frac{28}{256}$, $\frac{56}{256}$, $\frac{70}{256}$, $\frac{56}{256}$, $\frac{28}{256}$, $\frac{8}{256}$ y $\frac{1}{256}$ para 0, 1, 2, 3, 4, 5, 6, 7 y 8 1s respectivamente. Ahora deja que el flujo de bytes proporcione una serie de superposición de palabras de 5 letras, cada "letra"tomando valores A, B, C, D, E. Las letras se determinan por el número de 1s en un byte 0, 1, o 2 corresponde a una A, 3 produce una B, 4 produce una C, 5 produce una D y las secuencias de 6, 7 u 8 producen una E. Por lo tanto tenemos un generador de cinco letras con diferentes probabilidades ($\frac{37}{256}$, $\frac{56}{256}$, $\frac{70}{256}$, $\frac{56}{256}$ y $\frac{37}{256}$. Hay 5^5 posibles palabras de 5 letras de una serie de 256.000 de palabras de 5 letras, los recuentos se realizan sobre las frecuencias de cada palabra. La forma cuadrática de la matriz inversa de covarianza proporciona una prueba para los recuentos de las células de χ-cuadrado Q4 Q5, la diferencia de

las sumas de Pearson $\frac{(OBS-EXP)}{EXP}$ se usan en el recuento para contar las células de 5 y 4 letras.

La prueba del recuento 1s para bytes específicos: Se considera el archivo que se está probando como una corriente de enteros de 32 bits. De cada número entero, se elige un byte específico, por ejemplo los bits de más a la izquierda de 1 a 8. Cada byte pueden contener de 0 a 8 1s, con probabilidades $\frac{1}{256}$, $\frac{8}{256}$, $\frac{28}{256}$, $\frac{56}{256}$, $\frac{70}{256}$, $\frac{56}{256}$, $\frac{28}{256}$, $\frac{8}{256}$ y $\frac{1}{256}$ para 0, 1, 2, 3, 4, 5, 6, 7 y 8 1s respectivamente. Ahora se deja que los bytes especificados de los números enteros sucesivos proporcionen una cadena de palabras (por superposición) de palabras de 5 letras, cada una tiene valores de A, B, C, D ó E. las letras son determinados por el número de 1s, en un byte 0, 1, o 2 corresponde a una A, 3 produce una B, 4 produce una C, 5 produce una D y las secuencias de 6, 7 u 8 producen una E. Por lo tanto tenemos un generador de cinco letras con diferentes probabilidades ($\frac{37}{256}$, $\frac{56}{256}$, $\frac{70}{256}$, $\frac{56}{256}$ y $\frac{37}{256}$). Hay 5^5 posibles palabras de 5 letras de una cadena de 256.000 palabras de 5 letras, los recuentos se realizan sobre las frecuencias para cada palabra. La forma cuadrática de la matriz inversa de covarianza proporciona una prueba para los recuentos de las células de χ-cuadrado Q4 Q5, la diferencia de las sumas de Pearson $\frac{(OBS-EXP)}{EXP}$ se usan en el recuento para contar las células de 5 y 4 letras.

La prueba de estacionamiento: En un cuadrado de lado 100, al azar colocamos un círculo de radio 1. A continuación, tratamos de colocar un segundo, un tercero, y así sucesivamente. Es decir, si un intento para colocar un círculo provoca un choque con otro ya colocado, se intenta de nuevo en una nueva ubicación al azar. Cada intento conduce a ya sea un choque con otro círculo o a un éxito, este último seguido por un incremento a la lista de círculos ya colocados.

Si dibujamos n: número de intentos, en comparación con el número k de círculos colocados en el cuadrado correctamente, se obtiene una curva que debería ser similar a la proporcionada por un generador de números aleatorios perfecto. La teoría de comportamiento de una curva aleatoria de este tipo parece fuera de nuestro alcance, y como muestra gráficos no están disponibles para esta serie de pruebas, se utiliza un simple caracterización del experimento aleatorio: k, el número de círculos colocados con éxito después de n = 12.000 intentos. La simulación muestra que k debe valer, en promedio, 3523 con sigma 21,9 y está muy cerca de una distribución normal. Por lo tanto, $\frac{k-3523}{21,9}$ debe ser una variable estándar normal, que, convertida en una variable uniforme, proporciona entrada al test basado en una muestra de 10.

La prueba de mínima distancia: Se eligen 100 veces n=8000 puntos al azar en un cuadrado de lado 10000. Se busca d, que es la mínima distancia entre $\frac{n^2-n}{2}$ pares de puntos. Si los puntos son verdaderamente uniformes e independientes, a continuación, d^2, el cuadrado de la distancia debe estar exponencialmente distribuida con una media de 0.995. Así $1 - e^{\frac{-d^2}{0,995}}$ debe ser uniforme en [0,1) y se aplica el test a los 100 valores

resultantes, que sirve como una prueba de la homogeneidad en puntos al azar del cuadrado.

La prueba de las esferas 3D: Se elijen 4000 puntos al azar en un cubo de arista 1000. En cada punto, centrar una gran esfera suficiente para alcanzar el siguiente punto más cercano. A continuación, el volumen de las esferas más pequeños, están próximas a una distribución exponencial con una media de $\frac{120\pi}{3}$. Así, el cubo del radio es exponencial con una media de 30 (la media se obtiene por simulación).

La prueba de esferas 3D genera 4000 esferas 20 veces. Cada cubo del radio m conduce a una variable uniforme por medio de $1-e^{\frac{-r^3}{30}}$, y luego el test se realiza sobre los 20 valores de p.

La prueba de compresión Los enteros aleatorios se transforman a valores de punto flotante para obtener valores uniformes sobre [0,1). A partir de $k = 2^{31} = 2147483647$, la prueba encuentra j, el número de iteraciones necesarias para reducir k a 1, utilizando k = valor máximo de reducción (kU), con U proporcionada por números enteros desde el archivo flotante que se está probando. Tales js se encuentran 100.000 veces, y luego se cuentan para el número de veces de $j \leq 6, 7, ..., 47, \geq 48$ y se utilizan para proporcionar una prueba de χ-cuadrado.

La prueba de superposiciones de sumas: Los enteros aleatorios se transforman a valores de punto flotante para obtener una secuencia de variables uniformes U(1), U(2), ... del intervalo [0,1). A continuación, se forma, la superposición de sumas, S (1) = U (1) + ... + U (100), S (2) = U (2) + ... + U (101), ...

Los valores de S son prácticamente normales, con una cierta matriz de covarianza. Una transformación lineal de los valores de S los convierten en una secuencia estándar e independiente que es convertida en una secuencia de variables uniformes para los test.

La prueba de los dados: Ejecuta 200.000 partidas de dados, se encuentra el número de victorias y ejecuta el número necesario de tiradas para poner fin a cada partida. El número de victorias debe estar muy cerca de una normal con media $200000p$ y varianza $200000p(1-p)$, con $p = \frac{244}{495}$. El número de lanzamientos necesario para completar cada partida puede variar de 1 a ∞, pero para valores mayores que 21, se fijan a 21.

Se realiza una prueba de χ-cuadrado con el recuento del número de lanzamientos. Cada número entero de 32 bits cogido del archivo a evaluar proporciona el valor para el tiro de un dado normalizando a un intervalo [0,1) y, posteriormente, multiplicándolo por 6 y sumando 1 a la parte entera del resultado.

1.2. Conjunto de pruebas estadísticas del NIST (Instituto Nacional de Estándares y Tecnología).

Se trata de otra batería de pruebas incluída en Dieharder. La descripción de estos test es la obtenida en [**13**].

Prueba de frecuencia monobit: El objetivo de la prueba es determinar la proporción de ceros y unos. El propósito de esta prueba es determinar si el número de unos y ceros en una secuencia son aproximadamente la misma que se esperaría para una secuencia verdaderamente aleatoria.

La prueba evalúa si el número de unos y ceros en una secuencia es, aproximadamente, la misma. Todas las pruebas posteriores dependen de que se supere esta prueba.

Prueba de frecuencia de bloques internos: El objetivo de la prueba es determinar la proporción de los bloques internos de M-bits. El propósito de este ensayo es determinar si la frecuencia de los en un bloque de M bits es aproximadamente $\frac{M}{2}$, como sería de esperar bajo un supuesto de aleatoriedad. Para el caso de M=1, tendríamos el caso de la prueba monobit.

Prueba de carreras: El objetivo de este ensayo es determinar el número total de carreras en la secuencia donde una carrera es una secuencia ininterrumpida de bits idénticos. Una serie de longitud k consta de bits k exactamente idénticos y está delimitada antes y después con un valor opuesto. El propósito de la prueba de carreras es determinar si el número de carreras de unos y ceros de diferentes longitudes es tal y como se espera para una secuencia aleatoria. En particular, esta prueba determina si la oscilación entre estos ceros y unos es demasiado rápida o demasiado lenta.

Prueba para la carrera más larga de unos en un bloque: El objetivo de la prueba es buscar la carrera más larga de unos dentro de bloques de M bits. El propósito de este ensayo es determinar si la longitud de la carrera más larga dentro de la secuencia probada es consistente con la longitud de la carrera más larga de las que se esperaría en una secuencia aleatoria. Hay que tener en cuenta que una irregularidad en la duración prevista de la carrera más larga de unos, implica que también hay una irregularidad en la duración prevista de la carrera más larga de ceros.

Prueba de la transformada discreta de Fourier: El objetivo de esta prueba es determinar la altura de los picos de la Transformada Discreta de Fourier de la secuencia. El propósito de esta prueba es detectar las características periódicas (es decir, patrones repetitivos que están cerca unos de otros) en la secuencia de prueba que indicaría una desviación de la hipótesis de aleatoriedad. La intención es detectar si el número de picos que exceden el umbral del 95 % es significativamente diferente del 5 %.

Prueba de no solapamiento de plantillas: El propósito de esta prueba es detectar si el generador produce demasiadas ocurrencias de un patrón dado no periódico. Para esta prueba se utiliza una ventana de m bits para buscar un patrón específico de m bits. Si no se encuentra un patrón, la ventana se desliza una posición de bit. Si se encuentra un patrón, la ventana es reseteada al bit siguiente al patrón encontrado.

Prueba de solapamiento de plantillas La diferencia entre esta prueba y la prueba anterior, es que cuando se encuentra el patrón, la ventana se desliza sólo un bit.

Prueba estadística universal de Maurer: El objetivo de este ensayo es determinar el número de bits entre patrones de características determinadas. El propósito de la prueba es detectar si la secuencia se puede

comprimir de manera significativa o no sin pérdida de información. Una secuencia significativamente comprimible se considera que es no aleatoria.

Prueba de complejidad lineal: El objetivo de esta prueba es determinar la longitud de un registro de desplazamiento de realimentación lineal (LFSR). El propósito de este ensayo es determinar si la secuencia es lo suficientemente compleja como para ser considerada al azar. Las secuencias aleatorias se caracterizan por LFSR largos. Un LFSR que es demasiado corto implica no aleatoriedad.

Prueba en serie: El objetivo de esta prueba es determinar la frecuencia de todos los posibles patrones de bloques de m bits a través de toda la secuencia. El propósito de este ensayo es determinar si el número de ocurrencias de los distintos 2^m patrones superpuestos de m bits es aproximadamente el mismo que el que se esperaría para una secuencia aleatoria. Las secuencias aleatorias tienen la uniformidad; es decir, cada patrón de m bits tiene la misma probabilidad de aparecer que cualquier otro patrón de m bits. Tener en cuenta que para m = 1, la prueba de serie es equivalente a la prueba de frecuencia monobit.

Prueba de entropía aproximada: El objetivo de esta prueba es determinar la frecuencia de todos los posibles patrones de la superposición de m bits a través de toda la secuencia. El propósito de la prueba es comparar la frecuencia de los bloques de dos longitudes consecutivas/adyacentes (M y $M+1$) contra el resultado esperado para la superposición de una secuencia aleatoria.

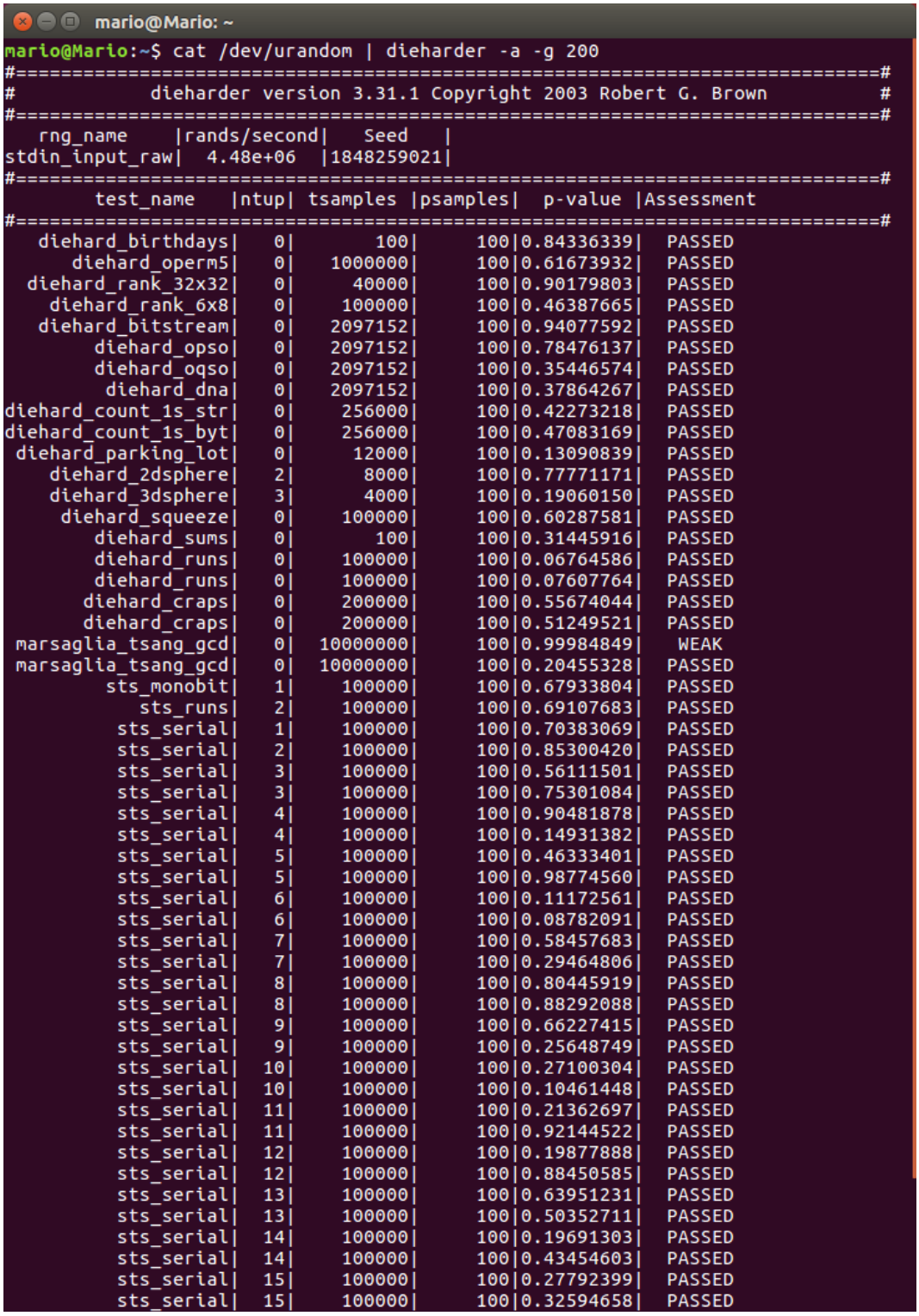
2. Test Dieharder con los generadores de Linux y Arduino

Para probar los test realizados por Dieharder y, la efectividad de los generadores que tienen Arduino y Linux, haremos las pruebas con dichos generadores. Para ello, tendremos que generar un archivo con números aleatorios lo suficientemente grande como para que pueda hacer los test correctamente (cuanto más grande sea el archivo generado, más exactos serán los resultados). Hay que destacar que Dieharder consume una enorme cantidad de números aleatorios para realizar sus test.
Tenemos dos formas de enviar los datos a Dieharder:

1. A través de un archivo guardado en el ordenador.
2. Directamente al programa según se van generando los números.

Cuando generemos los números con el generador de Linux (el generador se llama urandom), enviaremos los datos según se vayan generando porque es un generador muy rápido y genera datos suficiente en poco tiempo. Sin embargo, cuando generemos datos con la placa (tanto ahora como para los generadores que diseñemos) crearemos un archivo donde los guardaremos. Hay que destacar que, como no podemos generar archivos tan grandes como para poder realizar los test *de una pasada* el programa tendrá que recorrer varias veces el archivo para poder realizar sus test. Esto puede afectar negativamente al generador para pasar los test pero no hay otra posibilidad dado que algunos test requieren varios GB de datos.

Sin más demora vamos a comprobar los resultados (el código para generar números aleatorios con Arduino se encuentra en el anexo):



```
mario@Mario: ~
mario@Mario:~$ cat /dev/urandom | dieharder -a -g 200
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |rands/second|   Seed   |
stdin_input_raw|  4.48e+06  |1848259021|
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
   diehard_birthdays|   0|       100|     100|0.84336339|  PASSED
      diehard_operm5|   0|   1000000|     100|0.61673932|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.90179803|  PASSED
    diehard_rank_6x8|   0|    100000|     100|0.46387665|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.94077592|  PASSED
        diehard_opso|   0|   2097152|     100|0.78476137|  PASSED
        diehard_oqso|   0|   2097152|     100|0.35446574|  PASSED
         diehard_dna|   0|   2097152|     100|0.37864267|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.42273218|  PASSED
diehard_count_1s_byt|   0|    256000|     100|0.47083169|  PASSED
 diehard_parking_lot|   0|     12000|     100|0.13090839|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.77771171|  PASSED
    diehard_3dsphere|   3|      4000|     100|0.19060150|  PASSED
     diehard_squeeze|   0|    100000|     100|0.60287581|  PASSED
        diehard_sums|   0|       100|     100|0.31445916|  PASSED
        diehard_runs|   0|    100000|     100|0.06764586|  PASSED
        diehard_runs|   0|    100000|     100|0.07607764|  PASSED
       diehard_craps|   0|    200000|     100|0.55674044|  PASSED
       diehard_craps|   0|    200000|     100|0.51249521|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.99984849|   WEAK
 marsaglia_tsang_gcd|   0|  10000000|     100|0.20455328|  PASSED
         sts_monobit|   1|    100000|     100|0.67933804|  PASSED
            sts_runs|   2|    100000|     100|0.69107683|  PASSED
          sts_serial|   1|    100000|     100|0.70383069|  PASSED
          sts_serial|   2|    100000|     100|0.85300420|  PASSED
          sts_serial|   3|    100000|     100|0.56111501|  PASSED
          sts_serial|   3|    100000|     100|0.75301084|  PASSED
          sts_serial|   4|    100000|     100|0.90481878|  PASSED
          sts_serial|   4|    100000|     100|0.14931382|  PASSED
          sts_serial|   5|    100000|     100|0.46333401|  PASSED
          sts_serial|   5|    100000|     100|0.98774560|  PASSED
          sts_serial|   6|    100000|     100|0.11172561|  PASSED
          sts_serial|   6|    100000|     100|0.08782091|  PASSED
          sts_serial|   7|    100000|     100|0.58457683|  PASSED
          sts_serial|   7|    100000|     100|0.29464806|  PASSED
          sts_serial|   8|    100000|     100|0.80445919|  PASSED
          sts_serial|   8|    100000|     100|0.88292088|  PASSED
          sts_serial|   9|    100000|     100|0.66227415|  PASSED
          sts_serial|   9|    100000|     100|0.25648749|  PASSED
          sts_serial|  10|    100000|     100|0.27100304|  PASSED
          sts_serial|  10|    100000|     100|0.10461448|  PASSED
          sts_serial|  11|    100000|     100|0.21362697|  PASSED
          sts_serial|  11|    100000|     100|0.92144522|  PASSED
          sts_serial|  12|    100000|     100|0.19877888|  PASSED
          sts_serial|  12|    100000|     100|0.88450585|  PASSED
          sts_serial|  13|    100000|     100|0.63951231|  PASSED
          sts_serial|  13|    100000|     100|0.50352711|  PASSED
          sts_serial|  14|    100000|     100|0.19691303|  PASSED
          sts_serial|  14|    100000|     100|0.43454603|  PASSED
          sts_serial|  15|    100000|     100|0.27792399|  PASSED
          sts_serial|  15|    100000|     100|0.32594658|  PASSED
```

FIGURA 11. Resultados del generador de Linux (urandom).

Como hemos podido ver, los resultados nos demuestran que es un generador muy bueno capaz de pasar todos los test sin problema a pesar de ser pseudo-aleatorio. También cabe destacar que este generador es mucho más rápido de lo que pudiera ser nuestro generador creado dado que es capaz de producir 1GB de números aleatorios en unos pocos segundos.

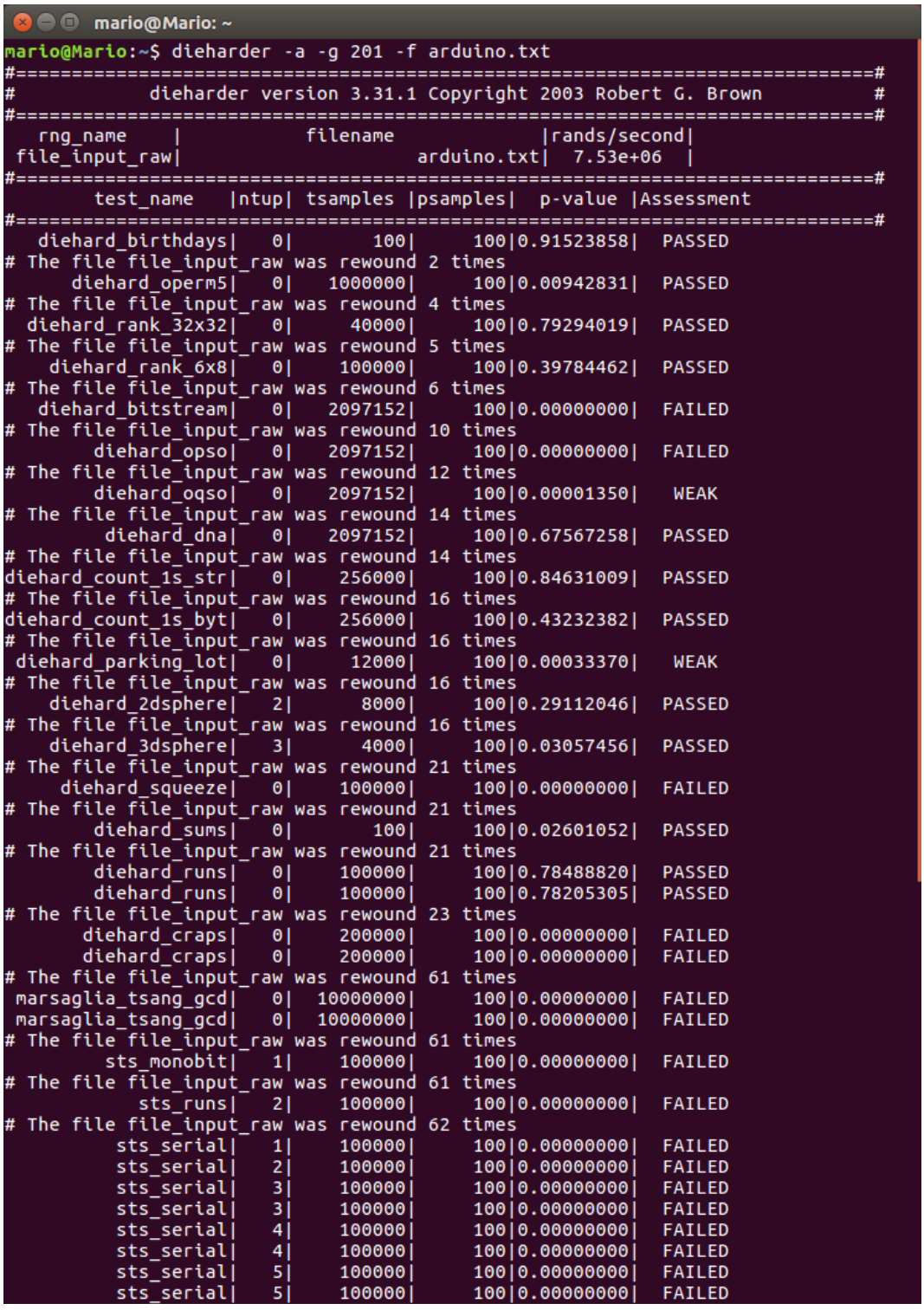


FIGURA 12. Resultados del generador de Arduino (función random).

Como se puede apreciar, el generador de Arduino es un poco peor pues, hay pruebas que no ha podido pasar. Dicho generador es pseudo-aleatorio y utiliza como semilla una lectura tomada del puerto analógico. Cabe destacar que el

archivo generado es de unos 200 MB y, por tanto hay muchos test que han tenido que pasar el archivo varias veces (viene indicado en la imagen cuántas veces ha tenido que recorrer el archivo) por lo que no podemos comparar directamente los resultados de uno con los de otro, podemos ver que los test que no han recorrido muchas veces el archivo han dado la secuencia como buena. Como última apreciación vemos que a la hora de elegir uno u otro nos decantaremos por el de Linux por la velocidad ya que es una velocidad inalcanzable por Arduino por mucho que transmitamos a la máxima velocidad permitida por el puerto serie.

Capítulo 7

Diseño del generador usando resistencias

La primera prueba que hacemos es un divisor de tensión básico que esté formado por las resistencias de mayor valor que tenemos (recordemos que, a mayor valor de la resistencia, mayor será el valor del ruido térmico), en este caso son de $1M\Omega$. Con esto, el circuito quedará como se muestra en la figura.

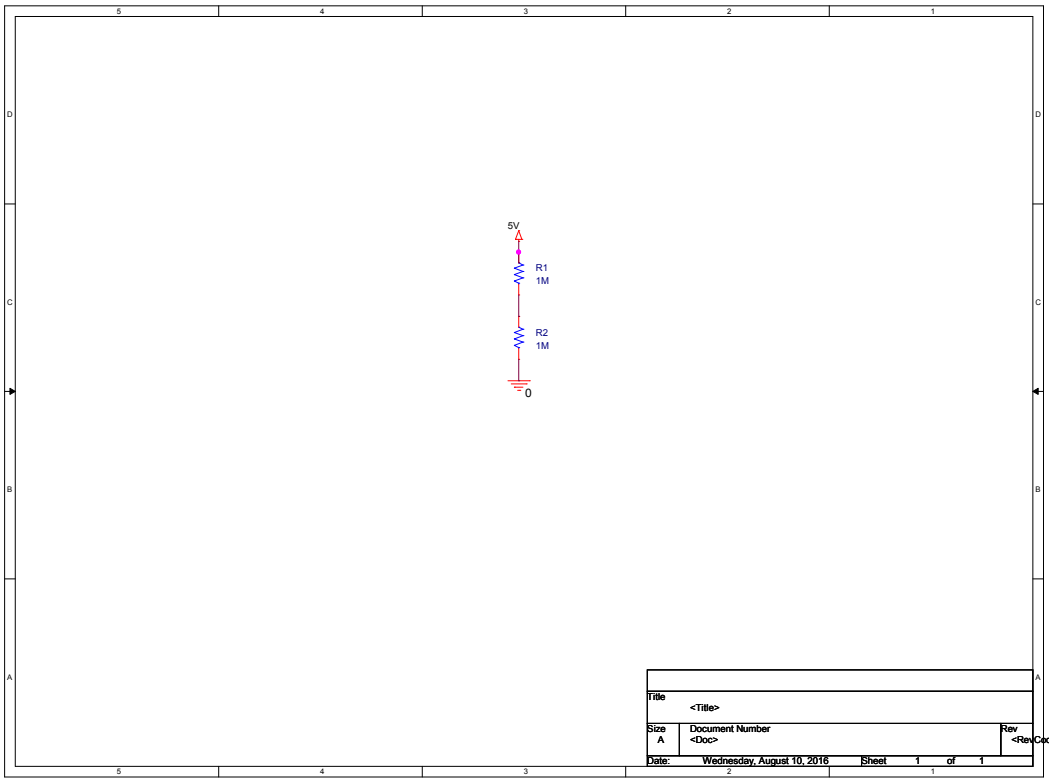


FIGURA 13. Divisor de tensión formado por dos resitencias de $1M\Omega$.

Como podemos ver, el valor de tensión medido entre las dos resistencias debería ser de 2.5 V, en este caso utilizaremos la lectura analógica de Arduino (archivo *LecturaAnalogicaPin0* de los códigos del proyecto) por lo que el valor medido debería ser de unos 512 (tener en cuenta las tolerancias) y al ejecutar la función, vemos lo siguiente en el Serial Plotter del IDE de Arduino.

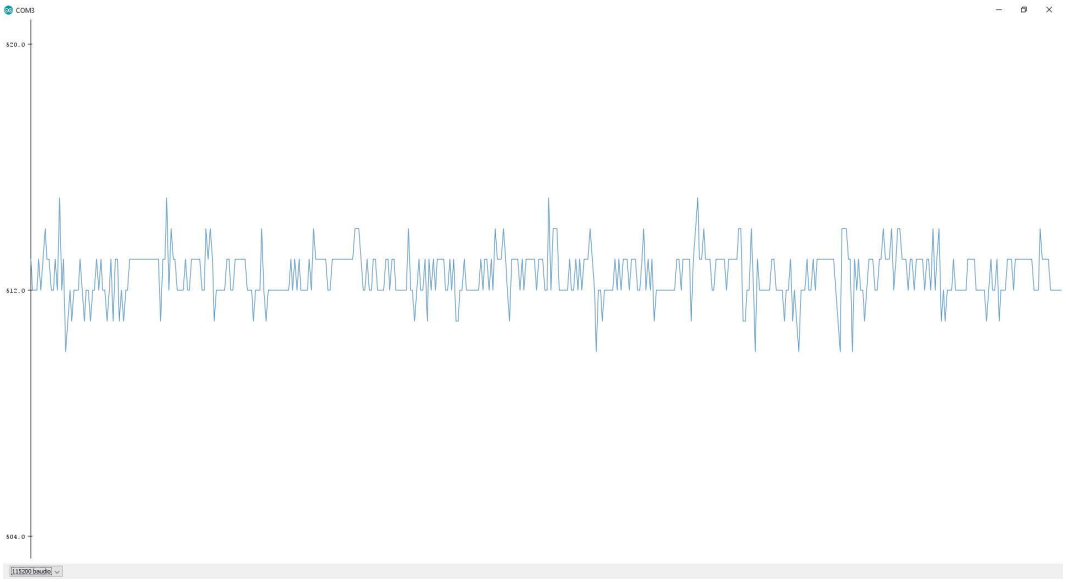


FIGURA 14. Tensión medida entre dos resitencias de $1M\Omega$.

Como podemos ver, la máxima variación detectada es de 5 puntos en la escala de Arduino sobre 1024 (Unos $0{,}0244V$). Nosotros vamos a intentar hacer que esa variación sea mayor utilizando un divisor de tensión que dé un valor de tensión mucho más pequeño y, posteriormente, amplificarlo mucho. El circuito diseñado para este propósito es el siguiente.

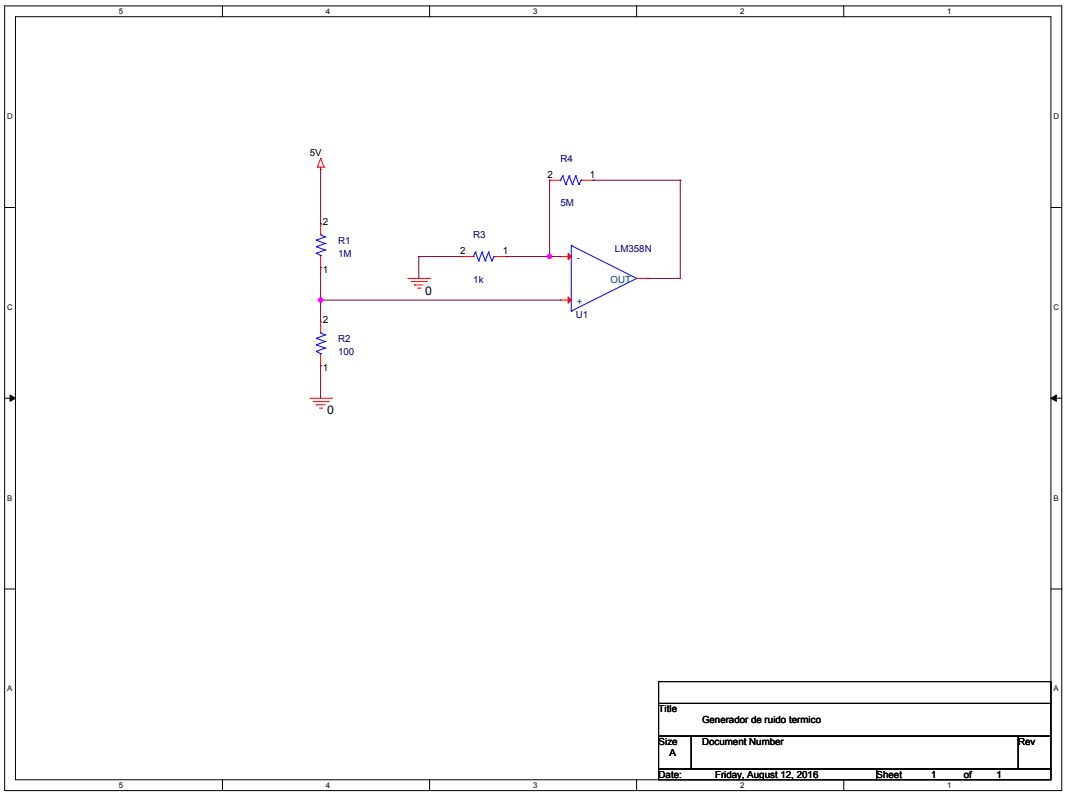


FIGURA 15. Circuito para la generación de ruido térmico.

Vamos a analizar el circuito diseñado. En primer lugar, encontramos un divisor de tensión que nos va a dar un valor de tensión de $4{,}99 \times 10^{-4}V$ aproximadamente (siempre tener en cuenta las tolerancias de las resistencias), después, tenemos un amplificador en configuración no inversora que multiplicará el valor de la tensión de entrada por 5001 dado que el valor de tensión de salida viene determinado por la ecuación $\frac{V_o}{V_i} = \frac{R_4}{R_3} + 1$. Esto nos da un valor teórico en la tensión de salida de $2{,}5V$. A continuación vemos el valor medido con la función Serial Plotter del IDE de Arduino.

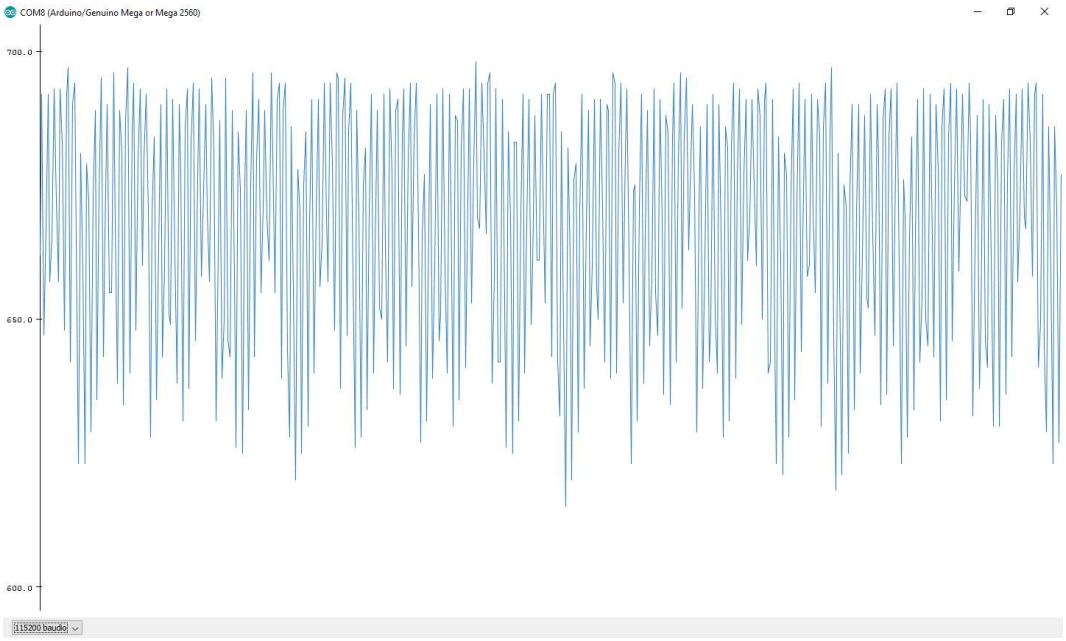


FIGURA 16. Tensión medida a la salida de nuestro generador diseñado.

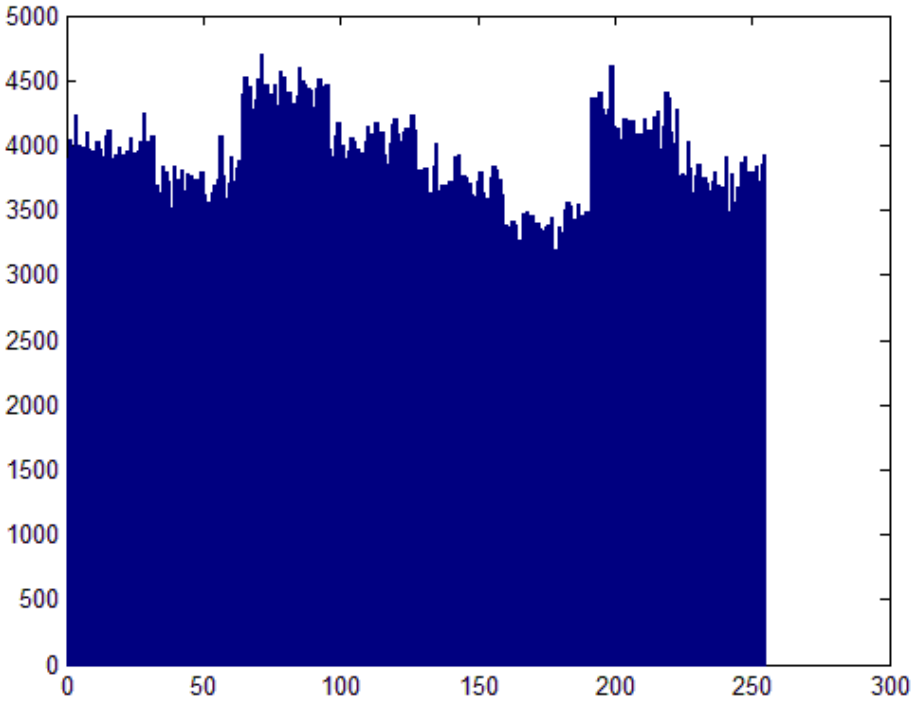
Ahora tenemos unas variaciones de tensión lo suficientemente grandes, el método que vamos a emplear para capturar los datos es mediante el bit menos significativo del valor tomado. Así mismo vamos a emplear un método para eliminar sesgos de forma que el número de 1s y 0s sea aproximadamente el mismo. Este método consiste en método de von Neumann, esto consiste en lo siguiente: tomamos 2 bits consecutivos y por cada par de capturas deberemos asignar el valor medido a lo que aparece en la siguiente tabla en función del valor de los 2 bits tomados. [**9**]

Entrada	Salida
00	Descartado
01	0
10	1
11	Descartado

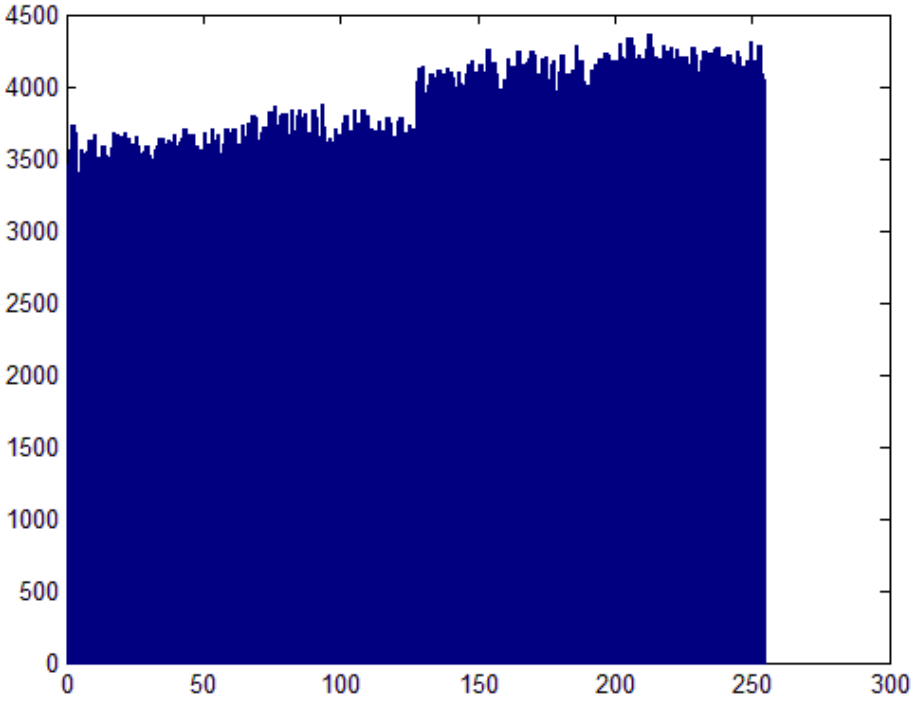
CUADRO 1. Método de von Neumann.

Como vemos, esto reduce la tasa, como mínimo a la mitad, pudiendo ser más baja cuanto más diferencia haya entre el número de 1s y 0s. Como vemos en el siguiente histograma, cuando generamos números de 8 bits (entre 0 y 256) el número de repeticiones de cada valor tiende a asemejarse sin llegar a ser perfecto del todo cuando aplicamos este método [1].

[1]El código para tomar los datos con Matlab se encuentra en el archivo Guarda_Datos_Serie.m y es una modificación del encontrado en [**3**].



(a) Sin aplicar von Neumann



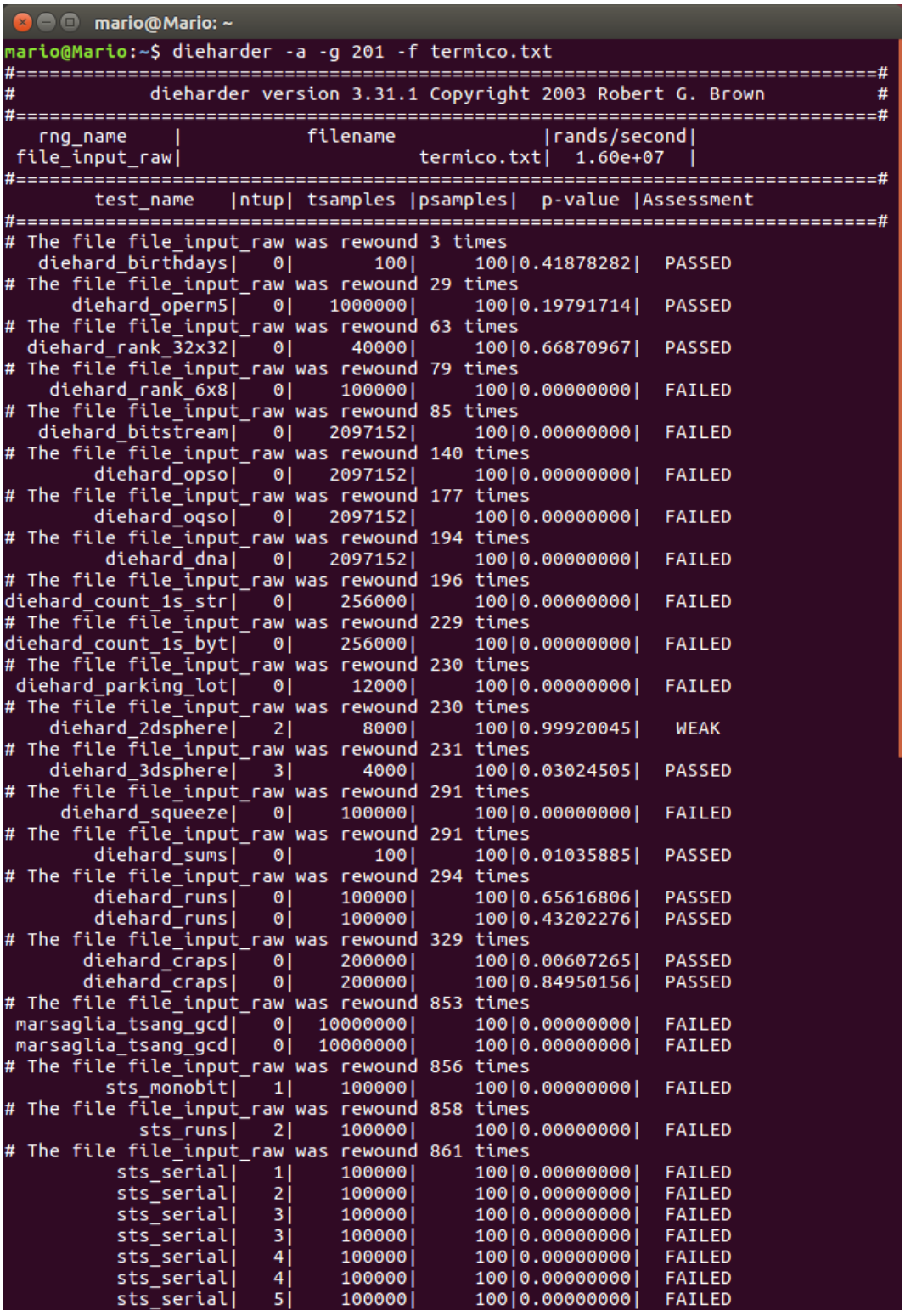
(b) Aplicando von Neumann

FIGURA 17. Histogramas del generador basado en ruido térmico.

Para determinar con exactitud la media y demás aspectos de la señal, utilizaremos el comando de linux ent (para usar dicho comando hay que escribir ent -b archivo.txt, donde archivo.txt es el nombre del archivo a evaluar). Los resultados de este comando son los siguientes:

- Entropy = 0.999938 bits per bit.
- Optimum compression would reduce the size of this 2612427480 bit file by 0 %.
- Arithmetic mean value of data bits is 0.4954 (0.5 = random).
- Monte Carlo value for Pi is 3.165519493 (error 0.76 percent).
- Serial correlation coefficient is -0.049430 (totally uncorrelated = 0.0).

De aquí sacamos que tiene una entropía muy alta, si buscamos en el manual de este comando (poner man ent en la linea de comandos) vemos que cuanto más alto sea, más dificil de comprimir es el archivo y, por lo tanto, más aleatorio será. Cuanto más similar a 1 bit por bit sea, más entropía tendrá. Vemos que el archivo se podría reducir en un 0 %. También vemos que la media es muy similar a 0.5, lo cual significa que el número de 1s y 0s es muy parecido siendo ligeramente mayor el de 0s. Utilizando el test de Monte Carlo (en este caso, este test toma valores como coordenadas y calcula la aproximación a pi) obtenemos 3.165519493, con una desviación del 0,76 % respecto de pi. Vemos que tenemos una muy baja correlación serie de los datos lo cual es muy bueno.
Con todo esto, obtenemos los siguientes resultados cuando aplicamos los datos tomados a Dieharder.



```
mario@Mario: ~
mario@Mario:~$ dieharder -a -g 201 -f termico.txt
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
 file_input_raw|                    termico.txt|  1.60e+07  |
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
# The file file_input_raw was rewound 3 times
   diehard_birthdays|   0|       100|     100|0.41878282|  PASSED
# The file file_input_raw was rewound 29 times
      diehard_operm5|   0|   1000000|     100|0.19791714|  PASSED
# The file file_input_raw was rewound 63 times
  diehard_rank_32x32|   0|     40000|     100|0.66870967|  PASSED
# The file file_input_raw was rewound 79 times
    diehard_rank_6x8|   0|    100000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 85 times
   diehard_bitstream|   0|   2097152|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 140 times
        diehard_opso|   0|   2097152|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 177 times
        diehard_oqso|   0|   2097152|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 194 times
         diehard_dna|   0|   2097152|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 196 times
diehard_count_1s_str|   0|    256000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 229 times
diehard_count_1s_byt|   0|    256000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 230 times
 diehard_parking_lot|   0|     12000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 230 times
    diehard_2dsphere|   2|      8000|     100|0.99920045|   WEAK
# The file file_input_raw was rewound 231 times
    diehard_3dsphere|   3|      4000|     100|0.03024505|  PASSED
# The file file_input_raw was rewound 291 times
     diehard_squeeze|   0|    100000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 291 times
        diehard_sums|   0|       100|     100|0.01035885|  PASSED
# The file file_input_raw was rewound 294 times
        diehard_runs|   0|    100000|     100|0.65616806|  PASSED
        diehard_runs|   0|    100000|     100|0.43202276|  PASSED
# The file file_input_raw was rewound 329 times
       diehard_craps|   0|    200000|     100|0.00607265|  PASSED
       diehard_craps|   0|    200000|     100|0.84950156|  PASSED
# The file file_input_raw was rewound 853 times
 marsaglia_tsang_gcd|   0|  10000000|     100|0.00000000|  FAILED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 856 times
         sts_monobit|   1|    100000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 858 times
            sts_runs|   2|    100000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 861 times
          sts_serial|   1|    100000|     100|0.00000000|  FAILED
          sts_serial|   2|    100000|     100|0.00000000|  FAILED
          sts_serial|   3|    100000|     100|0.00000000|  FAILED
          sts_serial|   3|    100000|     100|0.00000000|  FAILED
          sts_serial|   4|    100000|     100|0.00000000|  FAILED
          sts_serial|   4|    100000|     100|0.00000000|  FAILED
          sts_serial|   5|    100000|     100|0.00000000|  FAILED
```

FIGURA 18. Resultados de Dieharder del generador de ruido térmico.

Podemos ver que conseguimos pasar algunos test, aunque, probando Dieharder con archivos del generador de Linux (urandom) muy pequeños, estos pueden no pasar los test, esto puede suponer un problema dado que las lecturas analógicas de Arduino están algo limitadas dado que cada una de estas lleva 112μs y lleva mucho tiempo conseguir archivos muy grandes.
Para solucionarlo, vamos a tratar de crear variaciones que sean posibles de detectar por uno de los puertos digitales de Arduino. Para ello, utilizaremos un circuito comparador, en uno de los puertos conectaremos la salida de nuestro generador y, en el otro conectaremos la salida de un divisor de tensión en el cual, en una de las resistencias, pondremos una resistencia variable para poder adecuarlo a la tensión media medida. El esquemático del circuito usado será el mostrado a continuación [2].

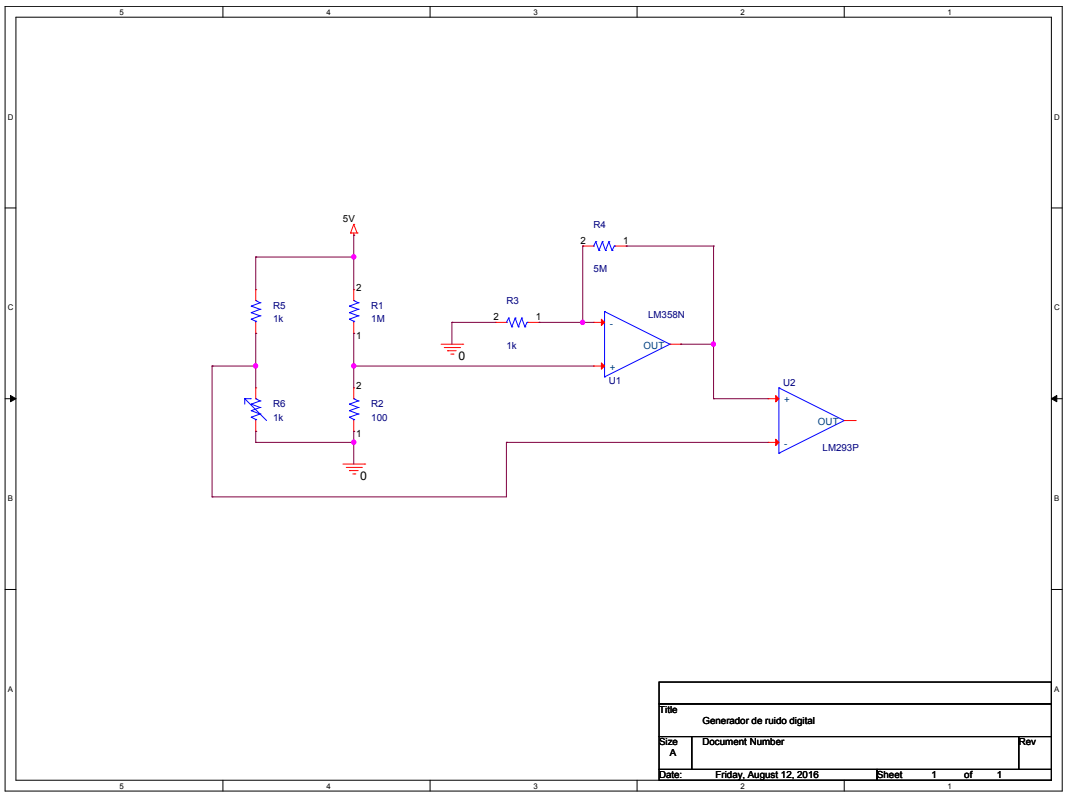


FIGURA 19. Circuito para la generación de ruido térmico digital.

A la salida de este circuito, tenemos una señal como la mostrada a continuación la cual podría no ser suficiente como para considerarla digital, esto puede deberse a que las variaciones no son suficientemente grandes como para generar una salida binaria, a pesar de que, como vemos en el apartado de los materiales usados, tendría que serlo.

[2]Aunque no viene especificado en el circuito, la salida del comparador es de drenador abierto, por lo que tenemos que conectar la salida a 5 V a través de una resistencia.

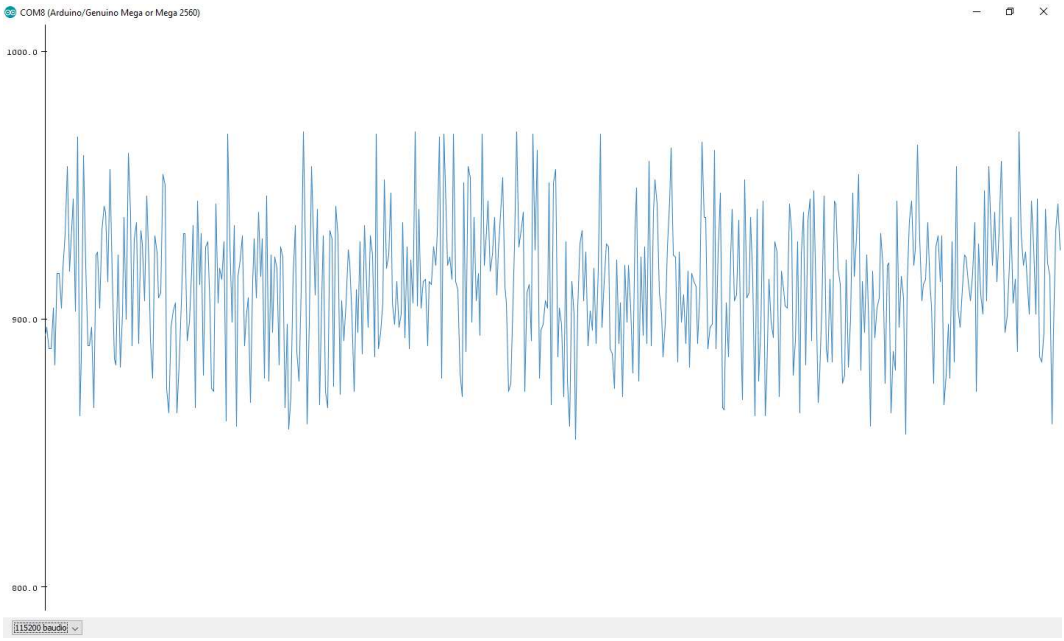


FIGURA 20. Tensión medida a la salida del generador digital.

Para poder solucionarlo podemos conectar esta salida a la salida de un nuevo comparador para tener variaciones aun más grandes. El circuito quedaría como el que aparece a continuación.

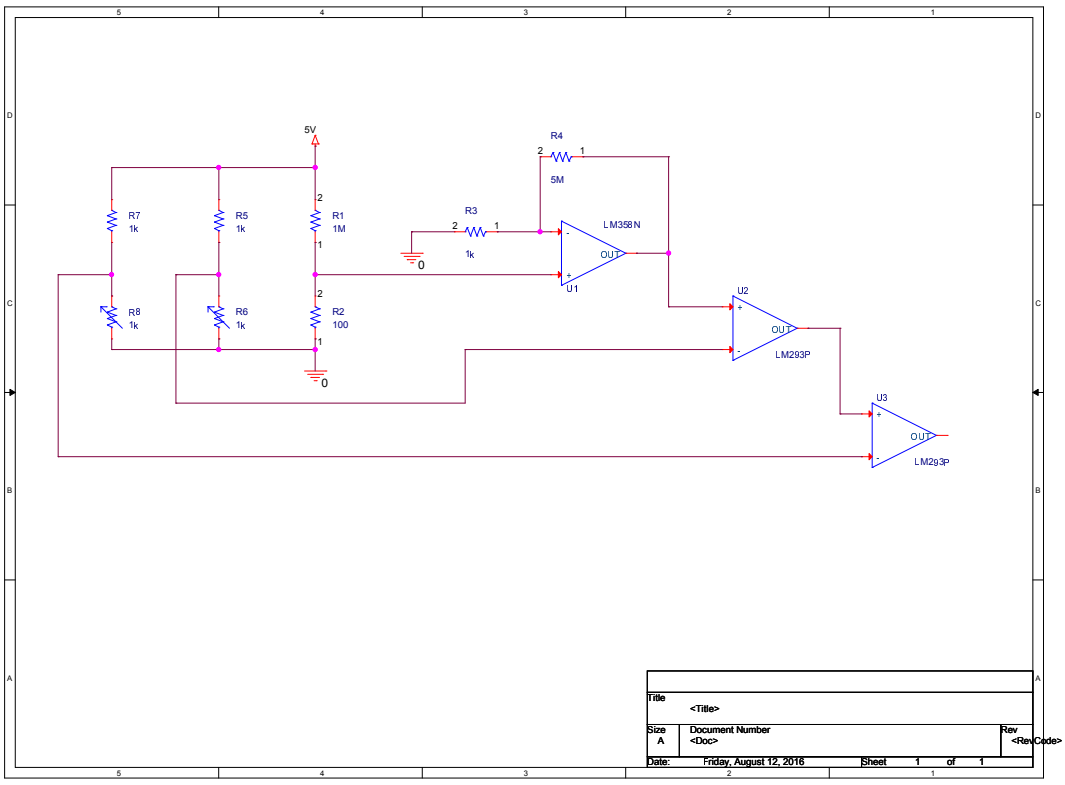


FIGURA 21. Circuito para la generación de ruido térmico digital mejorado.

Obtendremos los siguientes resultados en el caso de que midamos con una patilla analógica de Arduino.

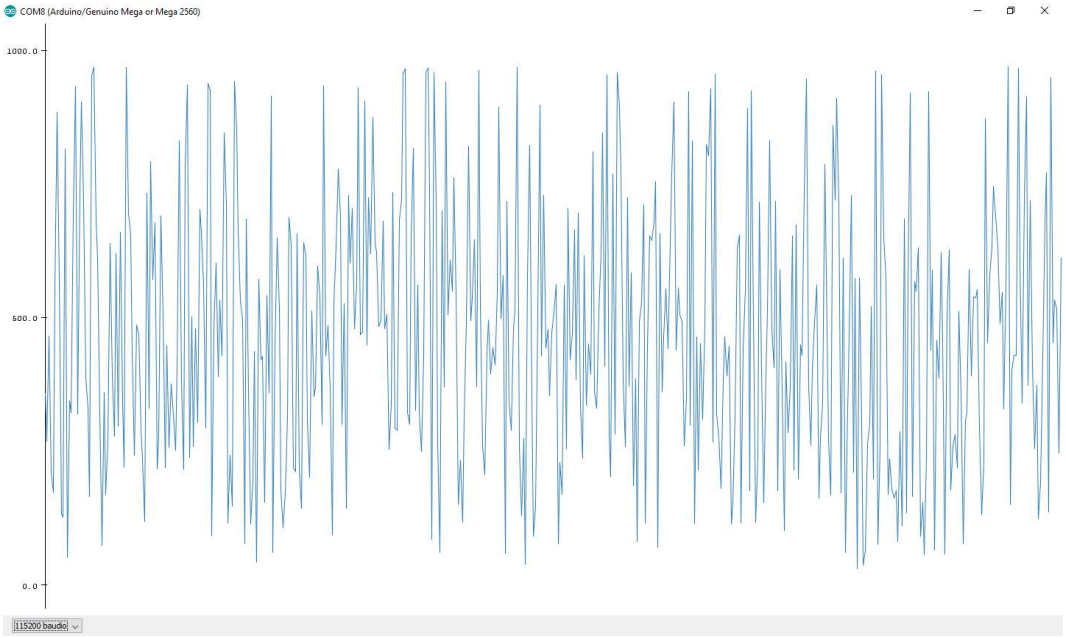


FIGURA 22. Tensión medida a la salida del generador digital mejorado.

Como podemos ver, tenemos unas variaciones lo bastante grandes como para poder tomar la señal con una patilla digital. Una vez medido con un pin digital, y utilizando el metodo de von Neumann para eliminar posibles sesgos, tenemos la siguiente señal:

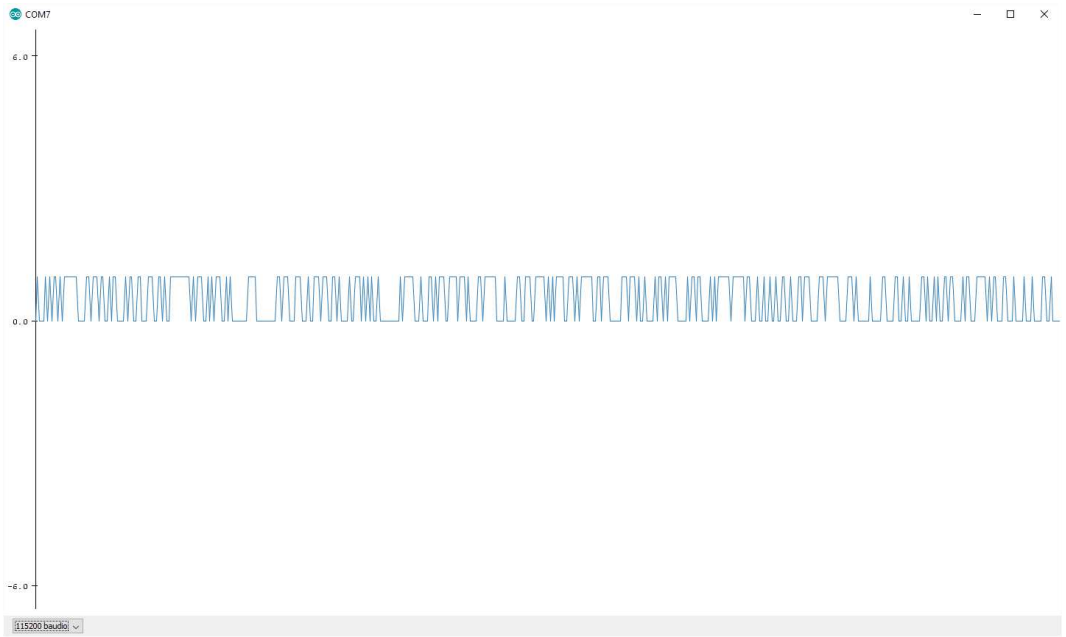


FIGURA 23. Niveles lógicos medidos a la salida del generador digital mejorado.

Tras tomar suficientes datos (para tomarlos hemos usado el Fubarino y los hemos ido almacenando en la tarjeta SD) usamos el comando ent como hemos hecho anteriormente para saber si el número de 1s es aproximadamente igual al de 0s:

- Entropy = 0.999876 bits per bit.
- Optimum compression would reduce the size of this 139942192 bit file by 0 %.
- Arithmetic mean value of data bits is 0.5066 (0.5 = random).
- Monte Carlo value for Pi is 3.057822054 (error 2.67 percent).
- Serial correlation coefficient is 0.004705 (totally uncorrelated = 0.0).

De aquí sacamos que tiene una entropía muy alta. Cuanto más similar a 1 bit por bit sea, más entropía tendrá. Vemos que el archivo se podría reducir en un 0 %. También vemos que la media es muy similar a 0.5, lo cual significa que el número de 1s y 0s es muy parecido siendo ligeramente mayor el de 1s. Utilizando el test de Monte Carlo obtenemos 3.057822054, con una desviación del 2,67 % respecto de pi, en este aspecto el empeoramiento ha sido significativo. Vemos que tenemos una muy baja correlación serie de los datos lo cual es muy bueno.
Tras finalizar esta comprobación, pasamos los datos a Dieharder y obtenemos los siguientes resultados:

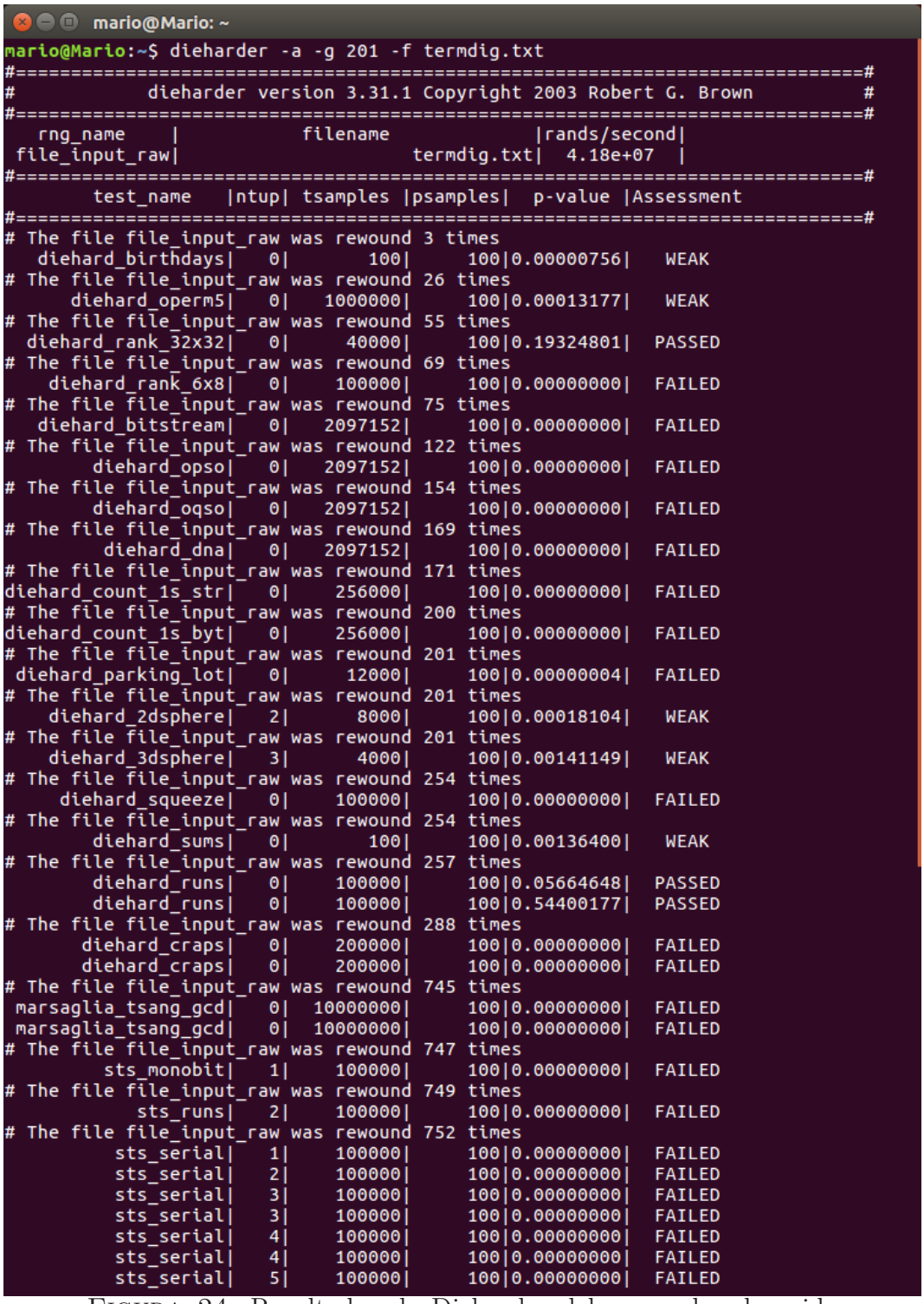


FIGURA 24. Resultados de Dieharder del generador de ruido térmico digital.

Vemos que la conversión de la señal en digital ha deteriorado la calidad de los números aleatorios. Algunos tests que pasaba el generador analógico no han sido pasados por el generador digital, sin embargo no hay ningún test

a mayores que haya sido pasado por el digital. A continuación tenemos una imagen del montaje del generador sobre una placa de prototipos:

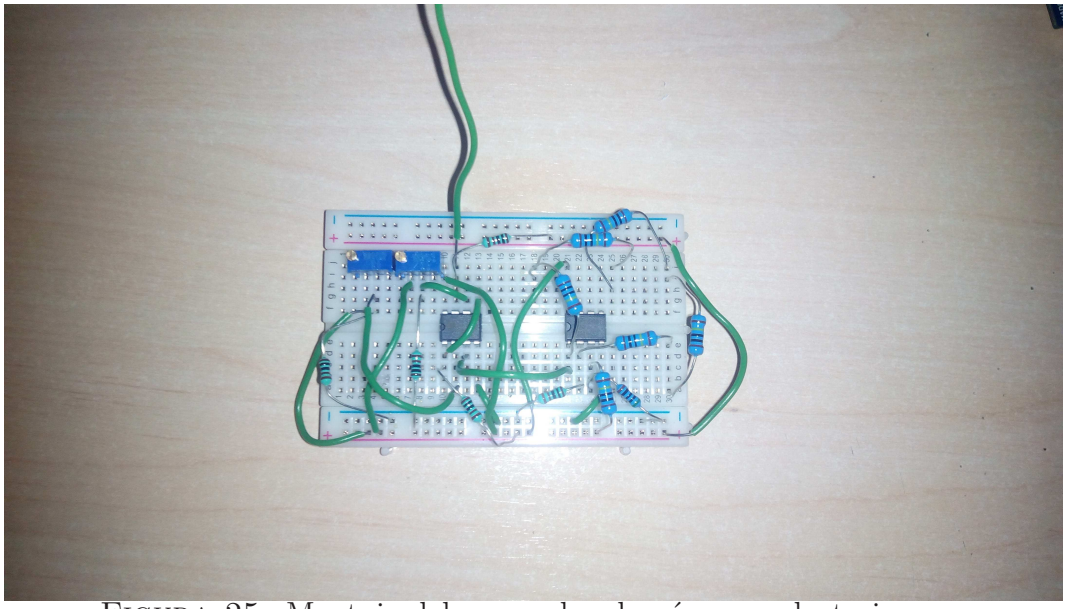


FIGURA 25. Montaje del generador de números aleatorios con ruido térmico.

Las resistencias azules de la izquierda son las resistencias variables. El circuito integrado de la izquierda es el comparador LM293P que incluye 2 comparadores, perfecto para nuestras necesidades. El circuito integrado de la derecha es el amplificador LM358N usado para amplificar el ruido generado. Las líneas de alimentación y tierra son las de los extremos superior e inferior.

Capítulo 8

Diseño del generador usando diodos Zener

Para el diseño del generador usando diodos Zener se ha decidido utilizar los diodos Zener de 2.4 V (consultar lista de materiales) dicho diodo tiene las siguientes características:

Electrical Characteristics

Values are at T_A = 25°C unless otherwise noted .

Device	V_Z (V) @ I_Z (2)			Z_Z (Ω) @ I_Z (mA)		Z_{ZK} (Ω) @ I_{ZK}(mA)		I_R (μA) @ V_R (V)		T_C (%/°C)
	Min.	Typ.	Max.							
1N5221B	2.28	2.4	2.52	30	20	1,200	0.25	100	1.0	-0.085

FIGURA 26. Características del diodo Zener de 2.4 V [**4**].

Teniendo en cuenta que un diodo Zener tiene una curva V-I con la siguiente forma (debemos tener en cuenta que I_{ZT} es lo mismo que I_Z en la tabla):

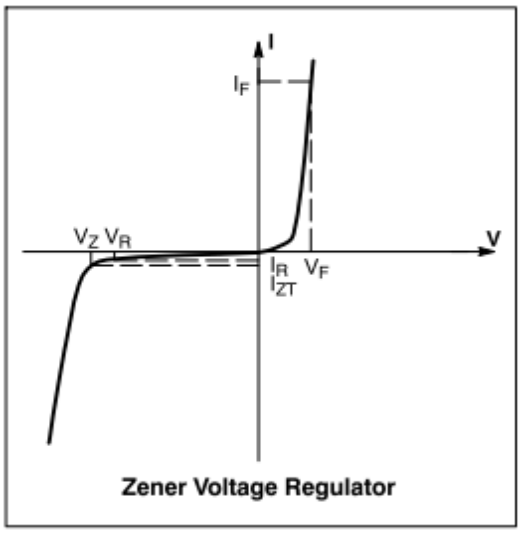


FIGURA 27. Curva característica de un diodo Zener [**10**].

Buscamos conectar un diodo en inversa y que se sitúe en el lado izquierdo de la gráfica, es decir, cuando alcanza la tensión de ruptura o incluso cuando sobrepasa la tensión Zener. Para ello, hemos diseñado el siguiente circuito:

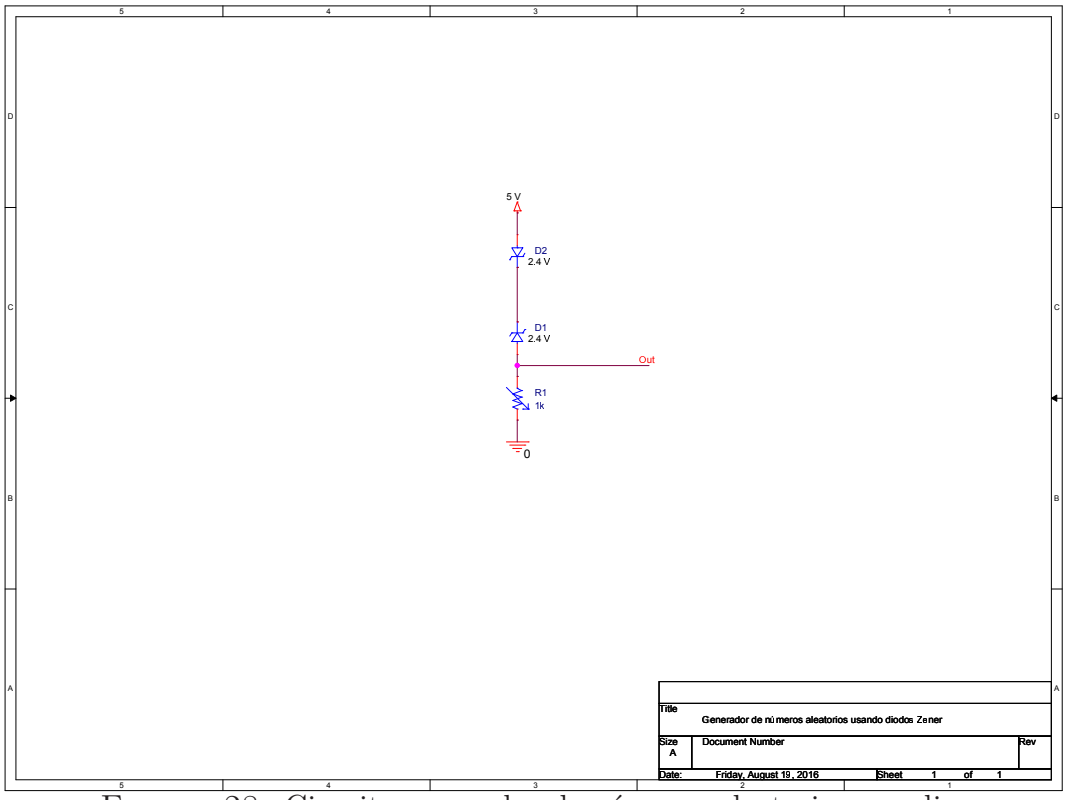


FIGURA 28. Circuito generador de números aleatorios con diodos Zener.

El ruido proviene del diodo Zener que se encuentra en inversa. Iremos disminuyendo el valor de la resistencia hasta el punto en el que el valor de tensión medido en el punto Out del circuito sea aleatorio. Con el diodo Zener en directa buscamos una caída fija de tensión de forma que si disminuimos mucho el valor de la resistencia, la caída de tensión en el diodo en inversa no sea tan excesiva.
Una vez ponemos un valor adecuado de la resistencia, podemos obtener unos valores de tensión en la salida como los que aparecen en la figura.

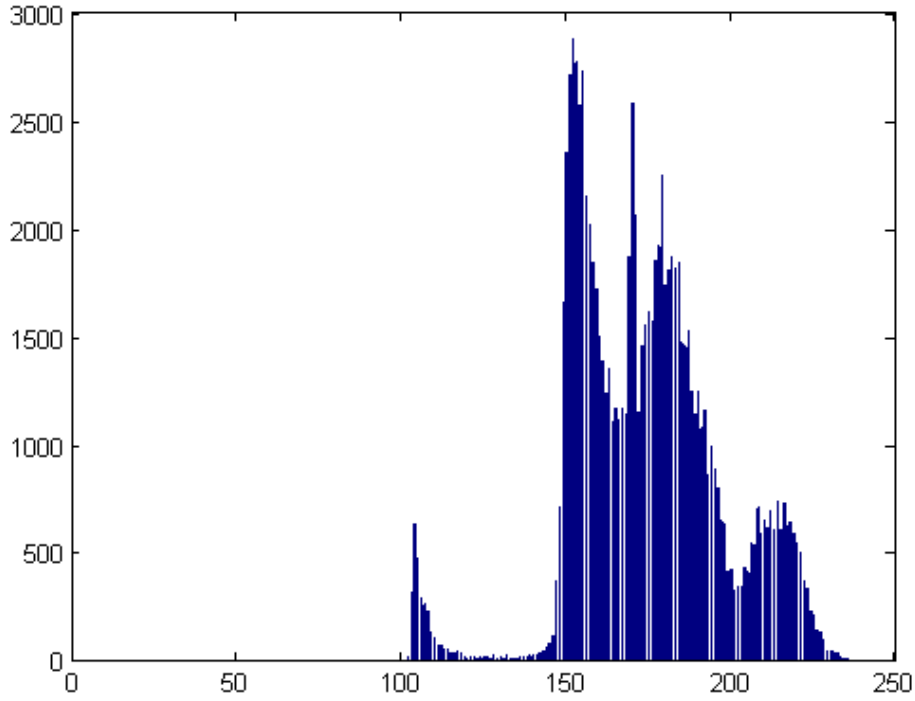


FIGURA 29. Histograma de valores medidos en la salida.

Para tomar la muestra, utilizaremos el programa mostrado en los códigos del proyecto llamado AnalogicaAleatorio.
Utilizando directamente los datos obtenidos de dicho generador a Dieharder obtenemos los siguientes resultados:

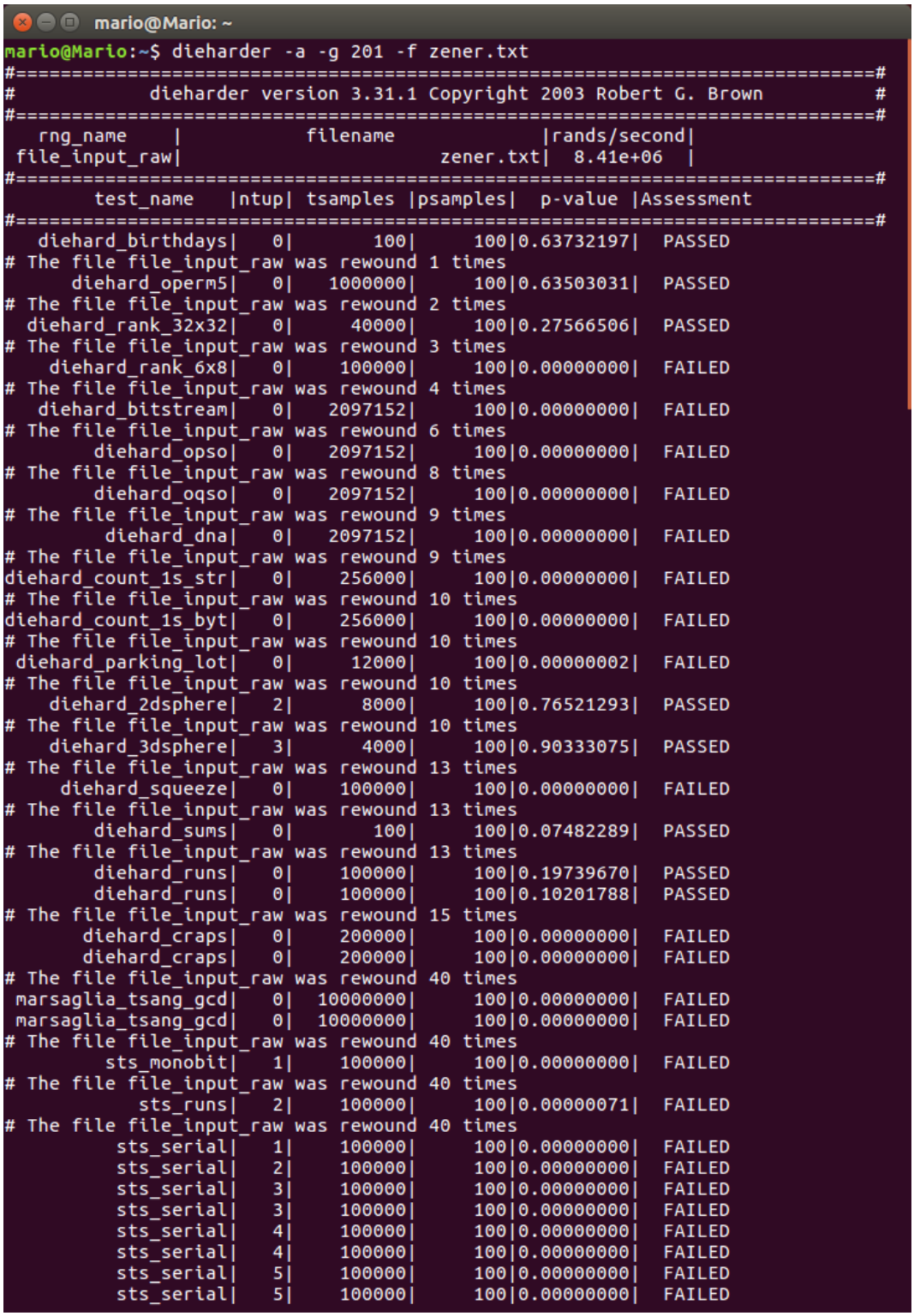


FIGURA 30. Resultados de Dieharder del generador de ruido Zener.

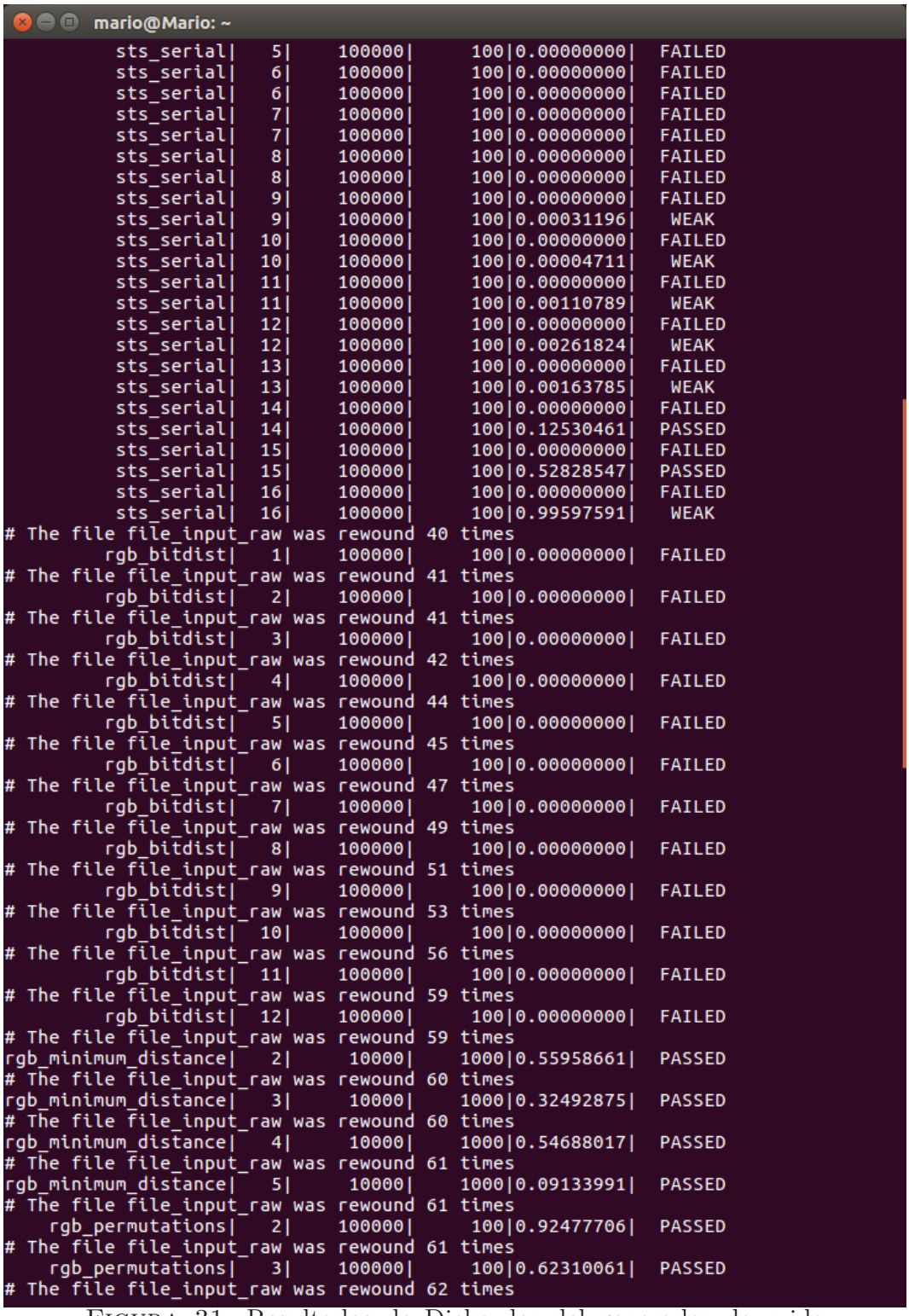


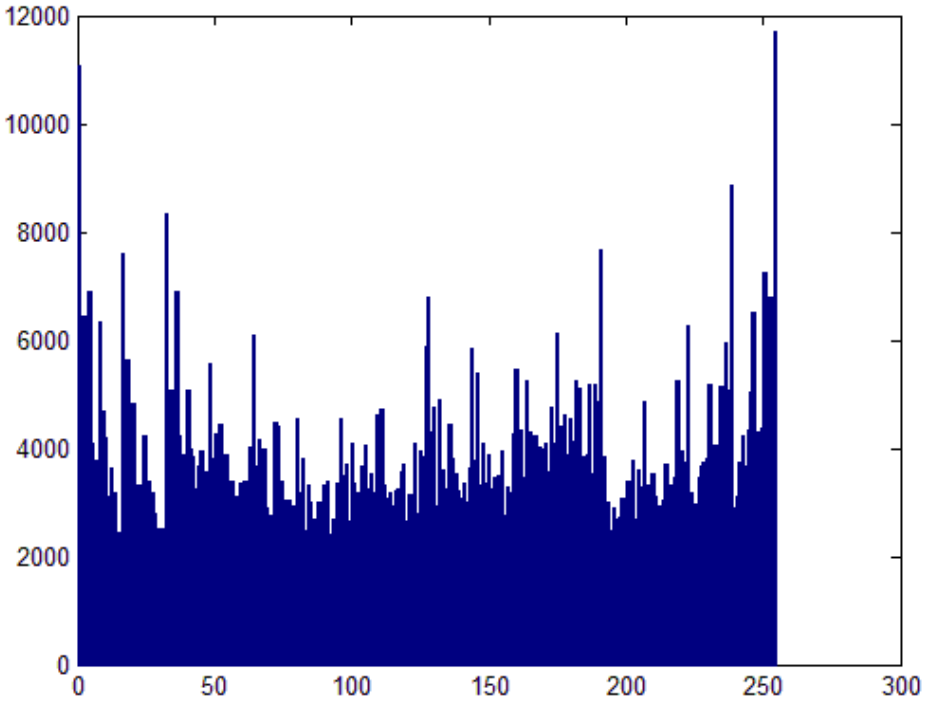
FIGURA 31. Resultados de Dieharder del generador de ruido Zener (cont).

Como podemos ver, hemos superado más tests que con el generador basado en ruido térmico pero seguimos sin superar todas las pruebas en parte por el

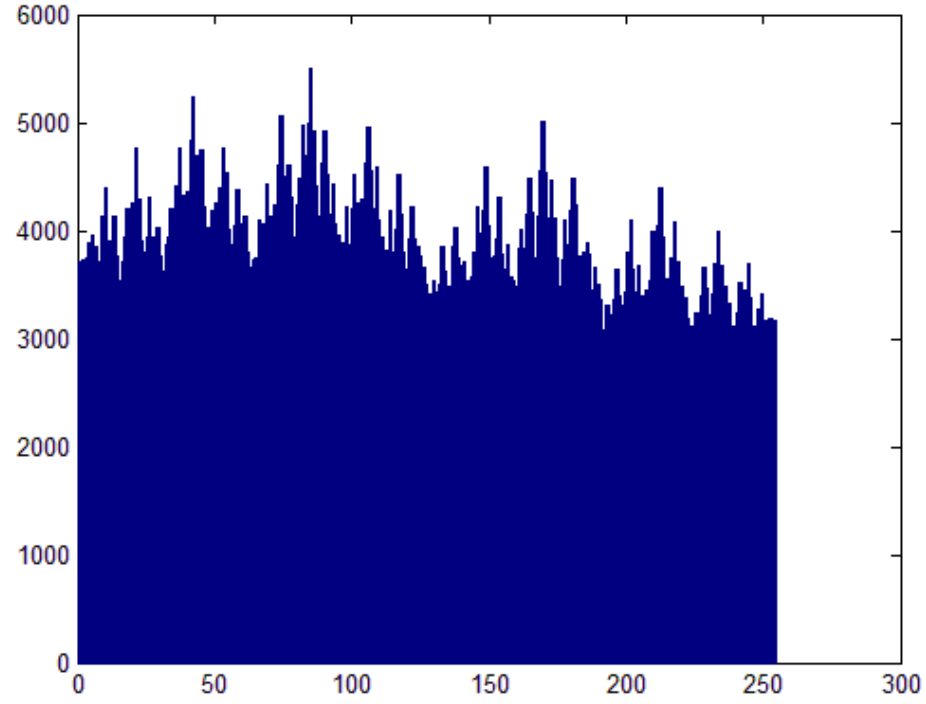
tamaño del archivo generado o por posibles problemas de carreras.
Para saber si el número de 1s es aproximadamente igual al de 0s, volveremos a utilizar el comando de linux ent. Los resultados de este comando son los siguientes:

- Entropy = 0.999971 bits per bit.
- Optimum compression would reduce the size of this 2612427480 bit file by 0 %.
- Arithmetic mean value of data bits is 0.5032 (0.5 = random).
- Monte Carlo value for Pi is 3.062634528 (error 2.51 percent).
- Serial correlation coefficient is 0.005958 (totally uncorrelated = 0.0).

De aquí sacamos que tiene una entropía muy alta. Cuanto más similar a 1 bit por bit sea, más entropía tendrá. Vemos que el archivo se podría reducir en un 0 %. También vemos que la media es muy similar a 0.5, lo cual significa que el número de 1s y 0s es muy parecido siendo ligeramente mayor el de 1s. Utilizando el test de Monte Carlo obtenemos 3.062634528, con una desviación del 2,51 % respecto de pi. Vemos que tenemos una muy baja correlación serie de los datos lo cual es muy bueno.
A continuación, vamos a comprobar las posibles carreras que podamos tener, para ello vamos a agrupar los bits tomados en conjuntos de 8 de forma que representen un número entre 0 y 255. Vamos a hacer este proceso tanto para el generador normal como para el generador utilizando von Neumann.



(a) Sin aplicar von Neumann



(b) Aplicando von Neumann

FIGURA 32. Histogramas del generador basado en Zener.

Como se ha podido ver, el problema de las carreras hace que sea más probable que sucedan valores próximos a los extremos (0 y 255) mientras que los valores centrales son menos probables.
Tratando de mejorar la velocidad de este generador, incluimos un circuito comparador como el mostrado en la figura. Dicho circuito tiene un divisor de tensión que variaremos para alcanzar el nivel de tensión equivalente al valor medio de los valores del circuito Zener.

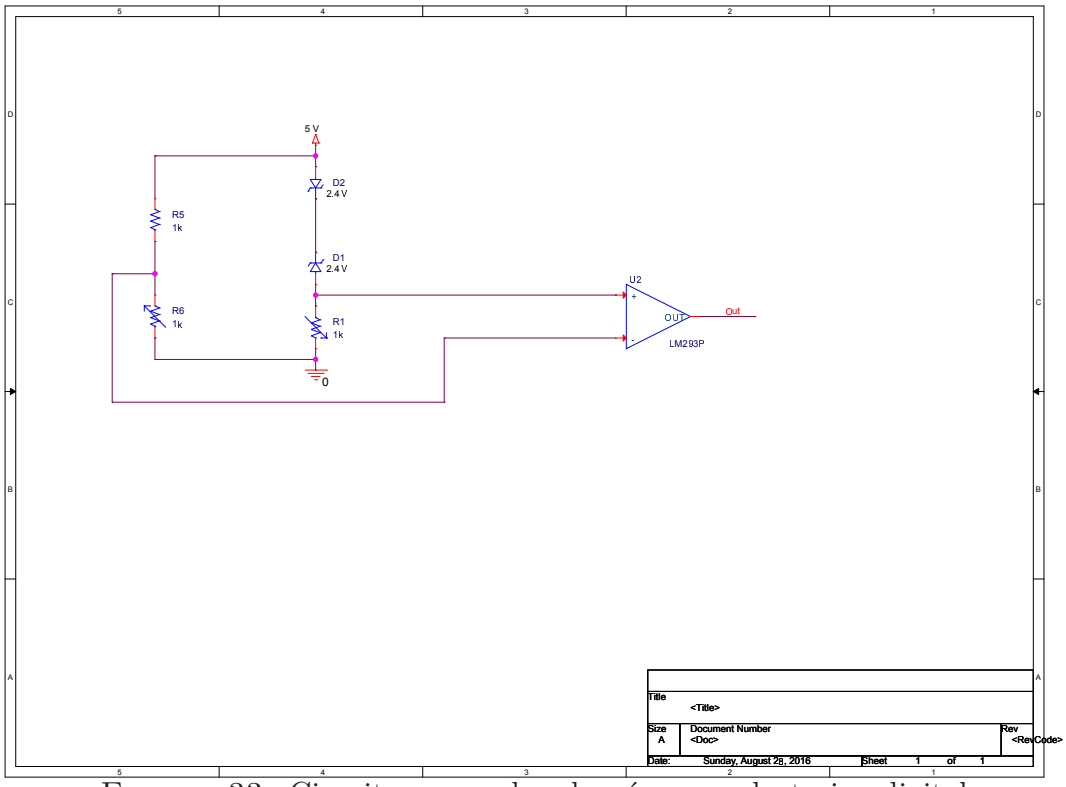


FIGURA 33. Circuito generador de números aleatorios digital con diodos Zener.

Como resultado de este circuito tenemos que decir que el valor medio de tensión en la salida no va a ser siempre igual variando demasiado por lo que no hemos podido ajustar el divisor de tensión a un valor fijo por lo que no he visto viable este circuito. A continuación tenemos una imagen del montaje del generador sobre una placa de prototipos:

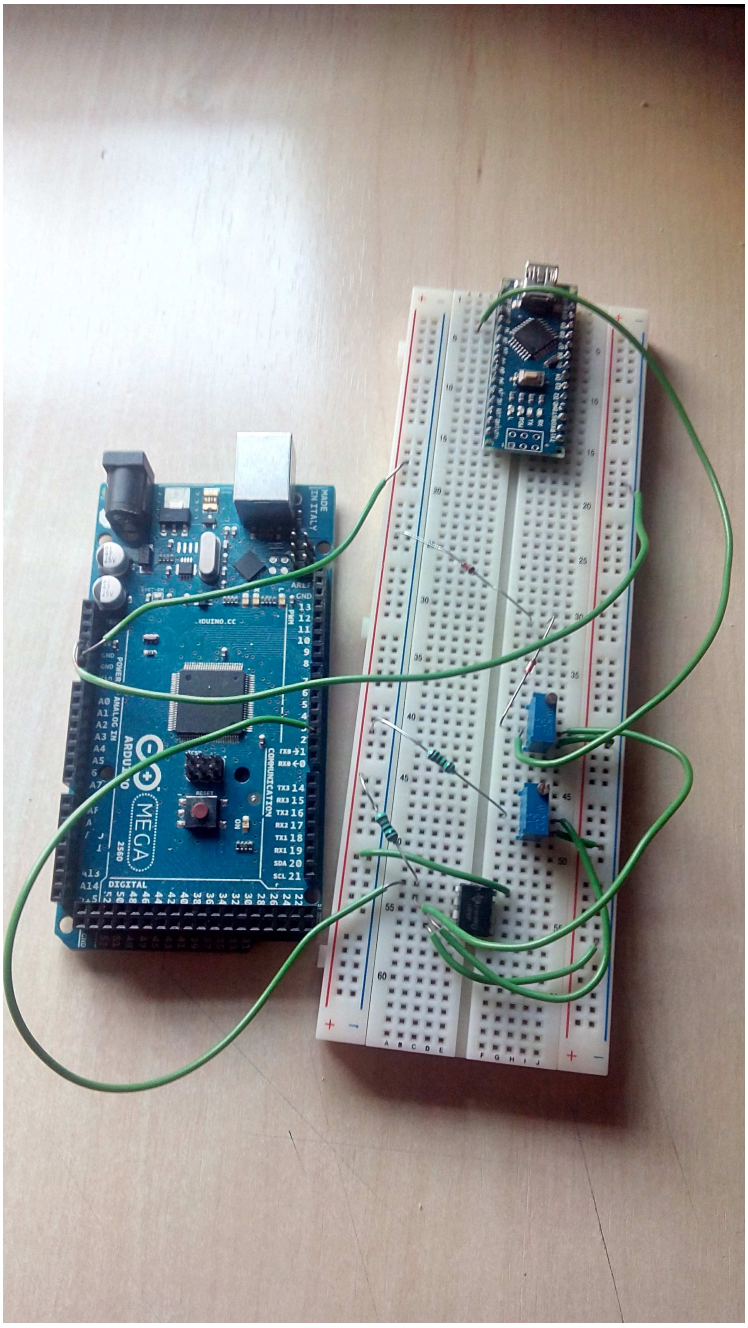


FIGURA 34. Montaje del generador basado en diodos Zener.

En el circuito vemos cómo la alimentación se ha realizado mediante el Arduino Mega para que proporcione toda la corriente que podamos necesitar mientras que el Arduino Nano se encarga de tomar los valores de tensión.
Vemos que se trata del generador digital pero con solo quitar el comparador que se ve abajo en la figura, las resistencias de color turquesa y la resistencia variable de abajo tendríamos el generador analógico.
Respecto a la parte analógica, podemos apreciar los dos diodos Zener que están en serie, solo que el que aparece conectado en un extremo a 5 V está

colocado en directa y el otro en inversa siguiendo el esquemático mostrado anteriormente. El ajuste de las resistencias variables se realiza observando la señal con la función serial plotter de Arduino IDE y modificando el valor de dichas resistencias hasta que apreciemos variaciones significativas de la tensión.

Capítulo 9

Conclusiones

Durante la realización del estudio teórico de mi trabajo de fin de grado he aprendido un montón de cosas sobre los números aleatorios y la necesidad de conseguirlos además de haber perfeccionado mis conocimientos sobre el ruido en la electrónica y diferentes fenómenos cuánticos que aparecen en estos. Por otro lado, durante el desarrollo de los dispositivos generadores, he mejorado mi soltura en el montaje de circuitos en placas así como en la facilidad de determinar el origen de los problemas que puede tener un circuito.
Respecto a los generadores diseñados, no hemos conseguido obtener un generador que supere tantos test ni que sea tan rápido como el que proporciona Linux. Si apreciamos bien los test que no han podido pasar nuestros generadores, vemos que muchos están relacionados con coger un conjunto de bits y asignarlos a un alfabeto determinado, estas pruebas serán difíciles de superar si tenemos problemas de carreras y el método de von Neumann puede no ser suficiente para corregirlo. A pesar de ello, yo creo que habiendo superado ciertos test de Dieharder y obteniendo generadores que consiguen tener una media muy próxima a 0,5 (muy similar número de 1s que de 0s) podemos estar satisfechos con los resultados que hemos ido obteniendo. La generación de números aleatorios es un tema que trae de cabeza a muchos expertos y no iba a ser posible alcanzar el nivel que tienen los generadores de hoy en día.
Con todo esto, hemos conseguido unos circuitos que podrían ser usados para uso doméstico de forma sencilla. Simplemente modificando el código que tiene nuestra placa podríamos hacer que devolviese un número en un rango que queramos obteniendo el bit menos significativo de los valores de tensión capturados por un pin.
Finalmente, por hacer una pequeña comparativa de los generadores caseros que hemos hecho, podemos decir que el generador basado en diodos Zener ha tenido mejores resultados que el basado en ruido térmico aunque no hayamos utilizado eliminación de sesgos con von Neumann al aplicar los datos a Dieharder. Esto se traduce en una mayor velocidad también dado que el método de von Neumann reduce la tasa de bits por lo menos a la mitad. A nivel teórico esta conclusión tiene sentido ya que el efecto Zener es un fenómeno cuántico y es mas impredecible que el efecto que pueda producir la temperatura sobre un componente electrónico.
En vista a futuros proyecto, tenemos unas cuantas mejoras que podrían aplicarse:

Mejoras en la velocidad Ha sido uno de los principales problemas que hemos tenido, una posible solución sería implementar los códigos que

hemos escrito para nuestras placas en una FPGA, ya que estas pueden alcanzar velocidades mayores que las placas que hemos utilizado.

Mejoras en el generador digital Un aspecto importante de una señal digital es que la anchura de los 1s y de los 0s sea la misma y esto se podría haber conseguido añadiendo a la salida de nuestros generadores digitales un Flip-Flop tipo D por ejemplo, esta solución podría implementarse vía software también.

Mejoras en la aleatoriedad de los circuitos Como hemos visto, nuestros circuitos no han podido superar todos los test que los hemos hecho pasar, posibles soluciones a este problema serían aumentar las variaciones que hemos obtenido con un valor de amplificación mucho mayor o solucionar posibles problemas de carreras que puedan existir. Para solucionar este último aspecto nosotros hemos optado por el método de von Neumann para solventarlo, sin embargo, existen otros meétodos como el de von Neumann iterado que se ha demostrado que se acerca mucho a la tasa óptima de bits conseguidos tal y como aparece en [**11**].

Apéndice A

Lista de materiales

Para hacer el cálculo generado por los materiales, hemos realizado una lista con todos los materiales que han tenido que ser adquiridos para las diferentes pruebas que se han realizado sobre los generadores y para el montaje de los mismo. Dicha lista contiene los siguientes apartados:

Cantidad Indica el número total de elementos de dicho componente que hay.

Valor Indica el valor de dicho componente, es un aspecto importante en el caso de las resistencias y condensadores.

Modelo Con este apartado sabemos el modelo exacto del componente adquirido.

Ref. Farnell referencia ofrecida por Farnell.

Precio ud Precio por unidad ofrecido por Farnell.

Precio total Resultado de multiplicar el contenido del campo cantidad por el contenido del campo Precio ud de una fila.

Aunque muchos materiales no aparecen en los generadores diseñados, se incluyen porque han formado parte del montaje durante las pruebas que se han ido realizado. Los Arduino Nano utilizados no son los originales y por lo tanto no han tenido un coste tan elevado. De hecho la explicación del coste tan alto del Arduino Nano mostrado en la lista de materiales es que se trata de una placa descatalogada, se podrían encontrar placas originales muy similares a Arduino Nano y que no tengan ese coste.
Por otra parte, esta lista no tiene en cuenta los costes generados por el uso de diferentes elementos como pueden ser ordenadores y gasto eléctrico.

Lista de materiales para generador de número aleatorios

Autor: Mario González de la Fuente

Total de componentes: 133

64 Resistencias:

Cantidad	Valor	Modelo	Ref. Farnell	Precio ud	Precio total
10	10 kΩ	MCMF0W4DF1002A50	1563077	0,2450 €	2,4500 €
10	330 Ω	MFR4-330RFI	1099879	0,0808 €	0,8080 €
10	4,7 Ω	SFR16S0004708JA500	9476091	0,0620 €	0,6200 €
10	100 Ω	MCMF0W4DF1000A50	1563074	0,2480 €	2,4800 €
10	1 kΩ	MCMF006FF1001A50	2401753	0,0649 €	0,6490 €
10	1 MΩ	MF50	9339809	0,1110 €	1,1100 €
4	1 kΩ (var)	3296W	9353178	3,0100 €	12,0400 €

33 Condensadores:

Cantidad	Valor	Modelo	Ref. Farnell	Precio ud	Precio total
10	10 pF	MCCHU5100J5	9411658	0,0373 €	0,3730 €
3	10 nF	D103K43Y5PH63L2R	1827843	0,2360 €	0,7080 €
10	47 pF	MCCHU5470J5	9411690	0,0438 €	0,4380 €
10	100 Pf	MCBU5101K5	9411747	0,0326 €	0,3260 €

25 Diodos zener:

Cantidad	Valor	Modelo	Ref. Farnell	Precio ud	Precio total
18	2,4 V	1N5221B	1467581	0,0396 €	0,7128 €
5	6 V	1N5340BG	9557970	0,3740 €	1,8700 €
2	3,3 V	1N5333BG	9557903	0,1120 €	0,2240 €

6 Circuitos integrados:

Cantidad	Valor	Modelo	Ref. Farnell	Precio ud	Precio total
3	--	LM293P	2292956	0,4130 €	1,2390 €
3	--	LM358N	2295980	0,2200 €	0,6600 €

5 Otros:

Cantidad	Valor	Modelo	Ref. Farnell	Precio ud	Precio total
1	1 m	MC6A-7/0,2T2-YW-100	2425406	0,0988 €	0,0988 €
2	--	Arduino nano	1848691	40,4100 €	80,8200 €
1	--	Fubarino SD	2345728	25,8700 €	25,8700 €
1	--	Arduino mega	2212779	35,8100 €	35,8100 €

Total: 169,3066 €

FIGURA 35. Lista de materiales

Apéndice B

Códigos del proyecto

Código para generar números aleatorios con Arduino:

. codigos/GeneradorArduino.ino

```
// Inicializo el puerto serie a 115200 y genero una semilla
int aleatorio;
void setup() {
  Serial.begin(115200);
  randomSeed(analogRead(0));
}

// Genero un numero aleatorio de que ocupe 1 byte y lo envio
void loop() {
  aleatorio = random(256);
  Serial.write(aleatorio);
}
```

Código para leer el valor de tensión (de 0 a 1023) del pin A0:

. codigos/LecturaAnalogicaPin0.ino

```
void setup() {
  // initialize serial communication at 115200 bits per second:
  Serial.begin(115200);
}


void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read:
  Serial.println(sensorValue);
}
```

Código para leer y enviar el bit menos significativo por bytes por el puerto serie:

. codigos/AnalogicaAleatorio.ino

```
// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 115200 bits per second:
  Serial.begin(115200);
}
char byteAEnviar;
int contador=0;
// the loop routine runs over and over again forever:
void loop() {
```

```
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  byteAEnviar= ((byteAEnviar<<1) | (sensorValue&1));
  contador++;
  // print out the value you read:
  if(contador==8){
      Serial.write(byteAEnviar);
      byteAEnviar=0;
      contador=0;
  }

}
```

Código para leer y guardar el bit menos significativo en bytes en la tarjeta SD:

. codigos/FubarinoAleatorio.ino

```
#include <SD.h>


const int chipSelect_SD_default = 27;

// chipSelect_SD can be changed if you do not use default CS pin
const int chipSelect_SD = chipSelect_SD_default;
int byteAEnviar;
int contador=0;
int elEnviados=0;
void setup()
{
  Serial.begin(115200);
  Serial.print("Initializing SD card...");

  // Make sure the default chip select pin is set to so that
  // shields that have a device that use the default CS pin
  // that are connected to the SPI bus do not hold drive bus
  pinMode(chipSelect_SD_default, OUTPUT);
  digitalWrite(chipSelect_SD_default, HIGH);

  pinMode(chipSelect_SD , OUTPUT);
  digitalWrite(chipSelect_SD , HIGH);


  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect_SD)) {
    Serial.println("Card failed , or not present");
    // don't do anything more:
    return;
  }
  Serial.println("card initialized.");
}

void loop()
{
  //String dataString = "";
  int valorUno = analogRead(8);
  byteAEnviar= ((byteAEnviar<<1) | (valorUno&1));
```

```
40    contador++;
41
42    // print out the value you read:
43    if(contador==8){
44      elEnviados++;
45      //dataString += String(byteAEnviar);
46
47      // open the file. note that only one file can be open at a
          time,
48      // so you have to close this one before opening another.
49      File dataFile = SD.open("termico.txt", FILE_WRITE);
50
51      // if the file is available, write to it:
52      if (dataFile) {
53        dataFile.write(byteAEnviar);
54        dataFile.close();
55        // print to the serial port too:
56       Serial.println(elEnviados);
57      }
58      // if the file isn't open, pop up an error:
59      else {
60        Serial.println("error␣abriendo␣termico.txt");
61      }
62      byteAEnviar=0;
63      contador=0;
64    }
65  }
```

Código para guardar en una matriz el número de datos que queramos y lo recibamos por el puerto serie que indiquemos:

. codigos/Guarda_Datos_Serie.m

```
1  function [valor_Arduino] = Guarda_Datos_Serie(numero_muestras)
2  close all;
3  valor_Arduino=zeros(1,numero_muestras);
4  delete(instrfind({'Port'},{'COM8'})); %El numero de puerto COM
       tiene que
5                        %ser modificado en funcion del
6                        %dispositivo conectado
7  puerto_serial=serial('COM8');
8  puerto_serial.BaudRate=115200;
9  warning('off','MATLAB:serial:fscanf:unsuccesfullRead');
10
11 fopen(puerto_serial);
12
13 contador_muestras=1;
14
15 while contador_muestras<=numero_muestras
16     a=fscanf(puerto_serial,'%d');
17     if (numel(a~=0)&&a(1)<2)  %Con esta condicion evitamos que
           matlab
18                  %no tome datos o tome datos que no se
19                  %encuentran en el rando esperado
20          valor_Arduino(1,contador_muestras)=a(1);
```

```
21          contador_muestras=contador_muestras+1;
22      end
23  end
24  fclose(puerto_serial);
25  delete(puerto_serial);
26
27  fid=fopen('DatosLeidos.txt','wt');  %Los datos leidos los guardo
        en un
28                      %archivo por si los necesitase
29
30  fprintf(fid,'%d',valor_Arduino);
31
32
33  fclose(fid);
34  end
```

Bibliografía

[1] Arduino. Arduino Mega, https://www.arduino.cc/en/Main/ArduinoBoardMega2560 . Última consulta 8 de Agosto de 2016.

[2] Arduino. Arduino Nano, https://www.arduino.cc/en/Main/ArduinoBoardNano . Última consulta 5 de Agosto de 2016.

[3] Esteso, M. P. Matlab + arduino: Serial port communication, https://geekytheory.com/matlab-arduino-serial-port-communication/ . Última consulta 14 de Agosto de 2016.

[4] Fairchild Semiconductos. 1n5221b - 1n5263b zener diodes, www.farnell.com/datasheets/2080977.pdf?_ga=1.3587635.2014530068.1470048205 . Última consulta 19 de Agosto de 2016.

[5] Fubarino. ChipKit Fubarino SD, http://fubarino.org/sd/index.html . Última consulta 6 de Agosto de 2016.

[6] Herrera, A. M. Números aleatorios. historia, teoría y aplicaciones. ingeniería y desarrollo, http://www.redalyc.org/articulo.oa?id=85200804 , 2000. Última consulta 15 de Agosto de 2016.

[7] Jones, R. V. Zener and avalanche breakdown/diodes, http://people.seas.harvard.edu/~jones/es154/lectures/lecture_2/breakdown/breakdown.html , Octubre 2001. Última consulta 4 de Agosto de 2016.

[8] Marsaglia, G. The marsaglia random number cdrom including the diehard battery of tests of randomness , http://stat.fsu.edu/pub/diehard/ . Última consulta 8 de Agosto de 2016.

[9] Moraleda, A. U., and Villalba, C. M. *Modelado y simulación de eventos discretos*. UNED, 2013.

[10] On Semiconductor. 1n5333b series, www.farnell.com/datasheets/111735.pdf?_ga=1.111107206.2014530068.1470048205 . Última consulta 19 de Agosto de 2016.

[11] Peres, Y. Iterating von neumanns procedure for extracting random bits, www.stat.berkeley.edu/~peres/mine/vn.pdf , 1992. Última consulta 31 de Agosto de 2016.

[12] Razavi, B. *Design of analog CMOS integrated circuits*. McGrawHill, 2001.

[13] Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., and Vo, S. A statistical test suite for random and pseudorandom number generators for cryptographic applications, https://www.google.es/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0ahUKEwjmwLDX1rTOAhVB1RQKHfsTAasQFggmMAE&url=http%3A%2F%2Fcsrc.nist.gov%2Fpublications%2Fnistpubs%2F800-22-rev1a%2FSP800-22rev1a.pdf&

usg=AFQjCNEKvUN79urhRAU-N9X6i8gvMZobbw&sig2=M0S5at55guE9pTKKcXzyxw&bvm=bv.129389765,d.d24&cad=rja
, Abril 2010. Última consulta 9 de Agosto de 2016.

[14] Universidad Central (Bogotá-Colombia). Ruido en comunicaciones, https://ruido.wikispaces.com/RUIDO+EN+COMUNICACIONES , Agosto 2011. Última consulta 4 de Agosto de 2016.

[15] van Zeghbroeck, B. Reverse bias breakdown, http://ecee.colorado.edu/~bart/book/book/chapter4/ch4_5.htm , Diciembre 2004. Última consulta 4 de Agosto de 2016.

[16] Villegas, R. M. C. Números aleatorios, http://numerosaleatorios2015.blogspot.com.es/2015/04/numeros-aleatorios.html , Abril 2015. Última consulta 4 de Agosto de 2016.

[17] Wikipedia. Número pseudoaleatorio, https://es.wikipedia.org/wiki/N%C3%BAmero_pseudoaleatorio , 2015. Última consulta 4 de Agosto de 2016.

[18] Wikipedia. Ruido de disparo, https://es.wikipedia.org/w/index.php?%20title=Ruido_de_disparo&oldid=87198980 , 2015. Última consulta 4 de Agosto de 2016.

[19] Wikipedia. Método de Montecarlo, https://es.wikipedia.org/wiki/M%C3%A9todo_de_Montecarlo , 2016. Última consulta 4 de Agosto de 2016.