UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

MÁSTER EN INVESTIGACIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES

Ataques contra generadores de números aleatorios físicos

Autor:
D. Jorge Javier Alarcón Cadena

Tutor:
D. Juan Carlos García Escartín

Valladolid, 21 de septiembre de 2018

TRABAJO FIN DE MÁSTER

TITULO:	**Ataques contra generadores de números aleatorios físicos.**
AUTOR:	**D. Jorge Javier Alarcón Cadena**
TUTOR:	**Dr. D. Juan Carlos García Escartín**
DEPARTAMENTO:	**Teoría de la señal**

TRIBUNAL

PRESIDENTE:	**D.**
SECRETARIO:	**D.**
VOCAL:	**D.**
PRESIDENTE SUPLENTE:	**D.**
SECRETARIO SUPLENTE:	**D.**
VOCAL SUPLENTE:	**D.**
FECHA:	**21 de septiembre de 2018**
CALIFICACIÓN:	

Resumen de TFM

El presente trabajo da cuenta de lo que son los números aleatorios y verdaderamente aleatorios, haciendo hincapié en sus diferentes aplicaciones y formas de conseguirlos. La generación de números verdaderamente aleatorios es fundamental para crear contraseñas seguras y preservar la privacidad de los datos. En muchos casos, estos números aleatorios proceden de fenómenos físicos impredecibles como el ruido térmico, sistemas electrónicos caóticos o fenómenos cuánticos, para ello será necesario hablar del ruido generado por los diferentes dispositivos pasivos y semiconductores, se describirán varios con especial atención a los que sean de interés en el trabajo. En este TFM se estudiarán dispositivos físicos basados en diversos principios y su comportamiento frente a ataques activos, como alteraciones en el voltaje de alimentación o inyección de radiofrecuencia. Se estudiarán y analizarán los puntos fuertes y débiles de cada implementación y se propondrán medidas para mejorar los sistemas existentes.

Palabras clave:
Números Aleatorios, TRNG, QRNG, Ruido, Araneus Alea II, Quantis USB

Abstract

The present work gives an approach to what are random numbers and truly random numbers, emphasizing their different applications and ways of achieving them. The generation of truly random numbers is essential to create secure passwords and preserve the privacy of the data. In many cases, these random numbers come from unpredictable physical phenomena such as thermal noise, chaotic electronic systems or quantum phenomena. It will be necessary to talk about the noise generated by the different passive and semiconductor devices, several will be described with special attention to those that are of interest in this document. In this TFM several physical devices will be studied based on different principles and their behavior against active attacks such as alterations in the power supply voltage or radiofrequency injection. The strengths and weaknesses of each implementation will be studied and analyzed, and measures will be proposed to improve existing systems

Keywords:
Random Numbers, TRNG, QRNG, Noise, Araneus Alea II, Quantis USB

Agradecimientos

Me gustaría comenzar mis agradecimientos reconociendo a quienes me han acompañado en mi paso por la Universidad de Valladolid.

Es grato para mí dedicar unas breves palabras para D. Juan Carlos García Escartín, quien me ha acompañado en el proceso de sacar adelante este trabajo. Su paciencia, atención y dedicación han sido fundamentales para la finalización de este documento.

A mis padres y hermanos; pilares fundamentales sobre los que me he apoyado durante todo este tiempo, su soporte y confianza han sido determinantes para alcanzar esta meta, a ellos mi gratitud infinita.

A mi esposa; mi compañera, amiga y confidente, quien ha sido participe de todo lo que he vivido durante mi paso por el Máster. Su apoyo incondicional es una muestra imborrable de su amor.

A mis abuelos; quienes partieron antes de nuestra despedida.

Por último, quiero agradecer a la Secretaría de Educación Superior, Ciencia, Tecnología e Innovación de Ecuador (SENESCYT) y a su programa de becas, por permitirme realizar este Máster.

A todos, muchas gracias.

“Las cosas no tienen que cambiar al mundo para ser importantes”.

– *Steve Jobs*

ÍNDICE

Contenido

ÍNDICE DE FIGURAS

1. Introducción

Hay que ser conscientes de que los ordenadores fueron creados para seguir determinadas instrucciones, las cuales son capaces de realizar mediante indicaciones que ya conocen de forma previa, es decir, que ya han identificado previamente para llevar a cabo la tarea que se le ha solicitado. A partir de aquí entran en escena los llamados números aleatorios, los cuales se basan en la medida de cuán impredecible es algo y, por otro lado, es fundamental destacar también el concepto de entropía, o lo que es lo mismo: la medida de desorden que posee un sistema.

Teniendo claro el significado de los términos anteriores, es fundamental señalar que la generación de números aleatorios se presenta como algo necesario dentro del campo de la seguridad informática y es aquí donde surge el problema: en el párrafo anterior hemos señalado que los ordenadores reciben órdenes, manipulando de esta manera aquello que ya conocen, pero no es posible pedir a un ordenador que busque algo desconocido, no puntual, ni específico. Buscando una solución a este problema fue cuando nacieron los generadores de números pseudoaleatorios (*PRNG)*, cuya base se trata de una función matemática en la que dada una entrada (*semilla o "seed"*), retorna una única salida, es decir, da lugar a un sólo posible resultado.

La particularidad del sistema es que, para obtener resultados altamente aleatorios será necesario proporcionarle un alto número de valores de entrada diferentes, para ello se utilizarán varios tipos de semillas, buscando una semilla lo suficientemente impredecible como para hacer funcionar correctamente el sistema. El uso de una u otra semilla determinará a la postre qué tan aleatorios serán los números de salida.

Por otro lado, cabe señalar aquellos fenómenos que son puramente impredecibles, siendo en éstos donde surgen los llamados Generadores de números verdaderamente aleatorios (*"TRNG", por sus siglas en ingles*), los cuales aportan una mayor robustez al sistema. Su implementación se puede hacer desde el empleo de un *hardware* externo, hasta por inclusión de un módulo de aleatoriedad pura en los microprocesadores, es decir, hay maneras muy variadas de incorporar estos generadores, como ejemplo se puede señalar el caso de unos investigadores de la Universidad de Cambridge [1], los cuales han descubierto una gran cantidad de cajeros y puntos de venta en los que las tarjetas de

crédito emplean números aleatorios con alto grado de predictibilidad, lo cual puede tener nefastas consecuencias para la seguridad del sistema, ya que los atacantes podrían descubrirlos fácilmente. Por lo tanto, es fundamental señalar el alto grado de importancia de la entropía en la generación de números aleatorios, ya que la implementación errónea de generadores de este tipo de números pone en peligro la seguridad de todo un sistema.

1.1. Justificación

En la sociedad actual la informática se ha convertido en un elemento fundamental, y esto se puede ver reflejado en la relevancia que presentan los ordenadores y dentro de los mismos los números aleatorios, los cuales representan un problema debido a que las computadoras tienen dificultades a la hora de generarlos por sí mismas. Por otro lado, cabe destacar la realización de un trabajo que utiliza el ruido, el cual es un elemento que suele considerarse problemático dentro del mundo informático. Esto consigue dar la visión de que, dependiendo del punto de vista, las cosas pueden no ser lo que parecen, ya que no todo ha de ser bueno o malo en función de las consideraciones previas. En este caso se utilizará al ruido como un elemento de análisis que nos permitirá discernir qué tan robustos pueden llegar a ser dispositivos generadores de números verdaderamente aleatorios. En último lugar cabe destacar la importancia que presentan estos números en la vida cotidiana, ya que abarcan procesos desde los más triviales y mundanos como juegos online hasta las múltiples aplicaciones que poseen con respecto al mundo de la ciberseguridad [2]. Por lo tanto, se puede destacar la importancia de éstos como un factor fundamental dentro de la sociedad contemporánea y es necesario poner el énfasis en la dificultad que presenta la generación de números realmente aleatorios a través de los diferentes sistemas electrónicos y de la posibilidad de interferir en su generación con la intención de predecirlos.

1.2. Objetivos

1.2.1. Objetivo general

a. Evaluar hasta qué punto los dispositivos Generadores de Números Verdaderamente Aleatorios son susceptibles a ruidos externos que afectan su correcto funcionamiento.

1.2.2. Objetivos específicos

a. Mostrar la utilidad de los números aleatorios.
b. Dar una correcta definición de los números verdaderamente aleatorios.
c. Explicar el cómo obtener números verdaderamente aleatorios a través de dispositivos diseñados para este efecto.
d. Presentar el funcionamiento de diferentes dispositivos para la obtención de números verdaderamente aleatorios.
e. Interferir los dispositivos generadores de números verdaderamente aleatorios con señales de ruido.
f. Comentar métodos de análisis y fenómenos relacionados.

1.3. Metodología

La metodología de este trabajo se basa inicialmente en una búsqueda exhaustiva de información en diversas fuentes documentales, con el fin de crear un grueso teórico que sirva para establecer un análisis correcto de la situación actual de la generación de números aleatorios, el ruido y sus efectos; además de las pruebas de análisis usadas para comprobación de aleatoriedad.

Las fuentes de documentación que aparecen en este trabajo se basan en una revisión exhaustiva de archivos de conocimiento. Esta investigación documental se constituye como una estrategia en la que se observa y se reflexiona sobre las realidades, teóricas o no, que repercuten en el estudio de la actual situación de la generación de números aleatorios a través de los diferentes dispositivos electrónicos, por lo tanto, en este trabajo nos basaremos en la documentación de diferentes fuentes bibliográficas, principalmente estadísticas y artículos.

Por otro lado, cabe resaltar que la parte fundamental de este TFM se basará en una metodología experimental, planteando un protocolo para la comprobación de los dispositivos, mediante el cual, a través de pruebas y experimentación obtendremos datos que nos permitirán llegar a una fase de análisis, utilizando baterías de pruebas de números aleatorios, para terminar con una reflexión crítica y objetiva de la información recolectada.

2. Contextualización

Para entender la importancia de los números aleatorios en nuestra sociedad, debemos hacer un repaso de su incidencia en nuestras costumbres y posteriormente en la necesidad que nos llevó a mejorar los procesos para generarlos.

En primera instancia, hay que situarse, aproximadamente, en el año 3.500 A.C, donde ya se practicaban juegos de azar con objetos hechos con hueso, los cuales pueden ser considerados precursores de los dados [3], [4]. Más adelante, en el siglo XVII, Antoine Gombauld (1607-1684) puso en tela de juicio el fundamento matemático del éxito y el fracaso en el mundo del juego, hasta que el teórico Pascal (1623-1662) resolvió el problema planteado por Gombauld, quién planteaba la pregunta de: << *¿Cuáles son las probabilidades de que salgan dos seises por lo menos una vez en veinticuatro lanzamientos de un par de dados?*>>. A los dos teóricos anteriores se les sumó Pierre de Fermat (1601-1655) y juntos dieron lugar al nacimiento de la primera revista académica sobre probabilidad [4].

Ya en el siglo XIX fue Pierre Simon (1749-1827), aunando las fórmulas y técnicas desarrolladas hasta el momento, formuló la primera teoría general de la probabilidad, la cual fue aplicada con éxito tanto a los juegos de azar como a la búsqueda de soluciones analíticas en lo que respecta a problemas de naturaleza no determinística [4].

El nacimiento de los *números aleatorios* puede situarse en la década de los cuarenta con la creación del método llamado *Simulación de Montecarlo* [4], [6]. Como pioneros en este ámbito se puede destacar a Ulam, Lehmer, Metrópolis o Neumann, siendo este último quien, aparentemente, infirió el potencial de los ordenadores para tratar problemas estocásticos en 1945 [4].

La publicación de El Método de Montecarlo (*The Montecarlo method)*, llevada a cabo por Metropolis y Ulam en 1949, se posiciona como el inicio de la historia oficial de este método [4]. Fueron dos años después cuando Lehmer propuso el generador lineal de congruencia, el cual se extendió rápidamente, siendo muy empleado para la generación de números aleatorios. En la actualidad este método tiene cuestionamientos respecto de su efectividad [3].

Antes de la llegada de los ordenadores, los números aleatorios eran generados por dispositivos físicos, un ejemplo sería el propuesto por Kendall y Babington-Smith en 1939, quienes publicaron 100.000 dígitos aleatorios que se obtuvieron con un disco giratorio iluminado por una lámpara [6]. Por último, cabe destacar que a pesar de que el Método de Montecarlo comenzó a ser utilizado con ruletas y dados en los problemas de difusión de neutrones, su actual auge se debe a que hoy en día se utilizan números aleatorios generados por ordenadores [4].

3. Los Números Aleatorios

Los números aleatorios se han convertido en un elemento fundamental de la vida cotidiana, esto se debe a que la aleatoriedad es aplicable a diferentes ámbitos como puede ser el caso de la criptografía o la coordinación de las redes de ordenadores. Hay que tener en cuenta que algunas aplicaciones precisan de una pequeña cantidad de números aleatorios, por lo que se emplean métodos manuales y mecánicos para generarlos, siendo el caso del lanzamiento en el aire de una moneda o del lanzamiento de un dado, no obstante, hay otra generación más compleja referida a aquellas cifras aleatorias que son generadas por ordenador [4].

Se entiende como número aleatorio todo aquel obtenido al azar; en otras palabras, todo conjunto de números que tengan la misma probabilidad de ser elegidos y en los que la elección de uno no dependa de otro, es decir, sin correlación alguna.

En el caso de una secuencia de números aleatorios generada mediante una fuente física o método matemático deben darse unas características básicas:

a. Periodo máximo: Los generadores presentan un periodo cíclico, es decir, lo ideal es que cada uno de los elementos del conjunto aparezcan exactamente una vez antes de repetir un ciclo completo. Es importante destacar que tanto el periodo como las propiedades de la secuencia no deberían tener dependencia del valor inicial.

b. Secuencias no correlacionadas: En la sucesión de números aleatorios ninguna subsecuencia debe estar relacionada con otra.

c. Uniformidad: Los números aleatorios pueden modelarse mediante la función de densidad de probabilidad de la variable aleatoria uniforme. En el caso de que A fuera la fuente generadora de la sucesión de números aleatorios, se vería de la siguiente manera:

$$A \sim U(r, a = 0, b = 1) \text{ siendo } A = X/m.$$

Ecuación 1

d. Independencia estadística y equiprobabilidad: Esto quiere decir que la probabilidad de que un número específico aparezca en la sucesión ha de ser la

misma para cada uno de los números que conforma el conjunto. A mayores, es importante señalar que la aparición de cualquiera de los números no puede implicar, ni excluir la aparición de cualquier otro número que conforme la sucesión.

En segundo lugar, cabe destacar la existencia de otro tipo de cifras: los números pseudo-aleatorios o PRNGs. Estos números pueden definirse como aquellos que producen números aleatorios a partir de un algoritmo determinista y aunque cualquier secuencia que se genere a través de algoritmos no puede ser realmente aleatoria, para muchas aplicaciones el empleo de estos números aparentemente aleatorios es suficiente. No obstante, su principal problema es que no son impredecibles del todo y esto es un verdadero inconveniente ya que se suelen emplear en cifrado, dándose la necesidad de generar números verdaderamente aleatorios.

Dentro de la utilidad de estos números pueden señalarse varios campos: estadística, diseño experimental o modelado por ordenador, entre otros. Los generadores poseen una amplia utilidad y destaca, especialmente, el uso del llamado sistema *Montecarlo* con aplicaciones tales como hallar volúmenes o áreas que se encuentran encerradas en una gráfica y cuyas integrales son prácticamente irresolubles [4]. La explicación es que la generación de puntos basados en números aleatorios permite hacer una aproximación con respecto al volumen total, encerrándolo en un cuadrado o en un cubo. Hay tres pasos determinantes para explicar esta metodología:

1. Primero, el software coloca el modelo dentro de un volumen conocido.
2. En segundo lugar, se da la generación de un punto aleatorio en el interior del volumen mencionado con anterioridad y se registra si este punto "ha caído" dentro o fuera del modelo. Esta segunda operación presenta un gran número de repeticiones, llegando a ser miles o millones, permitiendo de esta manera obtener un registro de cuántos puntos han quedado fuera y dentro.
3. Por último, es importante señalar que la probabilidad de que caiga dentro es proporcional al volumen del modelo, dando lugar a la conclusión de que la proporción de puntos que han caído dentro es la misma que la que presenta el volumen que ocupa el modelo dentro del cubo.

Como campos importantes para la aplicación de esta metodología se puede señalar en primer lugar la programación de juegos, en especial en aquellos donde el azar resulta fundamental, pero también aquellos en los que se deba garantizar fiabilidad, como es el caso de las apuestas económicas, donde el algoritmo más recurrente es el llamado Fisher-Yates; este algoritmo consiste en elegir diferentes segmentos de un vector y eliminándolos del mismo para, posteriormente, colocarlos en un vector vacío, de esta manera el proceso se repetirá hasta que el vector inicial se quede sin ningún elemento.

El campo de la criptografía también posee especial relevancia, ya que la seguridad informativa dentro del comercio electrónico tiene un peso capital [7]–[9]. Los servicios de criptografía hacen uso de números aleatorios y éstos se aplican en los diferentes servicios de seguridad, dentro de los cuales podemos encontrar mecanismos de firma digital, de confidencialidad o de autentificación, donde las claves secretas se generarán a partir de números aleatorios [4].

La siguiente cuestión para tener en cuenta es el cómo se obtienen estos números aleatorios, cuyo proceso más frecuente se denomina recolección de entropía [10]. En primer lugar, hay que tener en cuenta que los generadores de números aleatorios miden procesos físicos cuya predicción resulta compleja y emplean los resultados obtenidos para generar una secuencia de números aleatorios, pero también pueden basarse en valores impredecibles a los que se puede tener acceso a través de un software dentro de un ordenador. En segundo lugar, cabe señalar que dentro de la entropía se encuentran las fuentes estándar a las que el sistema operativo puede tener acceso, siendo el caso de los datos de la tarjeta de sonido, los tiempos de interrupciones, los datos interactivos del usuario, sistemas caóticos, parámetros biométricos, entre otros. Estos principios son usados por los Generadores de números verdaderamente aleatorios, denominados TRNG ("*True Random Number Generator"*), por sus siglas en inglés [7].

Los números verdaderamente aleatorios y los generadores de números aleatorios físicos no deterministas están incrementando cada vez más su importancia, ya que debido al principio de Kerckhoffs la definición de los generadores de números aleatorios adecuado para criptografía debe incluir que, aunque cada detalle sobre el diseño del generador sea conocido, este aun así debe proporcionar bits totalmente impredecibles. A diferencia de los Generadores de números pseudo-aleatorios (PRNG), los generadores de

números aleatorios físicos obtienen su aleatoriedad de procesos fundamentalmente no deterministas, siendo dispositivos independientes del ordenador, usualmente conectados a él a través de interfaces USB o PCI [7].

Tras la explicación anterior puede parecer que los TRNGs son mejores que los métodos deterministas, pero hemos de destacar también sus diferentes inconvenientes, entre los que se pueden señalar la complicación de añadir un dispositivo externo, la difícil detección de fallos dentro del software, la complicación de dar argumentos convincentes de que los números generados son realmente aleatorios, o su tasa de generación limitada; es decir, la producción de números aleatorios a un ritmo menor. A mayores también se dan fenómenos físicos, ya que cuando el sistema muestrea a una velocidad elevada, no se posee el tiempo suficiente para que el sistema cambie y debido a ellos los números generados podrían no ser totalmente independientes.

Otras propuestas han sido plateadas para mejoras los sistemas físicos de generación de números aleatorios, uno de ellos puede encontrarse en [11], un Generador de números aleatorios digital (DRNG) que usa ruido de alta frecuencia de estadística estacionaria de fuentes independientes para obtener las semillas necesarias para la producción de números aleatorios, también disponemos de los Generadores de números aleatorios cuánticos ampliamente tratados en [8].

4. El Ruido

La importancia del ruido en nuestro trabajo es primordial, ya que pretendemos perturbar la correcta generación de números verdaderamente aleatorios, para ello introduciremos ruido a los dispositivos generadores. Por esta razón debemos entender qué es el ruido y cómo se comporta.

Se considera ruido a toda señal anómala o perturbadora en un sistema transmisor que provoca que la información se distorsione o que no llegue de forma nítida. La composición del ruido se basa en una mezcla aleatoria de longitudes de onda y carece de información alguna [12].

Entendemos por ruido eléctrico todas aquellas señales eléctricas no deseadas unidas a una señal principal y que pueden alterarla, produciendo efectos perjudiciales, o lo que en términos físicos sería la fluctuación aleatoria de una magnitud eléctrica que puede llegar a enmascarar a la señal principal. No obstante, hay que ser conscientes de que para que el ruido sea perturbador ha de poseer una energía indeseada de la misma categoría que la señal prioritaria y tanto el ruido como la señal principal deben compartir una misma banda de frecuencia.

Otra de las características del ruido en canales de comunicación es que es aditivo [13], es decir, habrá una variación en la amplitud de la señal que se transmite, dando lugar a error, lo cual podría solucionarse limitando la amplitud, pero no hemos de olvidar que el ruido no deja de ser una señal aleatoria en la que tanto la amplitud como la fase también lo son.

La representación de este fenómeno es la de un vector que se suma a la señal principal, derivando en una portadora cuya fase y amplitud son variables, dificultando de esa manera la correcta transmisión de la información.

Algunos tipos de ruido se mencionarán a continuación:

4.1. Ruido externo

El ruido externo o de interferencias es aquel que se produce en un punto del sistema como resultado del acoplamiento eléctrico con otro punto distinto del mismo sistema, o de otros ajenos que pueden ser tanto naturales como construidos por el ser humano. Este tipo de ruido puede ser aleatorio, intermitente o periódico e incluye ruido artificial, espacial o atmosférico, el cual es producido por la estática que tiene la atmósfera y que se manifiesta a través de diferentes fenómenos naturales como los rayos. A mayores, cabe señalar que el ruido de interferencias puede reducirse minimizando el acoplo eléctrico o reorientando las conexiones y los diferentes componentes [14].

4.2. Ruido térmico

El ruido térmico se da debido al movimiento browniano de los electrones que van asociados a un conductor, fenómeno observado inicialmente por el científico Robert Brown en partículas de granos de polen, pero a nivel electrónico sería reconocido en 1927 por J. B. Johnson de Bell Telephone Laboratories [12]. En este caso será el movimiento aleatorio de los electrones el que producirá las fluctuaciones en la tensión, las cuales se medirán en sus extremos [15]. También puede describirse como fuente de tensión o corriente con una polaridad que será arbitrada y, a mayores, se ha de tener en cuenta que el espectro es proporcional a la temperatura absoluta, o lo que es lo mismo: cuanto menor sea la temperatura, menor será su ruido térmico. El ruido térmico se basa en las ecuaciones 2 y 3, que se presentan a continuación:

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

Ecuación 2

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

Ecuación 3

donde k es la constante de Boltzman, Δf es el ancho de banda del canal en Hertz y T es la temperatura absoluta expresada en Kelvin.

4.3. Ruido de parpadeo

El ruido de parpadeo también llamado ruido *"flicker",* ruido de corriente, ruido de exceso o ruido de baja frecuencia es aquel cuya densidad espectral de potencia es proporcional al inverso de la frecuencia $(1/f)$. En transistores MOS este tipo de ruido se da debido a los dopantes que liberan y capturan portadores aleatoriamente, y de esta manera la constante de tiempo que va asociada al proceso resulta en una señal de ruido que posee una energía asociada a bajas frecuencias. Para reducir este tipo de ruido lo correcto sería aumentar el área o trabajar con frecuencias más altas y, además, hay que tener en cuenta que en el ruido *"flicker"* es complejo medir la frecuencia media [16], lo cual se puede observar en las ecuaciones 4 y 5.

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

Ecuación 4

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

Ecuación 5

Donde $K \approx 10^{-25}V^2f$, C_{ox} es la capacidad del oxido, W es el ancho del canal, L es el largo del canal y g_m es la transconductancia del transistor [17].

Este tipo de ruido está presente todos los dispositivos activos y pasivos, dependiendo su origen del tipo de sistema [18], y será particularmente relevante en nuestro trabajo debido a que se espera que las perturbaciones a las que serán sometidos los dispositivos TRNG lo produzcan, afectando a su funcionamiento general.

En [16] podemos encontrar una explicación muy interesante respecto a los efectos que tienen las variaciones de voltaje sobre las resistencias, en la que se detalla como al mantiene una fuente de corriente constante a niveles dc, la fluctuación en la resistencia

$r(t)$ provocará una fluctuación en el voltaje $v(t)$ según la expresión: $v(t) = I\, r(t)$, dando como resultado que la densidad espectral de potencia de la fluctuación del voltaje sea: $\overline{Sv(f)} = I^2\, \overline{Sr(f)}$, siendo $Sr(f)$ la densidad espectral de potencia de la fluctuación de la resistencia. Aunque esto no explica el ruido como tal, si lleva a examinar a la resistencia como fuente de fluctuación, al depender la resistencia de la densidad y movilidad de los portadores de carga, se llega a la conclusión de que el ruido $1/f$ se deriva del número o fluctuaciones de movilidad.

4.4. Ruido de disparo

El ruido de disparo o ruido "*shot*" es aquel causado por oscilaciones aleatorias de corriente eléctrica a través de un conductor. Este ruido es causado debido a que la corriente se transfiere en cargas discretas y se origina al darse el movimiento de partículas cargadas, como pueden ser los electrones u otras diferentes, a través de una unión en cualquier conductor, inclusive en aquellos donde la carga no esté correctamente localizada [18]. Su estadística obedece por tanto a una distribución de Poisson. La contribución a este tipo de ruido puede ser realizada tanto por fotoelectrones como por electrones generados térmicamente (señal oscura, "*dark*") [19].

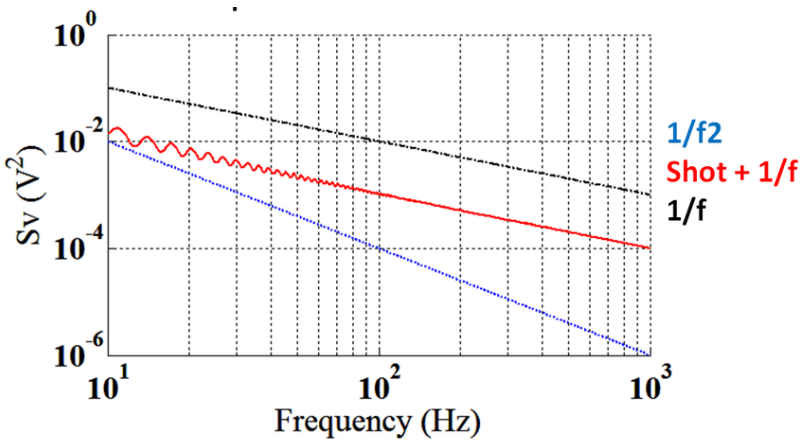


Figura 1: Interacción del ruido shot con la frecuencia 1/f. [20]

4.5. Ruido de tiempo de tránsito

El ruido de tiempo de tránsito es aquel que produce una variación aleatoria e irregular. Esto se explica en que cualquier cambio en una corriente de portadores conforme pasa

desde una entrada a la salida de un mismo dispositivo puede producir este fenómeno [12] [21].

4.6. Ruido de transición

Esta tipología es causada por desfases entre las tensiones y corrientes que se encuentran en el interior de un dispositivo y se debe a la cantidad de tiempo que los portadores tardan en franquearlos. Estos desfases aumentan con el incremento de la frecuencia, provocando de esta manera que la impedancia que entra en el mecanismo disminuya, un aumento de la retroalimentación positiva del ruido y una potencia adicional del mismo.

4.7. Ruido de amplitud

Este fenómeno produce un cambio en los niveles de frecuencia de forma repentina, lo cual es causado por contactos sucios con niveles de resistencia variable o amplificadores defectuosos.

4.8. Ruido de impulsos

Este tipo de ruido llamado también ruido de agujas es el culpable de los errores en comunicación de datos y la principal forma de reconocerlo es un “click” en las comunicaciones de voz. Lo que provoca este fenómeno es un error en ráfaga donde la variación del impulso puede ser tanto de 1 o 2 bits, como de centenas de ellos, lo cual es dependiente del índice de transferencia de la información [12] [22].

4.9. Ruido rosa

Éste consiste en una señal que posee un espectro de frecuencias cuya densidad espectral de potencia es proporcional a su igual en frecuencia, ya que su energía en lo que respecta a la frecuencia disminuye en 3 dB por octava, permitiendo de esa manera que cada banda tenga la misma frecuencia [23].

4.10. Ruido de intermodulación

En este caso se produce la intermodulación de dos tipos de líneas independientes, las cuales pueden estar en un tipo de frecuencia que se diferencia de ambas entradas, o pueden caer dentro de una banda de una tercera señal. Suele aparecer cuando el sistema de frecuencias es no lineal, pudiendo provocar de esa manera el surgimiento de nuevas frecuencias, las cuales se sumarán o se restarán a las originales generando frecuencias que previamente no existían y produciendo una distorsión de la señal real.

4.11. Ruido en canales telefónicos

Este tipo de ruido puede subdividirse en dos categorías según [23]:

- ***Diafonía*:** Se produce debido a las interferencias ocasionadas por pares de hilos telefónicos próximos, es decir, es fenómeno en el que una señal transitante perturba a otra que discurre de forma paralela.
- ***Eco*:** La señal que aquí se produce presenta las mismas características que la señal original, pero de forma atenuada y retardada.

4.12. Interferencia

La interferencia es unos de los puntos importantes en el presente trabajo, interferir es sinónimo de "perturbar o estorbar", que es lo que se buscará durante el desarrollo de este documento. Al agregar a los dispositivos motivo de análisis señales que puedan perturbar su correcto funcionamiento, a estas alteraciones las denominaremos perturbaciones. La interferencia eléctrica es producida cuando las señales provenientes de una fuente producen frecuencias fuera del ancho de banda designado para ellas, para interferir con las señales generadas por otra fuente [12].

5. Dispositivos generadores de números verdaderamente aleatorios a ser estudiados.

Como se ha detallado anteriormente en este documento, existen una gran variedad de dispositivos destinados a la generación de números verdaderamente aleatorios, que haciendo uso de variados principios para la obtención de entropía logran muy buenos resultados de generación de datos aleatorios.

De entre estos dispositivos se han seleccionado a tres que obedecen a diferentes procesos de generación, con la finalidad de analizarlos, un generador físico, un generador cuántico y un generador digital de números aleatorios.

Los dispositivos que se explicarán se listan a continuación:

- Araneus Alea II (TRNG)
- Quantis-USB-4M (QRNG)
- Intel Digital Random Number Generator (DRNG)

A continuación, se detallan ciertas particularidades de cada uno de ellos:

5.1. Araneus Alea II TRNG

El llamado comúnmente Alea II es un compacto generador de números verdaderamente aleatorios (TRNG), conocido también como generador de números aleatorios de hardware (HWRNG), dispone de una interfaz USB. Este generador está diseñado para aplicaciones criptográficas como la generación de claves o firmas DSA, pero también puede utilizarse en otras aplicaciones como loterías, juegos o muestreo aleatorio. Los datos aleatorios son leídos desde el dispositivo en formato binario usando un empaquetado de 64 bits [24].

En lo que respecta a su funcionamiento, Alea II emplea una polarización inversa en una unión p-n para generar ruido blanco gaussiano de banda ancha y este ruido se amplifica y digitaliza a través de un conversor analógico a digital (A/D). Los bits de salida que no han sido procesados por el convertidor A/D serán procesados por un

microprocesador incrustado con el objetivo de combinar la entropía de múltiples muestras para generar un bit de salida final, dando lugar a un flujo de bits aleatorios que estarán prácticamente libres de sesgo y correlación [25].

Una de las principales limitaciones de este dispositivo es su baja tasa de generación de datos, lo cual limita la recopilación de datos aleatorios.

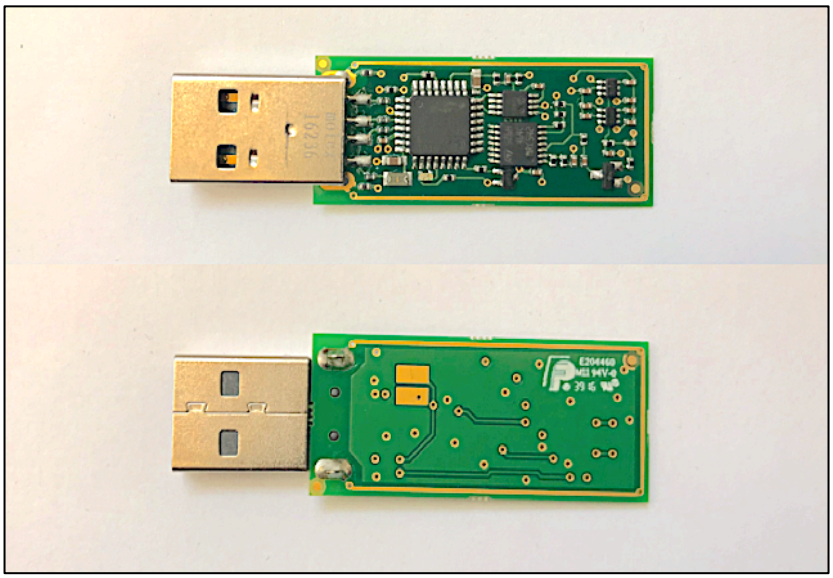


Figura 2: Apariencia física interna del USB TRNG Araneus Alea II

Especificaciones técnicas: [24]

- Tasa de datos aleatoria: 100 $kilobits\ /\ segundo$
- Interfaz: USB de alta velocidad
- Consumo de energía: menos de 250 mW
- Dimensiones: 59 x 21 x 12 mm incluyendo el conector USB

5.2. Quantis-USB-4M QRNG

El Quantis-USB-4M es un generador de números aleatorios cuánticos (QRNG). La empresa ID Quantique fue una de las primeras en aprovechar los fenómenos cuánticos como fuente de entropía en un producto comercial [26] luego de que Stefanov et al. propusieran los principios científicos y diseño de un generador de números aleatorios cuántico en el año 2000 [27].

Figura 3: Apariencia física externa del dispositivo Quantis-USB-4M

Este generador, destinado a cualquier operación que requiera criptografía o entropía y aleatoriedad, resuelve el problema de la necesidad de una aleatoriedad real con generación de datos a gran velocidad (~4Mb/s). Como fuente principal para generar aleatoriedad emplea la óptica cuántica, es decir, emplea un proceso físico en lugar de uno informático. Básicamente los eventos aleatorios se realizan mediante la elección de fotones individuales entre las dos salidas de un divisor de haz [27]. Esta metodología suele basarse en el empleo de un transductor que convierte el aspecto físico en señal eléctrica, por otro lado, hace uso de un amplificador y otros circuitos electrónicos de tal forma que aumenta la amplitud de las fluctuaciones y, por último, un conversor analógico a digital para transformar la señal en un número digital, el cual suele ser un dígito simple y binario, es decir, 0 o1. Tras muestrear de forma repetida la señal alternativa variable se obtendrá una serie de números aleatorios [27], [28].

El principio del funcionamiento de Quantis puede verse en la figura 4, los fotones se envían uno por uno a un espejo semitransparente y detectado. De esta manera las operaciones de reflexión y transmisión se asocian a los valores de bit 0 y1. Pero en realidad el dispositivo prescinde de los espejos usando la aleatoriedad en la dirección de emisión de un led.

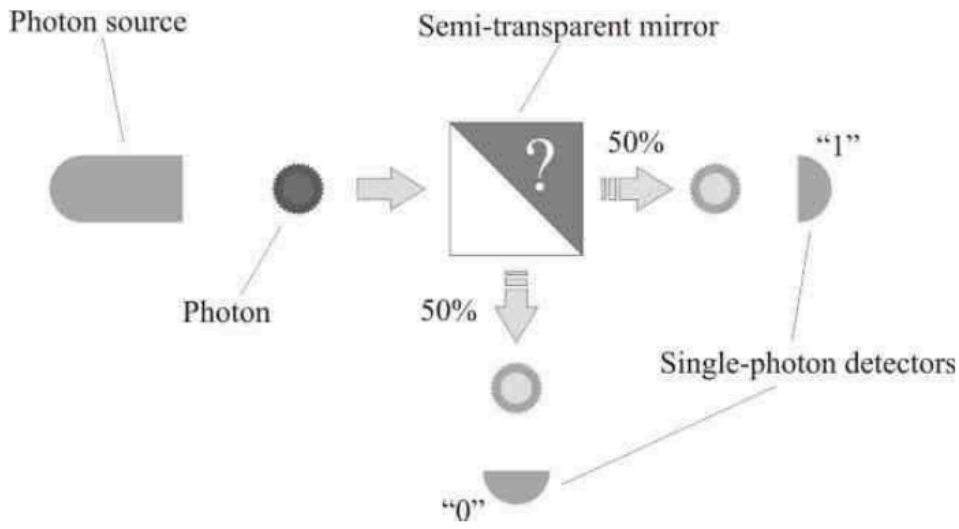


Figura 4: Principio de funcionamiento del dispositivo Quantis. [9]

Especificaciones técnicas: [29]

- Velocidad de bits aleatoria: 4 $Mbit/s \pm 10\%$ (Quantis-USB-4M)
- Contribución de ruido térmico: $< 1\%$ (Fracción de bits aleatorios derivados por ruido térmico)
- Temperatura de almacenamiento: $-25\ to\ +\ 85°C$
- Dimensiones: 61 mm x 31 mm x 114 mm
- Especificación USB: 2.0
- Requerimientos: PC con conexión vía puerto USB
- Batería: a través de puerto USB
- Permite la verificación de estado en vivo
- Altamente resistente a las perturbaciones ambientales
- Alta velocidad de bits de hasta 16 Mbits / s para tarjetas PCI Express
- Asequible, compacto y seguro
- Compatible con todos los principales sistemas operativos

5.3.Intel Digital Random Number Generator (DRNG)

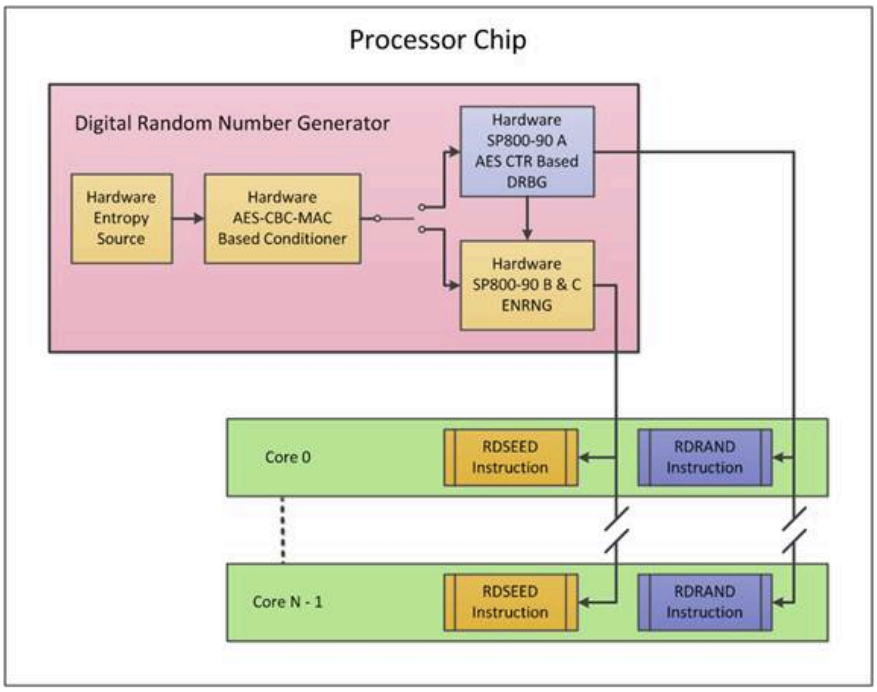


Figura 5: Diseño del generador digital de números aleatorios[30]

Este método para la generación de números aleatorios de forma digital presenta un enfoque que se basa en la implementación en el hardware y puede utilizarse con instrucciones que a su vez están complementadas a las del procesador Intel 64. A mayores, este software puede emplearse en todos los niveles omitiendo las llamadas pilas de software intermedio, la manipulación del sistema operativo o las bibliotecas [30].

Es importante señalar que, gracias a su construcción en cascada, este modelo presenta un PRNG criptográficamente seguro (CSPRNG), el cual permite la obtención de números aleatorios, con una apariencia verdaderamente aleatoria y que presentan un nivel de resistencia de ataque computacional bien definido.

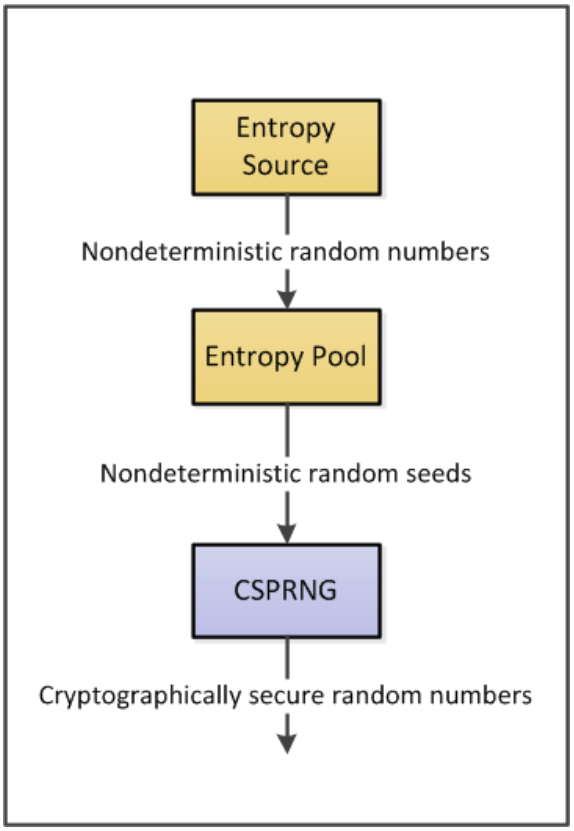


Figura 6: Generador de números aleatorios de Cascade Construction [30]

Especificaciones técnicas:

- RNG de construcción en cascada
- Empleo del Intel 64 RDRAND y RDSEED en el DRNG
- Posee dos salidas paralelas: un generador de bits aleatorio determinístico (DRBG) y un generador de números aleatorios mejorado y no determinista (ENRNG)
- Presenta una fuente de entropía (ES)
- El acondicionador toma pares de muestras de entropía sin procesar y las reduce a una sola muestra de entropía de 256 bits, y para ello emplea encriptación AES-CBC-MAC
- El DRBG utilizado es el CTR_DRBG con un cifrado de bloques AES
- Robustez y autovalidación
- Pruebas de salud en línea (OHT)
- Autopruebas incorporadas (BIST)

6. Programas de comprobación de aleatoriedad de los datos obtenidos

Las baterías de pruebas son conjuntos de pruebas estadísticas que tienen como finalidad comprobar la calidad de los generadores de números aleatorios [31]. Las pruebas están diseñadas para procurar la falla no para determinar la aleatoriedad de los datos analizados. Esto quiere decir que las baterías no garantizas que los números analizados sean completamente aleatorios, solo que han pasado varias pruebas diseñadas para alcanzar un nivel de confianza superior al 95%.

Para analizar la información recopilada durante la generación de datos aleatorios se emplearán las siguientes baterías de pruebas:

6.1. Prueba *Ent*:

"*Ent*" es un programa que aplica varias pruebas a las secuencias de bytes que se encuentran almacenadas en los archivos para, posteriormente, otorgar un resultado. La utilidad de esta modalidad es la evaluación de los generadores de números pseudoaleatorios en diversas aplicaciones entre las que podemos destacar los algoritmos de compresión, aplicaciones de cifrado y muestreo estadístico.

"*Ent*" realiza varias pruebas utilizando una secuencia de bytes en *infile*, dando como resultado una secuencia de salida estándar. A continuación se explican las pruebas que se realizan en esta batería:

- ***Entropía:***
  Se ha calcula primeramente la entropía de Shannon, la cual es la densidad informativa que contiene el archivo en cuestión, expresada en número de bits por carácter.
- ***Comprensión óptima:***
  En segundo lugar, se encuentra el proceso de compresión, esta prueba procurará disminuir el tamaño del archivo, logrando ser prácticamente cero en una sucesión aleatoria.

- ***La distribución de χ^2 (Chi cuadrado):***

  Luego se encuentra la prueba de χ^2, siendo la prueba más empleada para analizar la aleatoriedad de los datos. Esta operación consiste en calcular la secuencia de bytes que hay en un archivo, su resultado se indica con un número absoluto y un valor en porcentaje para indicar con qué frecuencia podría exceder el valor calculado la secuencia verdaderamente aleatoria.

- ***Valor medio aritmético de los bytes:***

  Más adelante se encuentra el valor aritmético, el cual es simplemente la suma de todos los bytes del archivo para, posteriormente, dividirlo entre la longitud de este. En este caso se considera que para que un dato sea prácticamente aleatorio debe aproximarse a la cifra 127.5.

- ***Valor de Montecarlo para Pi:***

  Para saber en qué consiste el valor de Montecarlo para π (Pi) hay que ser conocedores de que cada secuencia sucesiva de seis bytes emplea coordenadas X y Y de 24 bits dentro de un cuadrado. Si la distancia del punto aleatorio que se ha generado resulta más pequeña que el círculo que se encuentra inscrito dentro de ese cuadrado, se puede decir que la secuencia de seis bits es un "golpe". Este porcentaje de golpes puede ser empleado para calcular el valor de π.

- ***Coeficiente de correlación en serie:***

  Por último, cabe destacar el coeficiente de correlación en serie, que determina en qué medida cada byte del archivo depende del byte anterior. Hay que tener en cuenta que para las secuencias aleatorias este valor será cercano a 0.

En la siguiente gráfica se puede apreciar un ejemplo de aplicación de Ent a un banco de números aleatorios:

```
1.  jorgealarcon@Michelangelo ~/TFM/Alea II $ ent Aleatorios_AleaII_1GB.txt
2.  Entropy = 8.000000 bits per byte.
3.
4.  Optimum compression would reduce the size
5.  of this 1000000000 byte file by 0 percent.
6.
7.  Chi square distribution for 1000000000 samples is 265.81, and randomly
8.  would exceed this value 50.00 percent of the times.
9.
10. Arithmetic mean value of data bytes is 127.4990 (127.5 = random).
11. Monte Carlo value for Pi is 3.141739669 (error 0.00 percent).
12. Serial correlation coefficient is 0.000006 (totally uncorrelated = 0.0).
```

Figura 7: Prueba Ent a un banco de números aleatorios

6.2. Prueba RNG:

Conocido como *"rngtest"*, su utilidad es la verificación de la aleatoriedad de datos, para lo que emplea las llamadas pruebas del $FIPS\ 140-2$, el cual puede considerarse un programa de validación de módulos criptográfico, este programa es el resultado de la unión entre el Instituto Nacional de Estándares y Tecnología (NIST) [32] y el Centro de Seguridad de las Telecomunicaciones (CSE) del Gobierno de Canadá. El funcionamiento de "*rngtest"* se da en bloques de 20000 bits a la vez, para ello toma la entrada "*stdin"* para, posteriormente, enviar las estadísticas a "*stderr"* (dónde también se envían los errores) y de forma opcional pueden repetirse los bloques que superaron las pruebas FIPS a "*stdout"*.

Rngtest podrá desechar los primeros 32 bits de datos cuando trabaje en el llamado "*pipe mode"* y empleará los siguientes 32 para iniciar las pruebas FIPS, pero hay que tener en cuenta que estos bits no están probados por probabilidad.

La versión de software utilizada de *"rgntest"* será la disponible en el repositorio rng-tools, una versión de Henrique de Moraes Holschuh de 2004 [33]. En la siguiente gráfica se puede observar un ejemplo del programa en funcionamiento.

```
223. jorgealarcon@Michelangelo ~/TFM/Aleatorios $ cat Aleatorios_Quantis_1GB.dat
     |rngtest
224. rngtest 4
225. Copyright (c) 2004 by Henrique de Moraes Holschuh
226. This is free software; see the source for copying conditions.  There is NO
     warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
227.
228. rngtest: starting FIPS tests...
229. rngtest: entropy source drained
230. rngtest: bits received from input: 8000000000
231. rngtest: FIPS 140-2 successes: 399693
232. rngtest: FIPS 140-2 failures: 306
233. rngtest: FIPS 140-2(2001-10-10) Monobit: 41
234. rngtest: FIPS 140-2(2001-10-10) Poker: 27
235. rngtest: FIPS 140-2(2001-10-10) Runs: 109
236. rngtest: FIPS 140-2(2001-10-10) Long run: 129
237. rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
238. rngtest: input channel speed: (min=500000000.000; avg=4877715369.963;
     max=0.000)bits/s
239. rngtest: FIPS tests speed: (min=14.181; avg=60.868; max=61.927)Mibits/s
240. rngtest: Program run time: 127055357 microseconds
```

Figura 8: Ejemplo de rngtest

6.3. Prueba Dieharder:

La llamada "Prueba de Dieharder", desarrollado por George Marsaglia [34], consiste en una batería de pruebas destinadas a generadores de números aleatorios, cuya finalidad es comprobar que las secuencias generadas cumplen los requisitos que debería de tener una secuencia aleatoria, por tanto, su principal objetivo es procurar que los generadores no pasen las pruebas de aleatoriedad.

En la mayoría de estas pruebas si el archivo de entrada posee bits aleatorios y verdaderamente independientes, las operaciones otorgan un valor p, el cual ha de ser uniforme $[0, 1)$.

La forma en la que se obtiene p es la siguiente: $p = F(\mathrm{x})$, en el que F representa la función densidad de probabilidad de la variable aleatoria x, la cual suele estar normalizada. No obstante, es conveniente señalar que en este caso F representa una aproximación asintótica, por lo que para intervalos de confianza que ronden las asíntotas y que superen el 95% de confianza los valores de p dejarían de ser fiables.

Para el desarrollo del presente trabajo se está usando Dieharder versión 3.31.1 para Linux [31].

Dentro de la prueba de Dieharder hay diferentes tipologías, varias de las cuales se describirán a continuación:

6.3.1. Batería de pruebas de aleatoriedad de Marsaglia:

A continuación se describen varias de las pruebas realizadas en esta batería de pruebas según [35]:

- ***Prueba de separación de cumpleaños:*** Esta prueba se trata de elegir m cumpleaños en un año de n días y, después, enumerar las separaciones entre los cumpleaños seleccionados. Los intervalos que se repitan deberían distribuirse mediante la llamada distribución de Poisson y, además, se realizará la evaluación de chiqs y el valor de p probando la relación con la hipótesis nula.

- ***Prueba de las cinco permutaciones que se superponen:*** Esta operación se basa en buscar una secuencia de un millón de cifras enteras y aleatorias de 32 bits. Hay que tener en cuenta que cada grupo de cinco números con las características nombradas previamente puede estar en una de las 120 colocaciones que ofrece la prueba, las cuales difieren unas de otras. A medida que se contempla la gran cantidad de transiciones de estado, es cuando se hacen las cuentas acumulativas de los números de aparición de cada estado. Por último, es importante señalar que los 120 ordenamientos factibles deben ocurrir con la misma probabilidad estadística.

- ***Prueba de rango binario de matrices 31x31:*** Esta prueba consiste en coger los 31 bits que estén situados más a la izquierda de los 31 números aleatorios enteros de la secuencia de prueba para, de esa manera, generar una matriz binaria de 31x31 en el rango determinado de 0,1. Hay que ser conscientes de que el rango puede oscilar entre 0 y 31, pero los rangos que son menores de 28 son anómalos.

- ***Prueba de rango binario de matrices 32x32:*** En este caso se creará una raíz aleatoria binaria de 32x32 y cada fila supondrá un número aleatorio entero de 32 bits, nuevamente el rango estará determinado, como ya se señaló en el caso anterior. En este supuesto el rango puede oscilar entre 0 y 32, siendo las filas de 29 las que resultan anómalas. Tanto en este caso como en el previo los rangos fueron encontrados para 40.000 matrices aleatorias.

- ***Prueba de rango binario para matrices de 6x8:*** Este método consiste en coger un bit específico de cada uno de los seis números aleatorios y enteros de 32 bits del generador que se está probando. Como resultado, estos seis bytes formarán una matriz binaria de 6x8 con rango determinado, pero en este caso podrá oscilar de 0 a 6, considerándose anómalos el 0,1,2 y 3 y sus recuentos serán combinados con aquellos de rango 4. A mayores, cabe destacar que los rangos han sido encontrados para 100.000 matrices aleatorias.

- ***Prueba OPSO:*** La prueba OPSO (Pares superpuestos de ocupación dispersa) se basa en tomar palabras que tengan dos letras de un alfabeto que posee 1024 letras, de manera que a cada letra se le asignan de forma específica 10 bits de un

entero que posee 32. De esta manera se generan dos palabras, de forma superpuestas, las cuales constan de dos letras y, a mayores, realizará el recuento de las palabras de dos letras que no aparecen en la secuencia, es decir, las palabras que faltan; cabe señalar que este recuento debería encontrarse próximo a una distribución normal con una media de 141909 y sigma 290. Como conclusión, hay que destacar que esta metodología toma 32 bits al mismo tiempo del archivo de prueba y emplea un conjunto de diez bits consecutivos, lo cual se repetirá una y otra vez con el reinicio del archivo para los diez bits que siguen en la secuencia.

- ***Prueba OQSO:*** La prueba OQSO (Cuádruples superpuestos de ocupación dispersa) presenta grandes similitudes con la anterior, pero su principal diferencia es que se basa en tomar palabras de 4 letras dentro de un alfabeto de 32, dentro del cual cada letra estará determinada por una cadena de designada por la consecución de 5 bits del archivo de prueba. Es importante señalar que estos elementos deben de ser enteros y aleatorios de 32 bits. En este caso, el número medio de palabras que faltan es de nuevo 141909, pero con sigma 295.

- ***Prueba ADN:*** Esta metodología se basa en tomar un alfabeto que posee cuatro letras (A, C, G, T) y está determinado por la designación de dos bits en la secuencia entera y aleatoria que se está testeando. En este caso se tienen en cuenta las palabras de 10 letras por lo que, como en los dos casos previos, hay dos posibles palabras y el número medio de palabras faltantes es de 141909, pero con sigma 339.

- ***Prueba del recuento 1s en una secuencia de bytes:*** En esta tipología hay que tener en cuenta que el archivo está siendo probado como un flujo de bytes (cuatro por cada entero de 32 bits), en el que cada byte puede poseer de cero a ocho 1s, con unas probabilidades de 1, 8, 28, 56, 70, 56, 28, 8, 1 sobre 256. A continuación, hay que dejar que el flujo de bytes proporcione una cadena de palabras superpuestas de 5 letras, las cuales tomarán los valores de A, B, C, D y E. Estas letras podrán determinarse en función del número de 1s dentro de cada byte, de forma que 0, 1, 2 correspondan a A, 3 a B, 4 a C, 5 a D y 6, 7 u 8 a E. Esto da como resultado cinco posibles palabras con cinco letras y, mediante una

cadena de 256.000 palabras de cinco letras, podrá realizarse el recuento de la frecuencia de cada palabra.

Por otro lado, se encuentra la prueba del recuento 1s para bytes específicos, la cual presenta grandes similitudes con la explicada previamente, con la diferencia de que se elige un byte específico de cada número entero.

- ***Prueba de estacionamiento:*** Esta prueba se basa en tomar un cuadrado de tamaño 100 y en colocar, de forma aleatoria, un círculo de diámetro 1 para, más adelante, ir colocando un segundo, un tercero, un cuarto y sucesivos. Si un intento de colocación provoca el choque entre un círculo y otro hay que buscar otra situación al azar, de esta forma cada intento conducirá al fracaso o al éxito. De esta manera, tomando *n* como número de intentos y *k* como número de círculos colocados, al realizar su comparación, obtendremos una curva que debería asemejarse a la que garantiza un generador perfecto de números aleatorios.

- ***Prueba de distancia mínima:*** En este caso, el primer paso a tener en cuenta es la elección de 100 veces $n = 800$ puntos aleatorios en un cuadrado cuyo tamaño sea 1000. A continuación, hay que considerar d como la mínima distancia entre $(n2 - n) \div 2$ pares de puntos y si estos puntos resultan verdaderamente independientes, el cuadrado de la distancia, $d2$, debe de estar exponencialmente distribuido con una media bastante próxima a 0.995, $1 - exp\ (-d2\ \div\ 0.095)$ deberá, por lo tanto, ser igual en $[0,1)$ y esta prueba será aplicada a los 100 valores resultantes, sirviendo como prueba de homogeneidad de los puntos al azar que están dentro del cuadrado.

- ***Prueba de las esferas 3D:*** En este caso tomaremos 4000 puntos al azar en un cubo con una arista de 1000. En cada punto habrá que centrar una esfera lo suficientemente grande para alcanzar el siguiente más próximo. De esta manera, el volumen de las esferas más pequeñas será próximo a una distribución exponencial de media $120\pi\ \div\ 3$ y el radio del cubo será exponencial con media 30, la cual es obtenida por simulación. Esta prueba crea 4000 esferas veinte veces y cada cubo de radio m conducirá a una variable uniforme $(1 -$

$exp\ (-r3 \div 30))$ para, más adelante, realizar la prueba sobre los veinte valores de p.

- ***Prueba de compresión:*** En este caso, el primer paso será transformar los números enteros aleatorios a valores flotantes para tener los valores uniformes de [0,1). En segundo lugar, hay que tomar $k = 231 = 2147483648$ y la prueba encontrará *j,* es decir, el número de repeticiones necesarias para reducir k a 1 y para ello habrá que emplear la reducción $k = máximo\ (k\ x\ U)$, donde U procede de los números enteros flotantes que se están probando.

Test Name	ntup	t-samples	p-sample	Average P-value of 25 keys	Assessment
diehard_birthdays	0	100	100	0.653760029	OK
diehard_operm5	0	1000000	100	0.532907215	OK
diehard_rank_32x32	0	40000	100	0.510957518	OK
diehard_rank_6x8	0	100000	100	0.614881389	OK
diehard_bitstream	0	2097152	100	0.572861452	OK
diehard_opso	0	2097152	100	0.56342995	OK
diehard_oqso	0	2097152	100	0.564799784	OK
diehard_dna	0	2097152	100	0.49437631	OK
diehard_count_1s_str	0	256000	100	0.425854956	OK
diehard_count_1s_byt	0	256000	100	0.570959702	OK
diehard_parking_lot	0	12000	100	0.611109984	OK
diehard_2dsphere	2	8000	100	0.553310269	OK
diehard_3dsphere	3	4000	100	0.537745781	OK
diehard_squeeze	0	100000	100	0.589133353	OK
diehard_sums	0	100	100	0.13205638	OK
diehard_runs	0	100000	100	0.559359585	OK
diehard_craps	0	200000	100	0.563369301	OK
marsaglia_tsang_gcd	0	10000000	100	0.610666545	OK
sts_monobit	1	100000	100	0.606755002	OK
sts_runs	2	100000	100	0.501179924	OK
sts_serial	1-16	100000	100	0.504396057 -0.671142755	OK
rgb_bitdist	1-12	100000	100	0.481688763 - 0.702577895	OK
rgb_minimum_distance	2-5	10000	1000	0.445556424 - 0.652603025	OK
rgb_permutations	2	100000	100	0.601399862	OK
rgb_permutations	3	100000	100	0.630948728	OK
rgb_permutations	4	100000	100	0.534075445	OK
rgb_permutations	5	100000	100	0.461843218	OK
rgb_lagged_sum	0-32	1000000	100	0.388427623 - 0.709601808	OK
rgb_kstest_test	0	10000	1000	0.45008513	OK
dab_bytedistrib	0	51200000	1	0.48189395	OK
dab_dct	256	50000	1	0.500848326	OK
dab_filltree	32	15000000	1	0.491787382	OK
dab_filltree	32	15000000	1	0.44187268	OK
dab_filltree2	0	5000000	1	0.532810523	OK
dab_filltree2	1	5000000	1	0.59660472	OK
dab_monobit2	12	65000000	1	0.603623061	OK

Figura 9: Ejemplo de una tabla de resultados de las diferentes pruebas de Dieharder [36]

- ***Prueba de la superposición de sumas:*** En este caso, para obtener una secuencia $U(1), U(2), ...$ de variable uniforme $[1, 0)$, los números enteros aleatorios han de ser flotantes. De esta manera se forman las sumas superpuestas de modo que $S(1) = U(1) + ... + U(100)$, donde S posee unos valores normales con una

cierta matriz de covarianza. En el caso de realizar una transformación de forma lineal de los valores de S, éstos se convertirán en una secuencia estándar e independiente para, a su vez, servir de secuencia de variables uniformes para las pruebas.

- ***Prueba de carreras:*** Esta prueba se basa en contar, a través de una secuencia de variables uniformes $[0,1)$, el recorrido ascendente y descendente; esto se obtiene mediante los enteros flotantes de 32 bits que posee el archivo a testear. Es necesario destacar que, en este caso, las matrices de covarianza para los recorridos tanto ascendentes como descendentes se conocen bien y éstas son medidas a través de secuencias con una longitud 1000.

- ***Prueba de los dados:*** En esta tipología se realizan 200000 partidas de dados, donde hay que buscar tanto el número de victorias, como el número de tiradas necesarias para finalizar la partida. En lo que respecta al número de victorias, éste debe ser cercano a una normal con una media de $200000p$ y una varianza de $200000p\,(1-p)$, donde $p = 244 \div 495$. Además, cabe señalar que el número de lanzamientos necesarios para completar una partida puede oscilar entre uno e infinito y cada dígito entero de 32 bits otorga el valor al lanzamiento del dado, normalizado con un intervalo $[0,1)$, para multiplicarlo por seis a posteriori y sumando uno a la parte entera de la operación resultante.

Por último, dentro de este apartado, es necesario mencionar otro tipo de prueba incluidos dentro del Dieharder: El conjunto de pruebas estadísticas del NIST (Instituto Nacional de Estándares y Tecnología). Esta tipología se compone de quince pruebas, entre las que podemos encontrar la prueba de frecuencia de bloques internos, la prueba de solapamiento de plantillas o la prueba en serie, entre otras; estas pruebas se realizan sucesivamente sobre la secuencia de bits, de forma que, si cualquiera de ellas supera el umbral del 1%, la secuencia se considerará no aleatoria y no se aplicará ninguna prueba más.

7. Protocolo para la evaluación de los dispositivos generadores de números verdaderamente aleatorios.

Con la finalidad de mantener homogeneidad en el trabajo y establecer estándares básicos para alcanzar resultados fiables se establecerá el siguiente protocolo, que será aplicado en el desarrollo del proceso experimental.

A continuación se listan los pasos a seguir:

7.1. Elección de dispositivos a ser evaluados

Dos dispositivos generadores de números verdaderamente aleatorios han sido seleccionas para las pruebas, el ***Araneus Alea II*** y ***Quantis-USB-4M***. Los dispositivos disponen del mismo tipo de conector (USB) por medio del cual se transmiten los datos y la tensión necesaria para su puesta en funcionamiento.

7.2. Generación de datos

Para la generación de los datos en cada dispositivo se ha determinado las condiciones en las que se llevarán a cabo el proceso. Es importante para el resultado de este trabajo que los números aleatorios obtenidos usando los dispositivos generadores de números verdaderamente aleatorios sean confiables, por tanto, para el efecto se ha decido utilizar las librerías y recursos proporcionados por los fabricantes, y de esta manera garantizar la fiabilidad de los datos.

En el caso del dispositivo ***Araneus Alea II*** el proceso de generación se desarrollará en Linux. A mayores, es necesario precisar que se debe contar con la librería "*libusb",* ya que por su intermedio no será necesario instalar drivers específicos para el correcto funcionamiento del dispositivo. El fabricante proporciona el código fuente para instalar el programa "*randomfile"* que se encargará de obtener los datos aleatorios generados por Alea II.

La siguiente imagen muestra un ejemplo de código necesario para generar y almacenar números aleatorios usando el dispositivo físico.

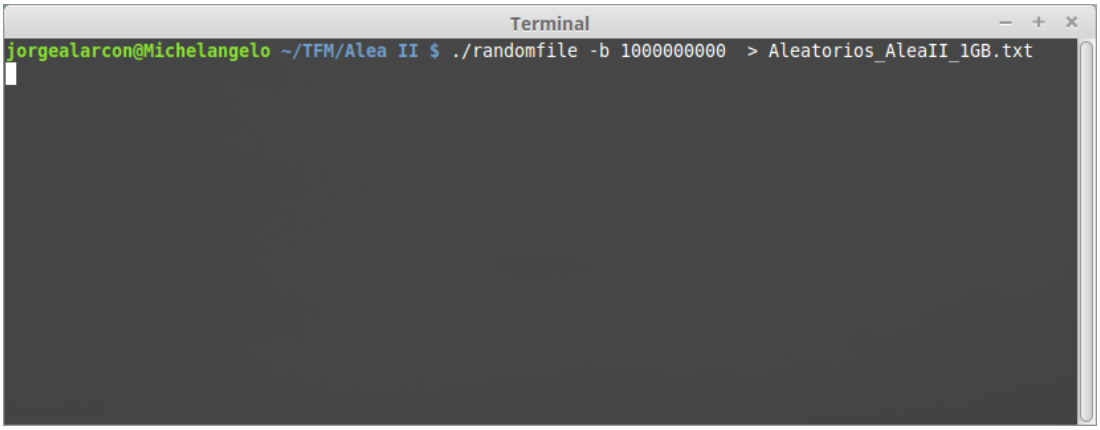


Figura 10: Terminal de Linux ejecutando la generación de números aleatorios en el dispositivo Araneus Alea II

Los números aleatorios generados por el dispositivo ***Quantis-USB-4M*** se obtendrán usando el software proporcionado por la empresa ID Quantique, fabricante del generador, en este caso la versión 2.1 de EasyQuantis para Windows [37].

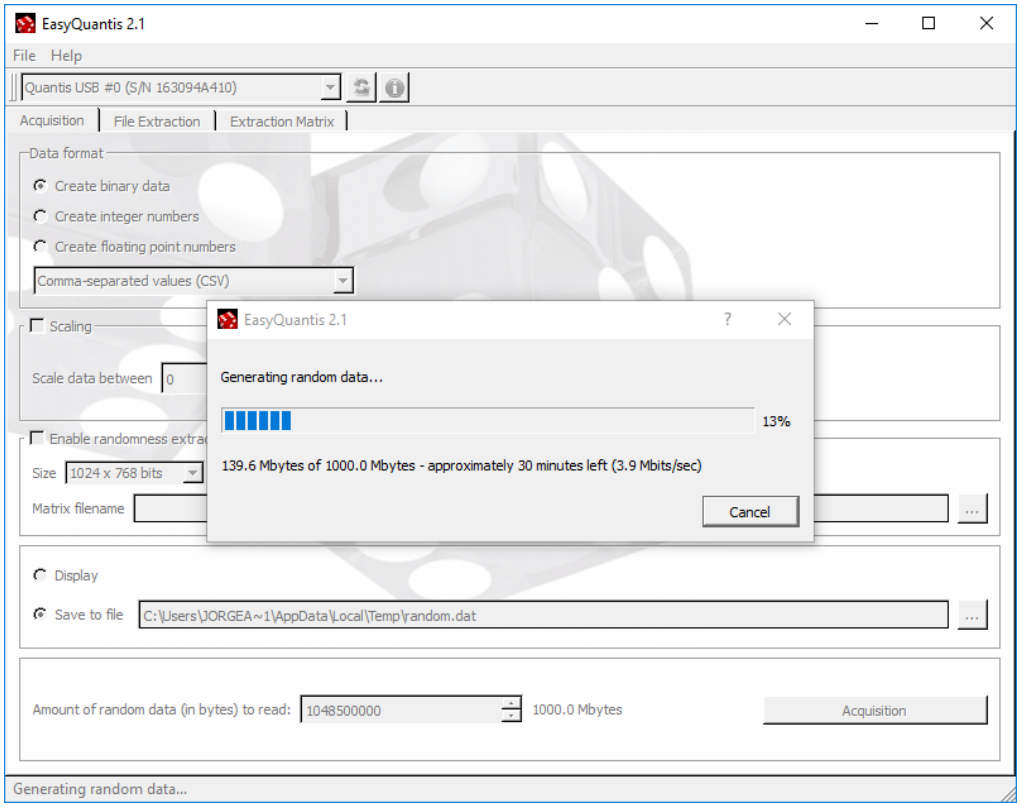


Figura 11: Interfaz del programa EasyQuantis 2.1 durante la generación.

Los datos aleatorios serán recopilados con los programas especificados anteriormente, tanto para el muestreo en estado estable como con presencia de perturbaciones.

7.3. Tamaño de archivo de datos aleatorios

Los dos dispositivos, tanto ***Araneus Alea II*** como ***Quantis-USB-4M***, tienen la capacidad de generar grandes cantidades de datos, dependiendo del modo seleccionado y de la necesidad presentada, para nuestro trabajo los archivos de números aleatorios generados serán de aproximadamente 1 Gigabyte, de esta manera podremos asegurar que los análisis tengan una cantidad suficiente de información y reducimos la probabilidad de errores.

Otro punto para considerar es la velocidad de generación de cada dispositivo, siendo estas completamente diferentes, mientras que ***Quantis-USB-4M*** posee una tasa de datos aleatorios de $\sim 4\ Mbit/s$ que permite obtener la cantidad de datos necesarios para el análisis en aproximadamente 40 minutos, ***Araneus Alea II*** solo dispone de una velocidad de $100\ Kbit/s$ lo cual denota una diferencia de tiempo de generación abismal al generar un gigabyte de datos aleatorios, siendo necesario esperar al menos 25 horas para obtenerlos.

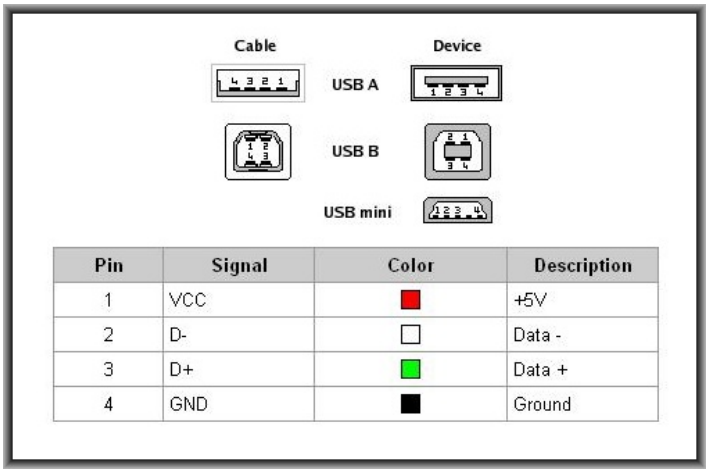
7.4. Tipo de perturbación a aplicarse

El método seleccionado para interferir el funcionamiento de los dispositivos ***Araneus Alea II*** y ***Quantis-USB-4M*** será mediante el ingreso de una perturbación a través de la variación de la tensión (voltaje) de alimentación, esperando generar ruido que altere el proceso de generación de datos.

7.5. Determinación de los mecanismos de ingreso de perturbaciones

Tanto el dispositivo ***Araneus Alea II*** como ***Quantis-USB-4M*** son energizados a través de la conexión USB 2.0, por tanto, la estrategia será emplear un circuito que nos permita tener el control del voltaje de energización de los dispositivos TRNG, luego determinar los voltajes mínimos de funcionamiento de cada dispositivo y generar números aleatorios a estos voltajes. Al finalizar el experimento podremos comparar los números aleatorios generados por el dispositivo en estado estable, con los generados durante aplicación de la perturbación, para determinar cómo fue afectado el proceso de generación.

Con la finalidad de realizar la variación de voltaje, se empleará un circuito que permite interferir el pin V_+ del conector USB tipo A y de esta manera controlar el voltaje de entrada que se suministra a los dispositivos generadores de números verdaderamente aleatorios. Hay que recalcar que en el proceso de interferir al voltaje de alimentación no se afectarán los pines de datos (D_+ y D_-) de los conectores USB, los cuales serán conectados directamente entre la entrada y la salida, sin pasarlos a través de ningún dispositivo externo que pudiera comprometer la correcta transferencia de los datos generados. También se cuidará de mantener una tierra (GND) común, tanto para el conector USB macho, que se conectaría al ordenador, como para el conector USB hembra, que se conectaría al dispositivo TRNG, usando para ello la tierra analógica proporcionada por un dispositivo Arduino MEGA.



Pin	Signal	Color	Description
1	VCC		+5V
2	D-		Data -
3	D+		Data +
4	GND		Ground

Figura 12: Diagrama para la conexión de USB A macho y hembra. [38]

Un dispositivo divisor de tensión será utilizado para suministrar el voltaje de interferencia a los dispositivos TRNG al cual se le suministran $5Vdc$ a través de la placa Arduino MEGA, como se ve en la figura 13.

La variación de tención de entrada de los dispositivos se realizará progresivamente, disminuyendo el voltaje de mayor a menor voltaje, hasta alcanzar el menor voltaje al que el dispositivo trabaja y genera los números aleatorios necesarios para realizar la toma de datos. El divisor de tensión trabaja en el rango de entre $1.2\ V$ y $4.4\ V$ lo que permite una ventana suficientemente amplia para encontrar el voltaje mínimo de trabajo de cada dispositivo.

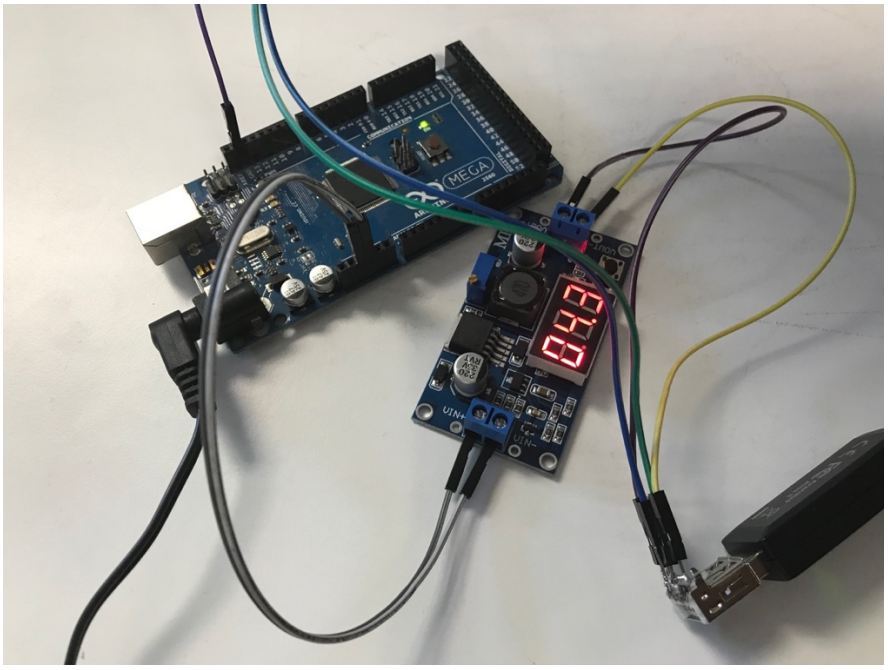


Figura 13: Circuito empleado para la variación del voltaje

7.6. Elección de métodos para el análisis de la información obtenida

El análisis de los datos obtenidos será analizado por baterías de prueba que se detallarán en el siguiente capítulo, en particular se emplearán tres: *"Ent", "Dieharder" y "rngtest"*. Las tres baterías de prueba se encuentran instaladas en un ordenador con Linux, todas las pruebas se realizarán bajo las mismas condiciones, en ningún caso se realizará un análisis de datos usando un ordenador diferente u otra versión de las baterías de prueba.

8. Resultados de experimentos en condiciones normales

A continuación se describirá la obtención de datos en condiciones normales (estables) bajo los parámetros establecidos en el protocolo planteado en el capítulo 7; se utilizarán los programas recomendados por los fabricantes y los dispositivos se conectarán a los ordenadores usando sus terminales, sin intervención de un agente externo que pueda afectar su funcionamiento.

8.1. Araneus Alea II sin perturbaciones

Los resultados obtenidos se basan en un proceso de experimentación, mediante el cual recopilamos datos de números aleatorios de cada dispositivo en estado estable. Los archivos generados no tendrán un tamaño inferior a 1 Gigabyte, con lo cual se espera obtener muestras suficientemente confiables para ser analizadas. Esta información nos servirá de medida de control ya que se compararán con las muestras tomadas bajo la presencia de interferencias logradas a través de la variación del voltaje de alimentación, como se explicó en la sección anterior.

Los datos obtenidos del dispositivo Alea II en condiciones normales, sin la presencia de las perturbaciones externas, fueron sometidos a las baterías de pruebas, comprobando que el dispositivo genera datos de muy buena calidad. La siguiente imagen muestras los resultados de la prueba *Ent* para los datos de Alea II sin perturbaciones.

```
1.  jorgealarcon@Michelangelo ~/TFM/Alea II $ ent Aleatorios_AleaII_1GB.txt
2.  Entropy = 8.000000 bits per byte.
3.
4.  Optimum compression would reduce the size
5.  of this 1000000000 byte file by 0 percent.
6.
7.  Chi square distribution for 1000000000 samples is 265.81, and randomly
8.  would exceed this value 50.00 percent of the times.
9.
10. Arithmetic mean value of data bytes is 127.4990 (127.5 = random).
11. Monte Carlo value for Pi is 3.141739669 (error 0.00 percent).
12. Serial correlation coefficient is 0.000006 (totally uncorrelated = 0.0).
```

Figura 14: Prueba Ent para Alea II sin presencia de perturbaciones

Para el dispositivo Alea II sin perturbaciones de voltaje, podemos observar que el valor de la entropía es muy alto, el más alto posible para esta prueba, obteniendo 8 bits por cada byte. Hay que recordar que mientras más cerca de 8 se encuentre la cantidad de bits en este punto, más entropía tendrá. También se puede observar que la media aritmética arroja

un valor de 127.499, lo que nos indica que la cantidad de 1s y 0s es muy equilibrada, sin embargo, hay más 0s que 1s en esta toma de datos.

Según la *rngtest* podemos analizar que, para 1 GB de datos, equivalentes a 8.000.000.000 bits, la prueba fue pasada con éxito 399.707 y tendremos tan solo 292 fallas. Según la documentación la prueba *Monobit* que busca una distribución de 50% de 1s y 50% de 0s en cada paquete de 20.000 bits, se espera que falle 1 de cada 9662 pruebas con una fuente aleatoria perfecta, según la ecuación:

$$\frac{\sum_{i=9726}^{10274}\binom{20000}{i}}{2^{20000}} \approx \frac{1}{9662}\ para\ FIPS\ 140-2$$

Ecuación 6

Para nuestro caso la suma entre los pases exitosos y fallados es 399.999, que dividido para 9662 nos da 41,399 que es un valor muy cercano al resultado de la prueba monobit que fue 38.

```
219. jorgealarcon@Michelangelo ~/TFM/Alea II $ cat Aleatorios_AleaII_1GB.txt |
     rngtest
220. rngtest 4
221. Copyright (c) 2004 by Henrique de Moraes Holschuh
222. This is free software; see the source for copying conditions.  There is NO
     warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
223.
224. rngtest: starting FIPS tests...
225. rngtest: entropy source drained
226. rngtest: bits received from input: 8000000000
227. rngtest: FIPS 140-2 successes: 399707
228. rngtest: FIPS 140-2 failures: 292
229. rngtest: FIPS 140-2(2001-10-10) Monobit: 38
230. rngtest: FIPS 140-2(2001-10-10) Poker: 43
231. rngtest: FIPS 140-2(2001-10-10) Runs: 111
232. rngtest: FIPS 140-2(2001-10-10) Long run: 102
233. rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
234. rngtest: input channel speed: (min=454.131; avg=4651.593;
     max=19073.486)Mibits/s
235. rngtest: FIPS tests speed: (min=13.663; avg=55.598; max=56.598)Mibits/s
236. rngtest: Program run time: 138937384 microseconds
```

Figura 15: Prueba rngtest para Alea II sin perturbaciones

La prueba Dieharder también mostró que el generador de datos aleatorios funciona correctamente ya que en las varias tomas de muestras no se detectaron pruebas fallidas "FAILED", únicamente algunas pruebas marcadas como débiles "WEAK", pero que no representan anomalías importantes en el planteamiento de los resultados. Una prueba completa puede encontrarse en el apartado Anexos.

8.2. Quantis-USB-4M sin perturbaciones

Los resultados para la prueba *Ent* en el dispositivo Quantis-USB-4M se presentan a continuación:

```
1.  jorgealarcon@Michelangelo ~/TFM/Aleatorios $ ent Aleatorios_Quantis_1GB.dat
2.  Entropy = 8.000000 bits per byte.
3.
4.  Optimum compression would reduce the size
5.  of this 1000000000 byte file by 0 percent.
6.
7.  Chi square distribution for 1000000000 samples is 256.77, and randomly
8.  would exceed this value 50.00 percent of the times.
9.
10. Arithmetic mean value of data bytes is 127.5028 (127.5 = random).
11. Monte Carlo value for Pi is 3.141519757 (error 0.00 percent).
12. Serial correlation coefficient is 0.000025 (totally uncorrelated = 0.0).
```

Figura 16: Prueba Ent para Quantis sin presencia de perturbaciones

Podemos observar que al igual que sucedió con las pruebas de Alea II el valor de la entropía el más alto posible para esta prueba, con 8 bits por cada byte. También se puede observar que la media aritmética arroja un valor de 127.4028, lo que nos indica que la cantidad de 1s y 0s es también muy equilibrada, sin embargo, hay más 1s que 0s.

```
223. jorgealarcon@Michelangelo ~/TFM/Aleatorios $ cat Aleatorios_Quantis_1GB.dat
     |rngtest
224. rngtest 4
225. Copyright (c) 2004 by Henrique de Moraes Holschuh
226. This is free software; see the source for copying conditions.  There is NO
     warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
227.
228. rngtest: starting FIPS tests...
229. rngtest: entropy source drained
230. rngtest: bits received from input: 8000000000
231. rngtest: FIPS 140-2 successes: 399693
232. rngtest: FIPS 140-2 failures: 306
233. rngtest: FIPS 140-2(2001-10-10) Monobit: 41
234. rngtest: FIPS 140-2(2001-10-10) Poker: 27
235. rngtest: FIPS 140-2(2001-10-10) Runs: 109
236. rngtest: FIPS 140-2(2001-10-10) Long run: 129
237. rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
238. rngtest: input channel speed: (min=500000000.000; avg=4877715369.963;
     max=0.000)bits/s
239. rngtest: FIPS tests speed: (min=14.181; avg=60.868; max=61.927)Mibits/s
240. rngtest: Program run time: 127055357 microseconds
```

Figura 17: Prueba rngtest para Quantis sin perturbaciones

Para la prueba rngtest que se muestra en la imagen 17 podemos apreciar que el número de pases totales son 399.999 teóricamente nos daría un valor *Monobit* de 41.399 muy cercano al resultado de 41 obtenido en la prueba.

La prueba *Dieharder* para este dispositivo arrojó resultados que merecen ser observados y analizados ya que se registraron pruebas fallidas que se presentarán y analizarán más adelante.

9. Resultados de experimentos con presencia de perturbaciones

Para la realización de los experimentos con presencia de perturbaciones, primero procedimos a determinar del voltaje mínimo de alimentación necesario con el que los dispositivos logran generar datos aleatorios, para ello, utilizamos el circuito de variación de voltaje. La variación se realizó en pasos de $1\,V$, comenzando en $5\,V$ hasta encontrar el menor valor de voltaje en el que cada dispositivo encendía.

Las mediciones determinaron que el dispositivo ***Araneus Alea II*** lograba generar números aleatorios con un voltaje de entrada de $2.9\,V$. Mientras que el dispositivo ***Quantis-USB-4M*** soportaba un voltaje mínimo de energización de $3.3\,V$. A estos voltajes, los dispositivos estaban en capacidad de generar archivos con números aleatorios del tamaño deseado.

Al realizar la toma de muestras se obtuvieron los siguientes resultados:

9.1. Araneus Alea II con perturbaciones

Si comparamos los resultados de Araneus Alea II sin perturbaciones con los obtenidos por el dispositivo con presencia de perturbaciones, podemos notar que no existen variaciones considerables que deban ser resaltadas, los resultados son tan buenos como los presentados para los datos sin perturbaciones.

```
1.  jorgealarcon@Michelangelo ~/TFM/Alea II $ ent pruebaRuido_1G_2.9.txt
2.  Entropy = 8.000000 bits per byte.
3.
4.  Optimum compression would reduce the size
5.  of this 1000000000 byte file by 0 percent.
6.
7.  Chi square distribution for 1000000000 samples is 222.03, and randomly
8.  would exceed this value 90.00 percent of the times.
9.
10. Arithmetic mean value of data bytes is 127.5012 (127.5 = random).
11. Monte Carlo value for Pi is 3.141661093 (error 0.00 percent).
12. Serial correlation coefficient is -0.000052 (totally uncorrelated = 0.0).
```

Figura 18: Prueba Ent para Alea II con presencia de perturbación.

Las pruebas *Dieharder* y *rngtest* fueron realizadas al dispositivo con presencia de perturbaciones, no encontrándose diferencias significativas. Las pruebas arrojaron resultados similares a los obtenidos sin presencia de perturbaciones en este dispositivo.

```
223. jorgealarcon@Michelangelo ~/TFM/Alea II $ cat pruebaRuido_1G_2.9.txt | rngtest
224. rngtest 4
225. Copyright (c) 2004 by Henrique de Moraes Holschuh
226. This is free software; see the source for copying conditions.  There is NO
     warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
227.
228. rngtest: starting FIPS tests...
229. rngtest: entropy source drained
230. rngtest: bits received from input: 8000000000
231. rngtest: FIPS 140-2 successes: 399662
232. rngtest: FIPS 140-2 failures: 337
233. rngtest: FIPS 140-2(2001-10-10) Monobit: 35
234. rngtest: FIPS 140-2(2001-10-10) Poker: 49
235. rngtest: FIPS 140-2(2001-10-10) Runs: 123
236. rngtest: FIPS 140-2(2001-10-10) Long run: 131
237. rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
238. rngtest: input channel speed: (min=397.364; avg=4688.903;
     max=19073.486)Mibits/s
239. rngtest: FIPS tests speed: (min=14.160; avg=61.160; max=62.332)Mibits/s
240. rngtest: Program run time: 126466198 microseconds
```

Figura 19: Prueba rgntest para dispositivo Alea II con perturbaciones

9.2. Quantis-USB-4M con perturbaciones

Como se puede apreciar en la figura 20, el dispositivo Quantis se comportó muy bien en el análisis de entropía, obteniendo el valor más alto posible para esta prueba, con 8 bits por cada byte. La media aritmética muestra que la relación entre 1s y 0s también es muy equilibrada.

```
1.   jorgealarcon@Michelangelo ~/TFM/Quantis $ ent Quantis_1G_3.3.dat
2.   Entropy = 8.000000 bits per byte.
3.
4.   Optimum compression would reduce the size
5.   of this 1000000000 byte file by 0 percent.
6.
7.   Chi square distribution for 1000000000 samples is 229.03, and randomly
8.   would exceed this value 75.00 percent of the times.
9.
10.  Arithmetic mean value of data bytes is 127.4973 (127.5 = random).
11.  Monte Carlo value for Pi is 3.141778789 (error 0.01 percent).
12.  Serial correlation coefficient is 0.000037 (totally uncorrelated = 0.0).
```

Figura 20: Prueba Ent para Quantis con presencia de perturbaciones

La prueba *rngtest* también arroja resultados favorables para este dispositivo en presencia de perturbaciones.

```
jorgealarcon@Michelangelo ~/TFM/Quantis $ cat Quantis_1G_3.3.dat | rngtest
rngtest 4
Copyright (c) 2004 by Henrique de Moraes Holschuh
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

rngtest: starting FIPS tests...
rngtest: entropy source drained
rngtest: bits received from input: 8000000000
rngtest: FIPS 140-2 successes: 399708
rngtest: FIPS 140-2 failures: 291
rngtest: FIPS 140-2(2001-10-10) Monobit: 39
rngtest: FIPS 140-2(2001-10-10) Poker: 34
rngtest: FIPS 140-2(2001-10-10) Runs: 122
rngtest: FIPS 140-2(2001-10-10) Long run: 99
rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
rngtest: input channel speed: (min=500000000.000; avg=4902921903.613;
max=0.000)bits/s
rngtest: FIPS tests speed: (min=14.428; avg=61.443; max=62.332)Mibits/s
rngtest: Program run time: 125891127 microseconds
```

Figura 21: Prueba rngtest para Quantis con perturbaciones

Por otra parte, la batería de pruebas Dieharder nuevamente, al igual que ocurrió con este dispositivo sin presencia de perturbaciones, ha vuelto a arrojar pruebas fallidas. La prueba fallada es *RGB Lagged Sums #31*. Esto lo analizaremos a detalle en el siguiente capítulo.

10. Análisis de Resultados

En base a los resultados obtenidos podemos notar que el dispositivo ***Araneus Alea II*** se ha comportado de acuerdo con lo esperado, tanto en las tomas de datos sin perturbaciones como con perturbaciones, no fallando en ninguna de las baterías a las que fue sometido. Como se puede observar en las figuras 14 y 18, los resultados de la prueba *Ent* sin y con perturbaciones en el dispositivo Alea II son similares, no pudiéndose destacar cambios significativos entre ellas. Lo mismo ocurre en este dispositivo con las pruebas *rngtest y Dieharder,* logrando resultados suficientemente buenos para pensar que el dispositivo está muy bien protegido ante posibles ataques externos generados por variaciones de voltaje.

También para el dispositivo ***Quantis-USB-4M*** se debe mencionar que los resultados de las pruebas *Ent* y *rgntes* fueron exitosas.

Por otra parte, es importante señalar que en la prueba de Dieharder hubo resultados muy interesantes que ahora analizaremos: La prueba de suma rezagada (RGB Lagged Sum Test) *#31,* durante las pruebas realizadas fallaba constantemente durante el análisis de los datos del dispositivo Quantis. En distintos pases de la batería, esta prueba siempre fue la más inestable, pasando siempre entre DÉBIL "WEAK" y FALLADA "FAILED"; un ejemplo de aquello se puede ver en la figura 18.

```
196. # The file file_input_raw was rewound 218 times
197.       rgb_lagged_sum|  30|   1000000|     100|0.87448505|  PASSED
198. # The file file_input_raw was rewound 231 times
199.       rgb_lagged_sum|  31|   1000000|     100|0.00000000|  FAILED
200. # The file file_input_raw was rewound 244 times
201.       rgb_lagged_sum|  32|   1000000|     100|0.41235925|  PASSED
```

Figura 22: Prueba Dieharder rgb_lagged_sum #31 fallada por el dispositivo Quantis-USB-4M sin perturbaciones

Un caso similar se da en la prueba *rgb_lagged_sums #24,* la cual también falló en el análisis de los números aleatorios generados en el dispositivo Quantis con interferencia de voltaje, mientras que, durante el proceso de análisis de los números generados sin interferencia, dio como resultado DEBIL "WEAK". En la siguiente figura se puede notar las pruebas #24 y #31 siendo falladas por el dispositivo Quantis sometido a perturbaciones de voltaje.

```
182. # The file file_input_raw was rewound 140 times
183.       rgb_lagged_sum|  23|   1000000|     100|0.15245138|  PASSED
184. # The file file_input_raw was rewound 150 times
185.       rgb_lagged_sum|  24|   1000000|     100|0.00000022|  FAILED
186. # The file file_input_raw was rewound 160 times
187.       rgb_lagged_sum|  25|   1000000|     100|0.44303160|  PASSED
188. # The file file_input_raw was rewound 171 times
189.       rgb_lagged_sum|  26|   1000000|     100|0.76326174|  PASSED
190. # The file file_input_raw was rewound 182 times
191.       rgb_lagged_sum|  27|   1000000|     100|0.40090598|  PASSED
192. # The file file_input_raw was rewound 194 times
193.       rgb_lagged_sum|  28|   1000000|     100|0.37333823|  PASSED
194. # The file file_input_raw was rewound 206 times
195.       rgb_lagged_sum|  29|   1000000|     100|0.00051041|   WEAK
196. # The file file_input_raw was rewound 218 times
197.       rgb_lagged_sum|  30|   1000000|     100|0.21674080|  PASSED
198. # The file file_input_raw was rewound 231 times
199.       rgb_lagged_sum|  31|   1000000|     100|0.00000003|  FAILED
200. # The file file_input_raw was rewound 244 times
201.       rgb_lagged_sum|  32|   1000000|     100|0.81078374|  PASSED
```

Figura 23: Prueba Dieharder rgb_lagged_sums #31 y #24 falladas por el dispositivo Quantis-USB-4M con perturbaciones

La prueba de suma rezagada (*RGB Lagged Sums Test*) fue diseñado para buscar correlaciones a nivel de bit en números generados que no aparecen hasta después de que cierta cantidad de datos fue consultada.

Estos casos presentados no pueden ser considerados concluyentes al momento de determinar si existen fallas en la generación de números verdaderamente aleatorios utilizando el dispositivo Quantis, pero plantean varias preguntas respecto de la posibilidad de que algún eslabón de la cadena de generación de datos esté causando dicho error.

Para procurar determinar la fuente que genera la falla en esta prueba, ser realizaron otras tomas de datos con las que esperamos aislar el problema y darnos luces de la causa que lo estaría generando. Es así que se realizaron tomas de datos con el software en otro sistema operativo, mayores volúmenes de datos se generaron y además se pasó la prueba *RGB Lagged Sums* en solitario. A continuación, se presentan los resultados obtenidos.

Un fichero de 6 GB fue generado en el mismo ordenador con sistema operativo Windows y bajo las mismas condiciones descritas anteriormente, con EasyQuantis 2.1 y sin presencia de perturbaciones; al analizar los datos no se detectaron problemas con las pruebas realizadas, pasándolas sin problemas.

Al tomar una muestra de 1 GB de datos usando la misma versión del programa EasyQuantis 2.1 en MacOS se obtuvo el resultado presentado en la siguiente imagen.

```
181. # The file file_input_raw was rewound 203 times
182.       rgb_lagged_sum|  30|   1000000|     100|0.88507780|  PASSED
183. # The file file_input_raw was rewound 215 times
184.       rgb_lagged_sum|  31|   1000000|     100|0.00000131|   WEAK
185. # The file file_input_raw was rewound 228 times
186.       rgb_lagged_sum|  32|   1000000|     100|0.63728670|  PASSED
```

Figura 24:Prueba RGB Lagged Sums #31 para Quantis en MacOS

Podemos observar que para los datos obtenidos en el sistema operativo MacOS también se aprecia que la prueba da como resultado débil o “WEAK”.

Se debe observar también que, al realizar la prueba de manera aislada, sin completar la batería completa, sino enfocándonos sólo en la prueba rng_lagged_sums #31, los datos aleatorios motivos de análisis siempre pasaron la prueba.

Por tanto, podemos decir que los datos no son suficientemente contundentes para realizar un análisis completo de los resultados obtenidos, se propondrá que se estudie más ampliamente en futuros trabajos.

11. Conclusiones y líneas futuras de trabajo

En este trabajo se han generado números verdaderamente aleatorios con dispositivos físicos específicamente diseñados para cumplir con dicha labor, los números generados han sido sometidos a diversas pruebas para comprobar su aleatoriedad, logrando determinar que los generadores funcionan correctamente y los números generados son de buena calidad. Luego de aquello, hemos procedido a interferir la entrada de voltaje de los dispositivos Araneus Alea II y Quantis-USB-4M a través de un circuito diseñado con este efecto, luego de determinar el voltaje mínimo de entrada que debía suministrarse con la finalidad de que los dispositivos funcionen y generen una cantidad adecuada de datos. Se logró generar números aleatorios en los dos dispositivos a pesar de interferir la alimentación de voltaje en con valores de $2.9\,V$ para Alea II y $3.3\,V$ para el dispositivo Quantis.

Los resultados de las pruebas en los dispositivos Alea II y Quantis-USB-4M no difieren notablemente y se mantienen en valores normales por lo que se concluye que los dispositivos generadores de números verdaderamente aleatorios no están siendo afectados por las perturbaciones externas aplicadas.

Existen casos puntuales que merecen un análisis más exhaustivo, como el caso de la prueba de suma rezagada (RGB Lagged Sum Test) *#31* de Dieharder la cual ha fallado en varias pruebas realizadas con el dispositivo Quantis. Al realizar otras pruebas en modo continuo y en condiciones diferentes, a las expresadas en el presente trabajo, se ha notado que la batería de pruebas funciona correctamente, lo que indicaría que el proceso de generación de los números aleatorios bajo las condiciones expuestas está afectando a los datos.

Algunas líneas futuras de investigación que se plantean son:

- Comparar el funcionamiento del generador de números verdaderamente aleatorios Quantis-USB-4M usando diferentes versiones de software y sistemas operativos, comparando sus resultados en Dieharder, para determinar en qué condiciones se producen los fallos de la prueba RGB Lagged Sum Test *#31 y #24.*

- Vulnerar las seguridades físicas del dispositivo Araneus Alea II para procurar alterar la integridad de los datos variando el voltaje de alimentación, ya que en las pruebas realizadas en este trabajo se ha notado muy robusto ante los ataques procurados.
- Se realizaron varias pruebas analizando los espectros de frecuencia de varios dispositivos generadores de números verdaderamente aleatorios, pero no se llegaron concluir los experimentos, por tanto, se propone la realización de las pruebas exponiendo a los dispositivos a ataques con equipos generadores de radiofrecuencia y determinar la afectación que se pueda producir.

12. Bibliografía:

[1] G. Charkiewicz, “La importancia de crear números verdaderamente aleatorios,” 2014. [Online]. Available: https://www.welivesecurity.com/la-es/2014/05/23/importancia-generar-numeros-aleatorios-seguridad-informatica/. [Accessed: 16-Sep-2018].

[2] P. M. A. Garau, J. M. G. Carrasco, and L. H. Encinas, “Diseño de un nuevo generador de secuencias de bits aleatorios por entrada de teclado,” *Novática*, vol. marzo-abri, no. 174, pp. 59–65, 2005.

[3] N. Moscoloni, *Las nubes de datos: Métodos para analizar la complejidad*, 1ra ed. Rosario: Editorial de la Universidad Nacional de Rosario, 2011.

[4] A. M. Mancilla Herrera, *Números Aleatorios. Historia, teoría y aplicaciones*, vol. 0, no. 8. Fundacion Universidad del Norte, 2000, pp. 49–69.

[5] L. F. Restrepo B, E. Esp, J. L. González, and I. Esp, “La Historia de la Probabilidad,” *Rev Col Cienc Pec*, vol. 16, no. 1, 2003.

[6] J. Hernandez, “Breve Historia de la Estadística,” Medellin, 2013.

[7] M. Stipčević and Ç. K. Koç, “True Random Number Generators,” in *Open Problems in Mathematics and Computational Science*, Cham: Springer International Publishing, 2014, pp. 275–315.

[8] J. C. Garcia-Escartin and M. Herrero-Collantes, “Quantum random number generators,” *Rev. Mod. Phys.*, vol. 89, no. 1, 2016.

[9] ID Quantique, “Random Number Generation using Quantum Physics,” p. 8, 2010.

[10] M. González de la Fuente, “Generación de números aleatorios físicos en dispositivos electrónicos,” *Escuela Técnica Superios de Ingenieros de Telecomunicación. Universidad de Valladolid*, 2016.

[11] D. H. Brocker, “Digital Random-Number Generator,” 1973.

[12] W. Tomasi, G. Mata Hernandez, and V. Gonzalez Pozo, *Sistemas de comunicaciones electrónicas*. Pearson Education, 2003.

[13] K. Abend and B. D. Fritchman, “Statistical Detection for Communication Channels with Intersymbol Interference,” *Proc. IEEE*, vol. 58, no. 5, pp. 779–785, 1970.

[14] Z. L. Lu and B. A. Dosher, “External noise distinguishes attention mechanisms,” *Vision Res.*, vol. 38, no. 9, pp. 1183–1198, May 1998.

[15] M. L. Gorodetsky, “Thermal noises and noise compensation in high-reflection multilayer coating,” *Phys. Lett. Sect. A Gen. At. Solid State Phys.*, vol. 372, no. 46, pp. 6813–6822, Nov. 2008.

[16] Á. B. Uscátegui, “El ruido 1/f,” *Ingeniería*, vol. 5, no. 1, pp. 28–36, 2000.

[17] F. Sandoval-Ibarra, N. Melchor-Hernández, and S. Ortega-Cisneros, “Analysis, modeling and simulation of flickernoise in MOS transistors,” *Acta Univ. Univ. Guanajuato*, vol. 23, no. 5, 2013.

[18] J. M. Moyano Drake, “Ruidos e Interferencias : Técnicas de reducción,” 2005.

[19] A. Ferrero and J. C. A. Pons, “Reducción del ruido en CCD,” *Opt. Eng.*, no. 34, pp. 1–3, 2003.

[20] “Interacción del ruido Shot con la frecuencia 1/f,” 2010. [Online]. Available: https://www.researchgate.net/figure/Figura-11-Interaccion-del-ruido-Shot-con-la-frecuencia-1-f_308087624. [Accessed: 16-Sep-2018].

[21] A. C. Hern, “Efectos del ruido en las comunicaciones electrónicas,” 2001.

[22] J. de J. Medel Juárez, “El ruido impulsivo, un problema en las telecomunicaciones con base en las líneas eléctricas de potencia | Academia de Ingeniería de México,” 2016. [Online]. Available: http://www.ai.org.mx/presentacion/el-ruido-impulsivo-un-problema-en-las-telecomunicaciones-con-base-en-las-líneas. [Accessed: 16-Sep-2018].

[23] J. Márquez Flores, “Curso Instrumentación y Señales.” p. 59, 2012.

[24] Araneus Information Systems Oy, “Araneus Alea 2: True Random Number Generator - User’s Guide.” Turku, Finland, 2014.

[25] “Araneus Alea II True Random Number Generator.” [Online]. Available: https://www.araneus.fi/products/alea2/en/. [Accessed: 13-Sep-2018].

[26] D. Hurley-smith and J. Hernandez-castro, “Quam Bene Non Quantum : Bias in a Family of Quantum Random Number Generators,” *https://eprint.iacr.org/2017/842*, 2017.

[27] A. Stefanov, N. Gisin, O. Guinnard, L. Guinnard, and H. Zbinden, “Optical quantum random number generator,” *J. Mod. Opt.*, vol. 47, no. 4, pp. 595–598, Mar. 2000.

[28] M. Jofre *et al.*, “True random numbers from amplified quantum vacuum,” *Opt. Express*, vol. 19, no. 21, p. 20665, Oct. 2011.

[29] T. J. Quirk and J. Palmer-Schuyler, *Random Number Generator*. Cham: Springer International Publishing, 2015.

[30] Intel, “Digital Random Number Generator (DRNG) Software Implementation Guide.” p. 35, 2014.

[31] R. G. Brown, “Dieharder: A Random Number Test Suite version 3.31.1,” 2018. [Online]. Available: https://webhome.phy.duke.edu/~rgb/General/dieharder.php. [Accessed: 13-Sep-2018].

[32] National Institute of Standards and Technology, “Random Bit Generation | CSRC,” 2016. [Online]. Available: https://csrc.nist.gov/projects/random-bit-generation/. [Accessed: 16-Sep-2018].

[33] H. de Moraes Holschuh, “Debian Package Tracking System - rng-tools.” [Online]. Available: https://packages.qa.debian.org/r/rng-tools.html. [Accessed: 10-Sep-2018].

[34] G. Castro Castilla and E. San Millán, “Implementación de un generador TRNG en FPGA,” Universidad Carlos III.

[35] G. Marsaglia, “The Marsaglia Random Number CDROM including the Diehard Battery of Tests,” *Florida State University*, 1995. [Online]. Available: http://www.csis.hku.hk/cisc/download/index2.htm. [Accessed: 10-Aug-2018].

[36] F. A. Khana, A. Alia, H. Abbasb, and N. A. H. Haldarc, “A cloud-based healthcare framework for security and patients ’ data privacy using wireless body area networks,” in *The 2nd International Workshop on Communications and Sensor Networks (ComSense-2014)*, 2014.

[37] “Quantis User Guide Version 2.10,” 2013. [Online]. Available: http://xepa15.fisica.ufmg.br/inetsec/uploadFiles/DOC/User_Manual.pdf. [Accessed: 22-Aug-2018].

[38] P. Puzzuto, “Cable Drawings - SLICE Micro/Nano USB Cables,” 2016. [Online]. Available: https://support.dtsweb.com/hc/en-us/articles/219537088-Cable-Drawings-SLICE-Micro-Nano-USB-Cables. [Accessed: 13-Sep-2018].

13. Anexos

ANEXO 1: Pruebas de aleatoriedad de 1 Gigabyte de datos generados sin perturbaciones con el dispositivo Araneus Alea II.

```
1.  jorgealarcon@Michelangelo ~/TFM/Alea II $ ent Aleatorios_AleaII_1GB.txt
2.  Entropy = 8.000000 bits per byte.
3.
4.  Optimum compression would reduce the size
5.  of this 1000000000 byte file by 0 percent.
6.
7.  Chi square distribution for 1000000000 samples is 265.81, and randomly
8.  would exceed this value 50.00 percent of the times.
9.
10. Arithmetic mean value of data bytes is 127.4990 (127.5 = random).
11. Monte Carlo value for Pi is 3.141739669 (error 0.00 percent).
12. Serial correlation coefficient is 0.000006 (totally uncorrelated = 0.0).
13.
14.
15. jorgealarcon@Michelangelo ~/TFM/Alea II $ dieharder -a -g 201 -f
Aleatorios_AleaII_1GB.txt
16. #=============================================================================#
17. #            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
18. #=============================================================================#
19.    rng_name    |           filename             |rands/second|
20.  file_input_raw|       Aleatorios_AleaII_1GB.txt|  4.18e+07  |
21. #=============================================================================#
22.         test_name   |ntup| tsamples |psamples|  p-value |Assessment
23. #=============================================================================#
24.    diehard_birthdays|   0|       100|     100|0.81414330|  PASSED
25.       diehard_operm5|   0|   1000000|     100|0.18494141|  PASSED
26.   diehard_rank_32x32|   0|     40000|     100|0.42088910|  PASSED
27. # The file file_input_raw was rewound 1 times
28.     diehard_rank_6x8|   0|    100000|     100|0.43213479|  PASSED
29. # The file file_input_raw was rewound 1 times
30.    diehard_bitstream|   0|   2097152|     100|0.42981271|  PASSED
31. # The file file_input_raw was rewound 2 times
32.         diehard_opso|   0|   2097152|     100|0.25349407|  PASSED
33. # The file file_input_raw was rewound 2 times
34.         diehard_oqso|   0|   2097152|     100|0.69314997|  PASSED
35. # The file file_input_raw was rewound 2 times
36.          diehard_dna|   0|   2097152|     100|0.58343227|  PASSED
37. # The file file_input_raw was rewound 2 times
38. diehard_count_1s_str|   0|    256000|     100|0.48322788|  PASSED
39. # The file file_input_raw was rewound 3 times
40. diehard_count_1s_byt|   0|    256000|     100|0.91425291|  PASSED
41. # The file file_input_raw was rewound 3 times
42.  diehard_parking_lot|   0|     12000|     100|0.71023270|  PASSED
43. # The file file_input_raw was rewound 3 times
44.     diehard_2dsphere|   2|      8000|     100|0.41707494|  PASSED
45. # The file file_input_raw was rewound 3 times
46.     diehard_3dsphere|   3|      4000|     100|0.60576322|  PASSED
47. # The file file_input_raw was rewound 4 times
48.      diehard_squeeze|   0|    100000|     100|0.75070480|  PASSED
49. # The file file_input_raw was rewound 4 times
50.         diehard_sums|   0|       100|     100|0.35165681|  PASSED
51. # The file file_input_raw was rewound 4 times
```

```
52.         diehard_runs|   0|    100000|     100|0.07665088|  PASSED
53.         diehard_runs|   0|    100000|     100|0.10230736|  PASSED
54. # The file file_input_raw was rewound 5 times
55.        diehard_craps|   0|    200000|     100|0.61252155|  PASSED
56.        diehard_craps|   0|    200000|     100|0.07491736|  PASSED
57. # The file file_input_raw was rewound 13 times
58.  marsaglia_tsang_gcd|   0|  10000000|     100|0.00103815|   WEAK
59.  marsaglia_tsang_gcd|   0|  10000000|     100|0.00418708|   WEAK
60. # The file file_input_raw was rewound 13 times
61.          sts_monobit|   1|    100000|     100|0.62232952|  PASSED
62. # The file file_input_raw was rewound 13 times
63.            sts_runs|   2|    100000|     100|0.35158116|  PASSED
64. # The file file_input_raw was rewound 13 times
65.           sts_serial|   1|    100000|     100|0.79544079|  PASSED
66.           sts_serial|   2|    100000|     100|0.83696695|  PASSED
67.           sts_serial|   3|    100000|     100|0.78608519|  PASSED
68.           sts_serial|   3|    100000|     100|0.30121879|  PASSED
69.           sts_serial|   4|    100000|     100|0.80084522|  PASSED
70.           sts_serial|   4|    100000|     100|0.41314656|  PASSED
71.           sts_serial|   5|    100000|     100|0.32418149|  PASSED
72.           sts_serial|   5|    100000|     100|0.08751799|  PASSED
73.           sts_serial|   6|    100000|     100|0.13422534|  PASSED
74.           sts_serial|   6|    100000|     100|0.85387743|  PASSED
75.           sts_serial|   7|    100000|     100|0.49312702|  PASSED
76.           sts_serial|   7|    100000|     100|0.96605003|  PASSED
77.           sts_serial|   8|    100000|     100|0.95879618|  PASSED
78.           sts_serial|   8|    100000|     100|0.68859181|  PASSED
79.           sts_serial|   9|    100000|     100|0.77432264|  PASSED
80.           sts_serial|   9|    100000|     100|0.83664494|  PASSED
81.           sts_serial|  10|    100000|     100|0.78395654|  PASSED
82.           sts_serial|  10|    100000|     100|0.23222495|  PASSED
83.           sts_serial|  11|    100000|     100|0.85693156|  PASSED
84.           sts_serial|  11|    100000|     100|0.80829288|  PASSED
85.           sts_serial|  12|    100000|     100|0.52596997|  PASSED
86.           sts_serial|  12|    100000|     100|0.20877856|  PASSED
87.           sts_serial|  13|    100000|     100|0.80359715|  PASSED
88.           sts_serial|  13|    100000|     100|0.60072625|  PASSED
89.           sts_serial|  14|    100000|     100|0.99732386|   WEAK
90.           sts_serial|  14|    100000|     100|0.99983019|   WEAK
91.           sts_serial|  15|    100000|     100|0.70173228|  PASSED
92.           sts_serial|  15|    100000|     100|0.72982036|  PASSED
93.           sts_serial|  16|    100000|     100|0.86997665|  PASSED
94.           sts_serial|  16|    100000|     100|0.26823959|  PASSED
95. # The file file_input_raw was rewound 13 times
96.          rgb_bitdist|   1|    100000|     100|0.83016804|  PASSED
97. # The file file_input_raw was rewound 13 times
98.          rgb_bitdist|   2|    100000|     100|0.02956725|  PASSED
99. # The file file_input_raw was rewound 13 times
100.              rgb_bitdist|   3|    100000|     100|0.85936973|  PASSED
101.     # The file file_input_raw was rewound 13 times
102.              rgb_bitdist|   4|    100000|     100|0.98672674|  PASSED
103.     # The file file_input_raw was rewound 14 times
104.              rgb_bitdist|   5|    100000|     100|0.54180757|  PASSED
105.     # The file file_input_raw was rewound 14 times
106.              rgb_bitdist|   6|    100000|     100|0.51003071|  PASSED
107.     # The file file_input_raw was rewound 15 times
108.              rgb_bitdist|   7|    100000|     100|0.79254888|  PASSED
109.     # The file file_input_raw was rewound 16 times
110.              rgb_bitdist|   8|    100000|     100|0.85240323|  PASSED
111.     # The file file_input_raw was rewound 16 times
112.              rgb_bitdist|   9|    100000|     100|0.58570153|  PASSED
```

```
113.      # The file file_input_raw was rewound 17 times
114.              rgb_bitdist|  10|    100000|     100|0.22296593|  PASSED
115.      # The file file_input_raw was rewound 18 times
116.              rgb_bitdist|  11|    100000|     100|0.45197301|  PASSED
117.      # The file file_input_raw was rewound 19 times
118.              rgb_bitdist|  12|    100000|     100|0.33657951|  PASSED
119.      # The file file_input_raw was rewound 19 times
120.      rgb_minimum_distance|   2|     10000|    1000|0.83553227|  PASSED
121.      # The file file_input_raw was rewound 19 times
122.      rgb_minimum_distance|   3|     10000|    1000|0.87365704|  PASSED
123.      # The file file_input_raw was rewound 19 times
124.      rgb_minimum_distance|   4|     10000|    1000|0.23054294|  PASSED
125.      # The file file_input_raw was rewound 19 times
126.      rgb_minimum_distance|   5|     10000|    1000|0.94276035|  PASSED
127.      # The file file_input_raw was rewound 20 times
128.          rgb_permutations|   2|    100000|     100|0.27560898|  PASSED
129.      # The file file_input_raw was rewound 20 times
130.          rgb_permutations|   3|    100000|     100|0.63941193|  PASSED
131.      # The file file_input_raw was rewound 20 times
132.          rgb_permutations|   4|    100000|     100|0.60670874|  PASSED
133.      # The file file_input_raw was rewound 20 times
134.          rgb_permutations|   5|    100000|     100|0.33127450|  PASSED
135.      # The file file_input_raw was rewound 20 times
136.               rgb_lagged_sum|   0|   1000000|     100|0.92064965|  PASSED
137.      # The file file_input_raw was rewound 21 times
138.               rgb_lagged_sum|   1|   1000000|     100|0.98815126|  PASSED
139.      # The file file_input_raw was rewound 22 times
140.               rgb_lagged_sum|   2|   1000000|     100|0.92418605|  PASSED
141.      # The file file_input_raw was rewound 24 times
142.               rgb_lagged_sum|   3|   1000000|     100|0.79084986|  PASSED
143.      # The file file_input_raw was rewound 26 times
144.               rgb_lagged_sum|   4|   1000000|     100|0.24845489|  PASSED
145.      # The file file_input_raw was rewound 28 times
146.               rgb_lagged_sum|   5|   1000000|     100|0.46558607|  PASSED
147.      # The file file_input_raw was rewound 31 times
148.               rgb_lagged_sum|   6|   1000000|     100|0.92080252|  PASSED
149.      # The file file_input_raw was rewound 34 times
150.               rgb_lagged_sum|   7|   1000000|     100|0.04549940|  PASSED
151.      # The file file_input_raw was rewound 38 times
152.               rgb_lagged_sum|   8|   1000000|     100|0.89790029|  PASSED
153.      # The file file_input_raw was rewound 42 times
154.               rgb_lagged_sum|   9|   1000000|     100|0.11682371|  PASSED
155.      # The file file_input_raw was rewound 46 times
156.               rgb_lagged_sum|  10|   1000000|     100|0.45173995|  PASSED
157.      # The file file_input_raw was rewound 51 times
158.               rgb_lagged_sum|  11|   1000000|     100|0.61969980|  PASSED
159.      # The file file_input_raw was rewound 56 times
160.               rgb_lagged_sum|  12|   1000000|     100|0.80534633|  PASSED
161.      # The file file_input_raw was rewound 62 times
162.               rgb_lagged_sum|  13|   1000000|     100|0.89547904|  PASSED
163.      # The file file_input_raw was rewound 68 times
164.               rgb_lagged_sum|  14|   1000000|     100|0.00343118|   WEAK
165.      # The file file_input_raw was rewound 74 times
166.               rgb_lagged_sum|  15|   1000000|     100|0.00945050|  PASSED
167.      # The file file_input_raw was rewound 81 times
168.               rgb_lagged_sum|  16|   1000000|     100|0.69889197|  PASSED
169.      # The file file_input_raw was rewound 88 times
170.               rgb_lagged_sum|  17|   1000000|     100|0.91917907|  PASSED
171.      # The file file_input_raw was rewound 96 times
172.               rgb_lagged_sum|  18|   1000000|     100|0.83828894|  PASSED
173.      # The file file_input_raw was rewound 104 times
```

```
174.            rgb_lagged_sum|  19|   1000000|     100|0.08132560|  PASSED
175.      # The file file_input_raw was rewound 112 times
176.            rgb_lagged_sum|  20|   1000000|     100|0.87831169|  PASSED
177.      # The file file_input_raw was rewound 121 times
178.            rgb_lagged_sum|  21|   1000000|     100|0.96343149|  PASSED
179.      # The file file_input_raw was rewound 130 times
180.            rgb_lagged_sum|  22|   1000000|     100|0.71740960|  PASSED
181.      # The file file_input_raw was rewound 140 times
182.            rgb_lagged_sum|  23|   1000000|     100|0.71288996|  PASSED
183.      # The file file_input_raw was rewound 150 times
184.            rgb_lagged_sum|  24|   1000000|     100|0.00005571|   WEAK
185.      # The file file_input_raw was rewound 160 times
186.            rgb_lagged_sum|  25|   1000000|     100|0.99632966|   WEAK
187.      # The file file_input_raw was rewound 171 times
188.            rgb_lagged_sum|  26|   1000000|     100|0.84559765|  PASSED
189.      # The file file_input_raw was rewound 182 times
190.            rgb_lagged_sum|  27|   1000000|     100|0.67223424|  PASSED
191.      # The file file_input_raw was rewound 194 times
192.            rgb_lagged_sum|  28|   1000000|     100|0.81834604|  PASSED
193.      # The file file_input_raw was rewound 206 times
194.            rgb_lagged_sum|  29|   1000000|     100|0.00665436|  PASSED
195.      # The file file_input_raw was rewound 218 times
196.            rgb_lagged_sum|  30|   1000000|     100|0.84956809|  PASSED
197.      # The file file_input_raw was rewound 231 times
198.            rgb_lagged_sum|  31|   1000000|     100|0.00003832|   WEAK
199.      # The file file_input_raw was rewound 244 times
200.            rgb_lagged_sum|  32|   1000000|     100|0.33942605|  PASSED
201.      # The file file_input_raw was rewound 244 times
202.           rgb_kstest_test|   0|     10000|    1000|0.68531366|  PASSED
203.      # The file file_input_raw was rewound 245 times
204.          dab_bytedistrib|   0|  51200000|       1|0.97443929|  PASSED
205.      # The file file_input_raw was rewound 245 times
206.                  dab_dct| 256|     50000|       1|0.63404632|  PASSED
207.      Preparing to run test 207.  ntuple = 0
208.      # The file file_input_raw was rewound 246 times
209.             dab_filltree|  32|  15000000|       1|0.24290648|  PASSED
210.             dab_filltree|  32|  15000000|       1|0.20546834|  PASSED
211.      Preparing to run test 208.  ntuple = 0
212.      # The file file_input_raw was rewound 246 times
213.            dab_filltree2|   0|   5000000|       1|0.78179046|  PASSED
214.            dab_filltree2|   1|   5000000|       1|0.94867846|  PASSED
215.      Preparing to run test 209.  ntuple = 0
216.      # The file file_input_raw was rewound 246 times
217.             dab_monobit2|  12|  65000000|       1|0.99607682|   WEAK
218.
219.
220.      jorgealarcon@Michelangelo ~/TFM/Alea II $ cat Aleatorios_AleaII_1GB.txt |
rngtest
221.      rngtest 4
222.      Copyright (c) 2004 by Henrique de Moraes Holschuh
223.      This is free software; see the source for copying conditions.  There is
NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
224.
225.      rngtest: starting FIPS tests...
226.      rngtest: entropy source drained
227.      rngtest: bits received from input: 8000000000
228.      rngtest: FIPS 140-2 successes: 399707
229.      rngtest: FIPS 140-2 failures: 292
230.      rngtest: FIPS 140-2(2001-10-10) Monobit: 38
231.      rngtest: FIPS 140-2(2001-10-10) Poker: 43
232.      rngtest: FIPS 140-2(2001-10-10) Runs: 111
```

```
233.      rngtest: FIPS 140-2(2001-10-10) Long run: 102
234.      rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
235.      rngtest: input channel speed: (min=454.131; avg=4651.593;
max=19073.486)Mibits/s
236.      rngtest: FIPS tests speed: (min=13.663; avg=55.598; max=56.598)Mibits/s
237.      rngtest: Program run time: 138937384 microseconds
238.      jorgealarcon@Michelangelo ~/TFM/Alea II $
239.
240.
```

ANEXO 2: Pruebas de aleatoriedad de 1 Gigabyte de datos generados con perturbaciones con el dispositivo Araneus Alea II.

```
1.  jorgealarcon@Michelangelo ~/TFM/Alea II $ ent pruebaRuido_1G_2.9.txt
2.  Entropy = 8.000000 bits per byte.
3.
4.  Optimum compression would reduce the size
5.  of this 1000000000 byte file by 0 percent.
6.
7.  Chi square distribution for 1000000000 samples is 222.03, and randomly
8.  would exceed this value 90.00 percent of the times.
9.
10. Arithmetic mean value of data bytes is 127.5012 (127.5 = random).
11. Monte Carlo value for Pi is 3.141661093 (error 0.00 percent).
12. Serial correlation coefficient is -0.000052 (totally uncorrelated = 0.0).
13.
14.
15.
16. jorgealarcon@Michelangelo ~/TFM/Alea II $ dieharder -a -g 201 -f
pruebaRuido_1G_2.9.txt
17. #=============================================================================#
18. #            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
19. #=============================================================================#
20.    rng_name    |           filename             |rands/second|
21.  file_input_raw|          pruebaRuido_1G_2.9.txt|  4.63e+07  |
22. #=============================================================================#
23.         test_name   |ntup| tsamples |psamples|  p-value |Assessment
24. #=============================================================================#
25.    diehard_birthdays|   0|       100|     100|0.85644157|  PASSED
26.       diehard_operm5|   0|   1000000|     100|0.09983099|  PASSED
27.   diehard_rank_32x32|   0|     40000|     100|0.19663834|  PASSED
28. # The file file_input_raw was rewound 1 times
29.     diehard_rank_6x8|   0|    100000|     100|0.68792723|  PASSED
30. # The file file_input_raw was rewound 1 times
31.    diehard_bitstream|   0|   2097152|     100|0.12933921|  PASSED
32. # The file file_input_raw was rewound 2 times
33.         diehard_opso|   0|   2097152|     100|0.64064226|  PASSED
34. # The file file_input_raw was rewound 2 times
35.         diehard_oqso|   0|   2097152|     100|0.23529010|  PASSED
36. # The file file_input_raw was rewound 2 times
37.          diehard_dna|   0|   2097152|     100|0.83006803|  PASSED
38. # The file file_input_raw was rewound 2 times
39. diehard_count_1s_str|   0|    256000|     100|0.93732286|  PASSED
40. # The file file_input_raw was rewound 3 times
41. diehard_count_1s_byt|   0|    256000|     100|0.52095477|  PASSED
42. # The file file_input_raw was rewound 3 times
43.  diehard_parking_lot|   0|     12000|     100|0.76062712|  PASSED
```

```
# The file file_input_raw was rewound 3 times
     diehard_2dsphere|   2|      8000|     100|0.54531714|  PASSED
# The file file_input_raw was rewound 3 times
     diehard_3dsphere|   3|      4000|     100|0.85750071|  PASSED
# The file file_input_raw was rewound 4 times
      diehard_squeeze|   0|    100000|     100|0.66553456|  PASSED
# The file file_input_raw was rewound 4 times
        diehard_sums|   0|       100|     100|0.19800827|  PASSED
# The file file_input_raw was rewound 4 times
        diehard_runs|   0|    100000|     100|0.39607842|  PASSED
        diehard_runs|   0|    100000|     100|0.29107720|  PASSED
# The file file_input_raw was rewound 5 times
       diehard_craps|   0|    200000|     100|0.99971432|   WEAK
       diehard_craps|   0|    200000|     100|0.30891471|  PASSED
# The file file_input_raw was rewound 13 times
 marsaglia_tsang_gcd|   0|  10000000|     100|0.10323664|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.00063807|   WEAK
# The file file_input_raw was rewound 13 times
         sts_monobit|   1|    100000|     100|0.55615041|  PASSED
# The file file_input_raw was rewound 13 times
            sts_runs|   2|    100000|     100|0.80036561|  PASSED
# The file file_input_raw was rewound 13 times
          sts_serial|   1|    100000|     100|0.37583264|  PASSED
          sts_serial|   2|    100000|     100|0.77441984|  PASSED
          sts_serial|   3|    100000|     100|0.05371048|  PASSED
          sts_serial|   3|    100000|     100|0.50166781|  PASSED
          sts_serial|   4|    100000|     100|0.96940196|  PASSED
          sts_serial|   4|    100000|     100|0.59101101|  PASSED
          sts_serial|   5|    100000|     100|0.16251152|  PASSED
          sts_serial|   5|    100000|     100|0.30766483|  PASSED
          sts_serial|   6|    100000|     100|0.33191137|  PASSED
          sts_serial|   6|    100000|     100|0.49127361|  PASSED
          sts_serial|   7|    100000|     100|0.48645826|  PASSED
          sts_serial|   7|    100000|     100|0.89636513|  PASSED
          sts_serial|   8|    100000|     100|0.84917349|  PASSED
          sts_serial|   8|    100000|     100|0.40035018|  PASSED
          sts_serial|   9|    100000|     100|0.66311121|  PASSED
          sts_serial|   9|    100000|     100|0.35988640|  PASSED
          sts_serial|  10|    100000|     100|0.16649752|  PASSED
          sts_serial|  10|    100000|     100|0.08153468|  PASSED
          sts_serial|  11|    100000|     100|0.55186862|  PASSED
          sts_serial|  11|    100000|     100|0.34234040|  PASSED
          sts_serial|  12|    100000|     100|0.19676475|  PASSED
          sts_serial|  12|    100000|     100|0.68271485|  PASSED
          sts_serial|  13|    100000|     100|0.51281239|  PASSED
          sts_serial|  13|    100000|     100|0.57515040|  PASSED
          sts_serial|  14|    100000|     100|0.11251742|  PASSED
          sts_serial|  14|    100000|     100|0.08599131|  PASSED
          sts_serial|  15|    100000|     100|0.05961985|  PASSED
          sts_serial|  15|    100000|     100|0.63410888|  PASSED
          sts_serial|  16|    100000|     100|0.28160784|  PASSED
          sts_serial|  16|    100000|     100|0.13511293|  PASSED
# The file file_input_raw was rewound 13 times
         rgb_bitdist|   1|    100000|     100|0.05923794|  PASSED
# The file file_input_raw was rewound 13 times
         rgb_bitdist|   2|    100000|     100|0.35032330|  PASSED
     # The file file_input_raw was rewound 13 times
              rgb_bitdist|   3|    100000|     100|0.97251048|  PASSED
     # The file file_input_raw was rewound 13 times
              rgb_bitdist|   4|    100000|     100|0.57690447|  PASSED
     # The file file_input_raw was rewound 14 times
```

```
105.              rgb_bitdist|   5|    100000|     100|0.43353301|  PASSED
106.     # The file file_input_raw was rewound 14 times
107.              rgb_bitdist|   6|    100000|     100|0.58970832|  PASSED
108.     # The file file_input_raw was rewound 15 times
109.              rgb_bitdist|   7|    100000|     100|0.00880373|  PASSED
110.     # The file file_input_raw was rewound 16 times
111.              rgb_bitdist|   8|    100000|     100|0.16252300|  PASSED
112.     # The file file_input_raw was rewound 16 times
113.              rgb_bitdist|   9|    100000|     100|0.41371783|  PASSED
114.     # The file file_input_raw was rewound 17 times
115.              rgb_bitdist|  10|    100000|     100|0.80619372|  PASSED
116.     # The file file_input_raw was rewound 18 times
117.              rgb_bitdist|  11|    100000|     100|0.14513437|  PASSED
118.     # The file file_input_raw was rewound 19 times
119.              rgb_bitdist|  12|    100000|     100|0.86679174|  PASSED
120.     # The file file_input_raw was rewound 19 times
121.     rgb_minimum_distance|   2|     10000|    1000|0.54641275|  PASSED
122.     # The file file_input_raw was rewound 19 times
123.     rgb_minimum_distance|   3|     10000|    1000|0.29564499|  PASSED
124.     # The file file_input_raw was rewound 19 times
125.     rgb_minimum_distance|   4|     10000|    1000|0.87795161|  PASSED
126.     # The file file_input_raw was rewound 19 times
127.     rgb_minimum_distance|   5|     10000|    1000|0.72742439|  PASSED
128.     # The file file_input_raw was rewound 20 times
129.         rgb_permutations|   2|    100000|     100|0.47140921|  PASSED
130.     # The file file_input_raw was rewound 20 times
131.         rgb_permutations|   3|    100000|     100|0.86479967|  PASSED
132.     # The file file_input_raw was rewound 20 times
133.         rgb_permutations|   4|    100000|     100|0.46118489|  PASSED
134.     # The file file_input_raw was rewound 20 times
135.         rgb_permutations|   5|    100000|     100|0.80696011|  PASSED
136.     # The file file_input_raw was rewound 20 times
137.              rgb_lagged_sum|   0|   1000000|     100|0.95502394|  PASSED
138.     # The file file_input_raw was rewound 21 times
139.              rgb_lagged_sum|   1|   1000000|     100|0.65444760|  PASSED
140.     # The file file_input_raw was rewound 22 times
141.              rgb_lagged_sum|   2|   1000000|     100|0.72176489|  PASSED
142.     # The file file_input_raw was rewound 24 times
143.              rgb_lagged_sum|   3|   1000000|     100|0.68236697|  PASSED
144.     # The file file_input_raw was rewound 26 times
145.              rgb_lagged_sum|   4|   1000000|     100|0.47501534|  PASSED
146.     # The file file_input_raw was rewound 28 times
147.              rgb_lagged_sum|   5|   1000000|     100|0.98602030|  PASSED
148.     # The file file_input_raw was rewound 31 times
149.              rgb_lagged_sum|   6|   1000000|     100|0.69684206|  PASSED
150.     # The file file_input_raw was rewound 34 times
151.              rgb_lagged_sum|   7|   1000000|     100|0.83311621|  PASSED
152.     # The file file_input_raw was rewound 38 times
153.              rgb_lagged_sum|   8|   1000000|     100|0.75792246|  PASSED
154.     # The file file_input_raw was rewound 42 times
155.              rgb_lagged_sum|   9|   1000000|     100|0.10075112|  PASSED
156.     # The file file_input_raw was rewound 46 times
157.              rgb_lagged_sum|  10|   1000000|     100|0.99022770|  PASSED
158.     # The file file_input_raw was rewound 51 times
159.              rgb_lagged_sum|  11|   1000000|     100|0.48711450|  PASSED
160.     # The file file_input_raw was rewound 56 times
161.              rgb_lagged_sum|  12|   1000000|     100|0.89478699|  PASSED
162.     # The file file_input_raw was rewound 62 times
163.              rgb_lagged_sum|  13|   1000000|     100|0.10469502|  PASSED
164.     # The file file_input_raw was rewound 68 times
165.              rgb_lagged_sum|  14|   1000000|     100|0.33254225|  PASSED
```

```
# The file file_input_raw was rewound 74 times
      rgb_lagged_sum|  15|   1000000|     100|0.34662898|  PASSED
# The file file_input_raw was rewound 81 times
      rgb_lagged_sum|  16|   1000000|     100|0.15519175|  PASSED
# The file file_input_raw was rewound 88 times
      rgb_lagged_sum|  17|   1000000|     100|0.11432485|  PASSED
# The file file_input_raw was rewound 96 times
      rgb_lagged_sum|  18|   1000000|     100|0.70337240|  PASSED
# The file file_input_raw was rewound 104 times
      rgb_lagged_sum|  19|   1000000|     100|0.00110249|   WEAK
# The file file_input_raw was rewound 112 times
      rgb_lagged_sum|  20|   1000000|     100|0.36018600|  PASSED
# The file file_input_raw was rewound 121 times
      rgb_lagged_sum|  21|   1000000|     100|0.68309052|  PASSED
# The file file_input_raw was rewound 130 times
      rgb_lagged_sum|  22|   1000000|     100|0.97248491|  PASSED
# The file file_input_raw was rewound 140 times
      rgb_lagged_sum|  23|   1000000|     100|0.08332477|  PASSED
# The file file_input_raw was rewound 150 times
      rgb_lagged_sum|  24|   1000000|     100|0.00000142|   WEAK
# The file file_input_raw was rewound 160 times
      rgb_lagged_sum|  25|   1000000|     100|0.26485943|  PASSED
# The file file_input_raw was rewound 171 times
      rgb_lagged_sum|  26|   1000000|     100|0.83745338|  PASSED
# The file file_input_raw was rewound 182 times
      rgb_lagged_sum|  27|   1000000|     100|0.35732885|  PASSED
# The file file_input_raw was rewound 194 times
      rgb_lagged_sum|  28|   1000000|     100|0.55518062|  PASSED
# The file file_input_raw was rewound 206 times
      rgb_lagged_sum|  29|   1000000|     100|0.01850995|  PASSED
# The file file_input_raw was rewound 218 times
      rgb_lagged_sum|  30|   1000000|     100|0.68371269|  PASSED
# The file file_input_raw was rewound 231 times
      rgb_lagged_sum|  31|   1000000|     100|0.39832225|  PASSED
# The file file_input_raw was rewound 244 times
      rgb_lagged_sum|  32|   1000000|     100|0.17752660|  PASSED
# The file file_input_raw was rewound 244 times
     rgb_kstest_test|   0|     10000|    1000|0.06868473|  PASSED
# The file file_input_raw was rewound 245 times
    dab_bytedistrib|   0|  51200000|       1|0.02334607|  PASSED
# The file file_input_raw was rewound 245 times
           dab_dct| 256|     50000|       1|0.13385443|  PASSED
Preparing to run test 207.  ntuple = 0
# The file file_input_raw was rewound 246 times
      dab_filltree|  32|  15000000|       1|0.95202535|  PASSED
      dab_filltree|  32|  15000000|       1|0.48672733|  PASSED
Preparing to run test 208.  ntuple = 0
# The file file_input_raw was rewound 246 times
     dab_filltree2|   0|   5000000|       1|0.64919250|  PASSED
     dab_filltree2|   1|   5000000|       1|0.15989084|  PASSED
Preparing to run test 209.  ntuple = 0
# The file file_input_raw was rewound 246 times
      dab_monobit2|  12|  65000000|       1|0.54598358|  PASSED
jorgealarcon@Michelangelo ~/TFM/Alea II $


jorgealarcon@Michelangelo ~/TFM/Alea II $ cat pruebaRuido_1G_2.9.txt |
rngtest
rngtest 4
Copyright (c) 2004 by Henrique de Moraes Holschuh
```

```
226.      This is free software; see the source for copying conditions.  There is
NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
227.
228.      rngtest: starting FIPS tests...
229.      rngtest: entropy source drained
230.      rngtest: bits received from input: 8000000000
231.      rngtest: FIPS 140-2 successes: 399662
232.      rngtest: FIPS 140-2 failures: 337
233.      rngtest: FIPS 140-2(2001-10-10) Monobit: 35
234.      rngtest: FIPS 140-2(2001-10-10) Poker: 49
235.      rngtest: FIPS 140-2(2001-10-10) Runs: 123
236.      rngtest: FIPS 140-2(2001-10-10) Long run: 131
237.      rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
238.      rngtest: input channel speed: (min=397.364; avg=4688.903;
max=19073.486)Mibits/s
239.      rngtest: FIPS tests speed: (min=14.160; avg=61.160; max=62.332)Mibits/s
240.      rngtest: Program run time: 126466198 microseconds
241.
242.
243.
```

ANEXO 3: Pruebas de aleatoriedad de 1 Gigabyte de datos generados sin perturbaciones con el dispositivo Quantis-USB-4M.

```
1.  jorgealarcon@Michelangelo ~/TFM/Aleatorios $ ent Aleatorios_Quantis_1GB.dat
2.  Entropy = 8.000000 bits per byte.
3.
4.  Optimum compression would reduce the size
5.  of this 1000000000 byte file by 0 percent.
6.
7.  Chi square distribution for 1000000000 samples is 256.77, and randomly
8.  would exceed this value 50.00 percent of the times.
9.
10. Arithmetic mean value of data bytes is 127.5028 (127.5 = random).
11. Monte Carlo value for Pi is 3.141519757 (error 0.00 percent).
12. Serial correlation coefficient is 0.000025 (totally uncorrelated = 0.0).
13.
14.
15.
16. jorgealarcon@Michelangelo ~/TFM/Aleatorios $ dieharder -a -g 201 -f
Aleatorios_Quantis_1GB.dat
17. #=============================================================================#
18. #            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
19. #=============================================================================#
20.    rng_name    |           filename             |rands/second|
21.  file_input_raw|      Aleatorios_Quantis_1GB.dat|  4.50e+07  |
22. #=============================================================================#
23.         test_name   |ntup| tsamples |psamples|  p-value |Assessment
24. #=============================================================================#
25.    diehard_birthdays|   0|       100|     100|0.99833526|   WEAK
26.       diehard_operm5|   0|   1000000|     100|0.48333904|  PASSED
27.   diehard_rank_32x32|   0|     40000|     100|0.97056134|  PASSED
28. # The file file_input_raw was rewound 1 times
29.     diehard_rank_6x8|   0|    100000|     100|0.54040588|  PASSED
30. # The file file_input_raw was rewound 1 times
```

```
31.    diehard_bitstream|   0|   2097152|     100|0.94743271|  PASSED
32. # The file file_input_raw was rewound 2 times
33.         diehard_opso|   0|   2097152|     100|0.99889146|   WEAK
34. # The file file_input_raw was rewound 2 times
35.         diehard_oqso|   0|   2097152|     100|0.41951223|  PASSED
36. # The file file_input_raw was rewound 2 times
37.          diehard_dna|   0|   2097152|     100|0.51679735|  PASSED
38. # The file file_input_raw was rewound 2 times
39. diehard_count_1s_str|   0|    256000|     100|0.14615206|  PASSED
40. # The file file_input_raw was rewound 3 times
41. diehard_count_1s_byt|   0|    256000|     100|0.26370926|  PASSED
42. # The file file_input_raw was rewound 3 times
43.  diehard_parking_lot|   0|     12000|     100|0.53057251|  PASSED
44. # The file file_input_raw was rewound 3 times
45.     diehard_2dsphere|   2|      8000|     100|0.30406483|  PASSED
46. # The file file_input_raw was rewound 3 times
47.     diehard_3dsphere|   3|      4000|     100|0.14326618|  PASSED
48. # The file file_input_raw was rewound 4 times
49.      diehard_squeeze|   0|    100000|     100|0.30796379|  PASSED
50. # The file file_input_raw was rewound 4 times
51.         diehard_sums|   0|       100|     100|0.00740304|  PASSED
52. # The file file_input_raw was rewound 4 times
53.         diehard_runs|   0|    100000|     100|0.87025860|  PASSED
54.         diehard_runs|   0|    100000|     100|0.91282442|  PASSED
55. # The file file_input_raw was rewound 5 times
56.        diehard_craps|   0|    200000|     100|0.43445238|  PASSED
57.        diehard_craps|   0|    200000|     100|0.07394927|  PASSED
58. # The file file_input_raw was rewound 13 times
59.  marsaglia_tsang_gcd|   0|  10000000|     100|0.12365817|  PASSED
60.  marsaglia_tsang_gcd|   0|  10000000|     100|0.00547920|  PASSED
61. # The file file_input_raw was rewound 13 times
62.          sts_monobit|   1|    100000|     100|0.27653513|  PASSED
63. # The file file_input_raw was rewound 13 times
64.             sts_runs|   2|    100000|     100|0.90230434|  PASSED
65. # The file file_input_raw was rewound 13 times
66.           sts_serial|   1|    100000|     100|0.15482183|  PASSED
67.           sts_serial|   2|    100000|     100|0.91274406|  PASSED
68.           sts_serial|   3|    100000|     100|0.94767332|  PASSED
69.           sts_serial|   3|    100000|     100|0.41814479|  PASSED
70.           sts_serial|   4|    100000|     100|0.77616826|  PASSED
71.           sts_serial|   4|    100000|     100|0.70644290|  PASSED
72.           sts_serial|   5|    100000|     100|0.61944262|  PASSED
73.           sts_serial|   5|    100000|     100|0.26018145|  PASSED
74.           sts_serial|   6|    100000|     100|0.65156466|  PASSED
75.           sts_serial|   6|    100000|     100|0.53551019|  PASSED
76.           sts_serial|   7|    100000|     100|0.99398167|  PASSED
77.           sts_serial|   7|    100000|     100|0.96215736|  PASSED
78.           sts_serial|   8|    100000|     100|0.49539030|  PASSED
79.           sts_serial|   8|    100000|     100|0.39508611|  PASSED
80.           sts_serial|   9|    100000|     100|0.83430556|  PASSED
81.           sts_serial|   9|    100000|     100|0.69058453|  PASSED
82.           sts_serial|  10|    100000|     100|0.44195229|  PASSED
83.           sts_serial|  10|    100000|     100|0.13290253|  PASSED
84.           sts_serial|  11|    100000|     100|0.81364231|  PASSED
85.           sts_serial|  11|    100000|     100|0.96338518|  PASSED
86.           sts_serial|  12|    100000|     100|0.12964797|  PASSED
87.           sts_serial|  12|    100000|     100|0.18370577|  PASSED
88.           sts_serial|  13|    100000|     100|0.48842583|  PASSED
89.           sts_serial|  13|    100000|     100|0.70270764|  PASSED
90.           sts_serial|  14|    100000|     100|0.54129631|  PASSED
91.           sts_serial|  14|    100000|     100|0.28598995|  PASSED
```

```
92.           sts_serial|  15|    100000|     100|0.91898506|  PASSED
93.           sts_serial|  15|    100000|     100|0.70129655|  PASSED
94.           sts_serial|  16|    100000|     100|0.56473846|  PASSED
95.           sts_serial|  16|    100000|     100|0.42423571|  PASSED
96. # The file file_input_raw was rewound 13 times
97.          rgb_bitdist|   1|    100000|     100|0.62017594|  PASSED
98. # The file file_input_raw was rewound 13 times
99.          rgb_bitdist|   2|    100000|     100|0.42978254|  PASSED
100.     # The file file_input_raw was rewound 13 times
101.              rgb_bitdist|   3|    100000|     100|0.97381750|  PASSED
102.     # The file file_input_raw was rewound 13 times
103.              rgb_bitdist|   4|    100000|     100|0.34016347|  PASSED
104.     # The file file_input_raw was rewound 14 times
105.              rgb_bitdist|   5|    100000|     100|0.84234565|  PASSED
106.     # The file file_input_raw was rewound 14 times
107.              rgb_bitdist|   6|    100000|     100|0.67720564|  PASSED
108.     # The file file_input_raw was rewound 15 times
109.              rgb_bitdist|   7|    100000|     100|0.64700802|  PASSED
110.     # The file file_input_raw was rewound 16 times
111.              rgb_bitdist|   8|    100000|     100|0.73540337|  PASSED
112.     # The file file_input_raw was rewound 16 times
113.              rgb_bitdist|   9|    100000|     100|0.35351236|  PASSED
114.     # The file file_input_raw was rewound 17 times
115.              rgb_bitdist|  10|    100000|     100|0.50663266|  PASSED
116.     # The file file_input_raw was rewound 18 times
117.              rgb_bitdist|  11|    100000|     100|0.46037474|  PASSED
118.     # The file file_input_raw was rewound 19 times
119.              rgb_bitdist|  12|    100000|     100|0.29395392|  PASSED
120.     # The file file_input_raw was rewound 19 times
121.     rgb_minimum_distance|   2|     10000|    1000|0.10676236|  PASSED
122.     # The file file_input_raw was rewound 19 times
123.     rgb_minimum_distance|   3|     10000|    1000|0.59915707|  PASSED
124.     # The file file_input_raw was rewound 19 times
125.     rgb_minimum_distance|   4|     10000|    1000|0.68056941|  PASSED
126.     # The file file_input_raw was rewound 19 times
127.     rgb_minimum_distance|   5|     10000|    1000|0.80866031|  PASSED
128.     # The file file_input_raw was rewound 20 times
129.         rgb_permutations|   2|    100000|     100|0.76762705|  PASSED
130.     # The file file_input_raw was rewound 20 times
131.         rgb_permutations|   3|    100000|     100|0.82434243|  PASSED
132.     # The file file_input_raw was rewound 20 times
133.         rgb_permutations|   4|    100000|     100|0.91419074|  PASSED
134.     # The file file_input_raw was rewound 20 times
135.         rgb_permutations|   5|    100000|     100|0.99631652|   WEAK
136.     # The file file_input_raw was rewound 20 times
137.              rgb_lagged_sum|   0|   1000000|     100|0.94531069|  PASSED
138.     # The file file_input_raw was rewound 21 times
139.              rgb_lagged_sum|   1|   1000000|     100|0.54989711|  PASSED
140.     # The file file_input_raw was rewound 22 times
141.              rgb_lagged_sum|   2|   1000000|     100|0.28065304|  PASSED
142.     # The file file_input_raw was rewound 24 times
143.              rgb_lagged_sum|   3|   1000000|     100|0.66530870|  PASSED
144.     # The file file_input_raw was rewound 26 times
145.              rgb_lagged_sum|   4|   1000000|     100|0.58412639|  PASSED
146.     # The file file_input_raw was rewound 28 times
147.              rgb_lagged_sum|   5|   1000000|     100|0.07176478|  PASSED
148.     # The file file_input_raw was rewound 31 times
149.              rgb_lagged_sum|   6|   1000000|     100|0.07773427|  PASSED
150.     # The file file_input_raw was rewound 34 times
151.              rgb_lagged_sum|   7|   1000000|     100|0.14944571|  PASSED
152.     # The file file_input_raw was rewound 38 times
```

```
153.            rgb_lagged_sum|   8|   1000000|     100|0.20157441|  PASSED
154.      # The file file_input_raw was rewound 42 times
155.            rgb_lagged_sum|   9|   1000000|     100|0.02934697|  PASSED
156.      # The file file_input_raw was rewound 46 times
157.            rgb_lagged_sum|  10|   1000000|     100|0.42788867|  PASSED
158.      # The file file_input_raw was rewound 51 times
159.            rgb_lagged_sum|  11|   1000000|     100|0.28923095|  PASSED
160.      # The file file_input_raw was rewound 56 times
161.            rgb_lagged_sum|  12|   1000000|     100|0.47349623|  PASSED
162.      # The file file_input_raw was rewound 62 times
163.            rgb_lagged_sum|  13|   1000000|     100|0.46304823|  PASSED
164.      # The file file_input_raw was rewound 68 times
165.            rgb_lagged_sum|  14|   1000000|     100|0.11007021|  PASSED
166.      # The file file_input_raw was rewound 74 times
167.            rgb_lagged_sum|  15|   1000000|     100|0.00169033|   WEAK
168.      # The file file_input_raw was rewound 81 times
169.            rgb_lagged_sum|  16|   1000000|     100|0.63644568|  PASSED
170.      # The file file_input_raw was rewound 88 times
171.            rgb_lagged_sum|  17|   1000000|     100|0.36408121|  PASSED
172.      # The file file_input_raw was rewound 96 times
173.            rgb_lagged_sum|  18|   1000000|     100|0.30921802|  PASSED
174.      # The file file_input_raw was rewound 104 times
175.            rgb_lagged_sum|  19|   1000000|     100|0.00004455|   WEAK
176.      # The file file_input_raw was rewound 112 times
177.            rgb_lagged_sum|  20|   1000000|     100|0.52341676|  PASSED
178.      # The file file_input_raw was rewound 121 times
179.            rgb_lagged_sum|  21|   1000000|     100|0.30141023|  PASSED
180.      # The file file_input_raw was rewound 130 times
181.            rgb_lagged_sum|  22|   1000000|     100|0.52157880|  PASSED
182.      # The file file_input_raw was rewound 140 times
183.            rgb_lagged_sum|  23|   1000000|     100|0.04870754|  PASSED
184.      # The file file_input_raw was rewound 150 times
185.            rgb_lagged_sum|  24|   1000000|     100|0.00092570|   WEAK
186.      # The file file_input_raw was rewound 160 times
187.            rgb_lagged_sum|  25|   1000000|     100|0.06202059|  PASSED
188.      # The file file_input_raw was rewound 171 times
189.            rgb_lagged_sum|  26|   1000000|     100|0.96072691|  PASSED
190.      # The file file_input_raw was rewound 182 times
191.            rgb_lagged_sum|  27|   1000000|     100|0.59907067|  PASSED
192.      # The file file_input_raw was rewound 194 times
193.            rgb_lagged_sum|  28|   1000000|     100|0.85454998|  PASSED
194.      # The file file_input_raw was rewound 206 times
195.            rgb_lagged_sum|  29|   1000000|     100|0.00147815|   WEAK
196.      # The file file_input_raw was rewound 218 times
197.            rgb_lagged_sum|  30|   1000000|     100|0.87448505|  PASSED
198.      # The file file_input_raw was rewound 231 times
199.            rgb_lagged_sum|  31|   1000000|     100|0.00000000|  FAILED
200.      # The file file_input_raw was rewound 244 times
201.            rgb_lagged_sum|  32|   1000000|     100|0.41235925|  PASSED
202.      # The file file_input_raw was rewound 244 times
203.         rgb_kstest_test|   0|     10000|    1000|0.12020363|  PASSED
204.      # The file file_input_raw was rewound 245 times
205.          dab_bytedistrib|   0|  51200000|       1|0.42936613|  PASSED
206.      # The file file_input_raw was rewound 245 times
207.                  dab_dct| 256|     50000|       1|0.58804424|  PASSED
208.      Preparing to run test 207.  ntuple = 0
209.      # The file file_input_raw was rewound 246 times
210.             dab_filltree|  32|  15000000|       1|0.26053536|  PASSED
211.             dab_filltree|  32|  15000000|       1|0.76608133|  PASSED
212.      Preparing to run test 208.  ntuple = 0
213.      # The file file_input_raw was rewound 246 times
```

```
214.            dab_filltree2|   0|   5000000|       1|0.21950465|  PASSED
215.            dab_filltree2|   1|   5000000|       1|0.66672546|  PASSED
216.      Preparing to run test 209.  ntuple = 0
217.      # The file file_input_raw was rewound 246 times
218.             dab_monobit2|  12|  65000000|       1|0.54561807|  PASSED
219.
220.
221.
222.
223.      jorgealarcon@Michelangelo ~/TFM/Aleatorios $ cat
Aleatorios_Quantis_1GB.dat |rngtest
224.      rngtest 4
225.      Copyright (c) 2004 by Henrique de Moraes Holschuh
226.      This is free software; see the source for copying conditions.  There is
NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
227.
228.      rngtest: starting FIPS tests...
229.      rngtest: entropy source drained
230.      rngtest: bits received from input: 8000000000
231.      rngtest: FIPS 140-2 successes: 399693
232.      rngtest: FIPS 140-2 failures: 306
233.      rngtest: FIPS 140-2(2001-10-10) Monobit: 41
234.      rngtest: FIPS 140-2(2001-10-10) Poker: 27
235.      rngtest: FIPS 140-2(2001-10-10) Runs: 109
236.      rngtest: FIPS 140-2(2001-10-10) Long run: 129
237.      rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
238.      rngtest: input channel speed: (min=500000000.000; avg=4877715369.963;
max=0.000)bits/s
239.      rngtest: FIPS tests speed: (min=14.181; avg=60.868; max=61.927)Mibits/s
240.      rngtest: Program run time: 127055357 microseconds
241.      jorgealarcon@Michelangelo ~/TFM/Aleatorios $
242.
243.
```

ANEXO 4: Pruebas de aleatoriedad de 1 Gigabyte de datos generados con perturbaciones con el dispositivo Quantis-USB-4M.

```
1.  jorgealarcon@Michelangelo ~/TFM/Quantis $ ent Quantis_1G_3.3.dat
2.  Entropy = 8.000000 bits per byte.
3.
4.  Optimum compression would reduce the size
5.  of this 1000000000 byte file by 0 percent.
6.
7.  Chi square distribution for 1000000000 samples is 229.03, and randomly
8.  would exceed this value 75.00 percent of the times.
9.
10. Arithmetic mean value of data bytes is 127.4973 (127.5 = random).
11. Monte Carlo value for Pi is 3.141778789 (error 0.01 percent).
12. Serial correlation coefficient is 0.000037 (totally uncorrelated = 0.0).
```

```
13.
14.
15.
16. jorgealarcon@Michelangelo ~/TFM/Quantis $ dieharder -a -g 201 -f
Quantis_1G_3.3.dat
17. #=============================================================================#
18. #            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
19. #=============================================================================#
20.    rng_name    |           filename             |rands/second|
21.  file_input_raw|              Quantis_1G_3.3.dat|  4.57e+07  |
22. #=============================================================================#
23.        test_name   |ntup| tsamples |psamples|  p-value |Assessment
24. #=============================================================================#
25.    diehard_birthdays|   0|       100|     100|0.78322143|  PASSED
26.       diehard_operm5|   0|   1000000|     100|0.69847151|  PASSED
27.   diehard_rank_32x32|   0|     40000|     100|0.46067143|  PASSED
28. # The file file_input_raw was rewound 1 times
29.     diehard_rank_6x8|   0|    100000|     100|0.74359534|  PASSED
30. # The file file_input_raw was rewound 1 times
31.    diehard_bitstream|   0|   2097152|     100|0.93078883|  PASSED
32. # The file file_input_raw was rewound 2 times
33.         diehard_opso|   0|   2097152|     100|0.03219919|  PASSED
34. # The file file_input_raw was rewound 2 times
35.         diehard_oqso|   0|   2097152|     100|0.22421033|  PASSED
36. # The file file_input_raw was rewound 2 times
37.          diehard_dna|   0|   2097152|     100|0.77293994|  PASSED
38. # The file file_input_raw was rewound 2 times
39. diehard_count_1s_str|   0|    256000|     100|0.40695442|  PASSED
40. # The file file_input_raw was rewound 3 times
41. diehard_count_1s_byt|   0|    256000|     100|0.99765530|   WEAK
42. # The file file_input_raw was rewound 3 times
43.  diehard_parking_lot|   0|     12000|     100|0.33078595|  PASSED
44. # The file file_input_raw was rewound 3 times
45.     diehard_2dsphere|   2|      8000|     100|0.61591103|  PASSED
46. # The file file_input_raw was rewound 3 times
47.     diehard_3dsphere|   3|      4000|     100|0.23520652|  PASSED
48. # The file file_input_raw was rewound 4 times
49.      diehard_squeeze|   0|    100000|     100|0.99982554|   WEAK
50. # The file file_input_raw was rewound 4 times
51.         diehard_sums|   0|       100|     100|0.40550098|  PASSED
52. # The file file_input_raw was rewound 4 times
53.         diehard_runs|   0|    100000|     100|0.33562986|  PASSED
54.         diehard_runs|   0|    100000|     100|0.77709668|  PASSED
55.           # The file file_input_raw was rewound 5 times
56.                diehard_craps|   0|    200000|     100|0.99999035|   WEAK
57.                diehard_craps|   0|    200000|     100|0.29223026|  PASSED
58.           # The file file_input_raw was rewound 13 times
59.            marsaglia_tsang_gcd|   0|  10000000|     100|0.00003203|   WEAK
60.            marsaglia_tsang_gcd|   0|  10000000|     100|0.04850459|  PASSED
61.           # The file file_input_raw was rewound 13 times
62.                    sts_monobit|   1|    100000|     100|0.47350342|  PASSED
63.           # The file file_input_raw was rewound 13 times
64.                      sts_runs|   2|    100000|     100|0.10281705|  PASSED
65.           # The file file_input_raw was rewound 13 times
66.                    sts_serial|   1|    100000|     100|0.25592726|  PASSED
67.                    sts_serial|   2|    100000|     100|0.99961799|   WEAK
68.                    sts_serial|   3|    100000|     100|0.57623813|  PASSED
69.                    sts_serial|   3|    100000|     100|0.24334407|  PASSED
70.                    sts_serial|   4|    100000|     100|0.59769686|  PASSED
71.                    sts_serial|   4|    100000|     100|0.95339048|  PASSED
72.                    sts_serial|   5|    100000|     100|0.82468446|  PASSED
```

```
                    sts_serial|   5|    100000|     100|0.47439714|  PASSED
                    sts_serial|   6|    100000|     100|0.90414945|  PASSED
                    sts_serial|   6|    100000|     100|0.63365543|  PASSED
                    sts_serial|   7|    100000|     100|0.83931669|  PASSED
                    sts_serial|   7|    100000|     100|0.89460150|  PASSED
                    sts_serial|   8|    100000|     100|0.71430001|  PASSED
                    sts_serial|   8|    100000|     100|0.18303073|  PASSED
                    sts_serial|   9|    100000|     100|0.17523929|  PASSED
                    sts_serial|   9|    100000|     100|0.34497942|  PASSED
                    sts_serial|  10|    100000|     100|0.59975575|  PASSED
                    sts_serial|  10|    100000|     100|0.32090622|  PASSED
                    sts_serial|  11|    100000|     100|0.39826363|  PASSED
                    sts_serial|  11|    100000|     100|0.54095454|  PASSED
                    sts_serial|  12|    100000|     100|0.89274338|  PASSED
                    sts_serial|  12|    100000|     100|0.92363087|  PASSED
                    sts_serial|  13|    100000|     100|0.98826711|  PASSED
                    sts_serial|  13|    100000|     100|0.68511852|  PASSED
                    sts_serial|  14|    100000|     100|0.85101935|  PASSED
                    sts_serial|  14|    100000|     100|0.84377088|  PASSED
                    sts_serial|  15|    100000|     100|0.65496827|  PASSED
                    sts_serial|  15|    100000|     100|0.40918277|  PASSED
                    sts_serial|  16|    100000|     100|0.88748738|  PASSED
                    sts_serial|  16|    100000|     100|0.15428356|  PASSED
          # The file file_input_raw was rewound 13 times
                   rgb_bitdist|   1|    100000|     100|0.58076683|  PASSED
          # The file file_input_raw was rewound 13 times
                   rgb_bitdist|   2|    100000|     100|0.17905497|  PASSED
          # The file file_input_raw was rewound 13 times
                   rgb_bitdist|   3|    100000|     100|0.20531490|  PASSED
          # The file file_input_raw was rewound 13 times
                   rgb_bitdist|   4|    100000|     100|0.37471283|  PASSED
          # The file file_input_raw was rewound 14 times
                   rgb_bitdist|   5|    100000|     100|0.66494933|  PASSED
          # The file file_input_raw was rewound 14 times
                   rgb_bitdist|   6|    100000|     100|0.23637227|  PASSED
          # The file file_input_raw was rewound 15 times
                   rgb_bitdist|   7|    100000|     100|0.02501407|  PASSED
          # The file file_input_raw was rewound 16 times
                   rgb_bitdist|   8|    100000|     100|0.01089293|  PASSED
          # The file file_input_raw was rewound 16 times
                   rgb_bitdist|   9|    100000|     100|0.59950994|  PASSED
          # The file file_input_raw was rewound 17 times
                   rgb_bitdist|  10|    100000|     100|0.98092456|  PASSED
          # The file file_input_raw was rewound 18 times
                   rgb_bitdist|  11|    100000|     100|0.47889814|  PASSED
          # The file file_input_raw was rewound 19 times
                   rgb_bitdist|  12|    100000|     100|0.60541837|  PASSED
          # The file file_input_raw was rewound 19 times
          rgb_minimum_distance|   2|     10000|    1000|0.67468456|  PASSED
          # The file file_input_raw was rewound 19 times
          rgb_minimum_distance|   3|     10000|    1000|0.38596283|  PASSED
          # The file file_input_raw was rewound 19 times
          rgb_minimum_distance|   4|     10000|    1000|0.14330869|  PASSED
          # The file file_input_raw was rewound 19 times
          rgb_minimum_distance|   5|     10000|    1000|0.20236833|  PASSED
          # The file file_input_raw was rewound 20 times
              rgb_permutations|   2|    100000|     100|0.64411238|  PASSED
          # The file file_input_raw was rewound 20 times
              rgb_permutations|   3|    100000|     100|0.98124537|  PASSED
          # The file file_input_raw was rewound 20 times
              rgb_permutations|   4|    100000|     100|0.97689358|  PASSED
```

```
          # The file file_input_raw was rewound 20 times
              rgb_permutations|   5|    100000|     100|0.20752672|  PASSED
          # The file file_input_raw was rewound 20 times
                   rgb_lagged_sum|   0|   1000000|     100|0.87194545|  PASSED
          # The file file_input_raw was rewound 21 times
                   rgb_lagged_sum|   1|   1000000|     100|0.27232485|  PASSED
          # The file file_input_raw was rewound 22 times
                   rgb_lagged_sum|   2|   1000000|     100|0.86236231|  PASSED
          # The file file_input_raw was rewound 24 times
                   rgb_lagged_sum|   3|   1000000|     100|0.00297114|   WEAK
          # The file file_input_raw was rewound 26 times
                   rgb_lagged_sum|   4|   1000000|     100|0.35917354|  PASSED
          # The file file_input_raw was rewound 28 times
                   rgb_lagged_sum|   5|   1000000|     100|0.00428497|   WEAK
          # The file file_input_raw was rewound 31 times
                   rgb_lagged_sum|   6|   1000000|     100|0.83318975|  PASSED
          # The file file_input_raw was rewound 34 times
                   rgb_lagged_sum|   7|   1000000|     100|0.55520240|  PASSED
          # The file file_input_raw was rewound 38 times
                   rgb_lagged_sum|   8|   1000000|     100|0.31035304|  PASSED
          # The file file_input_raw was rewound 42 times
                   rgb_lagged_sum|   9|   1000000|     100|0.00793600|  PASSED
          # The file file_input_raw was rewound 46 times
                   rgb_lagged_sum|  10|   1000000|     100|0.17539079|  PASSED
          # The file file_input_raw was rewound 51 times
                   rgb_lagged_sum|  11|   1000000|     100|0.25214293|  PASSED
          # The file file_input_raw was rewound 56 times
                   rgb_lagged_sum|  12|   1000000|     100|0.76126620|  PASSED
          # The file file_input_raw was rewound 62 times
                   rgb_lagged_sum|  13|   1000000|     100|0.38876448|  PASSED
          # The file file_input_raw was rewound 68 times
                   rgb_lagged_sum|  14|   1000000|     100|0.02008053|  PASSED
          # The file file_input_raw was rewound 74 times
                   rgb_lagged_sum|  15|   1000000|     100|0.00222996|   WEAK
          # The file file_input_raw was rewound 81 times
                   rgb_lagged_sum|  16|   1000000|     100|0.03990944|  PASSED
          # The file file_input_raw was rewound 88 times
                   rgb_lagged_sum|  17|   1000000|     100|0.09513460|  PASSED
          # The file file_input_raw was rewound 96 times
                   rgb_lagged_sum|  18|   1000000|     100|0.56686008|  PASSED
          # The file file_input_raw was rewound 104 times
                   rgb_lagged_sum|  19|   1000000|     100|0.00778824|  PASSED
          # The file file_input_raw was rewound 112 times
                   rgb_lagged_sum|  20|   1000000|     100|0.10873647|  PASSED
          # The file file_input_raw was rewound 121 times
                   rgb_lagged_sum|  21|   1000000|     100|0.07295273|  PASSED
          # The file file_input_raw was rewound 130 times
                   rgb_lagged_sum|  22|   1000000|     100|0.12030369|  PASSED
          # The file file_input_raw was rewound 140 times
                   rgb_lagged_sum|  23|   1000000|     100|0.15245138|  PASSED
          # The file file_input_raw was rewound 150 times
                   rgb_lagged_sum|  24|   1000000|     100|0.00000022|  FAILED
          # The file file_input_raw was rewound 160 times
                   rgb_lagged_sum|  25|   1000000|     100|0.44303160|  PASSED
          # The file file_input_raw was rewound 171 times
                   rgb_lagged_sum|  26|   1000000|     100|0.76326174|  PASSED
          # The file file_input_raw was rewound 182 times
                   rgb_lagged_sum|  27|   1000000|     100|0.40090598|  PASSED
          # The file file_input_raw was rewound 194 times
                   rgb_lagged_sum|  28|   1000000|     100|0.37333823|  PASSED
          # The file file_input_raw was rewound 206 times
```

```
195.                rgb_lagged_sum|  29|   1000000|     100|0.00051041|   WEAK
196.          # The file file_input_raw was rewound 218 times
197.                rgb_lagged_sum|  30|   1000000|     100|0.21674080|  PASSED
198.          # The file file_input_raw was rewound 231 times
199.                rgb_lagged_sum|  31|   1000000|     100|0.00000003|  FAILED
200.          # The file file_input_raw was rewound 244 times
201.                rgb_lagged_sum|  32|   1000000|     100|0.81078374|  PASSED
202.          # The file file_input_raw was rewound 244 times
203.             rgb_kstest_test|   0|     10000|    1000|0.02517813|  PASSED
204.          # The file file_input_raw was rewound 245 times
205.              dab_bytedistrib|   0|  51200000|       1|0.96916662|  PASSED
206.          # The file file_input_raw was rewound 245 times
207.                       dab_dct| 256|     50000|       1|0.69408881|  PASSED
208.          Preparing to run test 207.  ntuple = 0
209.          # The file file_input_raw was rewound 246 times
210.                 dab_filltree|  32|  15000000|       1|0.52403711|  PASSED
211.                 dab_filltree|  32|  15000000|       1|0.49563135|  PASSED
212.          Preparing to run test 208.  ntuple = 0
213.          # The file file_input_raw was rewound 246 times
214.                dab_filltree2|   0|   5000000|       1|0.91373406|  PASSED
215.                dab_filltree2|   1|   5000000|       1|0.57789380|  PASSED
216.          Preparing to run test 209.  ntuple = 0
217.          # The file file_input_raw was rewound 246 times
218.                 dab_monobit2|  12|  65000000|       1|0.80113084|  PASSED
219.
220.
221.
222.          jorgealarcon@Michelangelo ~/TFM/Quantis $ cat Quantis_1G_3.3.dat |
rngtest
223.          rngtest 4
224.          Copyright (c) 2004 by Henrique de Moraes Holschuh
225.          This is free software; see the source for copying conditions.  There
is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
226.
227.          rngtest: starting FIPS tests...
228.          rngtest: entropy source drained
229.          rngtest: bits received from input: 8000000000
230.          rngtest: FIPS 140-2 successes: 399708
231.          rngtest: FIPS 140-2 failures: 291
232.          rngtest: FIPS 140-2(2001-10-10) Monobit: 39
233.          rngtest: FIPS 140-2(2001-10-10) Poker: 34
234.          rngtest: FIPS 140-2(2001-10-10) Runs: 122
235.          rngtest: FIPS 140-2(2001-10-10) Long run: 99
236.          rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
237.          rngtest: input channel speed: (min=500000000.000; avg=4902921903.613;
max=0.000)bits/s
238.          rngtest: FIPS tests speed: (min=14.428; avg=61.443;
max=62.332)Mibits/s
239.          rngtest: Program run time: 125891127 microseconds
240.          jorgealarcon@Michelangelo ~/TFM/Quantis $
241.
242.
```

ANEXO 5: Pruebas de aleatoriedad de 1 Gigabyte de datos generados sin perturbaciones con el dispositivo Quantis-USB-4M en MacOS.

```
1.        jorgealarcon@Michelangelo ~/TFM/Quantis $ dieharder -a -g 201 -f
random_mac.dat
2.        #=======================================================================
=====#
3.        #            dieharder version 3.31.1 Copyright 2003 Robert G.
Brown          #
4.        #=======================================================================
=====#
5.           rng_name    |           filename             |rands/second|
6.         file_input_raw|                 random_mac.dat|  3.04e+07  |
7.        #=======================================================================
=====#
8.                test_name   |ntup| tsamples |psamples|  p-value |Assessment
9.        #=======================================================================
=====#
10.          diehard_birthdays|   0|       100|     100|0.28149807|  PASSED
11.             diehard_operm5|   0|   1000000|     100|0.84411274|  PASSED
12.         diehard_rank_32x32|   0|     40000|     100|0.97093304|  PASSED
13.       # The file file_input_raw was rewound 1 times
14.            diehard_rank_6x8|   0|    100000|     100|0.56999483|  PASSED
15.       # The file file_input_raw was rewound 1 times
16.          diehard_bitstream|   0|   2097152|     100|0.72448209|  PASSED
17.       # The file file_input_raw was rewound 2 times
18.                diehard_opso|   0|   2097152|     100|0.32255472|  PASSED
19.       # The file file_input_raw was rewound 2 times
20.                diehard_oqso|   0|   2097152|     100|0.92905060|  PASSED
21.       # The file file_input_raw was rewound 2 times
22.                 diehard_dna|   0|   2097152|     100|0.20790352|  PASSED
23.       # The file file_input_raw was rewound 2 times
24.       diehard_count_1s_str|   0|    256000|     100|0.52884548|  PASSED
25.       # The file file_input_raw was rewound 3 times
26.       diehard_count_1s_byt|   0|    256000|     100|0.84139091|  PASSED
27.       # The file file_input_raw was rewound 3 times
28.        diehard_parking_lot|   0|     12000|     100|0.92308018|  PASSED
29.       # The file file_input_raw was rewound 3 times
30.           diehard_2dsphere|   2|      8000|     100|0.54289017|  PASSED
31.       # The file file_input_raw was rewound 3 times
32.           diehard_3dsphere|   3|      4000|     100|0.60898216|  PASSED
33.       # The file file_input_raw was rewound 4 times
34.            diehard_squeeze|   0|    100000|     100|0.92498942|  PASSED
35.       # The file file_input_raw was rewound 4 times
36.                diehard_sums|   0|       100|     100|0.52995370|  PASSED
37.       # The file file_input_raw was rewound 4 times
38.                diehard_runs|   0|    100000|     100|0.68469420|  PASSED
39.                diehard_runs|   0|    100000|     100|0.99874791|   WEAK
40.       # The file file_input_raw was rewound 4 times
41.              diehard_craps|   0|    200000|     100|0.16115788|  PASSED
42.              diehard_craps|   0|    200000|     100|0.71718463|  PASSED
43.       # The file file_input_raw was rewound 12 times
44.        marsaglia_tsang_gcd|   0|  10000000|     100|0.98894396|  PASSED
45.        marsaglia_tsang_gcd|   0|  10000000|     100|0.10967269|  PASSED
46.       # The file file_input_raw was rewound 12 times
47.                sts_monobit|   1|    100000|     100|0.19534566|  PASSED
48.       # The file file_input_raw was rewound 12 times
49.                  sts_runs|   2|    100000|     100|0.88955578|  PASSED
50.       # The file file_input_raw was rewound 12 times
51.                sts_serial|   1|    100000|     100|0.40099533|  PASSED
52.                sts_serial|   2|    100000|     100|0.35366532|  PASSED
53.                sts_serial|   3|    100000|     100|0.21437941|  PASSED
54.                sts_serial|   3|    100000|     100|0.25889779|  PASSED
55.                sts_serial|   4|    100000|     100|0.19154900|  PASSED
```

```
56.                sts_serial|   4|    100000|     100|0.88901006|  PASSED
57.                sts_serial|   5|    100000|     100|0.79153405|  PASSED
58.                sts_serial|   5|    100000|     100|0.29751438|  PASSED
59.                sts_serial|   6|    100000|     100|0.01137542|  PASSED
60.                sts_serial|   6|    100000|     100|0.26731400|  PASSED
61.                sts_serial|   7|    100000|     100|0.09059163|  PASSED
62.                sts_serial|   7|    100000|     100|0.28047596|  PASSED
63.                sts_serial|   8|    100000|     100|0.09771472|  PASSED
64.                sts_serial|   8|    100000|     100|0.16571270|  PASSED
65.                sts_serial|   9|    100000|     100|0.28750586|  PASSED
66.                sts_serial|   9|    100000|     100|0.90903881|  PASSED
67.                sts_serial|  10|    100000|     100|0.19687008|  PASSED
68.                sts_serial|  10|    100000|     100|0.41684357|  PASSED
69.                sts_serial|  11|    100000|     100|0.91871130|  PASSED
70.                sts_serial|  11|    100000|     100|0.18716932|  PASSED
71.                sts_serial|  12|    100000|     100|0.74705769|  PASSED
72.                sts_serial|  12|    100000|     100|0.52378096|  PASSED
73.                sts_serial|  13|    100000|     100|0.76071253|  PASSED
74.                sts_serial|  13|    100000|     100|0.98855063|  PASSED
75.                sts_serial|  14|    100000|     100|0.99853728|   WEAK
76.                sts_serial|  14|    100000|     100|0.97022238|  PASSED
77.                sts_serial|  15|    100000|     100|0.83515402|  PASSED
78.                sts_serial|  15|    100000|     100|0.97576765|  PASSED
79.                sts_serial|  16|    100000|     100|0.75059217|  PASSED
80.                sts_serial|  16|    100000|     100|0.90391712|  PASSED
81.      # The file file_input_raw was rewound 12 times
82.                rgb_bitdist|   1|    100000|     100|0.40994017|  PASSED
83.      # The file file_input_raw was rewound 12 times
84.                rgb_bitdist|   2|    100000|     100|0.99680157|   WEAK
85.      # The file file_input_raw was rewound 12 times
86.                rgb_bitdist|   3|    100000|     100|0.59247780|  PASSED
87.      # The file file_input_raw was rewound 12 times
88.                rgb_bitdist|   4|    100000|     100|0.12262329|  PASSED
89.      # The file file_input_raw was rewound 13 times
90.                rgb_bitdist|   5|    100000|     100|0.54132757|  PASSED
91.      # The file file_input_raw was rewound 13 times
92.                rgb_bitdist|   6|    100000|     100|0.81790736|  PASSED
93.      # The file file_input_raw was rewound 14 times
94.                rgb_bitdist|   7|    100000|     100|0.15852566|  PASSED
95.      # The file file_input_raw was rewound 14 times
96.                rgb_bitdist|   8|    100000|     100|0.60956698|  PASSED
97.      # The file file_input_raw was rewound 15 times
98.                rgb_bitdist|   9|    100000|     100|0.42425120|  PASSED
99.      # The file file_input_raw was rewound 16 times
100.               rgb_bitdist|  10|    100000|     100|0.56429410|  PASSED
101.     # The file file_input_raw was rewound 17 times
102.               rgb_bitdist|  11|    100000|     100|0.37716425|  PASSED
103.     # The file file_input_raw was rewound 18 times
104.               rgb_bitdist|  12|    100000|     100|0.44217341|  PASSED
105.     # The file file_input_raw was rewound 18 times
106.     rgb_minimum_distance|   2|     10000|    1000|0.76668788|  PASSED
107.     # The file file_input_raw was rewound 18 times
108.     rgb_minimum_distance|   3|     10000|    1000|0.10922889|  PASSED
109.     # The file file_input_raw was rewound 18 times
110.     rgb_minimum_distance|   4|     10000|    1000|0.91634933|  PASSED
111.     # The file file_input_raw was rewound 18 times
112.     rgb_minimum_distance|   5|     10000|    1000|0.53510005|  PASSED
113.     # The file file_input_raw was rewound 18 times
114.         rgb_permutations|   2|    100000|     100|0.72611202|  PASSED
115.     # The file file_input_raw was rewound 18 times
116.         rgb_permutations|   3|    100000|     100|0.79432198|  PASSED
```

```
117.      # The file file_input_raw was rewound 18 times
118.          rgb_permutations|   4|    100000|     100|0.64911723|  PASSED
119.      # The file file_input_raw was rewound 19 times
120.          rgb_permutations|   5|    100000|     100|0.93071163|  PASSED
121.      # The file file_input_raw was rewound 19 times
122.              rgb_lagged_sum|   0|   1000000|     100|0.87055455|  PASSED
123.      # The file file_input_raw was rewound 20 times
124.              rgb_lagged_sum|   1|   1000000|     100|0.72773424|  PASSED
125.      # The file file_input_raw was rewound 21 times
126.              rgb_lagged_sum|   2|   1000000|     100|0.74763114|  PASSED
127.      # The file file_input_raw was rewound 22 times
128.              rgb_lagged_sum|   3|   1000000|     100|0.67796946|  PASSED
129.      # The file file_input_raw was rewound 24 times
130.              rgb_lagged_sum|   4|   1000000|     100|0.00614653|  PASSED
131.      # The file file_input_raw was rewound 26 times
132.              rgb_lagged_sum|   5|   1000000|     100|0.13142993|  PASSED
133.      # The file file_input_raw was rewound 29 times
134.              rgb_lagged_sum|   6|   1000000|     100|0.50184257|  PASSED
135.      # The file file_input_raw was rewound 32 times
136.              rgb_lagged_sum|   7|   1000000|     100|0.01978389|  PASSED
137.      # The file file_input_raw was rewound 35 times
138.              rgb_lagged_sum|   8|   1000000|     100|0.95312302|  PASSED
139.      # The file file_input_raw was rewound 39 times
140.              rgb_lagged_sum|   9|   1000000|     100|0.03448497|  PASSED
141.      # The file file_input_raw was rewound 43 times
142.              rgb_lagged_sum|  10|   1000000|     100|0.09494492|  PASSED
143.      # The file file_input_raw was rewound 48 times
144.              rgb_lagged_sum|  11|   1000000|     100|0.43659754|  PASSED
145.      # The file file_input_raw was rewound 52 times
146.              rgb_lagged_sum|  12|   1000000|     100|0.27077132|  PASSED
147.      # The file file_input_raw was rewound 58 times
148.              rgb_lagged_sum|  13|   1000000|     100|0.35770895|  PASSED
149.      # The file file_input_raw was rewound 63 times
150.              rgb_lagged_sum|  14|   1000000|     100|0.00000026|  FAILED
151.      # The file file_input_raw was rewound 69 times
152.              rgb_lagged_sum|  15|   1000000|     100|0.00000005|  FAILED
153.      # The file file_input_raw was rewound 76 times
154.              rgb_lagged_sum|  16|   1000000|     100|0.45698805|  PASSED
155.      # The file file_input_raw was rewound 82 times
156.              rgb_lagged_sum|  17|   1000000|     100|0.20565599|  PASSED
157.      # The file file_input_raw was rewound 89 times
158.              rgb_lagged_sum|  18|   1000000|     100|0.29280436|  PASSED
159.      # The file file_input_raw was rewound 97 times
160.              rgb_lagged_sum|  19|   1000000|     100|0.94574074|  PASSED
161.      # The file file_input_raw was rewound 105 times
162.              rgb_lagged_sum|  20|   1000000|     100|0.87669699|  PASSED
163.      # The file file_input_raw was rewound 113 times
164.              rgb_lagged_sum|  21|   1000000|     100|0.92719746|  PASSED
165.      # The file file_input_raw was rewound 121 times
166.              rgb_lagged_sum|  22|   1000000|     100|0.63201012|  PASSED
167.      # The file file_input_raw was rewound 130 times
168.              rgb_lagged_sum|  23|   1000000|     100|0.01002788|  PASSED
169.      # The file file_input_raw was rewound 140 times
170.              rgb_lagged_sum|  24|   1000000|     100|0.00002819|   WEAK
171.      # The file file_input_raw was rewound 149 times
172.              rgb_lagged_sum|  25|   1000000|     100|0.96249270|  PASSED
173.      # The file file_input_raw was rewound 159 times
174.              rgb_lagged_sum|  26|   1000000|     100|0.99051068|  PASSED
175.      # The file file_input_raw was rewound 170 times
176.              rgb_lagged_sum|  27|   1000000|     100|0.53989525|  PASSED
177.      # The file file_input_raw was rewound 181 times
```

```
178.            rgb_lagged_sum|  28|   1000000|     100|0.12844444|  PASSED
179.      # The file file_input_raw was rewound 192 times
180.            rgb_lagged_sum|  29|   1000000|     100|0.00000001|  FAILED
181.      # The file file_input_raw was rewound 203 times
182.            rgb_lagged_sum|  30|   1000000|     100|0.88507780|  PASSED
183.      # The file file_input_raw was rewound 215 times
184.            rgb_lagged_sum|  31|   1000000|     100|0.00000131|   WEAK
185.      # The file file_input_raw was rewound 228 times
186.            rgb_lagged_sum|  32|   1000000|     100|0.63728670|  PASSED
187.      # The file file_input_raw was rewound 228 times
188.          rgb_kstest_test|   0|     10000|    1000|0.75398022|  PASSED
189.      # The file file_input_raw was rewound 228 times
190.           dab_bytedistrib|   0|  51200000|       1|0.76480053|  PASSED
191.      # The file file_input_raw was rewound 228 times
192.                   dab_dct| 256|     50000|       1|0.16383064|  PASSED
193.      Preparing to run test 207.  ntuple = 0
194.      # The file file_input_raw was rewound 229 times
195.             dab_filltree|  32|  15000000|       1|0.25172329|  PASSED
196.             dab_filltree|  32|  15000000|       1|0.06544507|  PASSED
197.      Preparing to run test 208.  ntuple = 0
198.      # The file file_input_raw was rewound 229 times
199.            dab_filltree2|   0|   5000000|       1|0.57404025|  PASSED
200.            dab_filltree2|   1|   5000000|       1|0.89627673|  PASSED
201.      Preparing to run test 209.  ntuple = 0
202.      # The file file_input_raw was rewound 229 times
203.             dab_monobit2|  12|  65000000|       1|0.85854775|  PASSED
204.      jorgealarcon@Michelangelo ~/TFM/Quantis $
205.
206.
207.      jorgealarcon@Michelangelo ~/TFM/Quantis $ dieharder -d 203 -n 31 -f
random_mac.dat
208.      #=========================================================================
=====#
209.      #            dieharder version 3.31.1 Copyright 2003 Robert G.
Brown          #
210.      #=========================================================================
=====#
211.         rng_name    |           filename             |rands/second|
212.              mt19937|                 random_mac.dat|  1.13e+08  |
213.      #=========================================================================
=====#
214.             test_name   |ntup| tsamples |psamples|  p-value |Assessment
215.      #=========================================================================
=====#
216.            rgb_lagged_sum|  31|   1000000|     100|0.27187083|  PASSED
217.      jorgealarcon@Michelangelo ~/TFM/Quantis $
218.
219.
```