Dan Beninson

Instituto de Tecnología Nuclear

EQUIPO GENERADOR DE PULSOS ALEATORIOS Y DETERMINÍSTICOS PARA EMULACIÓN DE UN SISTEMA DE DETECCIÓN NEUTRÓNICA Y AJUSTE DE CADENAS DE MEDICIÓN NEUTRÓNICA

CARRERA: ESPECIALIZACIÓN EN REACTORES NUCLEARES Y SU CICLO DE COMBUSTIBLE

Alumno: **Franco Nicolás Ferrucci**

Director: **Claudio Verrastro**

Mes y año: **Marzo 2012**

CNEA

UNSAM
UNIVERSIDAD NACIONAL DE SAN MARTÍN

ÍNDICE

RESUMEN

En este trabajo se presenta la implementación del pre-prototipo de un emulador de un sistema de detección neutrónica en modo pulso basado en tecnología FPGA (*Field Programmable Gate Array*, o arreglo de compuertas programables en campo).

El equipo emula la salida digital del circuito discriminador presente en una cadena de medición neutrónica, generando trenes de pulsos con tiempos de interarribo de distribución aleatoria o determinística, dependiendo de la selección del usuario. Los pulsos son de 3.3V, con un rango que va de 1 pulso/seg a 100 Kpulsos/seg. Además, es posible generar un perfil de crecimiento exponencial con una tasa de crecimiento seleccionable, posibilitando así la emulación de pulsos generados por un reactor nuclear en la etapa de arranque.

La emulación se centra en la generación de una versión sincrónica de un de un proceso de Poisson mediante ensayos de Bernoulli, lo que posibilita una implementación muy económica en términos de procesamiento requerido. Además, de modo de considerar los efectos de tiempo muerto presentes en todo sistema de detección en modo pulso, el equipo implementa los modelos de tiempo muerto *paralizables* y *no paralizables*, seleccionables por el operador.

Este equipo fue ideado con el fin de utilizarlo como *calibrador* de los equipos *medidores de flujo* y de *tasa de variación de flujo* (o *impulsímetro*) presentes en la cadena de instrumentación en régimen de pulsos de un reactor nuclear. Esta aplicación requiere que el equipo funcione de manera autónoma, sin necesidad de estar conectado a una PC, de modo de pasar a formar parte de la cadena de instrumentación de manera directa.

La definición de *pre-prototipo* se debe a que la implementación aquí presentada fue realizada en una placa de desarrollo genérica y no en un hardware fabricado especialmente para el proyecto. De este modo, un pre-prototipo constituye una *implementación conceptual*. La realización de un pre-prototipo permite:

- validar el algoritmo principal en el que el equipo final estará basado,
- dimensionar los componentes de hardware, como ser chips de procesamiento, controles presentes en el frente del equipo, etc.

La tesina está organizada de la siguiente manera. En el Capítulo 1 se presenta la instrumentación neutrónica, haciendo hincapié en los detectores neutrónicos en modo pulso; se describe la estadística que siguen estos detectores y se introduce el concepto de tiempo muerto. El Capítulo 2 describe el hardware utilizado para implementar el diseño. En el Capítulo 3 se detalla el algoritmo diseñado para emular la realización un proceso estocástico de características continuas mediante un equipo digital de funcionamiento sincrónico. En el Capítulo 4 se discuten los resultados y en el Capítulo 5 se enuncian las conclusiones.

1. PRESENTACIÓN

1.1 OBJETIVOS

El objetivo planteado en esta tesina fue la de realizar el desarrollo conceptual de un emulador de un sistema de detección neutrónica en modo pulso con el fin de utilizarlo como *calibrador* de los equipos *medidores de flujo* y de *tasa de variación de flujo* (o *impulsímetro*) presentes cualquier cadena de instrumentación que opere en régimen de pulsos.

Dada la aplicación planteada, el equipo debe tener un funcionamiento autónomo, sin necesidad de estar conectado a una PC, de modo que pueda ser adosado a la cadena de instrumentación de manera directa.

El equipo debe generar pulsos aleatorios con una distribución equivalente a la encontrada en las cadenas neutrónicas de régimen de pulsos. Además, como también se pretende la emulación de un reactor nuclear en su etapa de arranque, el equipo debe tener un modo de operación donde dichos pulsos aleatorios tengan una tasa de crecimiento exponencial.

Cabe aclarar que no existe en el mercado *ningún* equipo que combine estas dos características planteadas: generador de pulsos aleatorios con una tasa de pulsos que siga un perfil exponencial.

1.2 INTRODUCCIÓN

Los detectores neutrónicos tienen un rol fundamental en la instrumentación de un reactor nuclear ya que ellos son pieza fundamental de los sistemas de control y de seguridad.

Entre los detectores de radiación existe una distinción fundamental en sus modos de operación; ellos son *modo pulso, modo corriente* y *modo Campbel* y se encuentran explicados en el Capítulo 4 de [1].

En el *modo pulso*, el instrumento está preparado para detectar cada uno de los cuantos individuales de radiación que interactúa con el detector, produciendo a la salida una señal digital que dé cuenta de cuándo se produjo el evento de interacción. El factor limitante de este modo de operación es que debe existir un tiempo mínimo de separación de dos eventos para que el sistema lo pueda detectar como eventos diferentes, lo que hace que el *modo pulso sea* útil hasta un cierto valor máximo de flujo. Para flujos más altos, el tiempo adyacente entre eventos se hace muy corto. En estas situaciones se utilizan los detectores en *modo corriente* y *modo Campbel*, este último también llamado *modo fluctuación*. Para este último, la idea básica radica en promediar en el tiempo la corriente generada por la radiación incidente de modo de obtener a la salida una señal analógica proporcional al flujo.

En las siguientes Secciones se desarrollará con más detalle el *modo pulso* de operación ya que en él se centra el equipo aquí presentado.

1.2.1 DEFINICIONES DE TÉRMINOS

Antes de continuar es necesario aclarar ciertos términos que se utilizarán a lo largo de la tesina de modo de evitar confusión o ambigüedades.

Flujo neutrónico

El flujo neutrónico se refiere al número de neutrones que atraviesan una unidad de área en la unidad de tiempo, medido usualmente en $cm^{-2}seg^{-1}$. El flujo neutrónico no es constante dentro del reactor nuclear y determinar su variación espacial es muy importante ya que determina, junto a otras variables, la distribución espacial de potencia del reactor. Por esta razón es que se colocan numerosos detectores tanto dentro como fuera del núcleo del

reactor. Este arreglo de sensores permite entonces hacer una estimación de la potencia instantánea del reactor.

Valores comunes de flujo neutrónico van desde unos pocos $cm^{-2}seg^{-1}$ a valores en el orden de 10^{13} $cm^{-2}seg^{-1}$, por lo que se han desarrollado diferentes tipos de detectores de modo de medir el rango completo del flujo.

Para un flujo bajo, los detectores que ayudan a determinarlo son los de *modo pulso*. Así, existe para cada detector una función que relaciona el flujo neutrónico con la cantidad de pulsos de corriente por unidad de tiempo generados en la salida del detector.

Tasa

La *tasa* es, según su definición, la relación entre dos magnitudes. El uso que se le da a este término en la tesina es para referirse a la cantidad de pulsos por unidad de tiempo producidos en un detector o en la salida de los circuitos que le siguen a él. Por esta razón es que también se puede llamar ***tasa de cuenta***.

Debido a que en los detectores en modo pulso existe una relación directa entre el flujo neutrónico y la tasa de cuenta, estos términos se usarán indistintamente.

Por otro lado, muchas veces es necesario conocer cómo varía esta tasa con el tiempo. A dicha magnitud se la llamará ***tasa de variación*** o ***tasa de variación de flujo***, y tiene una relación directa con el ***período de duplicación.***

Para los casos en que el flujo tenga una variación exponencial de la forma:

$$f(t) = f_0 \,.\, e^{\Phi.t},$$

al término Φ se lo denomina ***tasa de variación relativa****, pues se da que:*

$$\left.\frac{df}{dt}\right/_f = \Phi$$

La unidad de φ es seg^{-1}; sin embargo, muchas veces es expresada de manera porcentual $\%{\cdot}seg^{-1}$ (porciento por segundo).

Se debe tener en cuenta que en muchos ámbitos a la *tasa de variación relativa* se la llama simplemente *tasa*. Esto puede llevar a confusiones pues, como ya se mencionó, aquí la *tasa* es la magnitud relaciona con el flujo, y no con su variación relativa.

1.3 CADENA DE INSTRUMENTACIÓN NUCLEAR

1.3.1 MODELO DEL DETECTOR

En el modelo más simplificado de un detector, una única partícula o cuanto de radiación incidente produce una corriente cuya integral (que es la carga *Q*) está directamente relacionada con la energía depositada en el detector. Dicha energía además está relacionada con el tipo de radiación que interactuó con el detector (e.g. neutrón, rayo γ, etc.). De este modo, cada pulso de corriente producido da información del tiempo en que se generó el evento de interacción y del tipo de radiación incidente involucrado. Estos pulsos pasan finalmente por un circuito conformador y un circuito discriminador; este último permite descartar pulsos de entrada que hayan sido producido por radiación que no se esté interesado en medir; demás, dicho circuito produce una señal de salida con tensiones de rango estándar de los circuitos lógicos.

En la Figura 1.1 se muestra una posible evolución de la corriente de salida de un detector. Allí se observa que las distintas áreas de cada pulso provienen de eventos con diferentes energías detectadas. Es importante remarcar que si bien la interacción incidente tiene una duración prácticamente instantánea, la corriente que ésta genera sí se extiende

por un tiempo determinado. Esto hace que para flujos neutrónicos altos se pueda dar la situación que la corriente que esté circulando por el detector provenga de más de una interacción; sin embargo, si asumimos flujos bajos, esta situación nombrada se dará con una probabilidad muy baja.

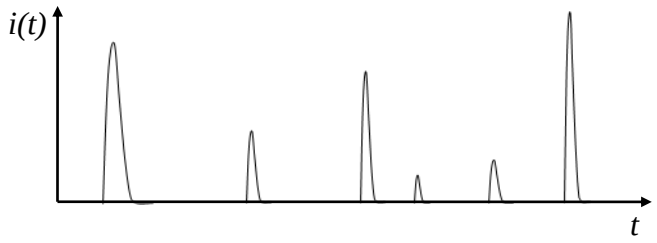


Figura 1.1. Ejemplo de evolución de la corriente de salida *de un detector de flujo neutrónico.*

Se debe resaltar también que el arribo de los cuantos de radiación al detector es un fenómeno aleatorio caracterizado por una estadística de *Poisson.* Así, si la probabilidad de que cada pulso de corriente del detector corresponda a un *único* arribo de radiación es muy alta, entonces $i(t)$ también tendrá una distribución de *Poisson*. Esto quiere decir que los tiempo de interarribo de los pulsos de $i(t)$ serán, en principio, variables aleatorias independientes y de distribución exponencial. Este punto es esencialmente importante para el diseño del equipo.

1.3.2 CIRCUITO CONFORMADOR + DISCRIMINADOR

Como ya se nombró anteriormente, seguido al detector se encuentran el conformador de pulsos y el discriminador, de modo de obtener una salida lógica con pulsos de una tensión constante. El discriminador incluye además un circuito monoestable previo a su salida que hace que todos los pulsos tengan la misma duración. La necesidad que los pulsos tengan la misma duración se justifica en la siguiente Sección, donde se hace el análisis de los tiempos muertos en los sistemas de detección. En la Figura 1.2 se observa un ejemplo de salida del conjunto detector-conformador-discriminador.

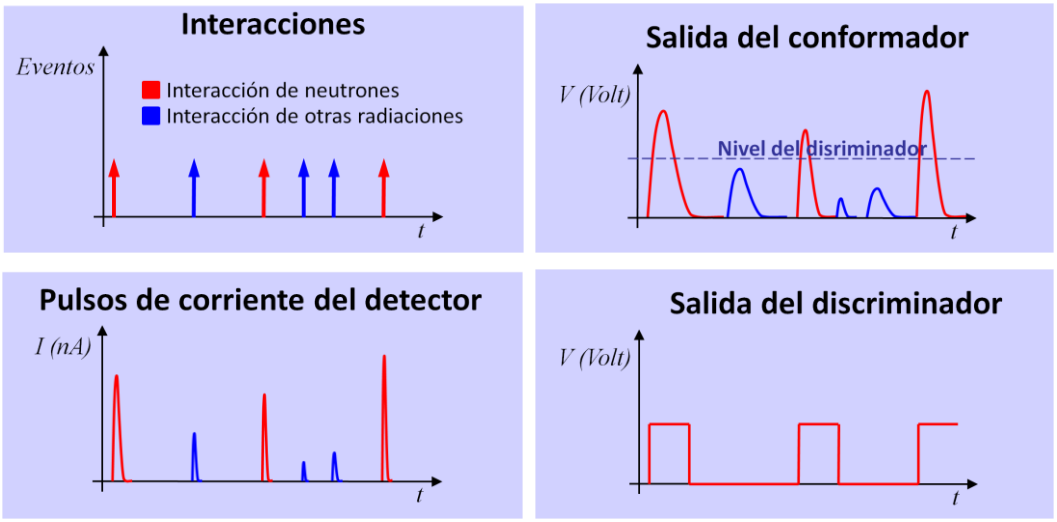


Figura 1.2. Ejemplo de evolución de la salida del conjunto detector-conformador de pulsos-discriminador.

1.4 TIEMPO MUERTO

Para que dos eventos de interacción puedan ser detectados como dos pulsos diferentes, aquéllos deberán estar separados por un tiempo mínimo. Dicho tiempo puede ser impuesto por el detector propiamente dicho, por la electrónica asociada al detector (*e.g.* circuito conformador - discriminador) o por una combinación de ambos. Este tiempo mínimo de separación se denomina *tiempo muerto*. Debido a la naturaleza aleatoria de la radiación, siempre habrá alguna probabilidad que un evento incidente sea perdido debido a que su llegada ocurre demasiado rápido luego de un evento anterior.

Las pérdidas por tiempo muerto pueden ser grandes en los casos de medición de altos valores de flujo neutrónico y por esta razón es muy importante caracterizar este fenómeno. Para ello se han desarrollado dos modelos de sistemas con tiempo muerto: *modelo paralizable* y *modelo no paralizable*. Estos modelos representan un comportamiento idealizado y en general los sistemas reales tienen características de ambos.

En la Figura 1.3 se muestra el comportamiento de los modelos frente a una sucesión de eventos que llega al detector. En ambos casos el arribo de un evento a la entrada hace que la salida se mantenga en alto durante un tiempo τ, llamado *tiempo muerto.* Los modelos difieren en qué sucede cuando llegan a la entrada eventos con una separación temporal menor a τ. En el sistema *no paralizable* los eventos arribados durante el tiempo muerto son perdidos y no quedan indicios de ellos en la salida. De la Figura se observa que de los 6 eventos que ingresan al sistema no paralizable, sólo 4 de ellos son acusados a la salida, pues 2 de esos 6 eventos ocurrieron en un tiempo menor a τ en relación al evento que generó el tiempo muerto correspondiente. En el sistema *paralizable,* los eventos que ocurren durante el tiempo muerto del sistema, si bien no son registrados como cuentas, extienden por un valor de τ el tiempo muerto de la salida. La Figura muestra que en el sistema paralizable sólo 3 cuentas son registradas, pues de los 6 eventos de entradas, 3 de ellos tienen una separación menor a τ con respecto al evento anterior.

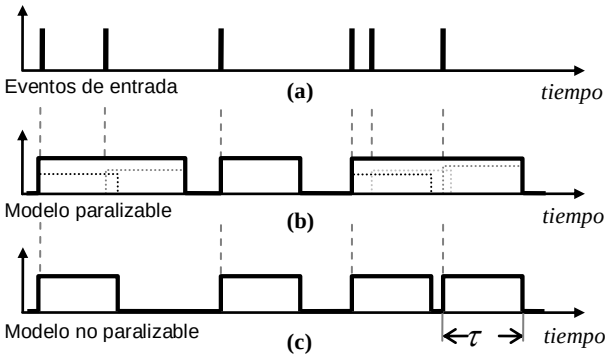


Figura 1.3. Modelos de tiempo muerto, ejemplo de sus comportamientos. (a) Secuencia de eventos de entrada. (b) Salida del modelo paralizable. (c) Salida del modelo no paralizable.

En el capítulo 4 de [1] se hace un estudio estadístico de estos sistemas y se llega a que, para el *modelo no paralizable* la relación entre la tasa de arribo de interacciones y la tasa de cuentas registradas a la salida es:

$$\lambda_{OUT} = \frac{\lambda_{IN}}{1 + \tau \cdot \lambda_{IN}} \ , \qquad (1\text{-}1)$$

mientras que para el *modelo paralizable*, la relación es:

$$\lambda_{OUT} = \lambda_{IN} \cdot e^{-\lambda_{IN} \cdot \tau} \qquad (1\text{-}2)$$

En ambos casos τ es la duración del tiempo muerto, λ_{IN} hace referencia a la tasa de eventos a la entrada y λ_{OUT} es la tasa de cuentas registradas a la salida del modelo. En la Figura 1.4 se ven graficadas las ecuaciones (1-1) y (1-2).

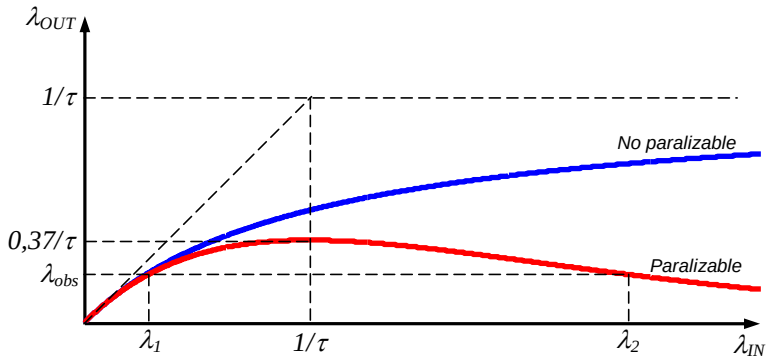


Figura 1.4. Relación entre tasa de eventos de entrada y tasa de cuentas registradas por los modelos paralizables y no paralizables.

Cuando las tasas son bajas los dos modelos dan prácticamente el mismo resultado; situación que no se mantiene para tasas elevadas. El sistema *no paralizable* alcanza un valor asintótico de tasa igual a $1/\tau$, que representa la situación en que el contador, apenas finalizado el tiempo muerto de un evento, arriba otro evento a su entrada haciéndolo producir un nuevo tiempo muerto. En el comportamiento *paralizable* se observa que a medida que se incrementa la tasa de entrada, también lo hace la tasa de salida, pero sólo hasta un valor máximo, comenzando luego a decrecer para valores de tasa de entrada cada vez mayor. Esto hace que para una determinada tasa observada, existan dos posibles valores de tasa de entrada (ver λ_{obs}, λ_1 y λ_2 en la Figura 1.4*).* Este funcionamiento puede resultar muy peligroso, pues se puede dar el caso de que la tasa real sea muy alta, pero la tasa de cuentas medida sea baja. Este peligro desaparece si, a partir de determinado flujo comienzan a funcionar otros medidores de flujo que funcionen en modo corriente o modo fluctuación.

En la Sección 1.3.2 se nombró la necesidad de que el circuito conformador fije un tiempo muerto constante para generar su salida, logrado en general mediante un circuito monoestable. El tiempo del monoestable debe ser mayor a todos los tiempos muertos presentes en los componentes del sistema (tanto del detector como del conformador de pulsos). Sin dicho monoestable es muy probable que la duración del tiempo muerto no sea constante y dependa de la amplitud o la forma del pulso de corriente de entrada. En esa situación no sería posible utilizar las *ec.* (1-1) y (1-2) *p*ara aplicar una corrección por tiempo muerto.

1.5 ESTADÍSTICA DE LOS SISTEMAS CON TIEMPO MUERTO. PROCESO DE POISSON

Como se dijo al inicio de la Sección 1.3.1, bajo determinada condición se puede asumir que los pulsos de salida del detector tienen una distribución de *Poisson*. Sin embargo, esta distribución queda alterada luego de pasar por el sistema conformador de pulsos. Es importante analizar entonces cuál será la distribución resultante, pues ello tendrá utilidad a la hora de verificar el funcionamiento del equipo.

Antes de continuar es necesario dar una descripción de un proceso de Poisson. El *proceso de Poisson* es un proceso estocástico en el cual ocurren eventos aleatorios de manera continua en el tiempo, e independientes unos de otros. El tiempo entre un par de eventos consecutivos tiene una *distribución exponencial* con parámetro λ y se asume que cada uno de estos tiempos de interarribo es independiente entre sí [6]. Existen numerosos

fenómenos naturales que pueden ser satisfactoriamente modelados por un proceso de Poisson. Por ejemplo, en un reactor, el número de impactos de neutrones sobre un detector de flujo neutrónico en un determinado tiempo se modela satisfactoriamente mediante un proceso de Poisson.

Las ec. (1-3) y (1-4) enuncian respectivamente la función de densidad de probabilidad *pdf* y la función de distribución *cdf* de los tiempos de interarribo de eventos de Poisson.

$$pdf_f = \lambda \cdot e^{-\lambda \cdot t}, \quad t \geq 0 \tag{1-3}$$

$$cdf_f = 1 - e^{-\lambda \cdot t}, \quad t \geq 0 \tag{1-4}$$

Dicho esto, consideremos el esquema de la Figura 1.5, donde se observa un sistema con tiempo muerto –paralizable o *no paralizable*– caracterizado por τ, y una señal de entrada *f(t)* que se asume *Poisson*, de parámetro λ.

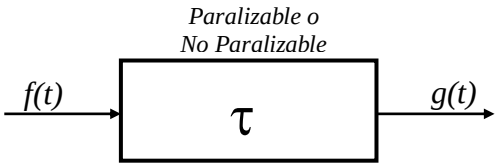


Figura 1.5. Diagrama de bloques de un sistema con tiempo muerto cuya señal de entrada tiene una distribución de Poisson.

En [3] se puede encontrar un estudio detallado de la modificación de la estadística que sufre una señal al pasar por sistemas paralizables y no paralizables. Allí se llega que para sistemas *no-paralizables* la *pdf* y *cdf* resultan:

$$pdf_{g\,NP} = \lambda \cdot e^{-\lambda \cdot (t-\tau)}, \; t \geq \tau \tag{1-5}$$

$$cdf_{g\,NP} = 1 - e^{-\lambda \cdot (t-\tau)}, \; t \geq \tau \tag{1-6}$$

$$PDF_{g\,NP} = \frac{\lambda}{s+\lambda} \cdot e^{-\tau \cdot s} \tag{1-7}$$

donde $PDF_{g\,NP}(s)$ corresponde a la transformada de *Laplace* de la función de densidad de probabilidad. Vemos que el sistema no-paralizable produce un simple corrimiento de la función de probabilidad de la señal de entrada. Para los sistemas *paralizables*, las funciones de distribución son más complicadas y vienen dadas por:

$$pdf_{g\,P}(t) = \lambda \cdot e^{-\lambda \cdot (t-\tau)} \cdot \sum_{j=1}^{floor(t/\tau)} \frac{\mu(t - j\tau) \cdot (-\lambda)^{j-1} \cdot (t - j\tau)^{j-1} \cdot e^{-(j-1)\lambda\tau}}{(j-1)!} \tag{1-8}$$

$$cdf_{g\,P} = -\sum_{k=1}^{floor(t/\tau)} \frac{\left(-\lambda \cdot e^{-\lambda \cdot \tau} \cdot (t - k\tau) \cdot \mu(t - k\tau)\right)^k}{k!} \tag{1-9}$$

donde $\mu(t)$ corresponde a la función escalón unitario. La transformada de *Laplace* para la *pdf* resulta:

$$PDF_{gP} = \frac{\lambda \cdot e^{-\tau \cdot (s+\lambda)}}{s + \lambda \cdot e^{-\tau \cdot (s+\lambda)}}, \qquad (1\text{-}10)$$

La Figura 1.6 *y* Figura 1.7 muestran respectivamente las gráficas de las funciones *pdf* y *cdf* mencionadas.

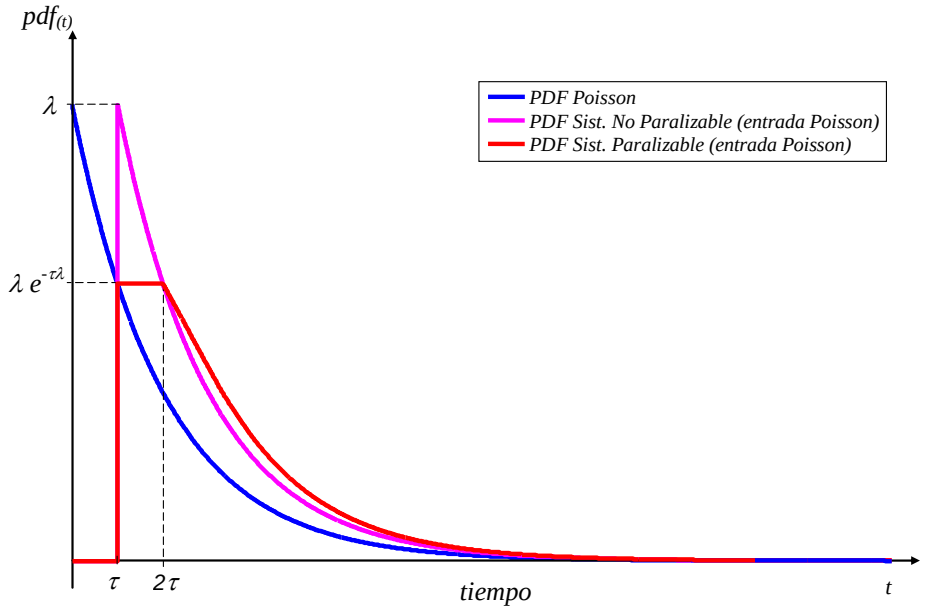


Figura 1.6. Gráficas de las funciones PDF de un proceso de Poisson y de la salida de un sistema paralizable (en rojo) y no-paralizable (en rosa); ambos sistemas poseen como entrada un proceso de Poisson.

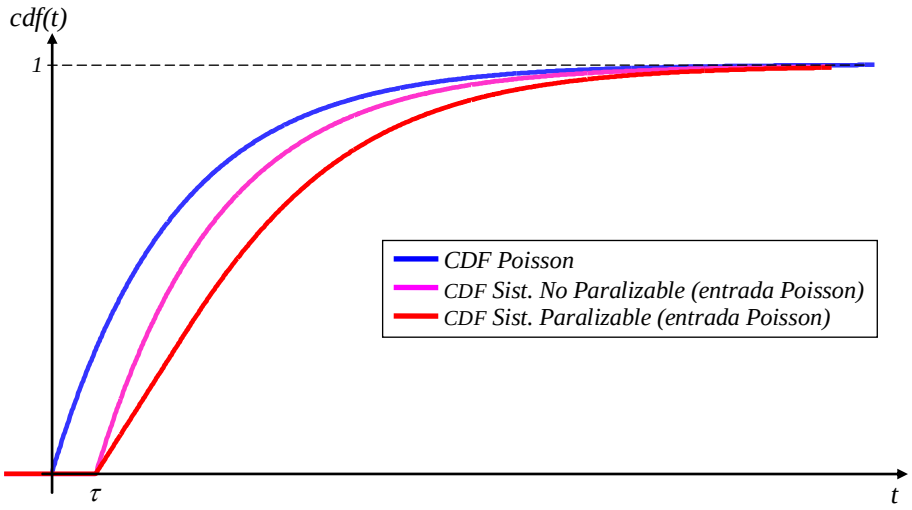


Figura 1.7. Gráficas de las funciones CDF correspondientes a la Figura 1.6.

Esta modificación que sufre la distribución exponencial al pasar por un sistema que inserta un tiempo muerto será tenida en cuenta en la Sección 0 al analizar los resultados del funcionamiento del equipo.

1.6 ANTECEDENTES

El pre-prototipo presentado aquí se basa en el equipo CNEA E023, *"Emulador de Pulsos Neutrónicos"* [2], también realizado por el autor de la presente tesina. Dicho equipo permite emular pulsos aleatorios con un perfil de flujo parametrizado a través de un software en una PC.

El contexto de este diseño era la verificación del funcionamiento de un equipo llamado *impulsímetro digital*, realizado por la Ing. Gloria Ríos. Así, el emulador posibilita su testeo sin necesidad que tuviera que ser llevado a campo y ser conectado a un detector real. En la Figura 1.8 se muestran el esquema de conexión del impulsímetro tanto en la etapa de testeo como en su funcionamiento en campo.

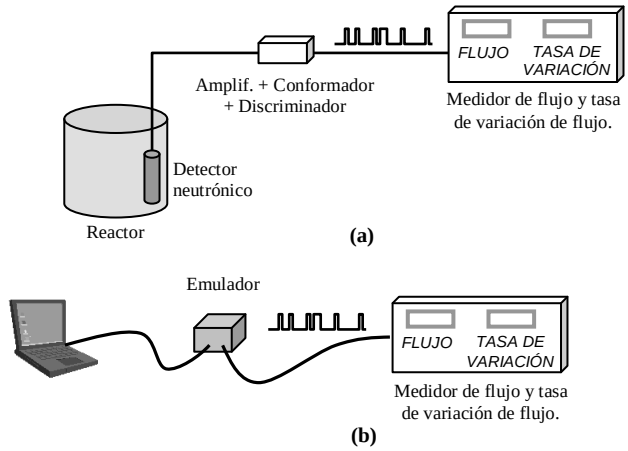


Figura 1.8. Diagrama de conexión de una cadena neutrónica en modo pulso. (a) Conectado a campo. (b) Conectado al equipo emulador

En la Figura 1.9 se muestra una fotografía del equipo E023. Allí se observa la presencia de un conector USB a través del cual el equipo es controlado. La Figura 1.10 muestra una pantalla del software de control; en ella se puede apreciar la gráfica de un ejemplo de perfil de flujo programado.

Figura 1.9. Equipo E023, antecedente del pre-prototipo presentado en esta tesina.

El equipo E023 es considerado un instrumento de laboratorio, por su máxima flexibilidad en el seteo de los parámetros, pudiendo emular perfiles de flujo arbitrarios. Esto lo hace ideal para la *validación* de los equipos *impulsímetros,* encargados de estimar el flujo.

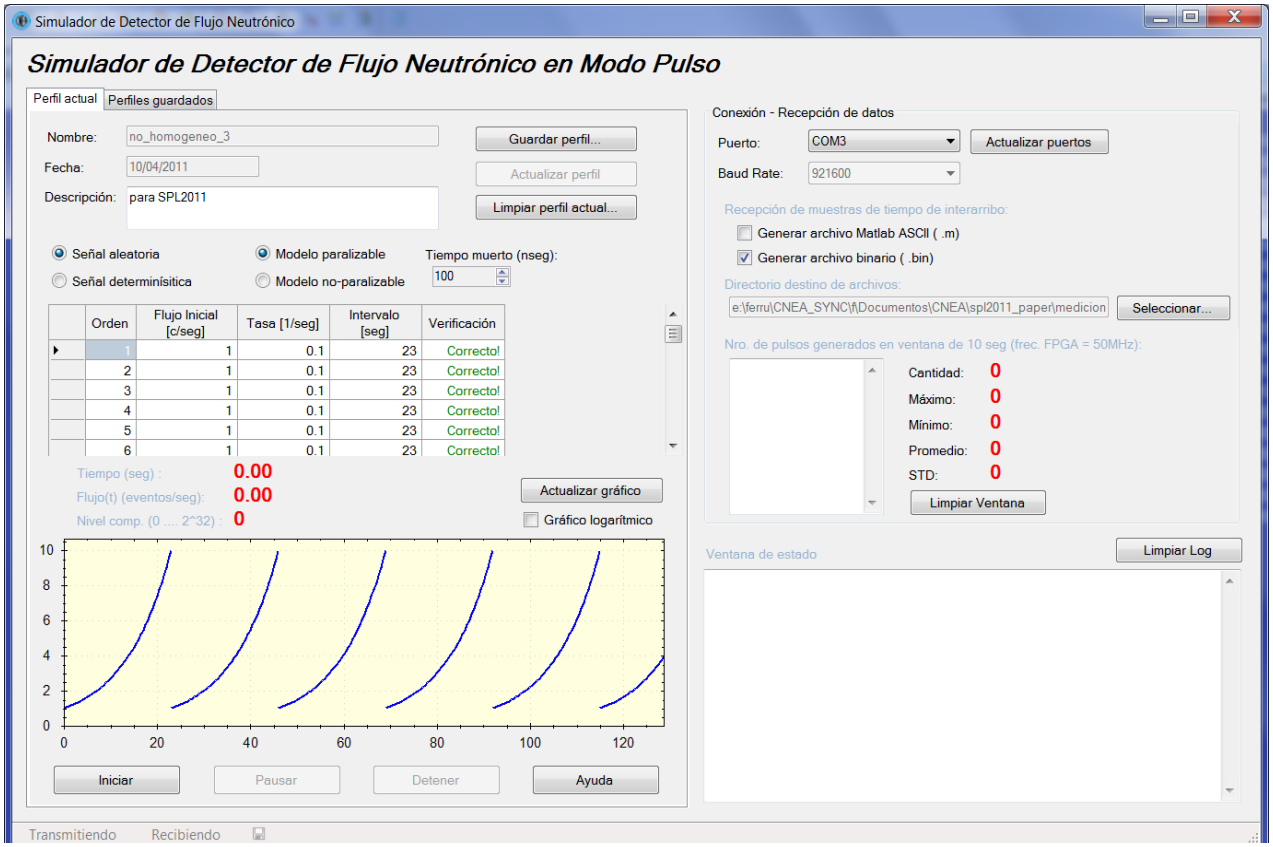


Figura 1.10. Pantalla del software que controla al equipo E023. Dicho equipo es el antecedente del pre-prototipo presentado en esta tesina.

Para que dicho equipo pueda emular un perfil totalmente arbitrario, instante a instante recibe desde la PC un parámetro está relacionado con el flujo instantáneo correspondiente. Este modo de funcionamiento no es compatible con el uso del equipo presentado en esta tesina, el cual es formar parte de una cadena de instrumentación en régimen de pulsos de modo de calibrar los equipos de medición de flujo y tasa de variación de flujo, por lo que requiere tener un funcionamiento totalmente autónomo.

2. DESCRIPCIÓN DEL HARDWARE

En esta Sección se describe el hardware utilizado para la implementación del equipo. Se comienza con la presentación de la tecnología FPGA (*Field Programmable Gate Array*, o arreglo de compuertas programables en campo) y luego se detalla el kit o la placa de desarrollo donde se programó el diseño.

2.1 FPGA, DESCRIPCIÓN

Las FPGA son circuitos integrados digitales que contienen bloques configurables (o programables) de lógica, interconectados a través de uniones también configurables. Los bloques lógicos pueden ser programados para realizar funciones combinacionales complejas o simples compuertas lógicas, como ser AND, OR ó XOR. En muchas FPGAs, estos bloques lógicos también incluyen de elementos de memoria; ellos van de simples flip-flops a unidades de memoria más complejas [4].

Las FPGAs pueden ser configuradas (o programadas) de modo de ser utilizadas en variadísimas aplicaciones, como ser procesamiento de señales, comunicaciones, audio, video, instrumentación industrial y científica, etc.

Dependiendo de la manera en que ellas son implementadas, algunas FPGAs pueden ser programadas sólo una vez (llamadas *OTP: one-time programmable*), mientras que otras pueden ser reprogramadas múltiples veces, cada vez que se necesite modificar su comportamiento.

La denominación *"programable en campo"* se refiere al hecho que la programación se realiza *en campo*, en contraposición de la "*programación en fábrica"*. Esto significa que las FPGAs pueden ser programadas en un laboratorio donde se está desarrollando el producto, o incluso (re)programadas cuando el equipo ha sido entregado al cliente.

La Figura 2.1 muestra un diagrama conceptual de una FPGA. Los signos de interrogación azules indican los bloques lógicos configurable, mientras que los rojos a las interconexiones.

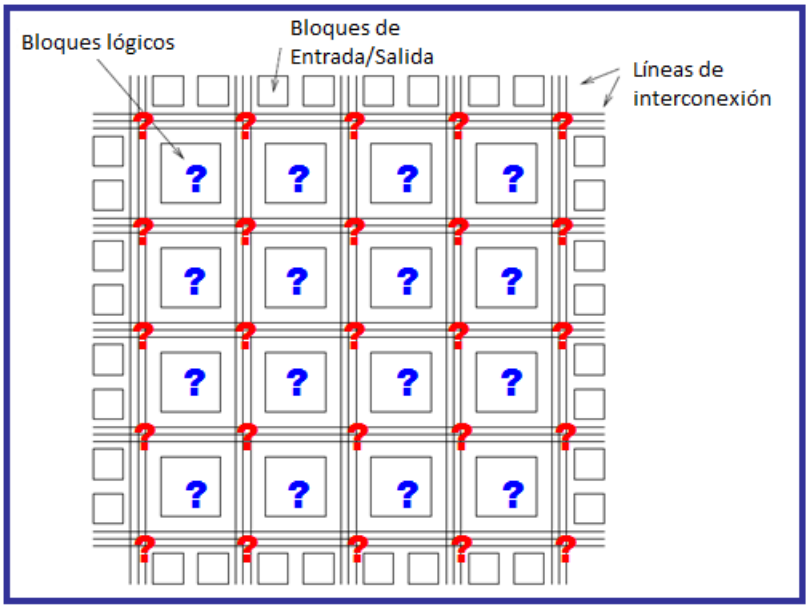


Figura 2.1. Esquema general de una FPGA.

En la Figura 2.2 se presenta un ejemplo de las posibilidades de interconexión que posee la serie Spartan-3 de la compañía Xilinx.

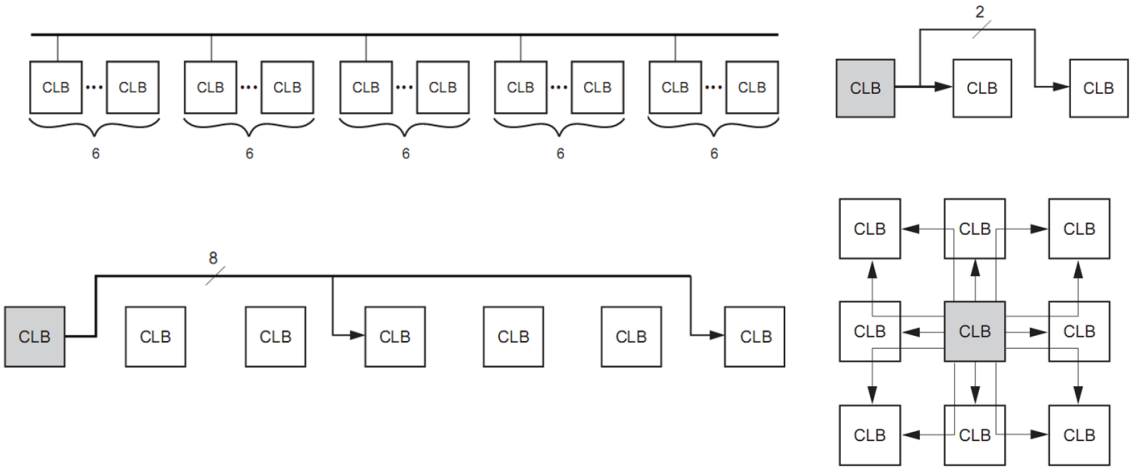


Figura 2.2. Ejemplo de posibilidades de interconexión de bloques lógicos en una FPGA "Spartan-3" de la compañía Xilinx. Los CLB hacen referencia a los bloques lógicos.

Es importante destacar que las FPGA funcionan bajo un paradigma diferente al los microprocesadores ya que no existe una unidad central de procesamiento o CPU que tome instrucciones de una memoria para decodificarlas y ejecutarlas; sólo existe un gran arreglo de celdas lógicas disponible para ser configuradas e interconectadas de la manera que se necesite. Sin embargo, dada la flexibilidad que poseen, es posible implementar un microprocesador utilizando dichas celdas lógicas. A este tipo de microprocesadores *sintetizados* dentro de una FPGA se los denomina *Soft Cores*.

Muchos modelos de FPGA disponen, además de la topología regular mencionada, otros periféricos que resultan de mucha utilidad, como ser:

- *bloques de memoria RAM*
- *memoria Flash integrada*
- *unidades de procesamiento aritmético (como ser multiplicadores)*
- *unidades de DSP (procesamiento digital de señales)*
- *conversores analógico-digital*
- *sintetizador de frecuencias arbitrarias (PLL: phase-locked loop)*

2.2 FPGA, FLUJO DE DISEÑO

La Figura 2.3 muestra los pasos que se siguen en el proceso de diseño utilizando dispositivos FPGA.

El flujo comienza en la *Entrada de Diseño*, donde se utiliza un *Lenguaje de Descripción de Hardware* para especificar el comportamiento que tendrá el dispositivo. Existen dos lenguajes principales utilizados: *VHDL* y *Verilog*. El código realizado en estos lenguajes se escribe en una computadora utilizando un software que interpreta el código ingresado. En general se dispone de paquetes de software que permiten detectar errores de sintaxis y hacer simulaciones sobre el comportamiento final que tendrá el diseño.

El siguiente paso se denomina *Síntesis Lógica*, en donde se realiza la transformación del código introducido a otro código de un nivel de *abstracción menor*. En este proceso se tiene en cuenta la tecnología específica de la FPGA destino. Así, si bien la *entrada de diseño* puede estar planteada para una FPGA genérica, la síntesis genera un código exclusivo para una FPGA en particular. La síntesis se realiza a través de un software, el cual en general

permite numerosas parametrizaciones, de modo de, por ejemplo, priorizar la velocidad del diseño, minimizar el número de celdas lógicas utilizadas, etc.

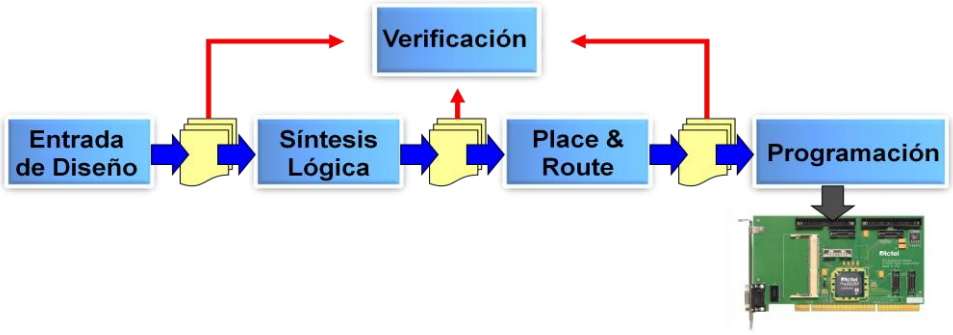


Figura 2.3. Flujo de diseño típico utilizando tecnología FPGA.

A continuación le sigue el proceso de *Place and Route*, en donde el paquete de software que lo realiza escoge las celdas lógicas a utilizar y las rutas de interconexionado, entre otras cuestiones. Esta herramienta permite además configurar opciones de optimización de velocidad. La salida de este proceso es un archivo que luego servirá para programar al dispositivo.

Durante todos los pasos mencionados se puede realizar, en paralelo al flujo normal, una *Verificación del Diseño.* Las herramientas de verificación dependen del paso en que se encuentre el diseño. Una verificación comúnmente realizada es *simulación* temporal del comportamiento. Se pueden realizar simulaciones más exhaustivas que tengan en cuenta el retardo que sufren las señales al atravesar las diferentes celdas lógicas.

El flujo de diseño termina con la *Programación* de la FPGA. Para ello se utiliza un dispositivo de hardware especial junto a un software de programación, provistos en general por el mismo fabricante de la FPGA. Una vez realizado este paso, la FPGA quedará lista para funcionar con el diseño que le ha sido introducido.

2.3 PLACA DE DESARROLLO *"SPARTAN-3 STARTER KIT"*

Como se explicó en el resumen de la tesina, el diseño presentado ha sido implementado en una placa de desarrollo de FPGA. En la Figura 2.4 se observa una fotografía de ella [5].

El kit cuenta con un chip FPGA *Spartan-3 XC3S200*. Además, cuenta con un display de 7 segmentos de cuatro caracteres, una botonera, llaves de selección, LEDs y numerosos pines de salida. Todos estos elementos hacen que la placa sea ideal para la implementación conceptual de equipo presentado en esta tesina.

Figura 2.4. Kit de desarrollo de la empresa Digilent para diseños digitales. La placa cuenta con una FPGA Xilinx Spartan-3.

2.3.1 OPERACIONES ARITMÉTICAS CON LA FPGA *XILINX XC3S200*

Como se describió en la Sección 2.1, la FPGA, además de la matriz regular de bloques lógicos, puede contener otros periféricos. En particular, el modelo *XC3S200* de *Xilinx* dispone de 12 multiplicadores de 18x18 bits, los cuales, combinándolos, permiten implementar multiplicaciones de otras cantidades de bits.

Cabe destacar que la FPGA no dispone de ninguna unidad aritmética que opere en punto flotante. En el caso que se requiera, dicha unidad debe implementarse utilizando celdas lógicas. Sin embargo, para la *XC3S200*, esto involucraría ocupar casi toda la FPGA con dicho bloque. Por esta razón es que para este equipo, las operaciones aritméticas deberán implementarse en punto fijo.

3. DESCRIPCIÓN DEL ALGORITMO

En esta Sección se presenta el método que fue utilizado para la implementación del generador de pulsos aleatorios en un equipo digital de característica sincrónica. Como se verá, dicho método resultó en una gran economía de utilización de recursos de la FPGA.

3.1 MODELO DEL CONJUNTO DETECTOR-CONFORMADOR-DISCRIMINADOR PARA UNA IMPLEMENTACIÓN DIGITAL

Basándose en las consideraciones descriptas en las Secciones 1.3 y 1.4, se propuso como modelo de la salida del discriminador al conjunto formado por un generador de un proceso de Poisson seguido de un sistema que introduzca un tiempo muerto (paralizable o no paralizable, según el caso elegido) Los parámetros de entrada del modelo serán entonces la tasa media λ del proceso de Poisson, el tipo de tiempo muerto a emular y su valor τ. La Figura 3.1 muestra un esquema de esta interconexión de modelos.

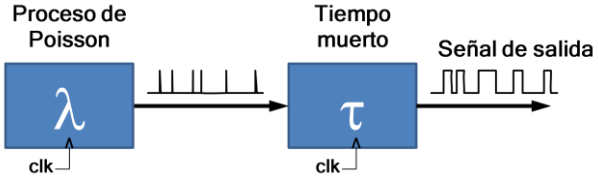


Figura 3.1. Esquema del modelo elegido para implementar el sistema de detección neutrónica en un dispositivo digital.

Se debe resaltar que debido a que en un proceso de Poisson el valor de los tiempos de interarribo entre eventos es una variable definida en el *dominio continuo*, es imposible implementar ese *mismo* comportamiento en un sistema de *tiempo discreto* ya que en estos sistemas los eventos sólo ocurren en instantes múltiplos del tiempo de muestreo T_S. Es por ello que el modelo de la Figura 3.1 es una aproximación a los sistemas reales funcionando en modo pulso. En la Sección 3.2 se explica el enfoque de la aproximación realizada en este trabajo.

3.2 GENERACIÓN DE REALIZACIONES DE PROCESO POISSON MEDIANTE UN SISTEMA DIGITAL SINCRÓNICO

Lo expresado en la Sección 1.5 indica que se puede implementar un proceso de Poisson mediante una secuencia de realizaciones de distribución exponencial. Como obviamente éstas también están definidas en el tiempo continuo, la idea fue aproximar la distribución exponencial mediante una *distribución geométrica*. Esta aproximación se basa en el hecho que la distribución geométrica puede verse como la contrapartida discreta de la distribución exponencial. En las siguientes Secciones se compararán ambas distribuciones y se analizará cómo generar realizaciones de una distribución geométrica.

3.2.1 DISTRIBUCIÓN EXPONENCIAL vs DITRIBUCIÓN GEOMÉTRICA

La distribución exponencial, de *cdf* dada por la ec. (1-4), está definida en el dominio continuo por el parámetro λ, la tasa media. En cambio, la distribución geométrica es una distribución de probabilidades discretas, y hace referencia a la distribución de probabilidad del número X de ensayos de Bernoulli necesarios para obtener un éxito. Se debe recordar que un ensayo de Bernoulli es un experimento aleatorio en el que sólo se pueden obtener dos resultados, habitualmente etiquetados como *éxito* y *fracaso*.

La *cdf* de la distribución geométrica también está definida por un solo parámetro p:

$$cdf_{geom}(k) = 1-(1-p)^k, \qquad k = 1,2,... \tag{3-1}$$

De las ec. (1-4) y (3-1) resulta natural relacionar ambas distribuciones evaluando la $cdf_{exp}(t)$ en los tiempos $t_k = k\,T_S$:

$$cdf_{\exp}(t_k = kT_S) = cdf_{\exp}(k) = 1-e^{\lambda.k.T_S} = 1-\left(e^{\lambda.T_S}\right)^k, \qquad k = 1,2,... \tag{3-2}$$

Igualando entonces las ec. (3-1) y (3-2) se obtiene la relación entre los parámetros que definen ambas distribuciones:

$$p = 1-e^{-\lambda \cdot T_S} \tag{3-3}$$

La Figura 3.2 muestra la superposición de las distribuciones.

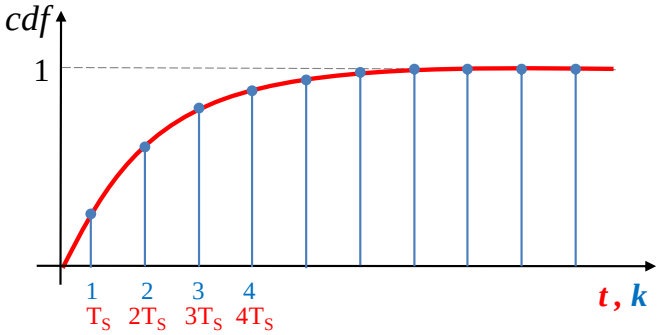


Figura 3.2. En rojo: distribución exponencial (continua); en celeste: distribución geométrica (discreta). La superposición se logra ajustando convenientemente los valores de λ y p.

La distribución geométrica tiene una media de valor $1/p$. Multiplicando esta media por T_S (de modo de pasar del tiempo discreto al tiempo continuo), se obtiene el valor de la media en el dominio continuo:

$$media_{geom(continua)} = \frac{T_S}{1-e^{-\lambda \cdot T_S}} \tag{3-4}$$

Este valor especifica el tiempo de interarribo medio obtenido al aproximar un proceso de Poisson a través de una secuencia de realizaciones de una distribución geométrica de parámetro $p=1-e^{\lambda Ts}$. Este valor difiere de la media de Poisson deseada, que es $1/\lambda$. A pesar que la desviación no es grande, existen aplicaciones donde cualquier offset en la media resulta inconveniente. Este es el caso, por ejemplo, cuando se usa el equipo para calibrar un instrumento que estima la tasa de cuentas. Por esta razón es que se recurrió a otra aproximación para relacionar ambas distribuciones y consiste en igualar sus medias, y no sus *cdfs*. Al hacer esto se obtiene la siguiente relación entre λ y p:

$$p = \lambda \cdot T_S \tag{3-5}$$

Utilizando este valor de p, la secuencia de realizaciones de distribución geométrica tendrá un tiempo de interarribo promedio de $1/\lambda$, coincidente con el proceso de Poisson que se desea emular.

Cabe aclarar que con la adopción de p dada en la ec. (3-5) no se logrará la superposición perfecta de las *cdf* tal como ocurre en la Figura 3.2. Por lo tanto, habrá una diferencia de "forma" entre ambas distribuciones. Sin embargo, más adelante se demostrará que con el valor de T_S adoptado esa diferencia es despreciable.

3.2.2 REALIZACIONES DE DISTRIBUCIÓN GEOMÉTRICA

En general, los métodos para generar números aleatorios de una distribución determinada parten de la generación de números aleatorios de *distribución uniforme*. Existe por ejemplo el *método de inversión*, el cual afirma que si U es una variable aleatoria de distribución uniforme *[0,1]*, entonces $Y=F^{-1}(U)$ posee la distribución F buscada. Este método implica entonces generar un número U de distribución uniforme y luego evaluar la función F^{-1} en U (que aquí correspondería evaluar la inversa de la ec. (3-1)) para finalmente obtener un número aleatorio de distribución geométrica. Sin embargo, este algoritmo tiene el problema que la evaluación de la función F^{-1} puede implicar la utilización de demasiados recursos de una FPGA.

Se pueden obtener realizaciones de distribución geométrica de una manera mucho más directa si se recuerda que dicha distribución está definida como una sucesión de ensayos de Bernoulli. De este modo, si en cada ciclo de reloj se realiza un ensayo de Bernoulli, la cantidad de ciclos entre los eventos "éxito" tendrá una distribución geométrica.

Cada ensayo de Bernoulli está caracterizado por un único parámetro p, que representa la probabilidad de obtener un éxito y la manera más directa de obtener una realización es generando un número aleatorio R de distribución uniforme *[0,1]* y luego compararlo con el valor de p, de modo que si $R<p$, entonces se obtiene un *éxito*, caso contrario se obtiene un *fracaso*. Cabe aclarar que el parámetro p de la distribución de Bernoulli coincide con el de la distribución geométrica.

En una FPGA, se puede realizar una sucesión de ensayos de Bernoulli de parámetro p generando en cada ciclo de reloj un número pseudoaleatorio R de N bits y rango $[0, 2^N-1]$ para luego compararlo con el valor:

$$P = round\left[p \cdot (2^N - 1)\right] \qquad (3\text{-}6)$$

de modo de expandir p en todo el rango de R. Si se da que $R < P$ entonces se debe generar un pulso de un ciclo de duración. El valor de N define la granularidad del número pseudoaleatorio y a su vez limita los valores que puede asumir p, discretizando su intervalo original *[0,1]* en 2^N valores. Esta discretización de p implica una discretización de λ. Se puede demostrar que el máximo error de resolución entre la tasa deseada λ_V y la tasa λ_O finalmente generada resulta:

$$\left|\lambda_V - \lambda_O\right|_{MÁX} = \frac{1}{2^{N+1} \cdot T_S}\ seg^{-1} \qquad (3\text{-}7)$$

La ec. (3-7) impone un límite en la *mínima* tasa alcanzable por el emulador, ya que para altas tasas el error es despreciable.

Para la implementación del equipo en cuestión se definieron los siguientes parámetros:

$$\begin{aligned} &N = 32 \\ &T_S = 20nseg\ \ (\Rightarrow Frec_{FPGA} = 50MHz) \end{aligned} \qquad (3\text{-}8)$$

con la posibilidad de seleccionar la tasas λ_i, abarcando 5 décadas:

$$\begin{aligned} &\lambda_1 = 1seg^{-1} \qquad &&\lambda_4 = 1Kseg^{-1} \\ &\lambda_2 = 10seg^{-1} \qquad &&\lambda_5 = 10Kseg^{-1} \\ &\lambda_3 = 100seg^{-1} \qquad &&\lambda_6 = 100Kseg^{-1} \end{aligned} \qquad (3\text{-}9)$$

Como se verá en la Sección 0, esta elección condujo a resultados muy satisfactorios, por lo que no hubo necesidad de incrementar aún más la frecuencia de trabajo de la FPGA ni utilizar una mayor cantidad de bits para generar la secuencia de ensayos de Bernoulli.

La Figura 3.3 muestra un diagrama de bloques de la generación los pulsos con distribución de Poisson en la FPGA para los diferentes valores λ_i de tasa. Cabe destacar que la FPGA no necesita realizar los cálculos para pasar de λ a p, ec. (3-5), y luego de p a R, ec. (3-6), pues ella almacena directamente los valores de comparación R_i y no las tasas λ_i. En dicha Figura, las siglas $PRNG$ significan *Pseudo Random Number Generator:* generador de números pseudo aleatorios. En la Sección 3.2.3 se dan más detalles del algoritmo utilizado.

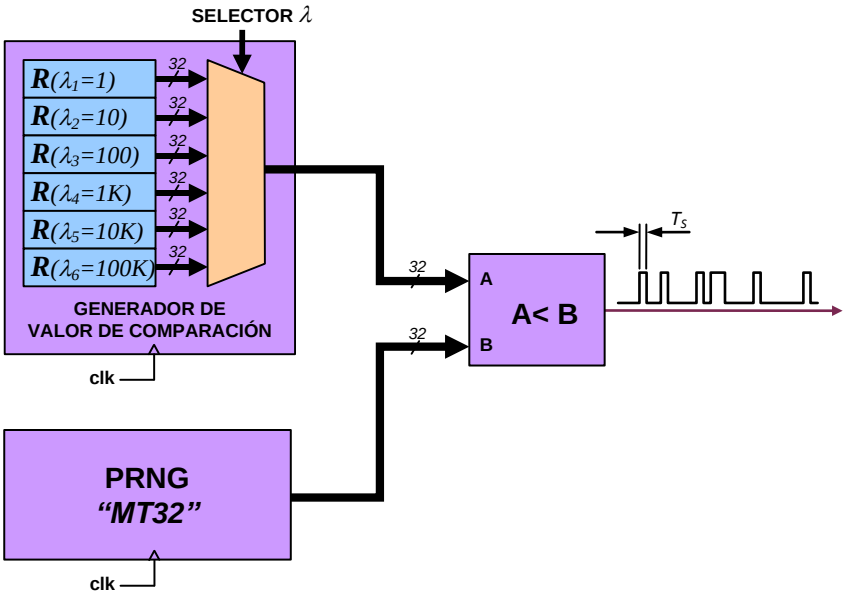


Figura 3.3. Diagrama de bloques de la implementación en FPGA de un generador de Poisson con posibilidad de emular diferentes tasas λ_i.

3.2.3 GENERACIÓN DE NÚMEROS PSEUDO ALEATORIOS MEDIANTE FPGA

El generador de números pseudo aleatorios es uno de los bloques fundamentales sobre el que se apoya el algoritmo que implementa el proceso de Poisson en el equipo; de él depende el buen comportamiento estadístico de las señales generadas.

El equipo incluye una versión de 32 bits del generador Mersenne-Twister (MT32) [7]. Este algoritmo genera números pseudo aleatorios de distribución uniforme con un período primo igual a 2^{19937}-*1*. La implementación para FPGA fue extraída de [8] y se denomina MT19937. Está escrita en el lenguaje VHDL y posee la capacidad de entregar un número aleatorio en cada ciclo de reloj. La semilla del algoritmo se encuentra almacenada en la memoria flash que acompaña a la FPGA.

Para testear la aleatoriedad de MT32 se utilizó una herramienta denominada *TestU01* [9], la cual se presenta como una librería programada en ANSI C y ofrece una colección de utilidades para el testeo empírico de secuencias de números de distribución uniforme. En la sección de anexos 7.1 se describe dicha herramienta y se dan ejemplos de su utilización.

El testeo de la implementación MT19937 se realizó generando un proyecto VHDL separado y simulando el algoritmo mediante el software *ModelSim*®. Se generó un archivo conteniendo alrededor de $550x10^6$ números aleatorios de 32 bits que luego se utilizó para correr las funciones `bbattery_pseudoDIEHARD` y `bbattery_SmallCrush` de la herramienta *TestU01*, pasando satisfactoriamente ambas baterías de tests.

3.3 GENERACIÓN DE UN PROCESO DE POISSON CON TASA DE CRECIMIENTO EXPONENCIAL

Hasta aquí se ha desarrollado la implementación de un generador de Poisson de tasa λ constante mediante un sistema de tiempo discreto, como es una FPGA la cual se basa básicamente en la comparación en cada ciclo de reloj de un valor de *límite*, convenientemente calculado, con un número pseudo aleatorio de modo que cuando el límite resulte *menor*, se genere un pulso de un ciclo de duración a la salida.

Para extender este algoritmo a un proceso de Poisson de tasa variable, la idea fue ir *actualizando* el valor de comparación a medida que transcurre la emulación. De este modo, si se desea obtener una tasa con crecimiento exponencial, entonces el valor de comparación también deberá crecer de la misma manera:

$$\begin{aligned} &\lambda = \lambda(t) = \lambda_0 \cdot e^{\phi \cdot t} \\ &\Downarrow \\ &R = R(t) = R_0 \cdot e^{\phi \cdot t}, \qquad R \in [0,\ 2^{32}-1] \end{aligned} \tag{3-10}$$

En la ec. (3-10) se observa que son necesarios dos parámetros para definir a la evolución exponencial: la comparación inicial R_0 (relacionada con λ_0 mediante las ec. (3-5) y (3-6)), y el valor de la tasa de variación Φ, en $\sec^{-1}$. Además, está definida en el tiempo continuo, por lo que deberá ser discretizada teniendo en cuenta el valor del período de muestreo T_S:

$$R(nT_S) = R[n] = R_0 e^{\phi \cdot T_S \cdot n} = R_0 \left(e^{\phi \cdot T_S}\right)^n \qquad \text{con } n = 0,1,2,... \text{ y } R \in [0,\ 2^{32}-1] \tag{3-11}$$

Vemos que para calcular el valor actualizado de R en principio se debería computar una ecuación de la forma $\boldsymbol{R_0 . K^n}$ en cada ciclo de reloj. En la FPGA utilizada en el proyecto, esto resultaría muy costoso pues involucraría la evaluación de una función exponencial en cada ciclo de reloj.

De modo de implementar la ec. (3-11) de manera económica se recurrió a un algoritmo recursivo. A continuación se expresa la justificación:

$$\begin{aligned} &R[n] = R_0 \cdot e^{\phi \cdot T_S \cdot n} \\ &\Downarrow \\ &R[n+1] = R_0 \cdot e^{\phi \cdot T_S \cdot (n+1)} = R[n] \cdot \underbrace{e^{\phi \cdot T_S}}_{\text{constante}} \end{aligned} \tag{3-12}$$

Este algoritmo reduce la función exponencial de la ec. (3-11) a una simple multiplicación:

$$\begin{aligned} &R[0] = R_0 \\ &R[1] = R[0] \cdot e^{\phi \cdot T_S} \\ &R[2] = R[1] \cdot e^{\phi \cdot T_S} \\ &\vdots \end{aligned}$$

Los valores adoptados en (3-8) hacen que sólo reste definir el valor de Φ. Para la implementación de este equipo se decidió poder emular 2 tasas de variación Φ_1 y Φ_2, las cuales se fijaron en:

$$\begin{aligned} &\Phi_1 = 0.025 \sec^{-1} \\ &\Phi_2 = 0.040 \sec^{-1} \end{aligned} \tag{3-13}$$

Estos valores no fueron adoptados de manera arbitraria, pues Φ_1 representa el límite de inhibición de barras para el reactor RA-1 y Φ_2 es un valor superior al límite de inhibición, lo cual debería activar caída de barras, pero las implementaciones actuales de los

impulsímetros de arranque no se incorporan a la lógica de SCRAM por un filtrado inadecuado del ruido estadístico.

Antes de continuar, se debe hacer una salvedad respecto esta manera de calcular una función exponencial. La característica iterativa del algoritmo hace que se tenga que prestar mucha atención en los errores que se van acumulando en cada cálculo. Debido a la necesidad de realizar las operaciones con punto fijo, los errores acumulados pueden resultar inadmisibles si no se toma la suficiente cantidad de bits para representar los números. Teniendo en cuenta este hecho, el valor de T_S fijado en (3-8) hace que la expresión $e^{\Phi Ts}$ necesite ser representada por una excesiva cantidad de bits para evitar una acumulación significativa de errores. Para salvar esta situación, se decidió *aumentar* el tiempo T_S que controla la actualización del valor de límite R, pasando a valer 1mseg. Con esto se logra poder representar $e^{\Phi Ts}$ con una menor cantidad de bits, manteniendo bajo los errores. Se debe aclarar que este nuevo tiempo no se aplica sobre el generador de números pseudoaletorios, que seguirá produciendo números a razón de 1 cada 20nseg.

Con este nuevo valor de T_S*, los valores de $e^{\Phi Ts*}$ valen:

$$\begin{aligned} &T_S{*} = 1\,\text{mseg} \\ &\Phi_1 = 0.025\,\text{seg}^{-1} \Rightarrow e^{\Phi_1 T_S{*}} \cong 1.0000250003125026321 \\ &\Phi_2 = 0.040\,\text{seg}^{-1} \Rightarrow e^{\Phi_2 T_S{*}} \cong 1.0000400008000107643 \end{aligned} \qquad \textit{(3-14)}$$

Debido a que estos valores son muy próximos a la unidad, se decidió restar 1 de los valores $e^{\Phi Ts*}$ oteniendo el resto *r(Φ)*:

$$\begin{aligned} &R[0] = R_0 \\ &R[1] = R[0] + R[0] \cdot r(\Phi) \\ &R[2] = R[1] + R[1] \cdot r(\Phi) \\ &\quad\vdots \end{aligned} \qquad \textit{(3-15)}$$

Expresado de esta manera, el cálculo de R involucra una operación de *suma* y una de *multiplicación*. Los valores de los *restos* son:

$$\begin{aligned} &\Phi_1 = 0.025\,\text{seg}^{-1} \Rightarrow r(\Phi_1) \cong 0.0000250003125026321 \\ &\Phi_2 = 0.040\,\text{seg}^{-1} \Rightarrow r(\Phi_2) \cong 0.0000400008000107643 \end{aligned} \qquad \textit{(3-16)}$$

En la Figura 3.4 se muestra el diagrama de bloques de la implementación en FPGA de las ec. (3-15). Allí se pueden ver las dimensiones fijadas para los números de punto fijo. Dicho diagrama de bloques se complementa con la Figura 3.5, donde se detalla la alineación de todos los números involucrados en las operaciones.

En el diagrama se observa que los bloques que implementan la suma y la multiplicación tienen asociado un clock de período T_S de 20nseg, mientras que el flip-flop FF encargado de registrar cada valor del límite R posee un reloj T_S *, de 1mseg de período. Esto es así pues los bloques multiplicadores y sumadores necesitan muchos ciclos de reloj para dar el resultado final a la salida (funcionamiento denominado *pipeline*). De esta manera, cada vez que el flip-flop FF actualice su estado, los datos a su entrada ya se habrán estabilizado en el valor definitivo. Además, el FF tiene un bus de *precarga* que se activa al iniciarse la emulación y tiene como objetivo fijar las condiciones iniciales del algoritmo.

Se debe aclarar aquí que en realidad el único reloj del cual se rigen todos los bloques es T_S, de 20nseg. De este modo, no es que el flip-flop FF tenga un clock diferente, sino que dispone de una señal de habilitación de clock (*clock enable*) que se activa generando un pulso cada T_S *. Así, la señal de "*clock enable"* hace que el FF se actualice sólo cada T_S *. La razón de este modo de operación es que los diseños con FPGA tienen un mejor desempeño cuando todos los bloques sincrónicos son comandados por la misma señal de clock.

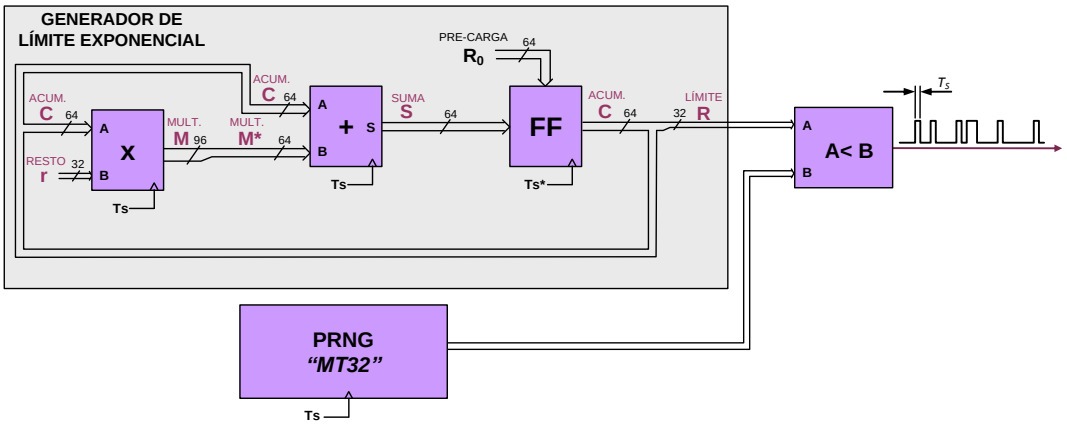


Figura 3.4. Diagrama de bloques del generador del valor de límite "R" para emular un proceso de Poisson con tasa de evolución exponencial.

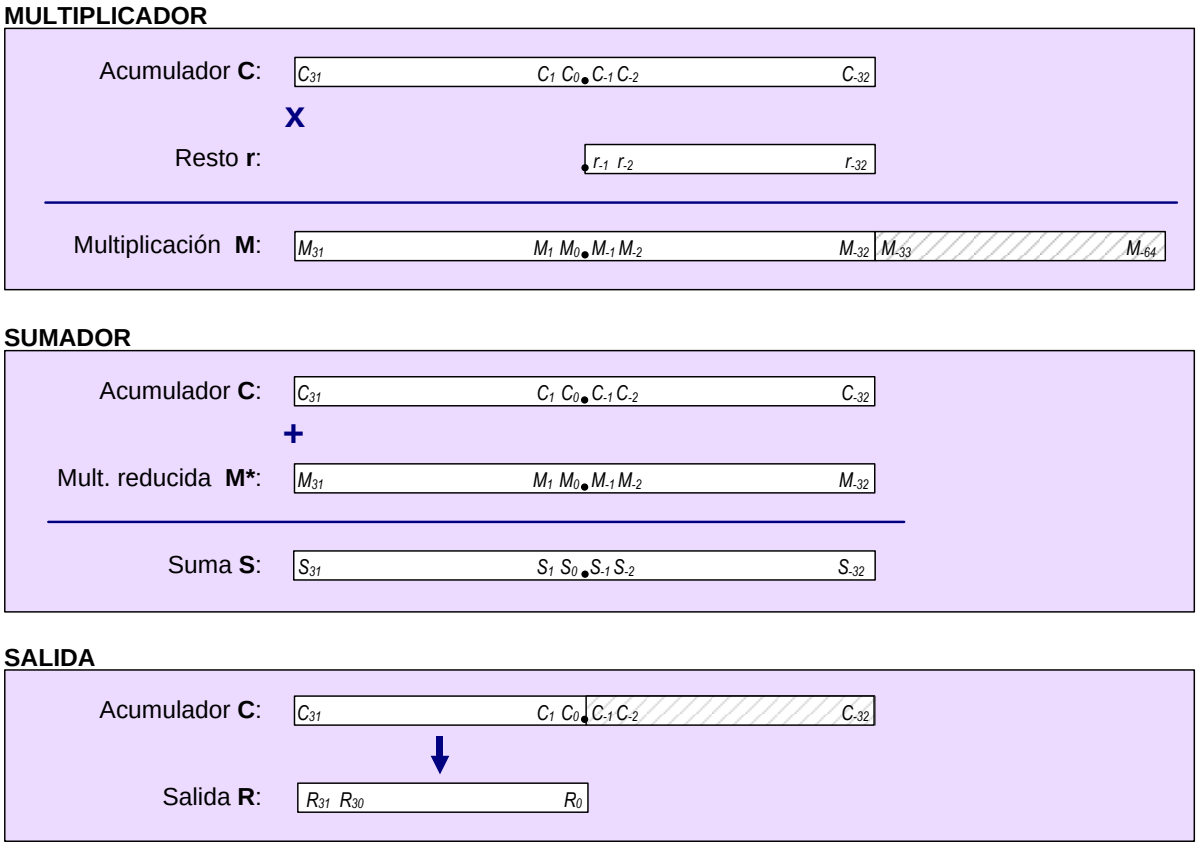


Figura 3.5. Diagrama del tamaño y alineación de los números de punto fijo utilizados para implementar el valor de límite "R" de evolución exponencial

Para verificar el algoritmo recursivo aquí presentado, se simuló el código VHDL mediante la herramienta *ModelSim* y se compararon los resultados con una evolución exponencial realizada de manera no recursiva con aritmética de punto flotante de 64bits. El algoritmo se verificó para los dos valores de Φ. En la Figura 3.6 se muestran los errores relativos obtenidos. La abscisa está expresada en segundos. En ambos casos, el tiempo de simulación es tal que el valor de límite *R* alcanzado genere una señal aleatoria de 100Kpulsos/seg (que es justamente la tasa máxima que implementa el equipo). El hecho que se observen áreas sólidas en la figura se debe a que los errores tienen una oscilación

continua. El máximo error relativo se da en bajas tasas y vale ≈ 0.012%, valor extremadamente bajo, lo que confirma el buen funcionamiento de la implementación.

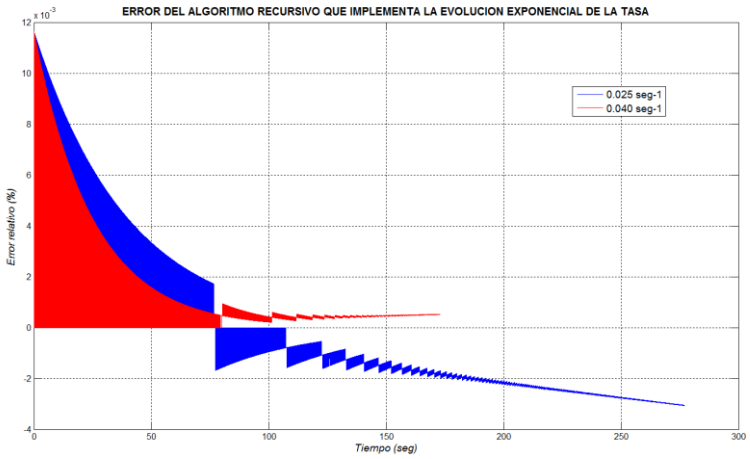


Figura 3.6. Error relativo de la implementación del algoritmo recursivo que genera un valor de límite "R" de evolución exponencial.

3.4 IMPLEMENTACIÓN DEL SISTEMA DE TIEMPO MUERTO

El módulo de tiempo muerto τ se implementó a través de un circuito monoestable con un valor de límite de cuenta dada por la ecuación:

$$\text{cuenta monoestable M} = round\left(\tau / T_S\right) \qquad (3\text{-}17)$$

Dicho valor corresponde a la versión discretizada del tiempo τ. La Figura 3.7 muestra un diagrama de entrada/salida del módulo. Internamente, el valor de *M* se encuentra guardado en un registro de 16 bits, lo que posibilita implementar un tiempo muerto que va desde 20nseg hasta ~1.3ms, en pasos de 20nseg.

Si bien en la implementación actual el valor almacenado genera un tiempo muerto de 500nseg, en el equipo final este valor podría ser seteado a través de controles desde el frente.

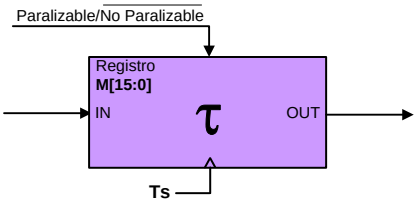


Figura 3.7. Diagrama entrada/salida del módulo que implementa en una FPGA un sistema con tiempo muerto.

La operación del módulo es la siguiente: cuando el circuito monoestable recibe un pulso (proveniente del generador de Poisson), extiende este pulso de entrada -de 1 ciclo de duración- de modo que dure *M* ciclos. En caso que ingrese otro pulso durante el transcurso de estos *M* ciclos, el comportamiento del módulo dependerá del tipo de tiempo muerto seleccionado.

Para el caso del modelo paralizable, el segundo pulso redisparará el contador monoestable, manteniendo en alto el pulso de salida por otros *M* ciclos a partir de este último pulso. De esta manera, salida volverá al estado bajo sólo después de que hayan transcurridos *M* ciclos sin recibir pulsos en la entrada.

En el caso que se seleccione el modelo no paralizable, cualquier pulso que se dé mientas la salida se encuentra activa será *descartado*. Esto hace que los pulsos de salida en este modo de funcionamiento tengan todos la misma duración de *M* ciclos.

3.5 GENERADOR DE PULSOS DETERMINÍSTICOS

La unidad de generación de pulsos de se implementó a través un contador ascendente seguido de un comparador. El funcionamiento es el siguiente: el contador incrementa su valor en 1 en cada ciclo de reloj y cuando la cuenta iguala al valor almacenado en el comparador, se genera un pulso de salida y la cuenta se reinicia.

La relación entre el valor de comparación L y la tasa de los pulsos generados λ está dada por:

$$\lambda = \frac{1}{T_S \cdot L} \Rightarrow L = \frac{1}{T_S \cdot \lambda} \qquad (3\text{-}18)$$

Con un valor de T_S de 20nseg y un registro de comparación L de 32 bits, la tasa de pulsos puede ir desde un pulso cada ~86 seg ($1/20x10^{-9}/2^{32}$) hasta los 25 Mpulsos/seg.

El equipo permite generar pulsos determinísticos, pudiéndose seleccionar las siguientes frecuencias:

$$f_1 = 1Hz \qquad f_4 = 1KHz$$
$$f_2 = 10Hz \qquad f_5 = 10KHz$$
$$f_3 = 100Hz \qquad f_6 = 100KHz$$

Notas que estas frecuencias coinciden con las tasas de pulsos seleccionables en el modo de generación aleatoria, Figura 3.3.

El equipo también implementa un módulo de generación de pulsos determinísticos con una tasa de variación exponencial en el tiempo, con los mismos valores de variación relativa que en el caso aleatorio, ec. (3-13): Φ_1 = 0.025 seg^{-1}y Φ_2 = 0.040 seg^{-1}. El funcionamiento de este módulo es similar al descripto en la Sección 3.3 y se basa en generar una función exponencial mediante un algoritmo recursivo, sólo que en este caso lo que se va variando exponencialmente es el valor de comparación L del contador.

Dada la relación (3-18) y la función $\lambda(t)$ se obtienen las siguientes ecuaciones:

$$\lambda(t) = \lambda_0 \cdot e^{\Phi \cdot t}$$
$$L = \frac{1}{T_S \cdot \lambda} \quad \Rightarrow \quad L = L(t) = L_0 \cdot e^{-\Phi \cdot t} \qquad L_0 = \frac{1}{T_S \cdot \lambda_0} \qquad (3\text{-}19)$$

Vemos que ahora la exponencial es decreciente, pues a medida que se incrementa la tasa, el valor de comparación del comparador se va reduciendo.

La versión recursiva de la ec. (3-19) resulta:

$$L[n] = L_0 \cdot e^{-\Phi \cdot T_S \cdot n}$$
$$L[n+1] = L_0 \cdot e^{-\Phi \cdot T_S \cdot (n+1)} = L[n] \cdot \underbrace{e^{-\Phi \cdot T_S}}_{\text{K (cte)}} \qquad (3\text{-}20)$$

En la ec. (3-20) se observa que el factor de multiplicación es ahora menor que la unidad (pues el exponente es negativo), por lo que no será necesario restarle 1 como se hizo en el caso aleatorio. Con T_S* = 1mseg, los valores de $e^{-\Phi T_S *}$ resultan:

$$T_S{}^* = 1\text{mseg}$$
$$\Phi_1 = 0.025\text{seg}^{-1} \Rightarrow e^{-\Phi_1 T_S{}^*} = K_1 \cong 0.999975000312497 \qquad (3\text{-}21)$$
$$\Phi_2 = 0.040\text{seg}^{-1} \Rightarrow e^{-\Phi_2 T_S{}^*} = K_2 \cong 0.999960000799989$$

El diagrama de bloques de la implementación en FPGA de las ec. (3-21) resulta similar al de la Figura 3.4 Figura 3.4, sólo que ahora se debe anular la operación de *suma*, pues ya no es necesaria. En realidad, el módulo de suma de la Figura 3.4 Figura 3.4posee una entrada denominada *bypass* que, al activarse, traslada el valor de la entrada *B* a la salida *S*. La Figura 3.8 muestra el esquema resultante:

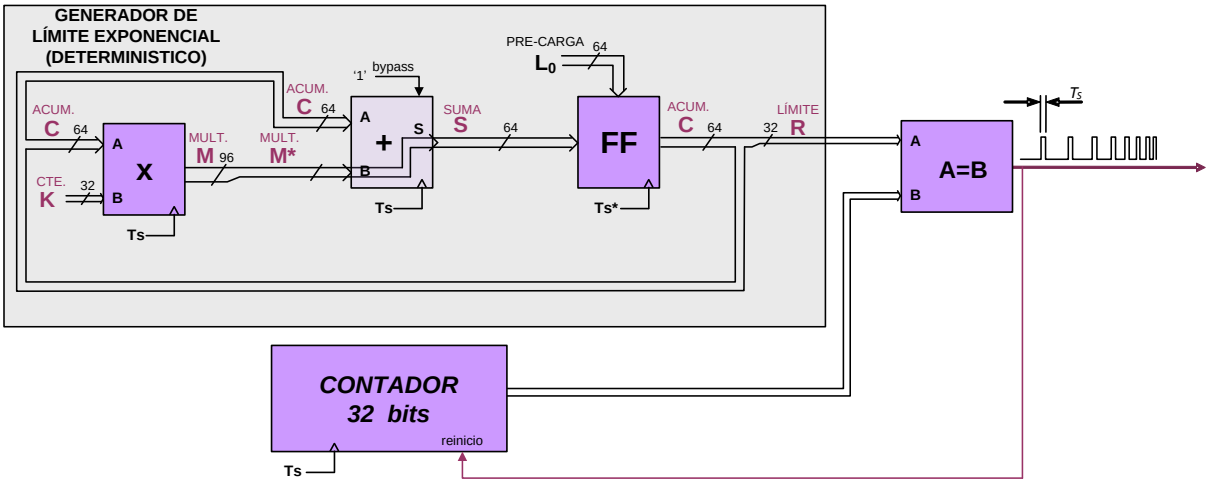


Figura 3.8. Diagrama de bloques del generador del valor de comparación "L" para emular un tren de pulsos determinístico con tasa de crecimiento exponencial.

Las dimensiones de los números de punto fijo son los mismos que los mostrados en la Figura 3.5, sólo que ahora desaparece la operación de suma.

En la Figura 3.9 se muestra el error relativo porcentual de la implementación recursiva. En este caso, el máximo error relativo se da en altas tasas y vale ≈ 0.2%, valor que confirma el buen desempeño del módulo.

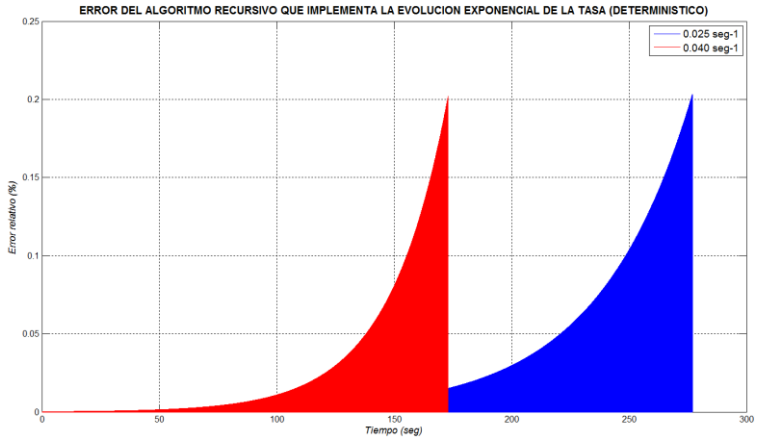


Figura 3.9. Error relativo porcentual de la implementación del algoritmo recursivo que genera un valor de comparación "L" de evolución exponencial.

3.6 MÓDULO DE REGISTRO DE DATOS

Los módulos de registro de datos se encargan de transmitir a una PC información acerca de los pulsos generados.

Existen dos tipos de datos que se transmiten. El primero corresponde al número de pulsos generados en ventanas sucesivas de 10 seg. Esta información permite obtener un *feedback* del equipo cada 10 segundos y el receptor puede hacer un análisis estadístico básico de los datos.

El otro tipo de información transmitida corresponde a muestras de valores de tiempos de interarribo contiguos de la señal que está siendo generada. Dichos datos posibilitan un análisis estadístico más profundo.

La Figura 3.10 muestra la interconexión de todos los módulos involucrados en el registro de datos. La entrada a estos módulos es la misma señal de salida de pulsos del equipo.

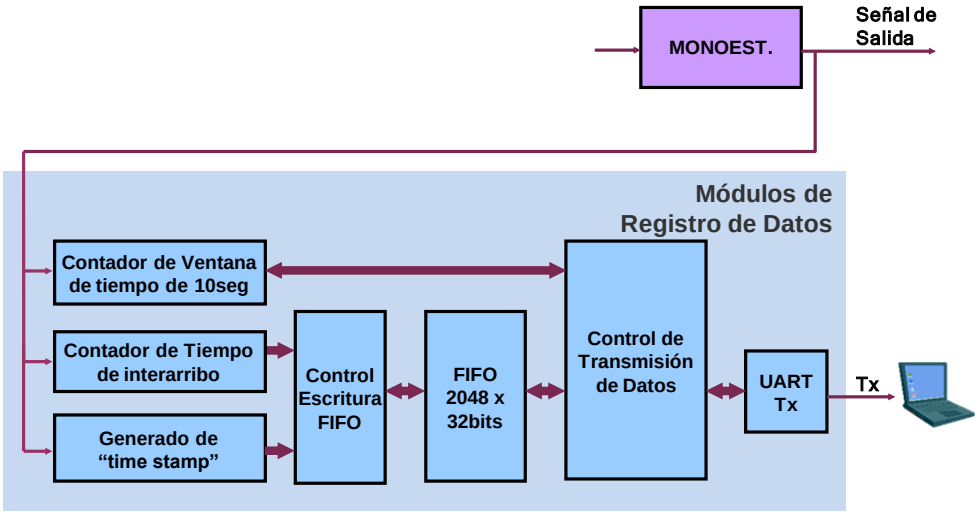


Figura 3.10. Diagrama de bloques de los módulos de registro de datos.

El *Contador de Ventana de Tiempo de 10 seg* registra el número de pulsos generados en una ventana de tiempo de 10 segundos y luego traslada el resultado al módulo de *Transmisión de Datos.* Una vez que la información se envía a la PC, se abre una nueva ventana de tiempo y el contador se reinicia. El valor de la cuenta se almacena en un registro de 24 bits (3 bytes).

El *Contador de Tiempos de Interarribo* computa los valores de tiempo de interarribo de la señal de salida. Esto se realiza contando el número de ciclos de reloj entre pulsos adyacentes. Cada valor de interarribo se pone en disposición del módulo *Control Escritura de FIFO* a través de un bus de 32 bits.

El *Generador de Time-Stamp* es un contador *free-running* de 45 bits que se reinicia al comienzo de cada emulación y es usado por el módulo *Control de Escritura de FIFO* del modo que se explica en el párrafo siguiente. Con un tamaño de 45 bits, operando a una frecuencia de 50MHz (T_S = 20nseg), esta unidad puede generar *time-stamps* válidos durante ≈8 días sin producir desbordamiento.

La función del módulo *Control de Escritura de FIFO* es la de alimentar a la memoria FIFO con valores de tiempo de interarribo *contiguos*. Estos valores se envían a la PC mediante el *Control de Transmisión de Datos*. Así, la velocidad de escritura de la FIFO (que depende de la tasa de pulsos que está siendo generada) es mayor que la velocidad de lectura de la FIFO (dependiente del *baudrate* de la transmisión por el canal serie), entonces la FIFO eventualmente se llenará completamente. Cuando esto sucede el *Control de Escritura de la FIFO* dejará de escribir en la memoria hasta que ella se vacíe completamente. Luego, antes

de recomenzar a grabar valores de interarribo, en la FIFO ya vacía se graba un registro en cero, que hace las veces de encabezado, y a continuación un valor de *time-stamp* extraído de la unidad de generación de time-stamp. Dicho valor de marca de tiempo ocupa 2 registros, ya que la FIFO tiene un ancho de 32 bits y la marca de tiempo 45 bits. De este modo, el valor de tiempos de interarribo contiguos junto a las marcas de tiempo permiten reconstruir parcialmente en la PC los pulsos generados en el equipo.

El *Control de Transmisión de Datos* toma los valores generados tanto por la memoria FIFO como por la ventana de cuenta de 10 seg para empaquetarlos en grupos de bytes, agregarles un encabezado y un byte de control de errores de transmisión. Finalmente, cada byte se envía al módulo *UART Tx* para finalmente transmitirlos a la PC.

Los módulos de registro de datos son muy útiles para capturar muestras de la señal generada de modo de analizar el comportamiento estadístico. En la Sección 0 se usará la información provista por el equipo a través de estos módulos para evaluar los resultados. Sin embargo, en una situación de uso normal de operación, no sería necesario que estos módulos queden en funcionamiento. Es por ello que el equipo permite deshabilitarlos mediante una llave que dispone la placa de desarrollo. En ese caso, los módulos de registro no quedarán totalmente apagados, ya que la ventana de cuenta de 10 segundos seguirá transmitiendo los valores registrados a la PC. Esto permite que siempre se disponga de un *feedback* desde el equipo hacia la PC.

3.7 OTROS COMPONENTES DEL SISTEMA

El diseño del equipo se complementa con los siguientes componentes:

MÓDULO DE CONTROL GENERAL

Compuesto por un conjunto de máquinas de estado que regula el comportamiento general del equipo. El módulo interpreta los comandos que el usuario ingresa a través los botones y llaves y en base a ellos regula opera sobre demás módulos del equipo.

CONTROL DE REBOTE DE BOTONES (DEBOUNCING)

Necesario para evitar que el rebote de los botones sean detectados como múltiples presionados sucesivos.

CONTROL DEL DISPLAY DE 7 SEGMENTOS

Los cuatros dígitos de 7 segmentos de la placa se encuentran multiplexados, por lo que el control de display implementa la lógica de barrido y actualización de cada uno de ellos.

COMPONENTES AUXILIARES

La siguiente es una lista de otros módulos presentes en el diseño:

- PLL para el ajuste de la frecuencia de trabajo,
- Control de parpadeo de LEDs,
- Contadores BDC, que implementan el cronómetro que se activa en la emulación con tasa variable.

3.8 DIGRAMA DE BLOQUES COMPLETO DEL EQUIPO

En la Figura 3.11 se muestra un diagrama con todos los módulos presentes en el equipo.

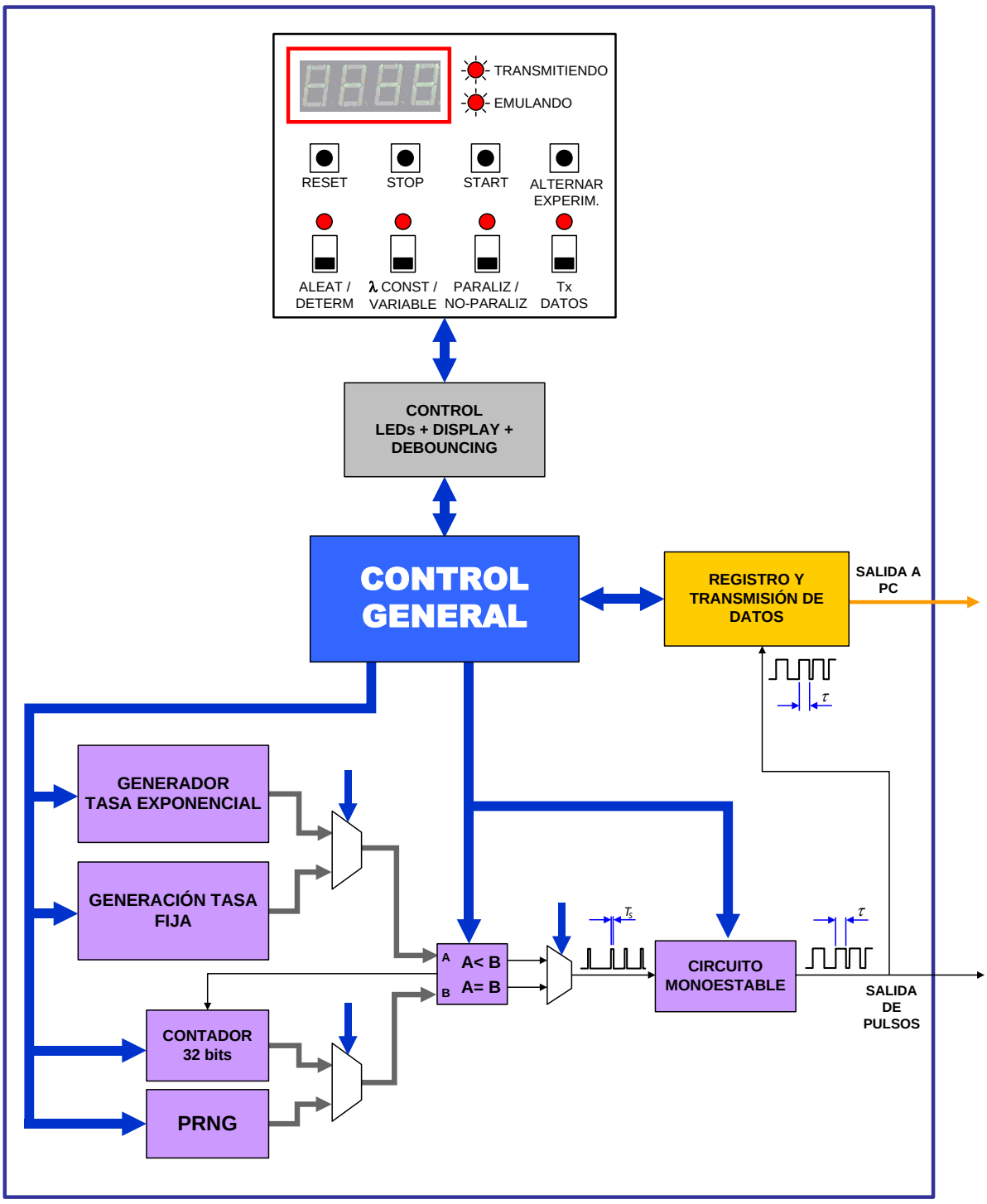


Figura 3.11. Diagrama de bloques completo del equipo

3.9 MODO DE USO DEL EQUIPO

En la Figura 3.12 se repite el esquema del panel de comando del equipo. A continuación se detalla el modo de utilización de cada uno de ellos.

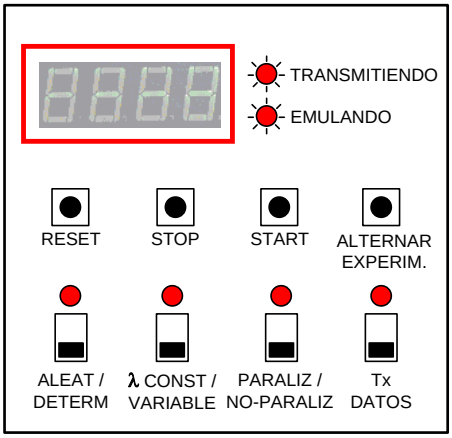


Figura 3.12. Panel de comando del equipo. Los controles observados corresponden a algunos de los disponibles en la placa de desarrollo.

3.9.1 DESCRIPCIÓN DEL PANEL DE COMANDO

RESET

Botón de reinicio del equipo. Al presionarlo se detiene la emulación en curso (si la hubiera) y reinicia todos los registros internos. Este botón también reinicia la semilla del generador pseudo aleatorio.

STOP

Botón de detención de la emulación en curso. Este botón NO reinicia la semilla del generador pseudo aleatorio.

START

Da inicio a una nueva emulación. Una vez que la generación de pulsos comienza, todos parámetros de emulación (determinados por los demás controles) son registrados internamente de modo que no sea posible alterarlos. START no tiene efecto mientras se encuentre la emulación en curso.

ALTERNAR EXPERIMENTO

Este botón funciona en conjunto con la llave ***λ CONSTANTE/VARIABLE***. Si esta llave está en modo de ***tasa constante***, al presionar repetidas veces el botón, se irá modificando el valor de la tasa λ a emular. Los posibles valores que puede adoptar λ son 1, 10, 100, 1e3, 10e3, 100e3 (seg^{-1}). En el caso que la llave esté en modo de ***tasa variable***, el botón se encarga de alternar los posibles valores de Φ: 0.025 ó 0.040 seg^{-1}. Tanto el valor de λ como de Φ son mostrados en el display.

ALEATORIO/DETERMINÍSTICO

Cuando esta llave se encuentra activa (posicionada hacia arriba) se selecciona el modo aleatorio, caso contrario se generarán pulsos determinísticos.

λ CONSTANTE/VARIABLE

Esta llave selecciona entre una tasa constante o variable.

PARALIZABLE/NO-PARALIZABLE

Selección del tipo de tiempo muerto a emular.

Tx DATOS

La activación de esta llave habilita la transferencia de los datos de tiempo de interarribo a través de una comunicación serie.

3.9.2 OPERACIÓN

La generación de pulsos comienza al presionar el botón START y finaliza cuando se presiona STOP ó RESET. En el caso que se estén generando pulsos de tasa constante, el display mostrará el valor de λ. Si se ha seleccionado una tasa variable, el display mostrará el valor del Φ seteado mientras no se haya presionado START, ya que al presionar dicho botón, junto con el inicio de la generación de pulsos se mostrará en el display un cronómetro con el tiempo de emulación.

Para el caso de λ variable, cuando la tasa de cuentas alcanza los 100e3 seg^{-1}, el cronómetro se detiene y λ deja de crecer, estableciéndose en un valor de aproximadamente 100e3 seg^{-1}.

3.10 IMPLEMENTACIÓN EN LA FPGA Spartan-3 XC3S200

La especificación del comportamiento del equipo se realizó utilizando el código VHDL (*Very-high-speed Hardware Description Language*). El diseño completo llevó *más de 3000 líneas de código* y en Sección 7.2 (Anexo) se muestra la estructura jerárquica de todas las entidades.

Para implementar el diseño en la FPGA se siguieron los pasos mencionados en la Sección 2.2 de *Flujo de Diseño.* En particular, los procesos de *síntesis, place & route* y *programación* se realizaron con las herramientas que Xilinx ofrece para sus FPGA.

La Figura 3.13 corresponde a un cuadro realizado por el software *XST* de Xilinx, donde se resume la cantidad de celdas lógicas y otros recursos utilizados en el diseño. Observando la columna que indica el porcentaje de utilización (marcada en rojo) se pude verificar que el diseño completo resultó en una implementación muy económica.

Es importante destacar que la Spartan-3 XC3S200 es una FPGA muy económica; en los distribuidores de EE.UU. (Digi-Key, Farnell, etc) el precio del chip ronda los U$S 10. Este es un punto no menor, pues si para la implementación se hubiera decidido optar por la tecnología de procesadores DSP (*Digital Signal Processor*), se debería utilizar modelos muy costosos (>U$S 200) de modo de equiparar el desempeño alcanzado con la FPGA. Además, el *DSP* difícilmente podría alcanzar a la FPGA en lo que respecta a la flexibilidad en el manejo de la sincronización entre los bloques que componen al diseño completo.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	1,423	3,840	37%	
Number of 4 input LUTs	1,358	3,840	35%	
Number of occupied Slices	1,167	1,920	60%	
Number of Slices containing only related logic	1,167	1,167	100%	
Number of Slices containing unrelated logic	0	1,167	0%	
Total Number of 4 input LUTs	1,597	3,840	41%	
Number used as logic	1,358			
Number used as a route-thru	239			
Number of bonded IOBs	27	173	15%	
Number of RAMB16s	9	12	75%	
Number of MULT18X18s	8	12	66%	
Number of BUFGMUXs	2	8	25%	
Number of DCMs	1	4	25%	
Average Fanout of Non-Clock Nets	2.84			

Figura 3.13. Resumen del porcentaje de utilización de la FPGA Spartan-3 XC3S200.

4. RESULTADOS

En esta Sección se presentan todos los resultados obtenidos al ensayar el equipo en todos sus modos de operación, los cuales se pueden dividir entre pulsos de tasa constante y de crecimiento exponencial, y a su vez en pulsos aleatorios y determinísticos. Todos los datos se colectaron a través de una conexión serie con una PC y haciendo uso del *módulo de registro de datos* que dispone el equipo. Las tramas fueron adquiridas por el mismo software desarrollado para el equipo E023, Figura 1.10. Luego cada experimento fue procesado en *Matlab* para su análisis.

La Tabla 4-1 sintetiza los tipos de experimentos que se realizaron para verificar el comportamiento del equipo.

ALEATORIEDAD	TASA	PARÁMETRO EXPERIMENTO	DESCRIPCIÓN
DETERMINÍSTICO	*CONSTANTE*	λ=1	*En cada caso se verificó la tasa de pulsos generada por el equipo. En todos los casos la tasa de pulsos medida coincidió con la especificada por el parámetro λ.*
		λ=10	
		λ=100	
		λ=1e3	
		λ=10e3	
		λ=100e3	
	VARIABLE	Φ = 0.025	*Se estimó Φ mediante un ajuste de curvas de los tiempos de interarribo.*
		Φ = 0.040	
ALEATORIO	*CONSTANTE*	λ=1	*En cada experimento se estimó λ y se calculó el error según el valor programado de la tasa corregido por tiempo muerto.* *Además, se verificó la distribución estadística de los valores de tiempo de interarribo comparando las cdf empíria con la analítica.*
		λ=10	
		λ=100	
		λ=1e3	
		λ=10e3	
		λ=100e3	
	VARIABLE	Φ = 0.025	*Se estimó Φ mediante un ajuste de curvas de los tiempos de interarribo.*
		Φ = 0.040	

Tabla 4-1. Resumen de los experimentos realizados para evaluar el desempeño del equipo.

4.1 GENERACIÓN DE PULSOS DETERMINÍSTICOS

4.1.1 TASA CONSTANTE

Tal como lo aclara la Tabla 4-1, en este caso se generaron pulsos determinísticos tanto de tasa constante como variable exponencialmente.

Para los casos de tasa constante se verificó que la frecuencia generada *coincidiera exactamente* con la programa en el equipo. Los resultados fueron exactos en todos los casos.

4.1.2 TASA VARIABLE

La Figura 4.1 muestra la gráfica de los tiempos de interarribo para el caso de generación de pulsos determinísticos con una tasa de variación relativa Φ de 0.025 seg^{-1} y 0.04 seg^{-1}.

Ambas evoluciones comienzan con una tasa de 100 seg^{-1} (tiempo de interarribo = 0.01 seg) y terminan cuando la tasa llega a 100 seg^{-1}.

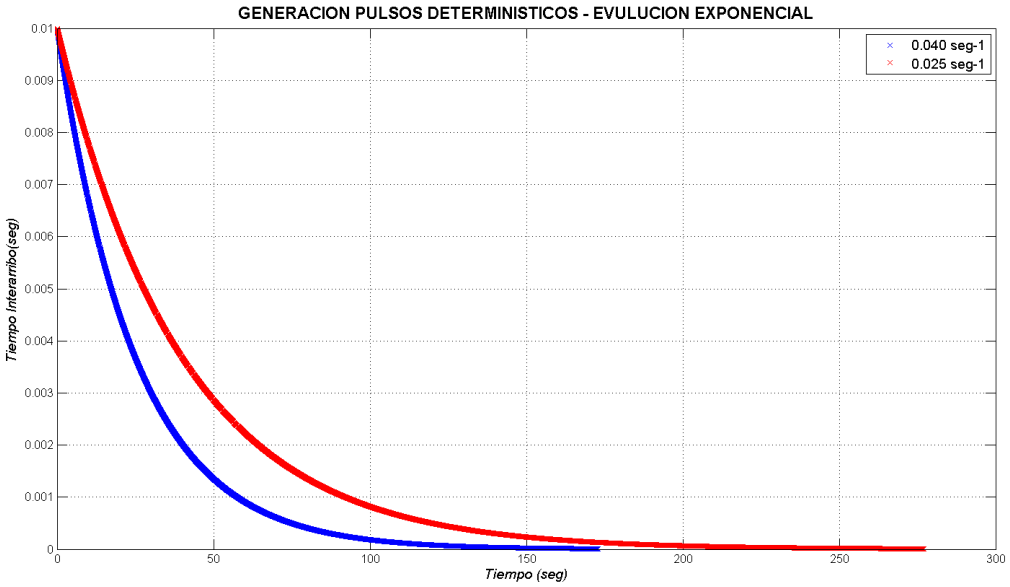


Figura 4.1. Generación de pulsos determinísticos con variación exponencial de la tasa. En rojo Φ=0.025 seg^{-1}; en azul: Φ =0.04 seg^{-1}. Para ambas curvas, λ_0 = 100 seg^{-1} y λ_{final}=100K seg^{-1}.

Para verificar los resultados de la Figura 4.1 se recurrió la herramienta *Curve Fitting Tool* de *Matlab* (comando `cftool`). El modelo de ajuste utilizado tiene la siguiente forma:

$$f(x) = a \cdot e^{b \cdot x} \qquad (4\text{-}1)$$

donde a (seg) corresponde al tiempo de interarribo inicial y b (seg^{-1}) a la tasa de variación relativa. La Figura 4.2 y Figura 4.3 muestran respectivamente las pantallas obtenidas para ambos valores de Φ (los resultados aparecen recuadrados). En la Tabla 4-4 se resumen los ajustes; de allí se observa que loe resultados son más que satisfactorios.

Φ (seg^{-1})	**PARÁMETRO**	**VALOR IDEAL**	**RESULTADO DEL AJUSTE**
0.025	a (seg)	0.01	0.009998
	b (seg^{-1})	-0.025	-0.25
0.040	a (seg)	0.01	0.009996
	b (seg^{-1})	-0.040	-0.04

Tabla 4-2. Resultados del ajuste de los experimentos determinísticos de tasa variable con Φ =0.025 seg^{-1} y Φ =0.04 seg^{-1}. En ambos casos λ_0 = 100 seg^{-1}.

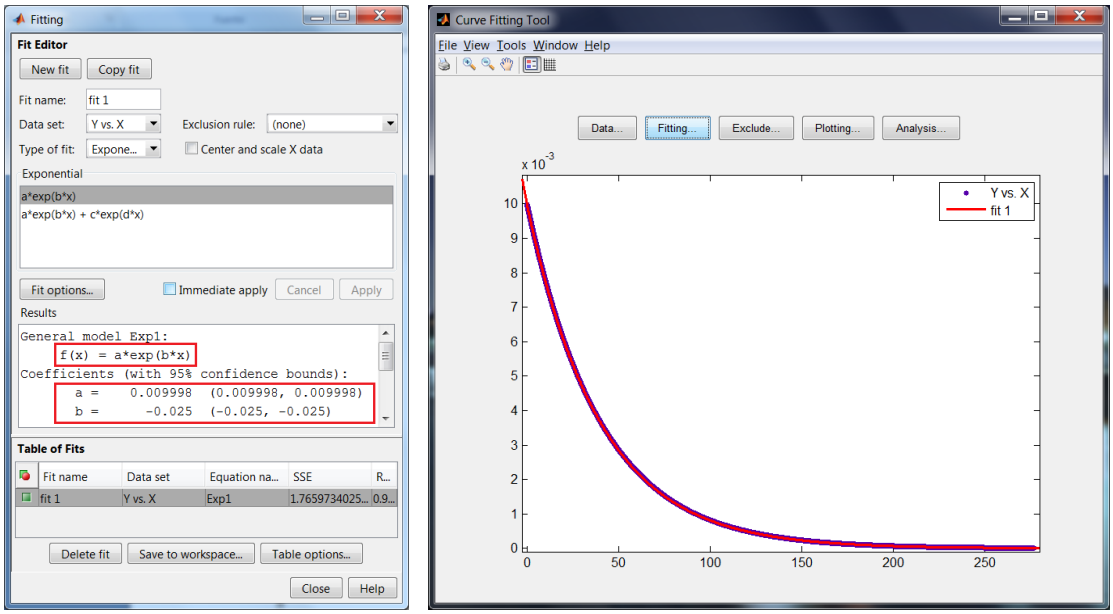


Figura 4.2. Aplicación de la herramienta "Curve Fit Tool" de Matlab para ajustar los pulsos determinísticos y tasa con variación exponencial con Φ =0.025 seg^{-1} y λ_0 = 100 seg^{-1}. La curva expresa los valores de tiempo de interarribo.

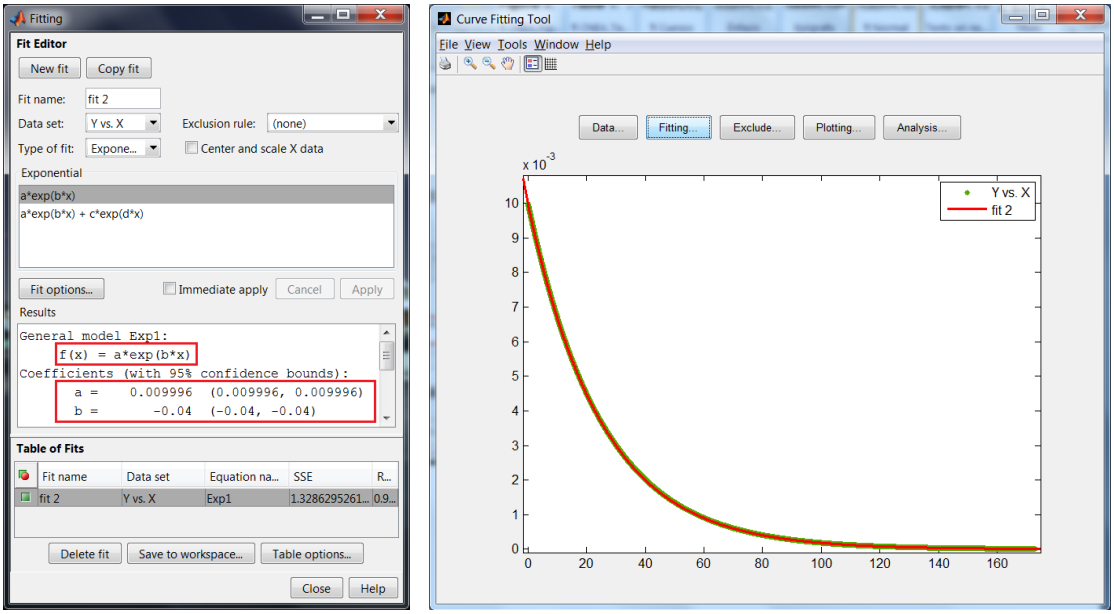


Figura 4.3. Aplicación de la herramienta "Curve Fit Tool" de Matlab para ajustar los pulsos determinísticos y tasa de variación exponencial con Φ =0.04 seg^{-1} y λ_0 = 100 seg^{-1}. La curva expresa los valores de tiempo de interarribo.

La Figura 4.1 también puede aprovecharse para verificar lo expresado en la Sección 3.3, en donde se aclaró que la actualización de los parámetros de la evolución exponencial se produce cada T_S*= 1 mseg. Así, la Figura 4.4 (que representa un detalle de la Figura 4.1) muestra un salto escalonado de los tiempos de interarribo. Se puede verificar allí que el tiempo de cada escalón corresponde a T_S*.

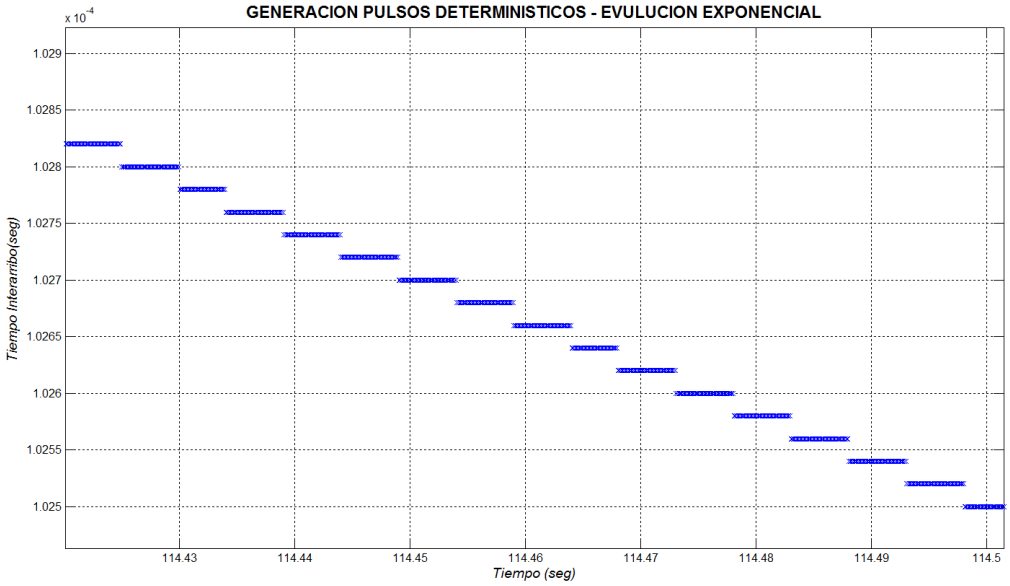


Figura 4.4. Detalle de la Figura 4.1 en donde se observa la actualización cada 1 mseg del parámetro que computa la evolución exponencial.

La Figura 4.5 muestra otro detalle de la Figura 4.1 en donde no sólo se observa el salto escalonado de la evolución exponencial, sino que además se verifica la existencia de intervalos en los cuales el equipo no envió valores de tiempo de interarribo. Tal como se explicó en la Sección 3.6, esto ocurre cuando la memoria FIFO interna se llena completamente. Es de destacar que los valores de *time-stamp* que envía el equipo permiten alinear de manera correcta los bloques de tiempos de interarribo contiguos.

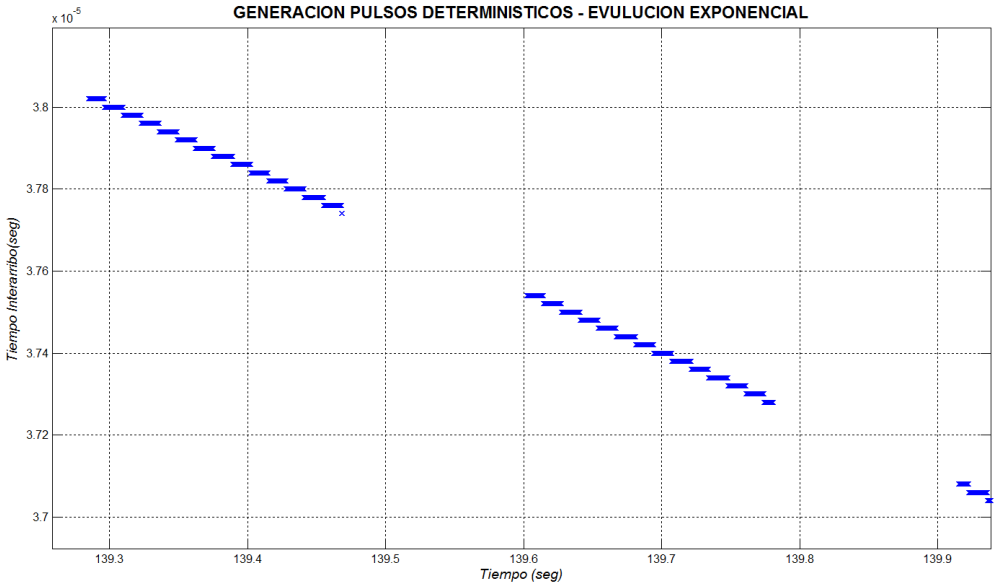


Figura 4.5. Detalle de la Figura 4.1 donde se observa el fenómeno de llenado completo de la memoria FIFO en el módulo de registro de datos del equipo.

4.2 GENERACIÓN DE PULSOS ALEATORIOS

En esta sección se presentan los resultados de la generación de pulsos con tiempos de interarribo aleatorios. Para todos los casos analizados se emuló un sistema de tiempo muerto de características no-paralizable, con un τ = 500nseg.

4.2.1 TASA CONSTANTE

En este modo de funcionamiento se emularon todas las tasas que permite setear el equipo, expresadas en la Tabla 4-1. Para cada caso se analizó la tasa media observada (λ_{MEDIDO}) y se calculó el error absoluto y relativo (porcentual) tomando como valor verdadero a la tasa programada luego de la corrección por tiempo muerto ($\lambda_{VERD.} = \lambda_{PROGR.\ CORREGIDO}$). Además, para verificar que la distribución de los pulsos coincida con la de un proceso de Poisson seguido de un sistema de tiempo muerto, se calculo el *error cuadrático medio MSE* entre la distribución empírica y la distribución teórica, ec. (1-6). Los resultados se resumen en la Tabla 4-3. Cabe aclarar que para cada experimento se tomaron 100e3 valores de tiempos de interarribo.

$\lambda_{PROGRAM.}$ (seg^{-1})	$\lambda_{PROGRAM.\ CORREGIDO}$ (seg^{-1})	λ_{MEDIDO} (seg^{-1})	$\lambda_{PROGRAM.\ CORREGIDO} - \lambda_{MEDIDO}$ (seg^{-1})	$\frac{\lambda_{PROGRAM.\ CORREGIDO} - \lambda_{MEDIDO}}{\lambda_{PROGRAM.\ CORREGIDO}} \cdot 100$ (%)	***MSE***
1	0.9999995	1.0067154	-0.0067159376	-0.67159376	5.0043e-006
10	9.99995	10.041763	-0.041813185	-0.41813185	1.829e-006
100	99.995	99.90697	0.088030463	0.088030463	2.6628e-006
1e3	999.50025	999.75336	-0.25310933	-0.025310933	8.0355e-007
10e3	9950.2488	9956.9543	-6.7055166	-0.067055166	9.3191e-008
100e3	95238.095	95299.471	-61.376179	-0.061376179	8.8273e-008

Tabla 4-3. Resultado de los experimentos con pulsos aleatorios y tasa media constante.

Como puede observase en dicha Tabla, los resultados son muy satisfactorios, con errores muy bajos tanto en la tasas medidas como en las funciones de distribución (esto último debido a los bajísimo valores de *MSE*). La Figura 4.6 resume gráficamente los valores de los errores.

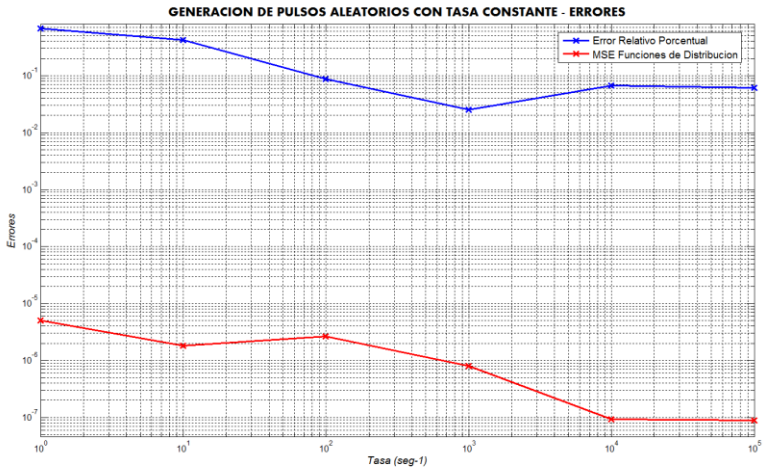
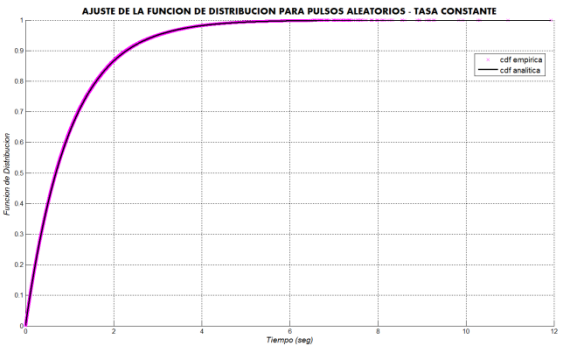
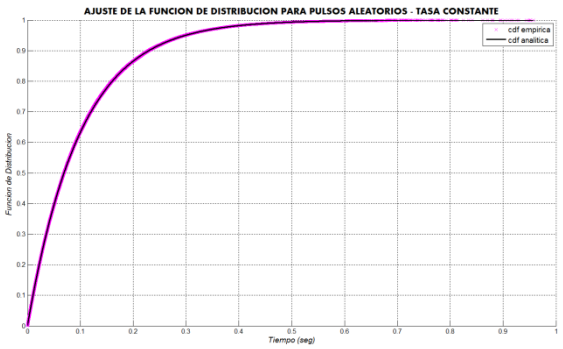


Figura 4.6. Gráfica de los errores en la generación de pulsos aleatorios de tasa media constante.

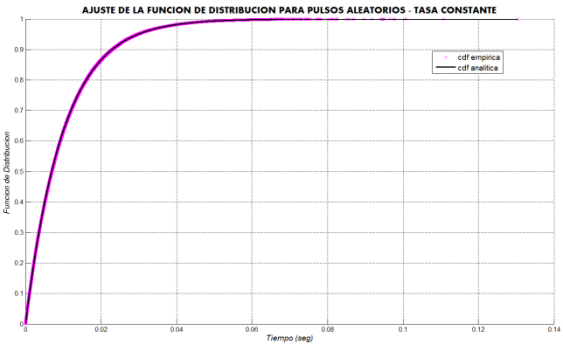
Los bajos valore de MSE indican que las funciones de distribución empíricas se superponen con las versiones teóricas. Las gráficas de la Figura 4.7 muestran este hecho.



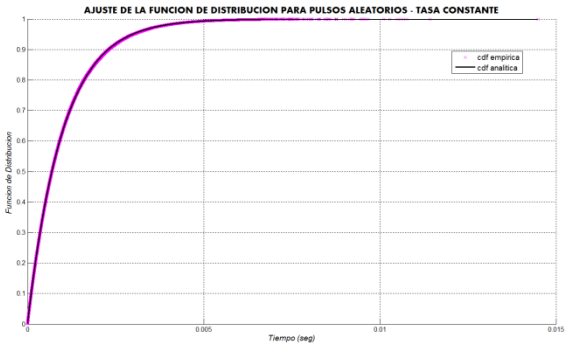
$\lambda_{PROGRAMADO}$ = 1 seg^{-1}



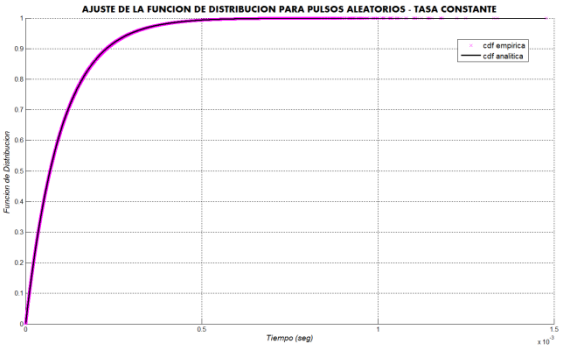
$\lambda_{PROGRAMADO}$ = 10 seg^{-1}



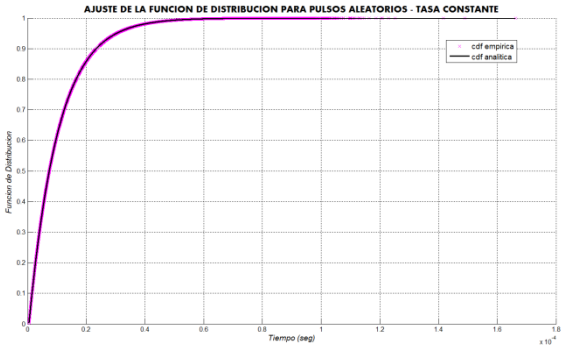
$\lambda_{PROGRAMADO}$ = 10 seg^{-1}



$\lambda_{PROGRAMADO}$ = 1e3 seg^{-1}



$\lambda_{PROGRAMADO}$ = 10e3 seg^{-1}



$\lambda_{PROGRAMADO}$ = 100e3 seg^{-1}

Figura 4.7. Gráficas de las funciones de distribución de los tiempos de interarribo de los experimentos correspondientes a la Tabla 4-3.

4.2.2 TASA VARIABLE

Para analizar el desempeño del equipo al generar pulsos aleatorios con tasa de cuentas de variación exponencial se siguieron los mismos pasos determinístico, Sección 4.1.2, es decir, se realizó un ajuste de curva de los tiempos de interarribo, utilizando el mismo modelo de ajuste de la ec. (4-1). Sin embargo, dada la característica aleatoria de los pulsos, para cada valor de Φ emulado se registraron 10 experimentos de modo de hacer análisis de la dispersión de los resultados. A modo de ejemplo, la Figura 4.8 y Figura 4.9 muestran la

aplicación de la herramienta de ajuste de curvas de *Matlab* sobre pulsos de Φ =0.025 seg^{-1} y 0.040 seg^{-1} respectivamente.

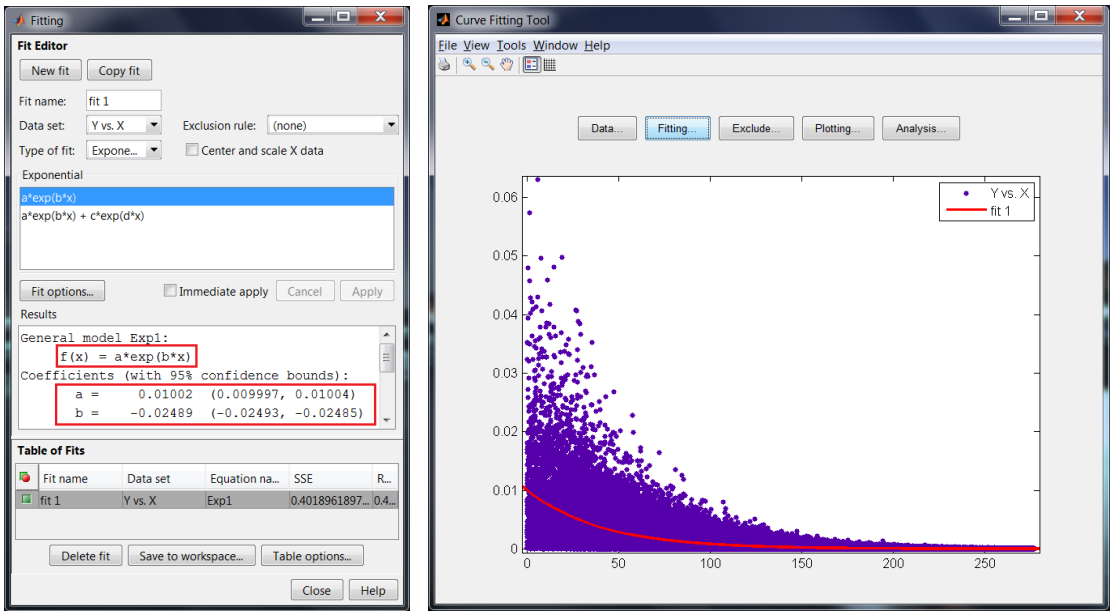


Figura 4.8. Aplicación de la herramienta "Curve Fit Tool" de Matlab para ajustar los pulsos aleatorios y tasa de variación exponencial con Φ =0.025 seg^{-1} y λ_0 = 100 seg^{-1}. La curva expresa los valores de tiempo de interarribo.

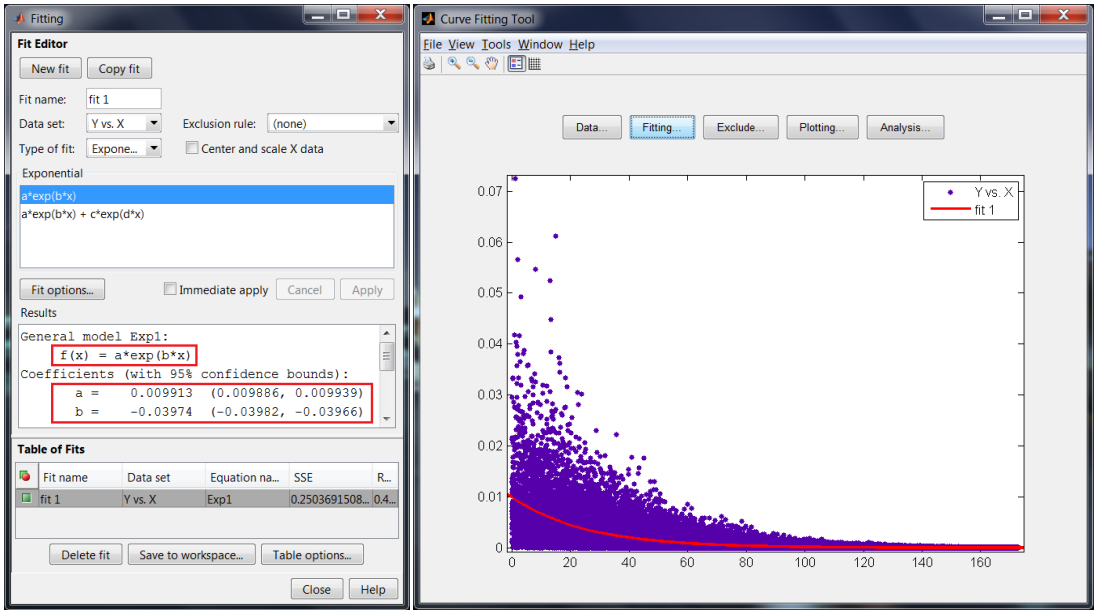


Figura 4.9. Aplicación de la herramienta "Curve Fit Tool" de Matlab para ajustar los pulsos aleatorios y tasa de variación exponencial con Φ =0.04 seg^{-1} y λ_0 = 100 seg^{-1}. La curva expresa los valores de tiempo de interarribo.

La Tabla 4-4 resume los ajustes realizados sobre los 10 experimentos para cada valor de Φ. De allí puede confirmarse que los valores estimados por el ajuste son muy próximos a los valores ideales. Además, la columna de *desviación estándar STD* indica que dispersión en los 10 experimentos es muy baja. Todo esto confirma que el equipo se comporta muy bien a la hora de emular pulsos aleatorios con una tasa de cuentas variable en el tiempo.

Φ (seg^{-1})	**PARÁMETRO**	**VALOR IDEAL**	**AJUSTE**	
			MEDIA 10 mediciones	**STD 10 mediciones**
0.025	a (seg)	0.01	0.0099509	1.1511e-4
	b (seg^{-1})	-0.025	-0.024933	1.4135e-4
0.040	a (seg)	0.01	0.0099995	1.2388e-4
	b (seg^{-1})	-0.040	-0.039975	2.6480e-4

Tabla 4-4. Resultados del ajuste de los experimentos aleatorios de tasa variable con Φ =0.025 seg^{-1} y Φ =0.04 seg^{-1}. En ambos casos λ_0 = 100 seg^{-1}.

5. CONCLUSIONES

En este trabajo se diseñó un equipo generador de pulsos aleatorios, con valores de tasa y tasa de variación seleccionables desde el frente, concebido para formar parte de la instrumentación de arranque, en régimen de pulsos. El equipo es compacto y puede ser alojado cómodamente en formato norma europea (Eurocard) o en formato americano (NIM). Mediante un preprototipo se validó la implementación, lográndose un resultado más que satisfactorio en la simulación del comportamiento de un reactor ya sea en estado estacionario en 6 décadas seleccionables, como en evoluciones a dos tasas de crecimiento, también seleccionables.

Se espera que con este instrumento, se puedan realizar comprobaciones y calibraciones de la instrumentación de arranque en forma rápida y segura. Adicionalmente el equipo es capaz de generar pulsos en forma determinística, tanto en forma estacionaria como con crecimiento exponencial. Este modo de funcionamiento permite evaluar distintos diseños de instrumentación nuclear en cuanto a su capacidad de filtrado de ruido estadístico.

La salida de transmisión serie del equipo posibilita el registro de las comprobaciones realizadas, pudiéndose realizar verificaciones *repetibles* aún en régimen aleatorio.

Para la versión final del equipo queda a considerar la posibilidad de modificar el valor de tiempo muerto desde el frente del módulo, así como también poder seleccionar valores adicionales de tasa de variación Φ.

Finalmente, se vuelve a insistir en que la posibilidad de generar pulsos aleatorios con un crecimiento exponencial de la tasa de cuentas es la característica más sobresaliente del diseño y no existe en el mercado un equipo con prestaciones similares.

6. REFERENCIAS

[1] *G. Knoll, Radiation Detection and Measurement, 2nd Ed., John Wiley & Sons.*

[2] Equipo CNEA E023: Generador de Pulsos Aleatorios Basado en FPGA para Emulación de un Sistema de Detección en un Reactor Nuclear.

[3] J. W. Müller, Dead-Time Problems*, Nucl. Instr. and Meth. 112 (1973), Pag. 47.*

[4] *Digital Systems Design with FPGAs and CPLDs* – Ion Grout - Elsevier - ISBN-13: 978-0-7506-8397-5.

[5] *Digilent Spartan-3 Kit: http://www.digilentinc.com/Products/Detail.cfm?Prod=S3BOARD*

[6] *http://en.wikipedia.org/wiki/Poisson_process - Wikipedia, Poisson Process*

[7] *M. Matsumoto, T. Nishimura - "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator" - ACM Transactions on Modeling and Computer Simulation, vol. 8, No. 1, pp. 3-30, January 1998.*

[8] *http://www.ht-lab.com/freecores/mt32/mersenne.html - Pseudo Random Number Generator for Xilinx FPGA*

[9] *http://www.iro.umontreal.ca/~simardr/testu01/tu01.html - Empirical Testing of Random Number Generators..*

[10] *http://www.iro.umontreal.ca/~simardr/testu01/guideshorttestu01.pdf - A Software Library in ANSI C for Empirical Testing of Random Number Generators - User's guide, compact version.*

7. ANEXOS

7.1 ANEXO 1: TEST DE NÚMEROS ALEATORIOS *"TestU01"*

Para analizar la aleatoriedad de los generadores de números aleatorios existen en la literatura numerosos algoritmos de testeo. Uno de los más conocidos es el test *DIEHARD* [Marsaglia 1996]. Diehard involucra numerosos test estadísticos, pero tiene muchas desventajas. Una de ellas es que los números aleatorios a testear sólo pueden ser enteros de 32 bits; esto es muy restristivo ya que no sirve, por ejemplo, para analizar la aleatoriedad de secuencia de bits. Otra desventaja de Diehard es que el tamaño de las muestras que se toman para el test no es muy grande de modo que la ejecución del test completo tarda unos pocos segundos en una computadora actual, desaprovechando así la velocidad de los CPUs de los que hoy se diponen.

Tests de aleatoriedad "TestU01"

Los autores P. L'ecuyer y R. Simard de la Universidad e Montreal crearon *TestU01* [9], una colección de tests de aleatoriedad mucho más potente que *Diehard*. *TestU01* implementa una variedad más grande de tests, tiene un funcionamiento más eficiente y puede trabajar con grandes bloques de datos (archivos binarios de decenas de Gbytes). Los tests están disponibles para cadenas de bits (no disponible en *Diehard*) y para números aleatorios de *n* cantidad de bits, no estando *n* restringido a *32* como en *Diehard*). La librería también dispone de una colección de generadores de números aleatorios; ellas se pueden utilizar para generar archivos binarios o para hacer de entrada en los tests de aleatoriedad.

TestU01 está implementada en ANSI C y en [9] se puede obtener el código fuente. En dicha página también se encuentra una guía de instalación y compilación de la librerías para Windows y Linux. En la guía de usuario [10] se encuentra una descripción detallada de las rutinas C. A continuación se muestra un ejemplo de código en C que utiliza las librerías para el análisis de un archivo binario que contiene números aleatorios uniformes generados por el algoritmo *MT32 (almacenados en el archivo* `rnd_500e6.bin`*)*:

```
#include <unif01.h>
#include <ufile.h>
#include <bbattery.h>

int main (void)
{
char*  path = "~/work_matlab/rnd_500e6.bin";
      unif01_Gen *gen = ufile_CreateReadBin(path,500e6);

bbattery_pseudoDIEHARD(gen);
ufile_InitReadBin();
bbattery_SmallCrush(gen);

ufile_DeleteReadBin(gen);
return 0;
}
```

El módulo *unif01* (`#include <unif01.h>`) debe ser incluído en todos los programas que utilicen las librerías *TestU01; ufile* (`#include <ufile.h>`) permite implementar generadores de números extraídos directamente de archivos y *bbatery* (`#include <bbattery.h>`) contiene baterías de tests de aleatoriedad. A continuación se da una descripción de las funciones utilizadas en el ejemplo anterior.

`unif01_Gen * ufile_CreateReadBin (char *fname, long nbuf);`

Lee números de un archivo de entrada. Se supone que dicho archivo se encuentra en formato binario. Los números son leídos en porciones de 4 `nbuf` *unsigned char* por vez, transformado en `nbuf` enteros sin signo de 32 bits. Esta función es utilizada para testear secuencias (aleatorias) guardadas en un archivo.

`void bbattery_pseudoDIEHARD (unif01_Gen *gen);`

Aplica la batería de test *PseudoDIEHARD*, que implementa la mayoría de los tests del popular conjunto de tests *DIEHARD*. No se recomienda esta batería de tests pues no son muy exigentes.

`void ufile_InitReadBin (void);`

Reinicia el generador -obtenido por el comando `ufile_CreateReadBin`- al inicio del archivo.

`void bbattery_SmallCrush (unif01_Gen *gen);`

Esta función aplica *SmallCrush*, una batería de 10 tests. Se utiliza principalmente para chequeos rápidos. Muchos de los tests en *SmallCrush* asumen que el generador devuelve números de al menos 30 bits de resolución, de modo que si éste no es el caso, dichos tests fallarán.

`void ufile_DeleteReadBin (unif01_Gen *);`

Cierra el archivo y libera la memoria dinámica utilizada por `ufile_CreateReadBin`.

La librería *TestU01-1.2.3* ha sido instalada en el servidor de *I&C*, el cual dispone de un SO Linux Debian Lenny con un Kernel 2.6.26-2-amd64. El programa se ha instalado en `/usr/local/TestU01-1.2.3` y las librerías y los includes también se han copiado en `/usr/lib` y `/usr/include` respectivamente de modo de facilitar la compilación de los programas que se generen. Así, por ejemplo, para compilar el ejemplo anterior se utilizó el compilador `gcc` con los siguientes parámetros:

```
gcc main.c -o test1 -ltestu01 -lprobdist -lmylib -lm
```

Esto genera el archivo ejecutable `test1`. Para correrlo, basta introducir `./test1`. Si se quiere direccionar la salida con los resultados a un archivo de texto se puede correr `test1` de la siguiente manera:

```
./test1 > test1.resultado
```

7.2 ANEXO 2: ARBOL DE JERARQUÍA DEL CÓDIGO VHDL

La Figura 7.1 muestra la estructura completa de la implementación en VHDL de equipo.

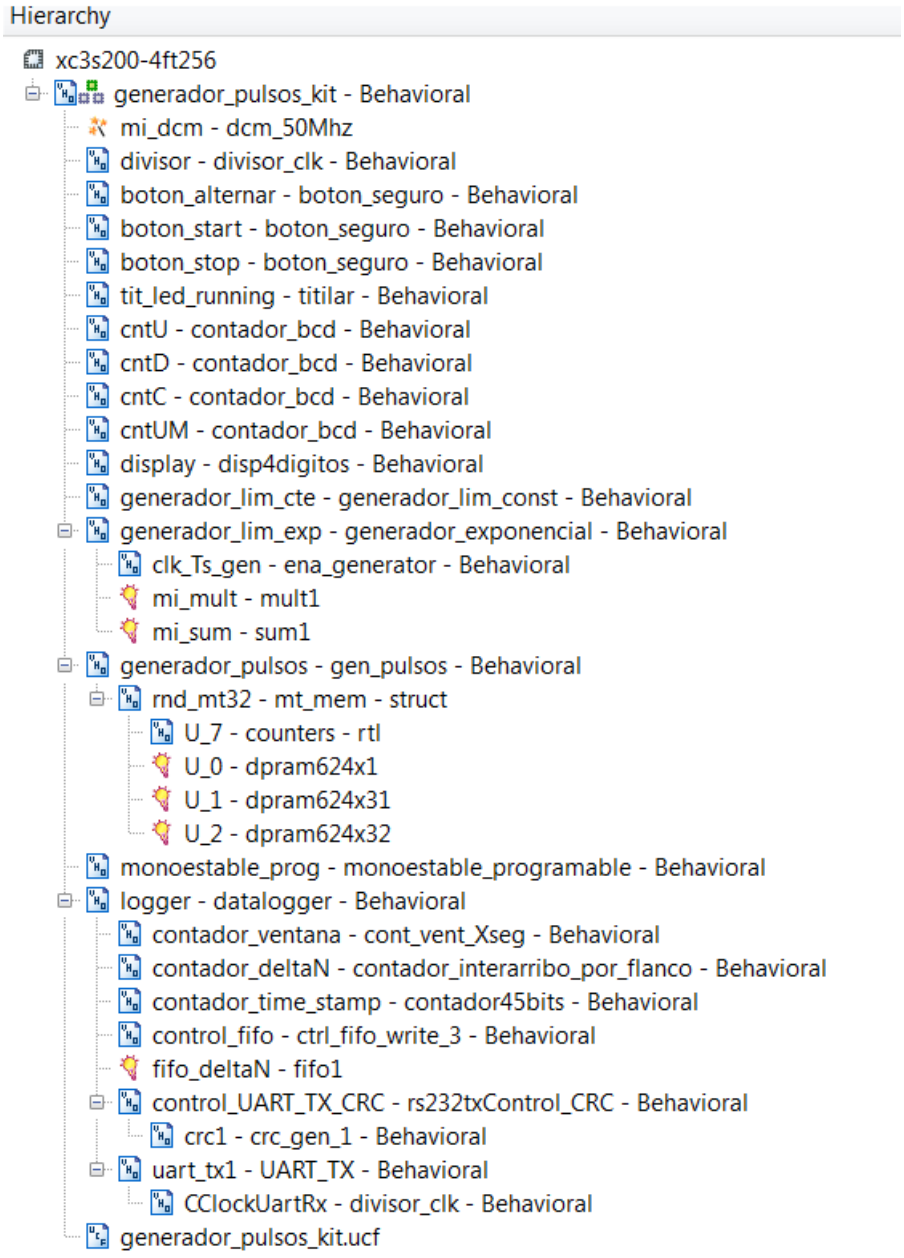


Figura 7.1. Estructura jerárquica de la implementación en VHDL del equipo

El código correspondiente a cada uno de los módulos de la Figura 7.1 se encuentra en el CD que acompaña esta tesina.