Diseño e implementación de un emulador de canal BPL en FPGA

por

Nicolás Matsunaga

Departamento de Electrónica
Facultad de Ingeniera, Universidad de Buenos Aires

Fecha: _____

_____
Directora: Phd. Ing. Cecilia G. Galarza

_____
Co-Director: Dr. Ing. Leonardo Rey Vega

_____
[Ing. Jorge A. Alberto]

_____
[Ing. Carlos F. Belaustegui Goitia]

_____
[Dr. Ing. Mario Hueda]

2012

Resumen

En el presente trabajo se realiza un estudio detallado del canal de comunicaciones en el ámbito de transmisiones de banda ancha a través de líneas de tensión, o ámbito BPL[1]. Analizaremos los distintos escenarios que se presentan y estudiaremos las características de cada uno.

Estudiaremos la respuesta en frecuencia y el ambiente de ruido, clasificando los distintos tipos de ruido según su característica espectral y/o temporal cuando sea más conveniente. Se plantearán modelos matemáticos para cada tipo de ruido.

El objetivo de este trabajo es el desarrollo e implementación de un emulador de canal BPL que es de suma importancia en las etapas de diseño, verificación y validación. La emulación plantea un desafío distinto al de la simulación debido a que se debe lograr un resultado similar pero en tiempo real. Para ello, utilizaremos tecnologías digitales para lograr nuestro objetivo. En particular emplearemos la tecnología FPGA[2] que brinda una gran flexibilidad y performance al mismo tiempo, además de ser una tecnología ubicua en estos tiempos.

[1] Broadband over Power Lines

[2] Field Programmable Gate Array

Imagination is more important than knowledge...
– Albert Einstein

Índice general

Índice de cuadros

Índice de figuras

Lista de Abreviaturas y Glosario

Abreviaturas

ADC	Analog to Digital Converter
AR	Auto Regressive process
API	Application Programming Interface
AWGN	Additve White Gaussian Noise
BIBO	Bounded Input Bounded Output
BPL	Broadband over Power Lines
DA	Distributed Arithmetic
DAC	Digital to Analog Converter
DDS	Direct Digital Synthesis
DFT	Discrete Fourier Transform, según [28]
DSO	Digital Storage Oscilloscope
DSP	Digital Signal Processor
EMC	ElectroMagnetic Compatibility
FFT	Fast Fourier Transform, implementación eficiente de la DFT
FIFO	First In First Out
FIR	Finite Impulse Response
FLOP	Floating-point OPeration
GMAC	Giga Multiply and ACcumulate
GPU	Graphics Processor Unit
GUI	Graphical User Interface
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
HomePNA	Home Phoneline Networking Alliance
HPA	HomePlug Powerline Alliance

IFFT	Inverse Fast Fourier Transform
ISI	InterSymbol Interference
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector
LFSR	Linear Feedback Shift Register
LUT	Look-Up Table
MAC	Multiply and ACcumulate
Msps	Mega samples per second
OFDM	Orthogonal Frequency-Division Multiplexing
OPERA	Open PLC European Research Alliance
PLC	Power Line Communications
PPDU	Physical Protocol Data Unit
PSD	Power Spectral Density
PVC	Polyvinyl Chloride o Policloruro de Vinilo
SCR	Silicon-Controlled Rectifier
SDK	Software Development Kit
RTL	Register Transfer Level
UPA	Universal Powerline Association
WPF	Windows Presentation Foundation

Glosario

NAYY	Tipo de cable de 1 o más conductores sólidos de aluminio con aislación de PVC según norma DIN VDE 0276
NKBA	Tipo de cable de 3 conductores con aislación de papel y vaina de plomo según norma DIN VDE 0276
NYY	Tipo de cable de 1 o más conductores sólidos de cobre con aislación de PVC según norma DIN VDE 0276

Agradecimientos

Ante todo quiero agradecer a mis padres por su apoyo incondicional y haberme dado la oportunidad de estudiar. A mi hermano por haberme hecho descubrir inconscientemente el gusto por la ingeniería electrónica.

A la Dra. Ing Cecilia Galarza por haber creído en mi, por su infinita paciencia y comprensión. Gracias Cecilia!!!. Por su humildad, calidad de persona y su sentido del humor siempre tan oportuno :)

Al Dr. Ing. Leonardo Rey Vega quién me guió durante una etapa de este trabajo. Por su atenta predisposición y las charlas amenas.

1

Introducción

Una de las características importantes de la red de distribución eléctrica es el nivel de omnipresencia que ésta posee. Tal es así que ofrece un enorme potencial para transportar comunicaciones de datos rápidas y confiables.

Las comunicaciones a través de líneas de tensión no son nuevas y datan de hace 8 décadas atrás [8, 14]. Sin embargo, las redes eléctricas fueron diseñadas con el propósito de minimizar las pérdidas de energía que existen en la transmisión y no contemplaron requisitos específicos para las telecomunicaciones. Aún así, no se tardó mayor tiempo en lograr una transmisión de energía "sin pérdidas" y una transmisión confiable de datos a niveles aceptables. En un principio, solamente las compañías proveedoras de servicios de electricidad podían obtener beneficios de este hecho, pero esta situación ha cambiado en algunos países con la desregulación de los mercados de telecomunicaciones y energía alrededor de fines de la década del 90.

La desregulación ha planteado un nuevo escenario donde las compañías proveedoras de servicios eléctricos deberán enfrentarse a una futura competencia. Y es por ello, que éstas querrán incorporar nuevas ramas de negocios en el mercado desregulado de las telecomunicaciones, donde existe un gran potencial de crecimiento. En este nuevo contexto la red de mayor penetración a nivel mundial podría ser utilizada como la última milla para servicios de comunicaciones y así constituir una interesante alternativa a lo ofrecido por las compañías telefónicas, actuales dominantes en este segmento. Por estos motivos, las proveedoras de servicios eléctricos desean expandir el producto de la electricidad y sumarle mayor valor agregado. Las posibles aplicaciones pueden ir desde la lectura remota del medidores de energía, los servicios para la automatización del hogar, las comunicaciones intra-vehículares, la transmisión de datos de banda ancha e Internet, variedad y transparencia de planes de tarifas, hasta la provisión de servicios necesarios para poder alcanzar el objetivo de las futuras redes inteligentes que hoy se conoce como *smart grid*.

Las posibilidades y consecuencias de esta transformación serán tan fundamen-

tales y comprensivas que son difíciles de predecir. Es decir que existe un enorme potencial de innovación ante la integración de las comunicaciones de datos junto con la distribución de energía.

1.1. Broadband over Power Lines

Las tecnologías PLC pueden dividirse en dos grupos: *narrowband* PLC y *broadband* PLC. Los sistemas *narrowband* tienen tasas de datos relativamente bajas (hasta 100 kbps) y se suelen utilizar en aplicaciones de automatización y control así como también en el transporte de algunos canales de voz. Los sistemas *broadband* son aquellos que alcanzan tasas superiores a los 2 Mbps y permiten la realización de varios servicios de telecomunicaciones en paralelo, como por ejemplo la telefonía y el acceso a Internet. A esta tecnología se la conoce con la sigla BPL, *Broadband over Power Lines*.

Actualmente existe una creciente actividad en BPL en varios países, y varias compañías ofrecen productos con tasas de transmisión de hasta 1 Gbps para redes in-house. A su vez se ha trabajado en la elaboración de varios estándares sobre el tema:

IEEE 1675	"Standard for Broadband over Powerline Hardware"
IEEE 1775	"Power Line Communication Equipement - Electromagnetic Compatibility (EMC) Requirements - Testing And Measurement Methods"
IEEE 1901	"Standard for Broadband over Power Line Networks: Medium Access Control and Physical Layer Specifications"
ITU-T G.9960	(G.hn) es el nombre conocido (*home network*) para el estándar desarrollado por la International Telecommunication Union, que a su vez tiene en cuenta las redes de acceso. La especificación de G.hn define la capacidad de montar redes sobre líneas de tensión, líneas de teléfono, cable coaxil y UTP categoría 5 con tasas de hasta 1 Gbps [23].

y varias alianzas u organizaciones que trabajan o trabajaron alrededor del mismo

OPERA	Open PLC European Research Alliance que fue un proyecto de investigación y desarrollo que trabajó en una nueva generación de red PLC integrada. (http://www.ist-opera.org)
HPA	HomePlug Powerline Alliance, cuya especificación HomePlug AV es totalmente interoperable con el estándar IEEE 1901. (http://www.homeplug.org/)

HomeGrid Forum	conformada por empresas como: AT&T, BT, Intel, Marvell, Motorola, Sigma Designs, Telefonica, Lantiq, Corinex, Best Buy. Esta alianza aboga por la norma ITU-T G.hn. (http://www.homegridforum.org/)
HomePNA	Home Phoneline Networking Alliance, conformada por empresas como: Cisco, AT&T, Motorola, K-micro, Sigma Designs, Sunrise Telecom. Esta alianza boga por la norma ITU-T G.hn. (http://www.homepna.org/)
UPA	Universal Powerline Association es una organización abierta que busca cubrir todos los mercados y todas las aplicaciones PLC

Muchas de estas organizaciones hicieron grandes aportes para lograr el desarrollo de los estándares de la IEEE como también el de la ITU-T.

Uno de los atractivos de BPL es la reutilización de la infraestructura para brindar servicios de datos de última milla. A diferencia de tecnologías como ADSL o redes de cable coaxil, BPL cuenta con la ventaja de requerir menores modificaciones en la infraestructura para ser operativa. Mientras que en una red de cable se deben adecuar la mayoría de los amplificadores para que soporten comunicaciones bidireccionales, en BPL deben instalarse inyectores y acondicionadores de señales que son de menor costo y que su instalación se realiza sin provocar interrupciones sobre el servicio de provisión eléctrica [13, p. 233]. A su vez, BPL es capaz de llegar a zonas rurales donde las empresas de telecomunicaciones o los servicios de radio no tienen penetración a nivel infraestructura o no son aptos para la transmisión de datos de banda ancha.

Sin embargo, no todo lo que brilla es oro ya que los sistemas PLC deben atenerse a las estrictas regulaciones para poder satisfacer requisitos de compatibilidad electromagnética (EMC). Si bien las frecuencias de trabajo podrían considerarse bajas para un ingeniero electrónico diseñando un circuito impreso, las longitudes de onda pueden ser del orden de los metros y los elementos conductores pueden llegar a tener cientos de metros, que son condiciones suficientes para que los cables se comporten como antenas. Por lo tanto, el rango dinámico y la calidad de la señal están acotados por la potencia máxima que se puede inyectar y por los ruidos propios de las redes eléctricas que son inusuales para un sistema de telecomunicaciones.

Por lo tanto, para poder lograr el objetivo de transmitir datos a tasas elevadas se requiere la aplicación de complejas técnicas de modulación y codificación para minimizar el impacto de las perturbaciones descriptas.

1.2. Necesidad de un emulador

Para el desarrollo eficiente y la prueba de sistemas PLC de alta velocidad es necesario contar con bancos de ensayo estandarizados. De esta manera, cada paso en el desarrollo de un módem puede ser verificado de forma inmediata y reproducible. Por ejemplo, diferentes esquemas de modulación pueden ser evaluados en etapas tempranas de un nuevo desarrollo. Más aún, en nuestro caso el ambiente de pruebas es de

vital importancia ya que las comunicaciones PLC plantean un desafío adicional para los ingenieros en telecomunicaciones debido a que deben lidiar con canales que nunca fueron diseñados para la transmisión de señales de alta frecuencia. Además, un emulador es útil para comparar la performance de módems de distintos fabricantes bajo las mismas condiciones repetibles de la capa física. Sin embargo, hay que saber cómo diseñar las pruebas ya que si éstas se enfocan en capas superiores el resultado puede ser engañoso y no comparable debido a que varios y distintos métodos de corrección de errores por codificación pueden estar enmascarando el verdadero comportamiento físico.

El problema en desarrollar un emulador de canal para sistemas PLC de alta velocidad es poder hacerlo en tiempo real. Eso implica que la emulación de la respuesta impulsiva y la generación de ruidos debe realizarse a tasas superiores a los 60 Msps[1]. Por ejemplo, la duración típica de la respuesta impulsiva en una red de acceso es de 10μs lo que equivale a 600 coeficientes si implementamos dicho filtro con un FIR a 60 Msps y a $3{,}6{\cdot}10^{10}$ operaciones MAC[2] por segundo. Las restricciones son tales que ni los microprocesadores de propósito general ni los procesadores de señales digitales (DSP) pueden satisfacer dichos requisitos. Entonces ahí es donde entran en escena los FPGAs, cuya solución es proveer grandes cantidades de cálculo computacional en paralelo. Aún así, veremos que deberemos modificar el modelo para que superar las limitaciones de hardware.[13]

1.3. Aportes

Los aportes producto de la realización de esta tesis son:

- el diseño de una arquitectura para la emulación del canal BPL. Para ello se estudió el canal y los modelos existentes. Se generó un diseño modularizado que sea fácilmente adaptable para la emulación de otros tipos de canales a través del simple intercambio de bloques. Además, las capacidades de emulación del diseño logradas excedieron las propuestas en [3].

- la implementación del emulador del canal BPL en una plataforma FPGA. Se hizo especial énfasis en el desarrollo de una interfaz de control pensada para que personas no interiorizadas con el desarrollo de esta tesis puedan utilizar el emulador.

- establecer una metodología *general* de control para la lógica de procesamiento de señales a través de un *soft-processor* implementado en el FPGA. De esta manera podemos aprovechar las ventajas que nos brinda el software respecto de una solución puramente por hardware. Ver §6.2.2.

[1] Mega samples per second. Millones de muestras por segundo.

[2] MAC: Multiply and ACcumulate

- generar documentación que refleje el *know-how* adquirido para facilitar la implementación de futuros proyectos del Laboratorio de Procesamiento de Señales de las Comunicaciones (LPSC) de la Facultad de Ingeniería de la UBA. Esto se logra en parte al considerar esta tesis como una guía de los pasos necesarios para desarrollar una implementación y por otra parte en base a los tutoriales creados para el uso de herramientas de *Xilinx*.

1.4. Organización del trabajo

La estructura de contenido de esta tesis se organiza como muestra la figura 1.1.

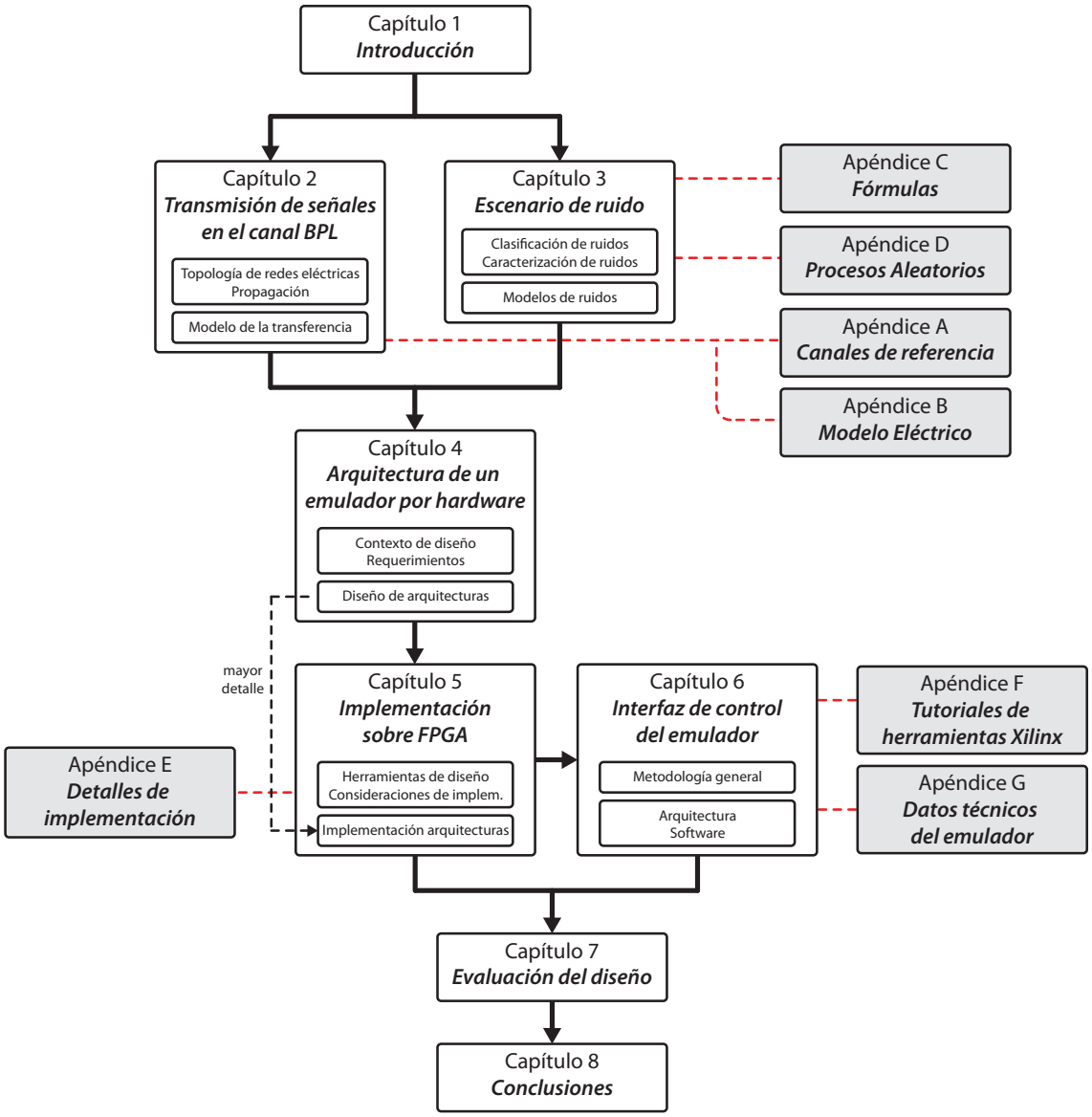


FIGURA 1.1: Estructura de la tesis

El Capítulo 1 provee una breve introducción al tema de las comunicaciones sobre líneas de tensión y la necesidad de una plataforma de pruebas. El Capítulo 2 introduce al lector en la topología de las redes eléctricas que representan el contexto en que se dan las comunicaciones de datos. Luego se trata la propagación de las señales en dichas redes y se plantean modelos para los canales: de las redes de acceso y las redes in-house. El Capítulo 3 estudia el escenario de ruido presente en el canal BPL que posee características especiales frente a los canales tradicionales para transmisión de datos. También se plantean modelos para cada uno de los tipos de ruido.

Los Capítulos 4, 5 y 6 tratan sobre el diseño e implementación del emulador. El Capítulo 4 y 5 tienen una relación especial ya que parte del índice parecería repetirse exceptuando por el intercambio de las palabras "arquitectura" e "implementación". Sin embargo, en el primero se desarrolla la arquitectura explicando como se implementarían los modelos propuestos en los Capítulos 2 y 3 en un FPGA. Mientras, que en el segundo se ve la ingeniería de detalle de la arquitectura propuesta en el anterior aplicando la técnica *Model-Based Design* mediante el software *Xilinx System Generator* y teniendo la plataforma de la que disponemos. El Capítulo 6 trata sobre el desarrollo de la interfaz de control del emulador y la introducción de una metodología general de control de lógica de procesamiento de señales aplicable a otros diseños.

El Capítulo 7 evalúa los diseños de implementación expuestos en el Capítulo 5 acorde a los requerimientos del emulador.

Finalmente, el Capítulo 8 presenta las conclusiones de este trabajo y propone distintas ramas de continuación del mismo.

Los apéndices contienen la siguiente información

Apéndice A presenta los canales de referencia propuestos por el consorcio OPERA [1].

Apéndice B trata algunos desarrollos matemáticos sobre el modelo eléctrico.

Apéndice C contiene un desarrollo matemático sobre una aproximación.

Apéndice D contiene un repaso de los procesos aleatorios que utilizamos junto con algún desarrollo nuestro.

Apéndice E trata sobre detalles de la implementación que merecían ser tratados apartes para no dificultar la lectura del Capítulo 5.

Apéndice F contiene los tutoriales sobre las herramientas de *Xilinx* utilizadas.

Apéndice G contiene datos técnicos del emulador implementado.

2

Transmisión de señales en el canal BPL

El estudio de los sistemas de comunicaciones nos ha enseñado que es necesario conocer en detalle las características del canal para poder lograr un alto rendimiento del mismo [17]. Es por eso que procederemos a describir las propiedades del mismo para poder encontrar modelos matemáticos apropiados para la simulación o emulación que serán aplicados en las etapas de diseño. Para ello debemos abordar al Canal BPL en su marco contextual.

2.1. Topología de las redes eléctricas

Las redes eléctricas están típicamente divididas en tres secciones de diferente tensión: alta tensión, media tensión y baja tensión. Desde el punto de vista de las comunicaciones no todas las partes de la red son de igual importancia.

2.1.1. Red de alta tensión

Las redes de alta tensión se utilizan para el transporte de energía eléctrica de larga distancia. Los cables suelen ser tendidos aéreos pero también pueden ser cables soterrados. Los cables soterrados son más costosos pero tienen la ventaja de tener menor impacto ambiental, muy bajo mantenimiento y mayor facilidad en la instalación y la unión de los mismos. La longitud de los cables suele ser decenas, cientos o hasta miles de kilómetros. Los cables soterrados pueden llegar a tener hasta decenas de kilómetros mientras que los tendidos aéreos pueden ser mucho más largos, llegando a récords de 2100 km en China en la línea Xiangjiaba-Shanghai [42].

Por otro lado, en la red de alta tensión suele haber tendidos de fibra óptica junto a la protección contra descargas atmosféricas que permiten velocidades de transmisión de varios gigabits por segundo. Sin embargo, en el caso de líneas viejas puede no existir la fibra óptica y en esos casos se utiliza PLC. Vale la pena recalcar que en

las líneas de alta tensión debido al elevado valor del campo eléctrico alrededor de los conductores o aisladores se producen algunos fenómenos particulares: el efecto Corona y las descargas luminiscentes. Las descargas por efecto Corona no sólo representan una pérdida de energía sino que son una fuente importante de interferencia para las altas frecuencias. Existen medidas para mitigar dicho efecto pero el estudio de las mismas excede al alcance de esta tesis. Por otro lado, las descargas luminiscentes también contribuyen como fuente de ruido para las altas frecuencias.

2.1.2. Red de media tensión

La red de media tensión se utiliza para el transporte de energía en áreas urbanas, suburbanas, rurales y plantas industriales. La tensión típica de las líneas suele ser de 10 kV a 20 kV y la longitud de las mismas de 5 km a 25 km. Los elementos conductores pueden ser tendidos aéreos o cables subterráneos dependiendo de la arquitectura de la zona y la densidad de urbanización. En zonas densamente pobladas se usan exclusivamente cables subterráneos. En algunos casos se acompaña al cableado de media tensión con cables coaxiles y fibras ópticas, a través de los cuales se establecen las comunicaciones de datos. De no ser así se utiliza PLC. El escenario rural puede ser un importante beneficiado de las comunicaciones BPL ya que suele haber poca penetración de las empresas de telecomunicaciones.

2.1.3. Red de baja tensión o red de distribución

La red de baja tensión o red de distribución es la de mayor interés ya que es vista como la "última milla" hacia el usuario. Este tramo de red entre la subestación y las instalaciones del cliente, llamado red de acceso, suele operar con una topología estrella.

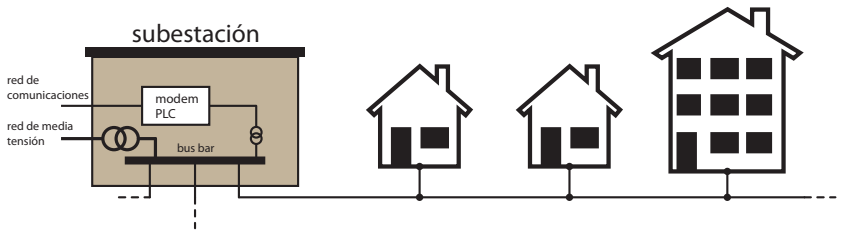


FIGURA 2.1: Topología de una red de distribución de baja tensión

La figura 2.1 muestra que la subestación alimenta la barra[1]. Si la subestación es utilizada para transportar comunicaciones de datos sobre líneas de tensión, o *Power Line Communications* en su versión anglosajona, la señal de datos o señal PLC también se aplica sobre la barra. Por otro lado, la conexión física entre la subestación y el backbone de la red puede ser realizada mediante medios convencionales como

[1] conocida como *bus bar* en inglés

fibra ópticas, radioenlaces o cables de banda ancha [48]. De la barra suelen salir varios cables de distribución (entre 3 y 10 típicamente) que van hacia las inmediaciones de los clientes. Cada uno de estos cables suele proveer a decenas o centenas de hogares. Dichos cables suelen estar soterrados, excepto en algunas zonas rurales donde pueden llegar a ser tendidos aéreos de cable. Cada tipo de cable tiene distintas características de transmisión. La longitud de los cables suele ser de algunos cientos de metros para mantener las pérdidas al mínimo pero puede variar de caso en caso.

Una característica de las redes de baja tensión es la gran variabilidad de las mismas, debido a que éstas están conformadas por el uso de diversas tecnologías (distintos tipos de cables, transformadores, protecciones, etc) y son instaladas de acuerdo a normas que son diferentes en cada país. Además, la topología de una red difiere de lugar en lugar y depende de varios factores como:

- *Ubicación*: una red PLC puede estar situada en un área residencial, industrial o rural.
- *Densidad de usuarios*: los usuarios pueden estar ubicados en una zona de casas aisladas (baja densidad) (area rural), en barrios residenciales (media densidad), en edificios con un gran número de departamentos u oficinas (alta densidad) o en edificios en torre de departamento u oficinas (muy alta densidad),
- *Longitud de la red*: la distancia entre un transformador y un usuario es variable de lugar en lugar pero usualmente se puede decir que existe una gran diferencia de longitud entre el caso urbano y el caso rural.
- *Diseño de la red*: la red de baja tensión suele constar de varias secciones (ramificaciones) de red que también varían de red en red.

Aún considerando el grado de variabilidad, también es bueno hablar de dimensiones típicas [21, 22] para tener una visión concreta. Vale la pena recalcar que estos números representan usuarios *potenciales* de servicios de datos que usan PLC.

- Número de usuarios en la red: $\sim$ 250 a 400
- Número de secciones de la red: $\sim$ 5
- Número de usuarios en una sección de red: $\sim$ 50 a 80
- Longitud de una sección de red: $\sim$ 500 m

Desde el punto de vista de los límites, la red de acceso termina en las cajas de conexión de las viviendas (ver figura 2.2). Luego de la caja de conexión el cable puede separarse hacia distintos medidores de energía si es necesario. Se aprecia en la figura 2.2 que el módem PLC se encuentra conectado en paralelo a la caja de conexión, separando la red de acceso de la red interna del edificio. Este método permite el uso de distintos rangos de frecuencia para la red acceso y la red interna. La ventaja es que las redes de acceso se caracterizan por una fuerte respuesta pasa bajos, por lo que se utilizan bajas frecuencias, y en las redes internas no teniendo tales limitaciones se pueden usar frecuencias mayores.

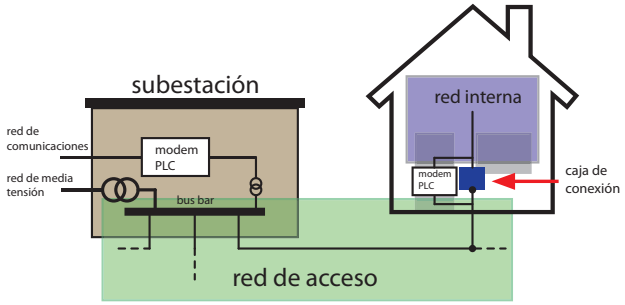


FIGURA 2.2: Topología de una red de distribución de baja tensión

2.1.3.1. Extendiendo la capacidad

Hasta ahora hemos observado un caso simplificado donde hay un único módem que se conecta a la barra y por lo tanto, todos los cables de distribución (ver nomenclación en figura 2.6) comparten la misma señal PLC y por ende abarca a todos los clientes de dicha subestación. Una manera de aumentar la capacidad de transmisión sería tener varios canales de comunicación independientes. Esto se puede hacer instalando bloqueadores de RF y alimentando cada cable de distribución con un módem propio tal como muestra la figura 2.3. También se pueden utilizar bloqueadores similares en el límite entre la red de acceso y la red hogareña para permitir el reuso de frecuencias.

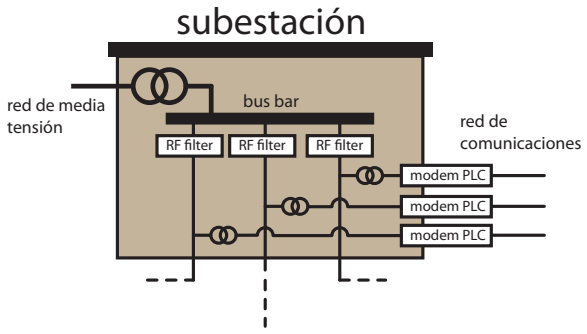


FIGURA 2.3: Inyección múltiple de señal PLC

Para la implementación del filtro y el acoplamiento de la señal PLC se ha estudiado y desarrollado [16] una variedad de implementaciones tanto inductivas como capacitivas que exceden al alcance de esta tesis.

2.1.3.2. Extendiendo la distancia

En el caso de que la red de accesso presente una característica muy dañina para la señal PLC es posible utilizar repetidores intermedios. De hecho la red PLC propuesta por OPERA consta de tres tipos de nodos:

- Head End equipement (HE) que conecta a la red PLC con la red de *backbone* de telecomunicaciones.
- Frequency/Time Repeater Equipement (R) que se utiliza para extender el rango de cobertura.
- Customer Premise Equipement (CPE) que conecta al usuario final con la red PLC.

En la figura 2.4 se puede observar un ejemplo de una topología de red que contiene a los tres tipos de nodos mencionados.

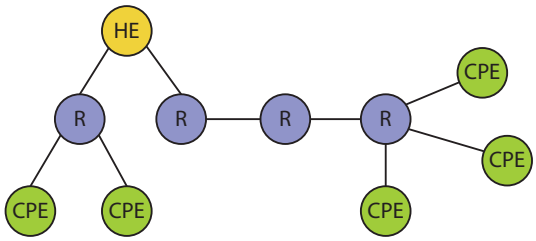


FIGURA 2.4: Topología de una red PLC según OPERA

2.2. Propagación de señales en un ámbito PLC

Como en todo sistema de comunicaciones, la señal que contiene la información se ve alterada a medida que atraviesa el medio de propagación. En el caso de PLC dicho medio se caracteriza a grandes rasgos por dos efectos: la característica pasabajos y el desvanecimiento selectivo de frecuencias. Como ambos efectos son independientes pueden analizarse por separado.

2.2.1. Característica Pasabajos

Como mencionamos anteriormente, los cables de la red eléctrica suelen estar soterrados. Dichos cables fueron diseñados para una óptima transferencia de energía con bajas pérdidas a frecuencias muy bajas, lo cual no implica que sean buenos para la transferencia de información. Múltiples mediciones han determinado que estos cables tienen una marcado característica pasabajos, que depende del tipo de cable y de la longitud. La figura 2.5 [1] muestra la atenuación en función de la frecuencia y la longitud para distintos tipos de cables. La característica pasabajos es causada por pérdidas dieléctricas en la aislación del cable. Esto explica también por qué los cables aéreos no evidencian un comportamiento pasabajos. La figura 2.5 muestra que el cable tipo N(A)YY con su aislación de PVC presenta mayor atenuación que el cable tipo NKBA con aislación de papel. Los cables de una red eléctrica hogareña muestran una *débil* característica pasabajos debido a su corta longitud.

La característica pasabajos de la red de acceso limita la máxima distancia que pueden soportar los sistemas de comunicaciones PLC y a su vez el rango máximo de

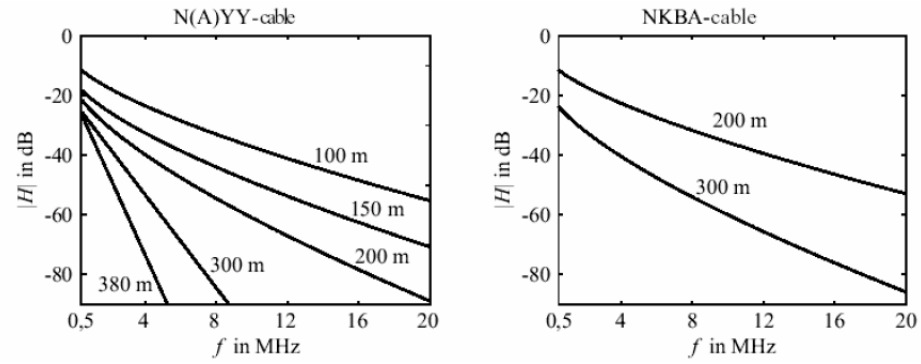


FIGURA 2.5: Atenuación en función de la longitud y el tipo de cable

frecuencia útil. Como mencionamos anteriormente, se ha encontrado que es razonable dividir el rango de frecuencias de manera que las bajas frecuencias (menores a 10 MHz) sean utilizadas para la red de acceso y las altas frecuencias, entre 10 MHz y 30 MHz, para la red eléctrica hogareña. De todas maneras, debe tomarse en cuenta que para largas distancias en la red de acceso a veces sólo es posible obtener una tasa de datos reducida o incluso no ser viable transmisión alguna. La atenuación afecta la funcionalidad de los sistemas PLC debido a que la potencia de transmisión no puede incrementarse irrestricamente, ya sea debido a razones de compatibilidad electromagnética o a motivaciones físicas prácticas o económicas. Como consecuencia, si la atenuación excede un valor límite la señal no podrá recibirse ya que será irrecuperable.

2.2.2. Desvanecimiento selectivo de frecuencias

La red de acceso entre la subestación y las inmediaciones del clientes suele operar en una topología estrella. Desde un punto de vista de comunicaciones es equivalente a un sistema de radio que consiste en celdas y radiobases y que el medio es compartido. A diferencia de la telefonía terrestre, en la red de acceso PLC no se tienen conexiones punto a punto entre la subestación y las instalaciones del cliente, sino que se tiene un *bus* formado por los cables de distribución y los cables de servicio a las casas. En la figura 2.6 observamos la nomenclatura utilizada para los cables.

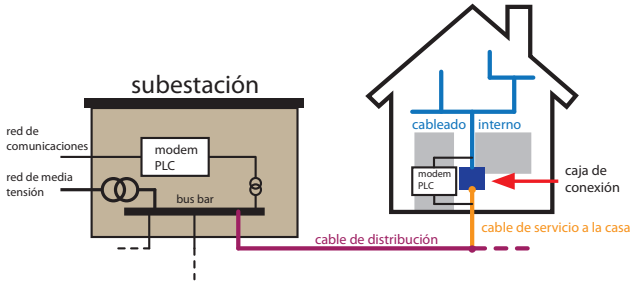


FIGURA 2.6: Convención de nombres de cables

Una típica red de acceso entre la subestación y el cliente consiste en el cable de distribución (o conexiones en series de cables de distribución) con impedancia característica Z_{L_i} (en cada tramo serie) y las conexiones ramificadas hacia las casas con impedancias características $Z_{L,H}$.

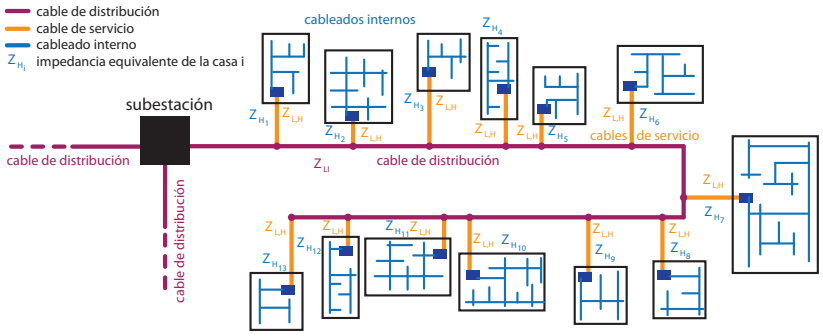


FIGURA 2.7: Ejemplo de red de distribución

Los cables de servicio a las casas terminan en la caja de conexión de la casa. El conjunto de los cables internos de la casa, del otro lado de la caja de conexión, es modelado como una terminación con impedancia $Z_{H_j}(f)$. Cada una de las transiciones en las conexiones de cables representa un cambio de impedancia que genera reflexiones. En la figura 2.7 vemos un ejemplo de red de distribución donde se indica cada uno de los elementos que intervienen.

Debido a las ramificaciones y las reflexiones, la señal no sólo se propaga en forma directa de transmisor a receptor, sino que existen caminos de propagación adicionales que deben ser considerados. Estos caminos tienen una longitud mayor y por eso causan ecos. El resultado es una señal que toma múltiples caminos y que presenta desvanecimientos selectivos de frecuencia. En casos extremos, un conjunto de frecuencias puede ser anulado completamente. Otra consecuencia de una propagación de caminos múltiples es la extensión temporal de la respuesta al impulso que es de suma importancia en los sistemas PLC para poder obtener tasas de datos elevadas. Esto se debe a que la duración de la respuesta al impulso puede ser mayor que la duración de un símbolo transmitido y esto produce interferencia inter-símbolo (ISI).

2.3. Efectos físicos en la propagación

En la anterior sección hemos descripto en forma calificativa aspectos generales de la propagación de la señal en un ámbito PLC. Ahora analizaremos los mismos desde un punto de vista científico y cuantitativo.

2.3.1. Atenuación por pérdidas óhmicas

En la subsección §2.2.1 vimos que las señales PLC sufren una atenuación creciente con la longitud de la línea y con la frecuencia, lo que tiene como resultado una

marcada característica pasabajos. Ahora analizaremos más en detalle las causas de estas pérdidas y derivaremos un modelo matemático para las mismas.

El modelo que utilizaremos es el de las líneas de transmisión con pérdidas. El elemento diferencial de la línea puede verse en la figura 2.8 con la definición de las tensiones y corrientes en la posición x, $v(x)$ e $i(x)$, y en un diferencial de distancia $x+\Delta x$, $v(x+\Delta x)$ e $i(x+\Delta x)$. También se observan las magnitudes R' y G' asociadas a las pérdidas. R' se refiere a las pérdidas óhmicas en el material conductor mientras que G' hace referencia a las pérdidas en el aislante dieléctrico. Los parámetros L' y C' corresponden a la inductancia y a la capacitancia por unidad de longitud respectivamente que se asocian a la velocidad de propagación.

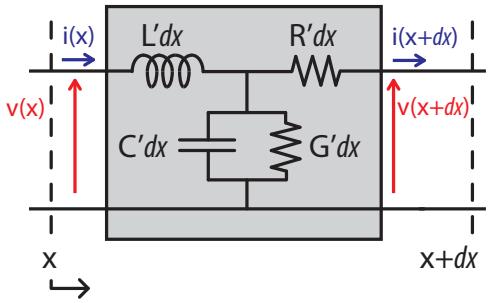


FIGURA 2.8: Modelo circuital diferencial de una línea de transmisión

Con dicho modelo se pueden derivar las ecuaciones del telegrafista.

$$\frac{\partial^2 v}{\partial x^2} = R'G'v + (R'C' + L'G')\frac{\partial v}{\partial t} + L'C'\frac{\partial^2 v}{\partial t^2} \tag{2.1}$$

$$\frac{\partial^2 i}{\partial x^2} = R'G'i + (R'C' + L'G')\frac{\partial i}{\partial t} + L'C'\frac{\partial^2 i}{\partial t^2} \tag{2.2}$$

En este caso no existe una solución general a estas ecuaciones a diferencia del caso ideal sin pérdidas ($R' = 0$, $G' = 0$) que es la conocida ecuación de d'Alembert. Pero si resolvemos dichas ecuaciones para una solución del tipo exponencial compleja tenemos las siguientes ecuaciones que describen la tensión y la corriente a lo largo de la línea según el modelo de la figura2.9

$$V(x) = V_2 \cosh(\gamma x) + I_2 Z_0 \sinh(\gamma x) \tag{2.3}$$

$$I(x) = I_2 \cosh(\gamma x) + \frac{V_2}{Z_0} \sinh(\gamma x) \tag{2.4}$$

donde γ es la constante de propagación y Z_0 es la impedancia característica de la línea, cuyas ecuaciones son.

$$\gamma = \sqrt{(R' + j\omega L')\,(G' + j\omega C')} = \alpha + j\beta \tag{2.5}$$

$$Z_0 = \sqrt{\frac{R' + j\omega L'}{G' + j\omega C'}} \tag{2.6}$$

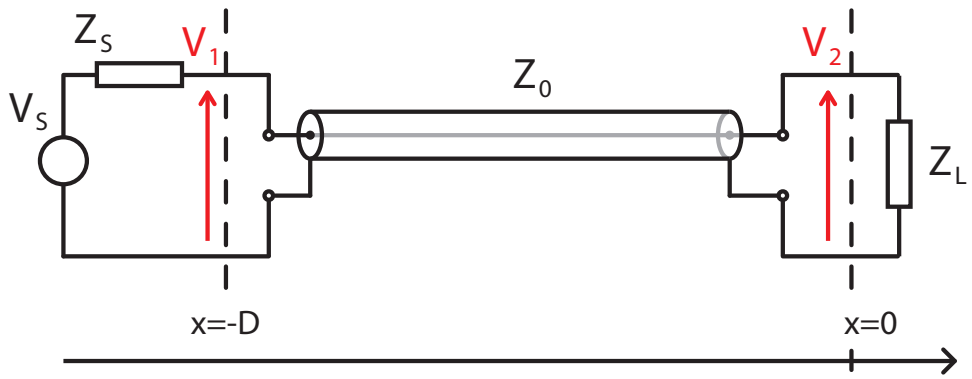


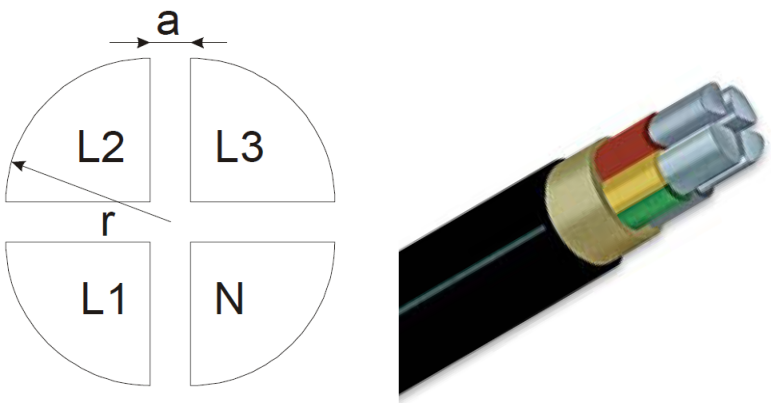
FIGURA 2.9: Línea de transmisión

Considerando una línea de transmisión adaptada, que es equivalente a sólo tener en cuenta el sentido de propagación de fuente a destino, la función de transferencia de una línea de longitud l puede ser expresada de la siguiente manera:

$$\begin{aligned} H(f) &= \frac{V(x=0)}{V(x=-l)} = \frac{\frac{1}{2}\left(V_2 - I_2 Z_0\right)}{\frac{1}{2}e^{\gamma l}\left(V_2 - I_2 Z_0\right)} \\ &= e^{-\gamma l} = e^{-\alpha(f) l} e^{-j\beta(f) l} \end{aligned} \tag{2.7}$$

donde explicitamos la dependencia de la parte real e imaginaria de la constante de propagación γ con la frecuencia $f = \frac{\omega}{2\pi}$.

La figura 2.10(a) nos muestra un corte de un típico cable de transmisión de energía con cuatro conductores. Cuando se alimenta la señal entre dos conductores adyacentes la mayoría del campo eléctrico está concentrada entre estos dos conductores. Para una primera aproximación se puede estimar el campo eléctrico y magnético por ecuaciones que describen una micro-strip. Los parámetros del cable puede ser estimados por sus dimensiones geométricas y las propiedades de los materiales.



(a) Sección transversal de un cable de 4 conductores. Se muestran las fases vivas L1, L2, L3 y el neutro N.

(b) Cable NAYY

FIGURA 2.10: Cable de distribución

La inductancia por unidad de longitud y capacidad por unidad de longitud son descriptas por las siguientes ecuaciones [34]

$$L' = \mu_0 \mu_r \frac{a}{r} \tag{2.8}$$

$$C' = \epsilon_0 \epsilon_r \frac{r}{a} \tag{2.9}$$

Considerando la utilización de frecuencias en el rango de los megahertz la resistencia por unidad de longitud es dominada por el efecto skin y puede ser aproximado por

$$R' = \sqrt{\frac{\pi \mu_0}{\kappa r^2} f} \longrightarrow R' \sim \sqrt{f} \tag{2.10}$$

en un cable circular con radio r, donde $\kappa = 4\pi^2\sigma$ y σ es la conductividad específica del conductor. La conductancia por unidad de longitud es

$$G' = 2\pi f C' \tan\delta \longrightarrow G' \sim f \tag{2.11}$$

y está principalmente influenciada por el factor de disipación (o tangente de pérdidas), $\tan\delta$, del material dieléctrico, que usualmente es PVC.

Usando las dimensiones geométricas y las propiedades de los materiales en las ecuaciones anteriormente descriptas resulta que $R' \ll \omega L'$ y $G' \ll \omega C'$ en el rango de frecuencias de interés. Por lo tanto, los cables pueden considerarse de bajas pérdidas y su impedancia característica Z_0 así como su constante de propagación γ pueden

ser simplificadas en las expresiones [2]

$$Z_0 = \sqrt{\frac{L'}{C'}} \in \mathbb{R} \tag{2.12}$$

$$\gamma = \underbrace{\frac{1}{2}\frac{R'}{Z_0} + \frac{1}{2}G'Z_0}_{\mathrm{Re}(\gamma)=\alpha} + j\underbrace{\omega\sqrt{L'C'}}_{\mathrm{Im}(\gamma)=\beta} \tag{2.13}$$

Resumiendo los parámetros característicos del cable en tres constantes (k_1, k_2, k_3) da como resultado

$$\gamma = k_1\sqrt{f} + k_2 f + jk_3 f \tag{2.14}$$

La parte real de la constante de propagación γ, el factor de atenuación α, aumenta con la frecuencia f. La relación entre α y f de un cable específico puede ser proporcional a $\sqrt{f}$ ó proporcional a f ó proporcional a una mezcla de ambos, donde k_1 o k_2 pueden dominar. Basado en derivaciones provenientes de suposiciones físicas y una extensiva investigación de mediciones de respuestas en frecuencia, según [48], las pérdidas del cable puede modelarse como:

$$\alpha(f) = a_0 + a_1 f^k \tag{2.15}$$

Es decir que si bien se utilizó la teoría de líneas de transmisión y las propiedades materiales y dimensiones de los cables para derivar la expresión matemática del modelo, en última instancia se optó por conservar el modelo pero ajustar los parámetros en base a resultados empíricos. Esto es debido a que las leyes generales deducidas son válidas pero la influencia real de todos los efectos es más compleja como para poder ser predicha simplemente por las características físicas del cable. Entonces, con una adecuada selección de parámetros a_0, a_1 y k la atenuación de un cable de transmisión de energía puede caracterizarse como

$$A(f, d) = e^{-\alpha(f)d} = e^{-(a_0 + a_1 f^k)d} \tag{2.16}$$

2.3.2. Propagación multi-camino

Debido a la estructura de la red de baja tensión, la propagación de la señal difiere en gran medida del caso de uso de líneas adaptadas. En el ámbito PLC existen numerosas reflexiones causadas por uniones entre cables de servicio a las casas, cajas de conexión y empalmes entre cables de distinta impedancia característica. La propagación de la señal no sólo se da en el camino directo entre el transmisor y el receptor sino que deben considerarse otros caminos (ecos). El resultado es una propagación multi-camino de la señal con desvanecimiento selectivo de frecuencias.

Estudiaremos la propagación multi-camino con un simple ejemplo que pueda ser analizado. En este caso en el camino tenemos una ramificación y la topología consta de tres segmentos (1) (2) (3) con longitudes l_1, l_2, l_3 cuyas impedancias características

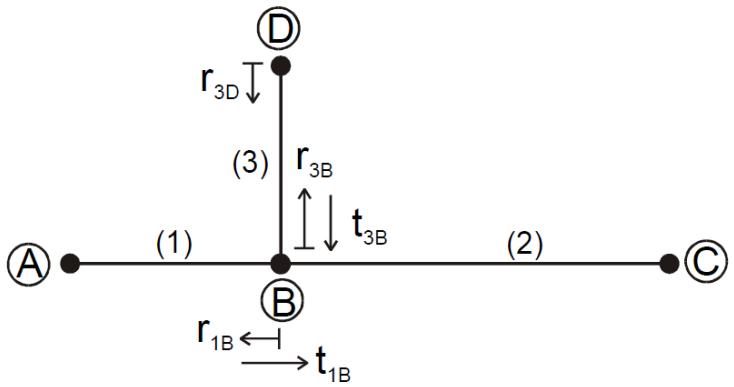


FIGURA 2.11: Propagación multi-camino con una ramificación. En el diagrama se indican los coeficientes de reflexión r_{1B}, r_{3B}, r_{3D} y los de transmisión t_{1B}, t_{3B}, además de los tres segmentos de cable (1), (2) y (3)

son Z_{0_1}, Z_{0_2} y Z_{0_3} respectivamente tal como muestra la figura 2.11. Para simplificar el análisis del caso supondremos que A y C tienen sus impedancias adaptadas, lo que significa $Z_A = Z_{0_1}$ y $Z_C = Z_{0_2}$. Los puntos restantes B y D tienen los siguientes coeficientes de reflexión

$$r_{1B} = \frac{Z_{0_2} \| Z_{0_3} - Z_{0_1}}{Z_{0_2} \| Z_{0_3} + Z_{0_1}} \tag{2.17}$$

$$r_{3D} = \frac{Z_D - Z_{0_3}}{Z_D + Z_{0_3}} \tag{2.18}$$

$$r_{3B} = \frac{Z_{0_2} \| Z_{0_1} - Z_{0_3}}{Z_{0_2} \| Z_{0_1} + Z_{0_3}} \tag{2.19}$$

y los siguientes coeficientes de transmisión

$$t_{1B} = 1 + r_{1B} \tag{2.20}$$

$$t_{3D} = 1 + r_{3D} \tag{2.21}$$

$$t_{3B} = 1 + r_{3B} \tag{2.22}$$

Con estas suposiciones los caminos de propagación posibles están listados en la tabla 2.1. Cada camino i tiene un peso g_i, que representa el producto de los coeficientes de reflexión y transmisión a través del camino. Vale la pena notar que el valor de g_i depende de la frecuencia si alguno de sus factores depende de la frecuencia. A su vez, el retardo del camino es τ_i dado por

$$\tau_i = \frac{d_i}{v_p} \tag{2.23}$$

[2] ver apéndice B.1 para la demostración de la ecuación (2.13)

donde d_i es la longitud recorrida y v_p es la velocidad de fase. Las pérdidas en cables reales causan una atenuación $A(f, d)$ creciente con la longitud y la frecuencia. Debemos suponer[3] que la velocidad de fase es igual en los tres medios de transmisión para que sea válido el cálculo de τ_i.

N° camino	Camino de la señal	peso g_i	Longitud del camino d_i
1	$A \to B \to C$	t_{1B}	$l_1 + l_2$
2	$A \to B \to D \to B \to C$	$t_{1B} r_{3D} t_{3B}$	$l_1 + 2l_3 + l_2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
N	$A \to B(\to D \to B)^{N-1} \to C$	$t_{1B} r_{3D} (r_{3B} r_{3D})^{N-2} t_{3B}$	$l_1 + 2(N-1)l_3 + l_2$

Tabla 2.1: Caminos de propagación

Las componentes de la señal de los caminos deben ser sumadas debido a la superposición y entonces la transferencia de A hacia C puede expresarse como:

$$H(f) = \sum_{i=1}^{N} g_i \cdot A(f, d_i) \cdot e^{-j2\pi f \tau_i} \tag{2.24}$$

Es importante recordar que la ecuación (2.24) fue derivada para A y C con impedancias adaptadas, de modo que entre A y C el canal resulta ser simétrico. Debemos entender por simétrico una transferencia en el sentido opuesto que es proporcional a la directa.

$$H_{A \to C}(f) \quad \propto \quad H_{C \to A}(f)$$

Sin embargo, no vale la pena detenerse en analizar los casos de simetría ya que para la gran mayoría de configuraciones la transferencia no va a ser simétrica.

En general, el módulo de los coeficientes de reflexión son menores o iguales a 1 y para el caso de los coeficientes de transmisión menores o iguales a 2.

$$\begin{aligned} &|r_{jX}| \leqslant 1, \ |t_{jX}| \leqslant 2 \\ &j = 1, 2, 3, \ldots \ X = A, B, C, \ldots \end{aligned} \tag{2.25}$$

Los pesos g_i son el producto de los anteriores y parecería que en principio no podemos afirmar nada acerca de si el módulo de estos tenderán a disminuir a medida que el camino sea más largo. Sin embargo, si tenemos en cuenta el principio de conservación de energía sabemos que es imposible que la potencia de la señal aumente a medida que se propaga por un medio con bajas pérdidas. En un análisis preliminar podríamos decir que:

- a medida que el camino es más largo, hay más rebotes y por ende hay más productos de coeficientes de reflexión menores o iguales a 1 que harán que ese g_i tienda a disminuir.

[3] Más adelante comprenderemos que esta suposición sólo es útil para llegar a la expresión matemática y que no implica limitaciones en el modelo.

- aunque haya coeficientes de transmisión mayores a 1, el coeficiente de transmisión en el otro sentido (cuando haya rebotes) será menor a 1

$$t_{0\to1} = 1 + r_{0\to1} = \frac{2Z_1}{Z_1 + Z_0}$$
$$\text{suponiendo } t_{0\to1} > 1 \Rightarrow Z_1 > Z_0$$
$$t_{1\to0} = 1 + r_{1\to0} = \frac{2Z_0}{Z_0 + Z_1} \Rightarrow t_{1\to0} < 1$$

  y además su producto será menor o igual a 1 (que es importante cuando se tiene en cuenta el doble transpaso de ida y vuelta de una desadaptación de impedancias).

$$t_{0\to1} \cdot t_{1\to0} = \frac{4Z_0Z_1}{(Z_1 + Z_0)^2}$$
$$\text{si } Z_0 = kZ_1 \text{ y } 0 < k \leqslant 1 \text{ entonces}$$
$$t_{0\to1} \cdot t_{1\to0} = \frac{4k}{(1+k)^2} \leqslant 1 \quad \text{si } 0 < k \leqslant 1$$

- una manera de acotar los coeficientes de transmisión se basa en que la impedancia en las uniones suele ser menor porque es el paralelo de otros cables y por esta razón el coeficiente de reflexión es negativo y el coeficiente de transmisión es menor que 1. Entonces g_i será producto de valores cuyo módulo es menor que 1 y por lo tanto tenderá a cero.

En el apéndice se pueden observar dos ejemplos donde existen coeficientes de transmisión mayores a 1 pero veremos que esto no tiene ninguna implicancia acerca de la tendencia decreciente de los g_i. Ver apéndice B.2 y B.3. Entonces concluimos con que a mayor distancia de camino, más chico tenderá a ser el valor de g_i sin importar que algunos coeficientes de transmisión sean mayores que 1. Es por ello que para el modelo alcanza con que tomemos los caminos de mayor importancia y obviemos aquellos cuyo g_i ya es demasiado chico como para aportar significativamente pero que considerarlos siempre aumenta la complejidad del cálculo.

El ejemplo analizado es uno de los más simples, sin embargo la metodología de particionamiento en caminos sigue siendo perfectamente válida para analizar la propagación en topologías más complicadas.

2.4. Modelo general multi-camino de la transferencia

En base a los fenómenos analizados Zimmermann y Dostert plantean [48, 50] un modelo teórico de la transferencia que comprenda los mismos. Este es un modelo *top-down*, es decir un modelo en el cual sus parámetros son determinados en base a mediciones y no en base cada uno de los componentes más elementales de la topología

(cables, uniones, dispositivos conectados). A su vez también podría clasificarse como un modelo fenomenológico que formula una estructura matemática que explica las mediciones y luego busca los valores que se adapten mejor a las mediciones. Aún así, en topologías simples los parámetros derivados en base a las mediciones tienen una correlación con la topología real de red, es decir que las distancias son similares a las dimensiones reales de los caminos. En general, en las topologías de redes que se presentan en la práctica es casi imposible hacer un seguimiento de las mediciones hacia sus causantes en busca de una explicación física. Peor aún, por ser justamente un modelo *top-down* puede no existir explicación alguna.

Anteriormente habíamos llegado a la expresión de la transferencia en función de los múltiples caminos (ecuación (2.24)). Sin embargo, ésta no explicitaba que las impedancias involucradas dependían de la frecuencia. A su vez, asumiremos el modelo de atenuación propuesto en 2.15. Entonces, teniendo en cuenta dichas consideraciones obtenemos:

$$H(f) = \sum_{i=1}^{N} \underbrace{|g_i(f)|e^{\varphi_{g_i}(f)}}_{\text{peso}} \cdot \underbrace{e^{-(a_0+a_1 f^k)d_i}}_{\text{atenuación}} \cdot \underbrace{e^{-j2\pi f\tau_i}}_{\text{retardo}} \quad f > 0 \tag{2.26}$$

Por lo tanto, el tiempo de recorrida de un camino está descripto por el término de retardo. La característica pasabajos es considerada en el factor de atenuación que depende de la longitud y de la frecuencia. Y por último, el factor de peso g_i comprende la reflexión y transmisión de la señal a lo largo de su camino de propagación. Vale la pena notar que g_i puede ser complejo y dependiente de la frecuencia. Finalmente, la señal en el receptor será aquella que esté conformada por las distintas contribuciones de los N caminos dominantes.

2.4.1. Modelo Simplificado

Afortunadamente, en la mayoría de los casos prácticos los factores de peso g_i pueden considerarse complejos pero no dependientes de la frecuencia. Recapitulando la ecuación (2.23) obtenemos relación entre el retardo τ_i, la longitud del camino d_i y la velocidad de fase v_p

$$\tau_i = \frac{di}{v_p} = \frac{d_i\sqrt{\epsilon_r}}{c_0} \tag{2.27}$$

donde c_0 es la velocidad de la luz en el vacío y ϵ_r la constante dieléctrica relativa del material aislante. Esto permite llegar a la siguiente función de transferencia

$$H(f) = \sum_{i=1}^{N} \underbrace{g_i}_{\text{peso}} \cdot \underbrace{e^{-(a_0+a_1 f^k)d_i}}_{\text{atenuación}} \cdot \underbrace{e^{-j2\pi f\frac{d_i}{v_p}}}_{\text{retardo}} \quad f > 0 \tag{2.28}$$

A modo de resumen explicaremos los parámetros de dicha transferencia

Parámetro	Descripción
i	número del camino. El camino con el menor retardo tendrá el índice $i = 1$
a_0,a_1	parámetros de atenuación, donde a_0 expresa la atenuación debida *sólo* a la distancia y a_1 la atenuación debida a la distancia y la frecuencia.
k	exponente del factor de atenuación (usualmente toma valores entre $0,5$ y 1.
g_i	peso del camino i. En general es un número complejo y puede ser interpretado físicamente como la combinación de los efectos de los coeficientes de reflexión y transmisión a lo largo del camino.
d_i	longitud del camino i.
τ_i	retardo del camino i.

Este modelo puede interpretarse como una extensión del modelo de Philipps [30] para redes in-house, donde se agregan los factores de característica pasabajos $e^{-(a_0+a_1 f^k)d_i}$ luego de los retardos, tal como muestra la figura 2.12. Estos factores se corresponden en tiempo con los impulsos individuales $h_i(t)$ descriptos por la ecuación (2.29) para el caso de $k = 1$.

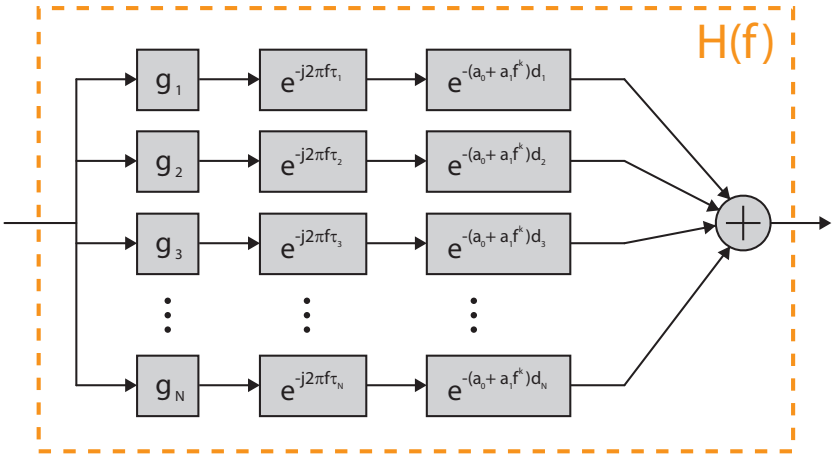


FIGURA 2.12: Modelo de la transferencia del canal de Zimmermann-Dostert

En síntesis, la ecuación (2.28) representa las bases de los modelos que describen la función de transferencia compleja de un típico canal de comunicaciones sobre líneas de tensión. Este modelo comprende todos los efectos substanciales que un canal puede tener en un rango de frecuencias entre 500 kHz y 40 MHz, y más aún, permite que el modelado sea haga con un conjunto reducido de parámetros. El parámetro N permite controlar fácilmente la precisión del modelo, a mayor N mayor precisión [50].

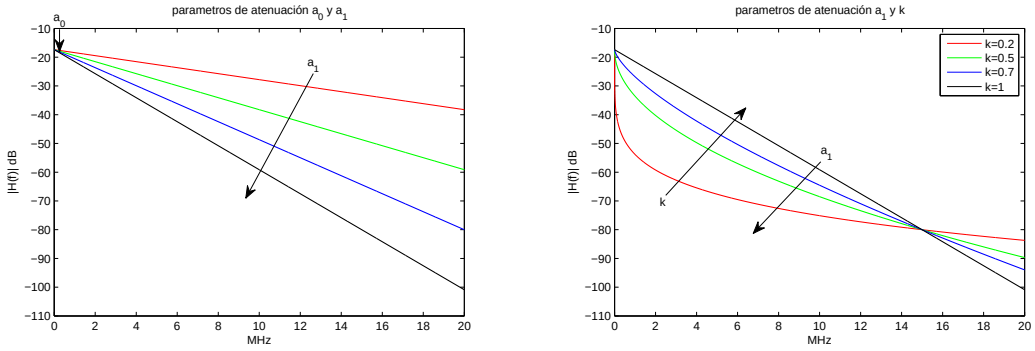
2.4.2. Explicación de los parámetros del modelo simplificado

A continuación haremos una explicación más detallada de los parámetros ya vistos.

2.4.2.1. Parámetros de atenuación

En la ecuación (2.15), dado un exponente k constante el parámetro a_1 influencia la dependencia en frecuencia de la atenuación. A medida que a_1 aumenta, la característica pasabajos sea hace más marcada. Por este motivo, las pérdidas por filtrado pasabajos pueden ser modeladas variando a_1. Por otro lado, el parámetro a_0 representa la atenuación *independiente* de la frecuencia que aumenta con la longitud. De esta manera se tiene en cuenta que los caminos más largos debidos a reflexiones sufren una mayor atenuación. Es este hecho el que también ayuda a limitar la cantidad de caminos dominantes en el modelo. A su vez observamos que cuanto más grande sea a_0 más distintivo es el offset en la curva de atenuación.

De acuerdo a la figura 2.13(b) el comportamiento en frecuencia puede ser ajustado con el exponente k. En escala logarítmica con $k = 1$, cuando las pérdidas dieléctricas dominan, la atenuación aumenta proporcionalmente en frecuencia, es decir $\alpha(f) \sim f$ para $k = 1$. Para valores de $k < 1$, la atenuación aumenta en menor medida con la frecuencia. Por ejemplo, cuando las pérdidas óhmicas por efecto *skin* predominan, tenemos un aumento de la atenuación con la raíz cuadrada de la frecuencia, entonces $k = 0,5$, es decir $\alpha(f) \sim \sqrt{f}$.



(a) variación de a_1 (b) variación de k

FIGURA 2.13: Influencia de los parámetros de atenuación sobre $|H(f)|$

Si utilizamos valores absolutos para ajustar la curva de atenuación, debemos tener en cuenta la correlación entre a_1 y k. Para valores crecientes de k, los valores de a_1 deben disminuir para obtener una atenuación del mismo orden. En el dominio del tiempo, un aumento en a_1 influencia el ancho del impulso en el camino principal junto con una menor amplitud máxima. Esto se puede observar en la figura 2.14.

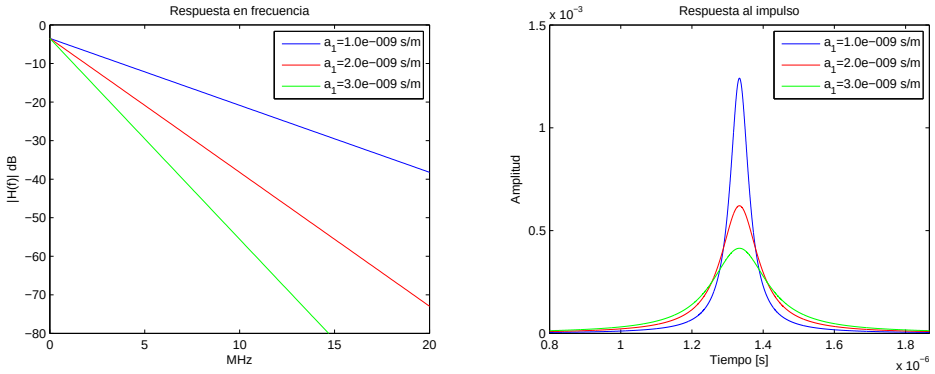
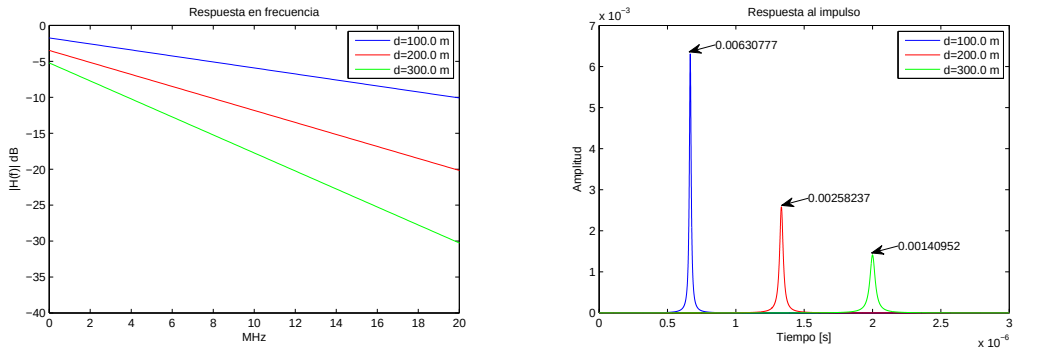


FIGURA 2.14: Influencia de a_1 sobre la respuesta al impulso

2.4.2.2. Parámetros de camino

Una vez que tenemos el perfil de atenuación del canal, debemos modelar los desvanecimientos selectivos de frecuencia usando parámetros de camino. Mientras que los parámetros de atenuación pueden obtenerse por medio de mediciones características de atenuación, para determinar los parámetros de camino necesitamos conocer la respuesta impulsiva del canal. El primer parámetro d_i indica la longitud del camino, el cual está directamente relacionado con el retardo de la señal en su propagación. Cuanto más grande elijamos dicho parámetro, mayor será el retardo del camino. Esto se ve en la figura 2.15(b). Duplicar la longitud no sólo duplica el retardo de propagación sino que disminuye la amplitud del impulso a menos de la mitad. La influencia de la longitud del camino en la pendiente es similar a la de a_1 (ver figura 2.15(a)). Sin embargo, a_1 sólo influenciaba la pendiente mientras que d_i también influye en el piso de la atenuación.



(a) Amplitud de $|H(f)|$ en función de d_i (b) respuesta al impulso en función de d_i

FIGURA 2.15: Influencia del parámetro d_i

La respuesta al impulso para $k = 1$ puede expresarse en forma cerrada haciendo la transformada inversa de Fourier de (2.28) para el camino i-ésimo. Para el camino i

tenemos la respuesta al impulso en tiempo continuo

$$h_i(t) = \Re(g_i)\frac{e^{-a_0 d_i}}{2\pi^2}\frac{a_1 d_i}{\left(\frac{a_1 d_i}{2\pi}\right)^2 + \left(t - \frac{d_i}{v_p}\right)^2} - \Im(g_i)\frac{e^{-a_0 d_i}}{\pi}\frac{t - \frac{d_i}{v_p}}{\left(\frac{a_1 d_i}{2\pi}\right)^2 + \left(t - \frac{d_i}{v_p}\right)^2} \tag{2.29}$$

La respuesta al impulso $h(t)$ es la superposición de todos los caminos y por lo tanto la suma de todos los impulsos $h_i(t)$.

El segundo parámetro de camino g_i tiene implicancias dependiendo si es puramente real o es un número complejo. Si es un número real afecta la amplitud del impulso de manera proporcional dejando las otras características intactas. Si es número complejo alterará la forma del pulso en función de la fase que tenga. La ecuación (2.29) define la forma exacta del impulso pero en ella no es fácil ver la influencia de la fase de g_i. Es más claro de apreciar esta relación en una visualización gráfica en la figura 2.16.

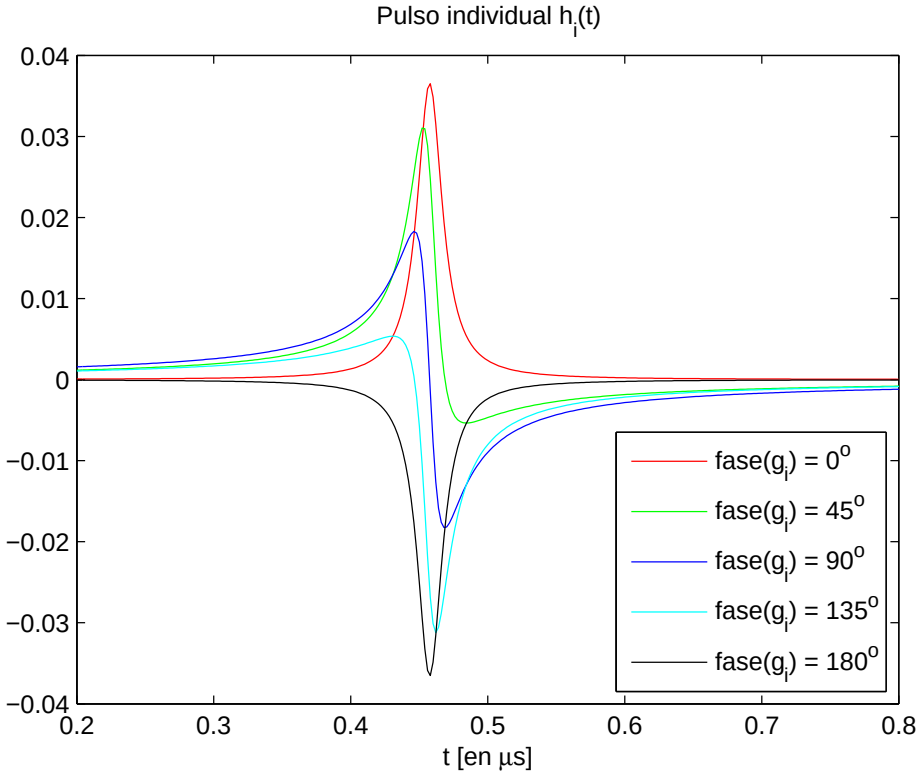


FIGURA 2.16: Influencia de $g_i \in \mathbb{C}$ en función de su fase

2.4.3. Consideraciones acerca de la estimación de parámetros

La estimación de parámetros requiere una estrategia sofisticada. La estrategia utilizada en [50] consta de tres pasos. En el primer paso, los coeficientes de atenuación

se obtienen del perfil de atenuación usando un modelo de un solo camino con una estimación de mínimos cuadrados. En el segundo paso, se determina la cantidad, posición y amplitud de los caminos significativos por simple inspección de los picos de la respuesta impulsiva. En casos de baja complejidad estos pasos ya logran suficiente precisión. Sin embargo, si se requiere una mayor cantidad de caminos, los parámetros deben ser optimizados con una estrategia de evolución donde la respuesta impulsiva o la amplitud y fase son utilizadas como métricas de calidad. Esta complejidad en el método de estimación se debe a que a mayor cantidad de caminos la influencia de las colas de cada impulso descripto por la ecuación (2.29) no puede despreciarse, y un cambio en el valor de un camino influencia a otros y obliga el recálculo de los mismos.

En la figura 2.17 se observa la medición de un enlace de larga distancia (> 300m) en un área residencial urbana con doce ramificaciones alimentando principalmente pequeños grupos de departamentos. Además del módulo de la transferencia también se gráfica el piso de ruido. A su vez se gráfica el modelo del perfil de atenuación que se calculó con un $N = 1$. Para estimar los parámetros de atenuación se debió preprocesar los datos medidos de manera que sólo los rangos de frecuencia no afectados por los *notches* sean considerados para la estimación. Además se observa que las frecuencias superiores a 7 MHz se encuentran por debajo del piso de ruido, una atenuación fuertemente marcada que es característica de los enlaces de larga distancia.

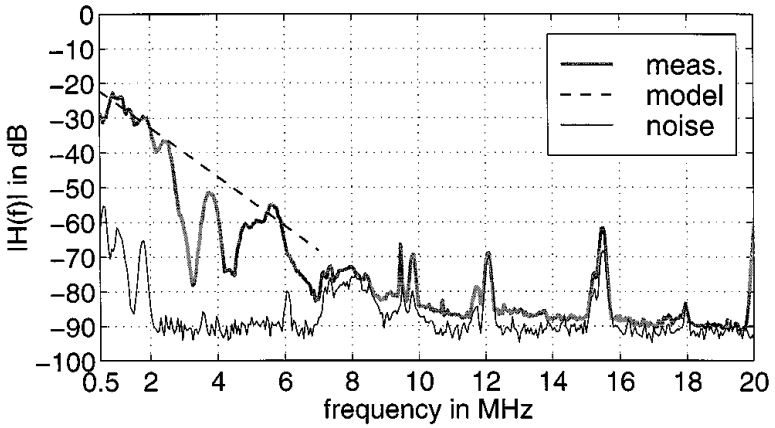


FIGURA 2.17: Estimación del perfil de atenuación

Por otro lado, en la figura 2.18 se muestra el modelado de una respuesta al impulso medida [1]. En la subfigura izquierda se observa la respuesta en frecuencia medida y la modelada. Sólo seis caminos fueron considerados por lo que no se espera una correspondencia exacta entre la medición y el modelo. Sin embargo, las propiedades típicas están bien reproducidas. Luego, en la subfigura del centro observamos la medición de la respuesta al impulso en comparación con la modelada. La limitación de seis caminos impuesta nos lleva a no considerar aquellos ecos que ocurren más allá de 2,5 μs. Finalmente, en la subfigura derecha podemos ver la descomposición

de la respuesta al impulso modelada en los seis impulsos individuales. En general, no es fácil ver cuando múltiples impulsos sucesivos se superponen con signos opuestos por simple observación de la respuesta al impulso. Vale la pena notar que las escalas del gráfico central y del de la derecha son distintas.

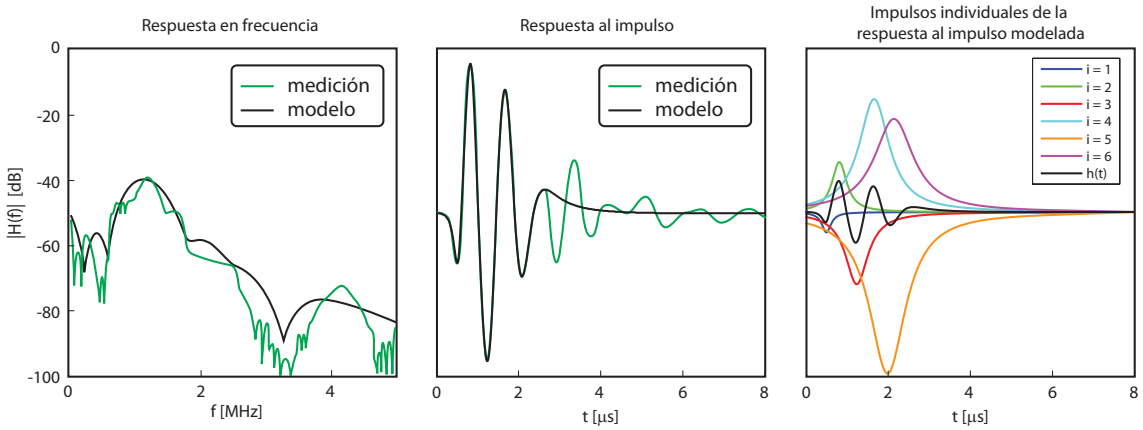


FIGURA 2.18: Modelado de una medición

Se pueden observar varios hechos inesperados en los valores de los parámetros de camino estimados

- Las amplitudes de los impulsos individuales, que se corresponden con los pesos g_i son considerablemente más grandes que el mínimo y máximo de la respuesta al impulso.
- Las amplitudes de los impulsos individuales son crecientes con la longitud creciente del camino aún cuando la envolvente de la respuesta al impulso está decreciendo. Por ejemplo, en nuestro caso particular vemos que el quinto impulso tiene el factor de peso más grande aunque en la respuesta al impulso el pico esté en el segundo.
- Los impulsos individuales que obtuvimos tienen un ancho mayor y una amplitud mayor. Eso implica que cada impulso que agregue a mi modelo influencia a los precedentes. Por esta razón, los parámetros anteriormente estimados ($N = 6$) deben reajustarse (para $N > 6$).

Este ejemplo[4] da una idea aproximada de las dificultades que se encuentran al estimar dichos parámetros y de que estamos frente a un modelo *top-down* donde los parámetros pueden no tener ningún correlato físico. A fin de cuentas, la estimación de los parámetros resulta ser un problema de optimización donde no hay un algoritmo general para resolver estas dificultades en su completitud.

[4] en particular este caso es el modelo de referencia 8 del canal de acceso (§A.1.3.3)

Sin embargo, este alejamiento de la realidad de los coeficientes g_i se puede justificar debido a la no utilización de números complejos. Es fácil ver que las oscilaciones generadas podrían corresponden a impedancias complejas (elementos que acumulan energía) que dan lugar a g_i complejos. De esta manera, se podría obtener un correlato físico más acorde. Esta restricción en la estimación parámetrica que se autoimpuso OPERA[5] obliga al modelo a apartarse de una explicación desde la teoría del electromagnetismo y a convertirse en un modelo más fenomenológico. Una posible explicación de este hecho podría ser que decidieron condicionar su estimación de parámetros adelantándose a una posible arquitectura de emulación y otra explicación sería la de la complejidad de la estimación de parámetros complejos que disuelve la estrategia expuesta al inicio de esta subsección.

2.4.4. Verificación experimental del modelo

En [50] verifican los resultados basados en simulaciones del modelo de (2.28) al compararlos con mediciones experimentales. Esta experiencia fue realizada con una red de topología simple y dimensiones conocidas. La figura 2.19 muestra la topología de la red. El transmisor fue ubicado en la posición A y el receptor en la posición C. Las impedancias en A y C estaban adaptadas a la impedancia característica del cable. En D se dejó el circuito abierto lo que resulta en un coeficiente de reflexión igual a 1. La sección 1 tiene una longitud de 30 m, la sección 2 de 170 m y la sección 3 12 m de largo. Mientras que las secciones 1 y 2 consisten de un cable de distribución NAYY150 con una impedancia característica de alrededor de 45 Ω, la sección 3 es una cable de servicio a las casas de tipo NAYY35 con una impedancia característica de aproximadamente 70 Ω.

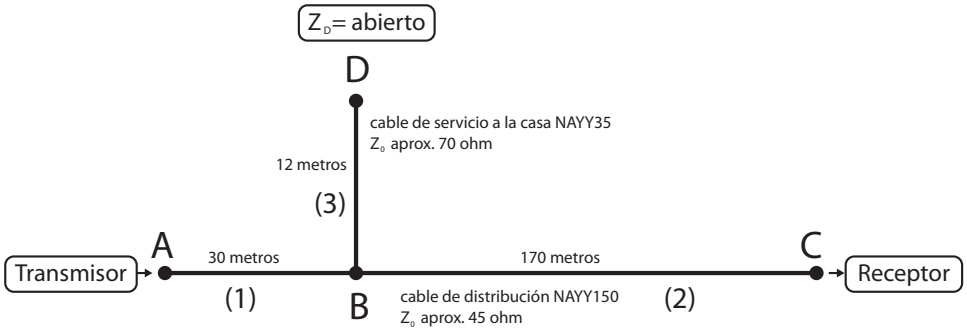


FIGURA 2.19: Red conocida para la verificación del modelo

La señal transmitida y recibida fueron almacenadas digitalmente utilizando un DSO (Digital Storage Oscilloscope). La transferencia compleja y el retardo de grupo fueron calculados off-line. Debido a que la propagación de la señal da lugar a grandes valores de fase, por esta razón para tener un mejor conocimiento de la fase se ha generado un gráfico de detalle de fase en donde se resta la parte lineal de la misma.

[5] Ver que en el apéndice §A todos los g_i son reales

Las reflexiones en el extremo abierto generan *notches* en la respuesta en frecuencia que pueden observarse fácilmente en la figura 2.20. En las mismas secciones del espectro se observan distorsiones en la fase y cambios en el retardo de grupo. Además, a causa de que la adaptación en las ubicaciones A y C no es ideal se genera un pequeño *ripple* en la respuesta en frecuencia que es visible en el gráfico.

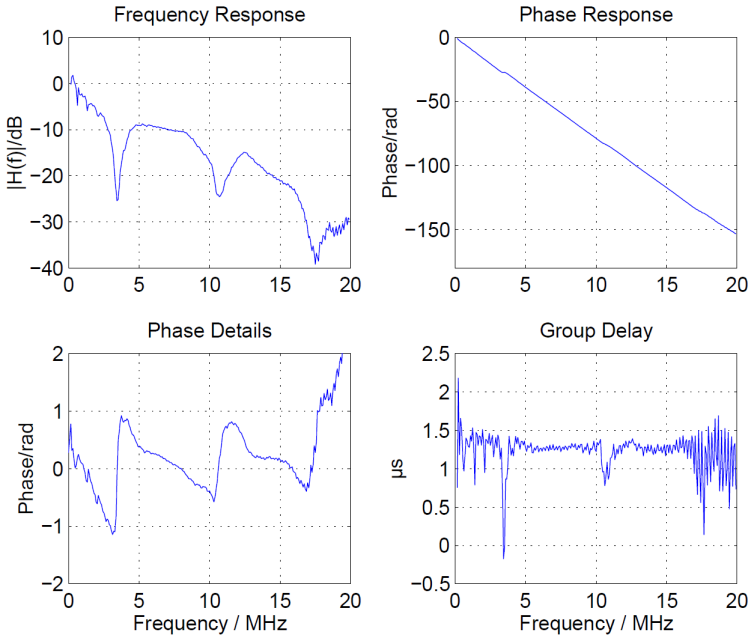


FIGURA 2.20: Medición de la red conocida

En la figura 2.21 se observa el resultado de la simulación de la transferencia con un modelo basado en la ecuación 2.28 con 6 caminos, es decir $N = 6$. Los parámetros se listan en la tabla 2.2 y fueron derivados de las mediciones de la respuesta en frecuencia. Es evidente que la simulación y las mediciones del módulo y fase difieren en algunos detalles. Sin embargo, lo importante es que el modelo comprende todos los efectos esenciales.

# Camino	**1**	**2**	**3**	**4**	**5**	**6**
d_i [m]	200	221	242	259	266	530
g_i	0,54	0,275	-0,15	0,08	-0,03	-0,02
$k = 1$	$a_0 = -2,1 \cdot 10^{-3}$		$a_1 = 8,1 \cdot 10^{-10}$			

Tabla 2.2: Parámetros de simulación

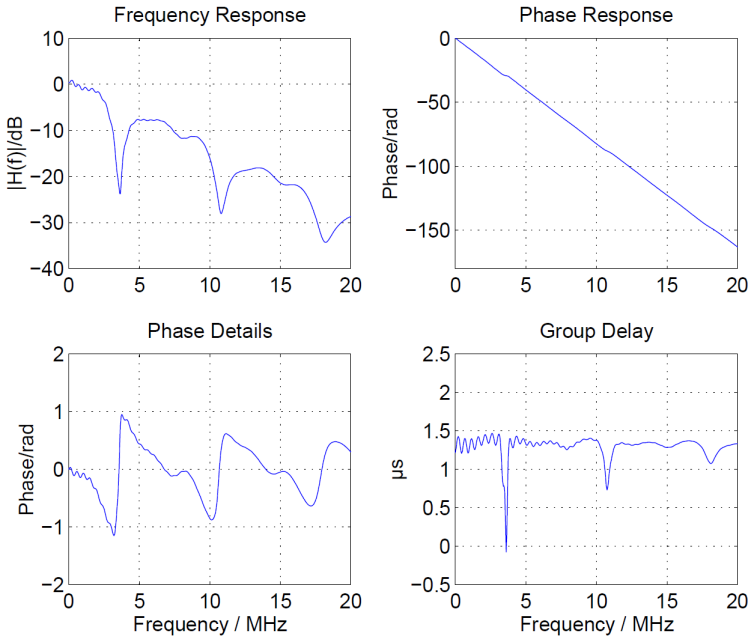


FIGURA 2.21: Simulación de la red con $N = 6$

Además de este validación experimental en condiciones de laboratorio, muchas mediciones realizadas para el consorcio OPERA [2] y otras anteriores también probaron adecuarse correctamente al uso de este modelo.

2.4.5. Clasificación de canales de acceso

Una clasificación posible para caracterizar a los canales de acceso se puede realizar en una jerarquías de dos niveles. Un primer nivel que es en función de la distancia entre el lugar de la inyección de la señal y los hogares receptores de dicha señal. Este nivel fue pensado desde el punto de vista del perfil de atenuación que depende mayormente de la distancia y el tipo de cables. El segundo nivel se refiere a la calidad del canal dentro de dicho rango de distancias y tendrá correlación con el grado de ramificación de la topología. Una topología muy ramificada tendrá como consecuencia muchos *notches* en frecuencia debido a las reflexiones de señal y será de baja calidad.

Desde el punto de vista de la distancia se proponen tres clases de longitudes

- distancias cortas, alrededor de 150m
- distancias medias, alrededor de 250m
- distancias largas, alrededor de 350m

y luego se distinguirá una calidad buena, media y mala. Los modelos de referencia para cada categoría definidos por OPERA [1] los podemos ver en el apéndice §A.

2.4.6. Modelo para redes in-house

El modelo de redes in-house ampliamente adoptado es el modelo de ecos de Philipps [30]. Otros modelos fueron planteados desde el enfoque de simulación física de la realidad, es decir conociendo bien la topología de la red y las cargas conectadas en cada enchufe y el comportamiento de dicha carga, como por ejemplo el modelo de Cañete et al. [9, 10, 36]. El modelo expuesto por Cañete plantea la variación de corto tiempo del canal debido a impedancias no lineales que termina conformando un canal lineal pero periódicamente variable en el tiempo. Sin embargo, él mismo y sus colegas desestiman este efecto por ser demasiado complejo y no tener aportes significativos pero que podrían ser considerados para la construcción de un sistema eficiente de transmisión. Por ello nos quedaremos con el modelo de Philipps.

Philipps desarrolló el modelo de ecos basándose en mediciones intensivas en ambientes in-house realizadas por él. En su estudio caracterizó a las redes hogareñas por contener reflexiones debido a las discontinuidades de impedancias y por este motivo realizó un planteo de ecos. Como mencionamos en §2.2.1, la característica pasabajos no se manifiesta debido a que las distancias en juego son muy cortas.

Por lo tanto, este modelo podría considerarse un caso especial del modelo de Zimmermann-Dostert donde no se considera el factor de atenuación si pasamos por alto la cronología de aparición de los modelos. Esta simplificación requiere menores recursos computacionales a la hora de la emulación. Sin embargo, es esencial modelar una mayor cantidad de ecos debido a la alta ramificación que existe en las redes internas. La figura 2.22 ilustra el modelo de ecos que queda descripto por la ecuación (2.30).

$$H(f) = \sum_{i=1}^{N} \underbrace{g_i}_{\text{peso}} \cdot \underbrace{e^{-j2\pi f \frac{d_i}{v_p}}}_{\text{retardo}} \quad f > 0 \tag{2.30}$$

La transferencia de las redes in-house suele tener un perfil de atenuación plano junto con la aparición de desvanecimientos selectivos de frecuencias causados por el alto grado de ramificación. Por ello, esta situación permite utilizar frecuencias superiores a los 30 MHz. En la figura 2.23 [30] vemos la comparación entre la medición de un canal y su modelo, que en este caso utiliza 5 caminos. Se puede observar como el replica los valles de amplitud y las características de fase.

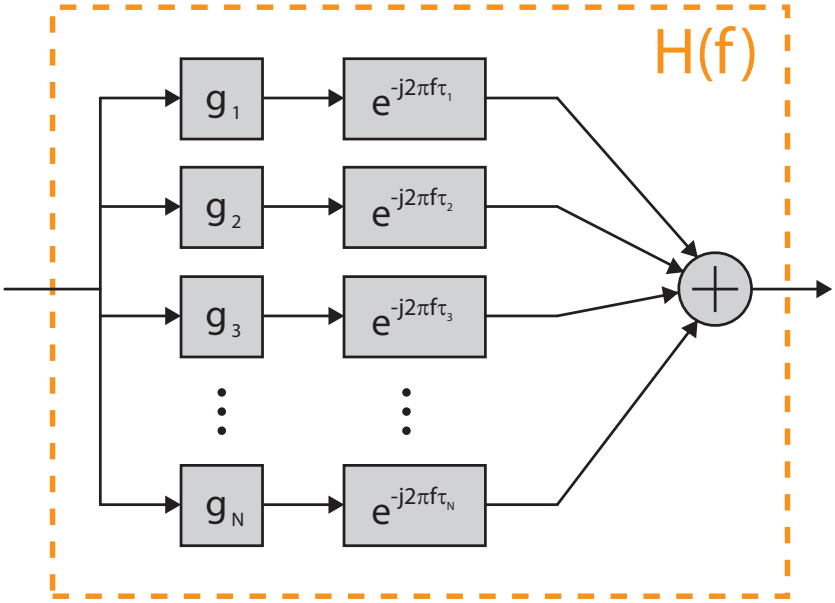


FIGURA 2.22: Modelo de la transferencia del canal para redes in-house de Philipps

Para el modelado se extraen los ecos más relevantes de la respuesta al impulso medida. Este método de extracción de parámetros es considerablemente más simple que el caso con característica pasabajos que es un problema de optimización no trivial.

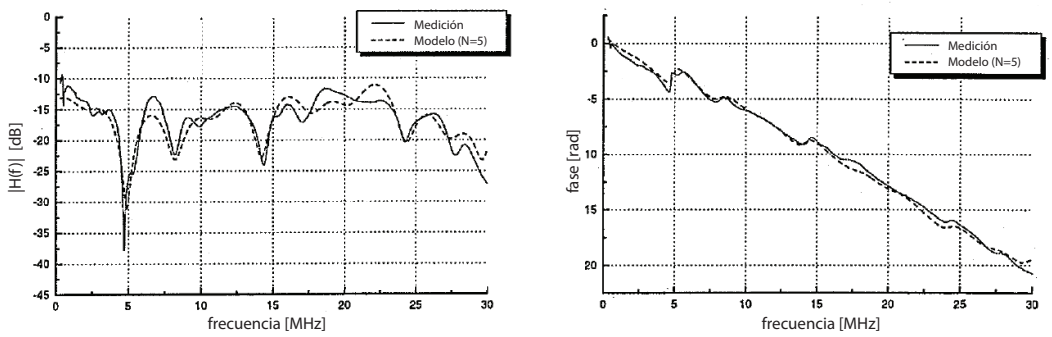


FIGURA 2.23: Medición de módulo y fase de un canal in-house

2.4.6.1. Caracterización de canales in-house

Existen diferentes enfoques a la hora de clasificar canales, por ejemplo Philipps [31] utiliza una clasificación que distingue el tipo de propiedad: comercial o residencial. Nosotros utilizaremos un criterio basado en la cantidad de caminos que tiene la respuesta al impulso. Las cuatro categorías se muestran en la tabla 2.3 y fueron obtenidas a través del análisis de una base de mediciones de canal [1]. Cuanto mayor

es el número de modelo, mayor es la cantidad de caminos y mayor el retardo inicial de la respuesta al impulso. A su vez, cuanto mayor es la cantidad considerada de caminos, la característica de atenuación se vuelve más dentada y adicionalmente se observa un mayor piso de atenuación cuyo efecto está reflejado en una menor amplitud de la respuesta al impulso. Estos valores tabulados son caracterizaciones de la respuesta al impulso, y se necesita más información para poder construir una respuesta al impulso. El procedimiento basa su validez en que en una red interna la topología, que describe longitudes, posicionamiento y ramificaciones de cables, puede suponerse aleatoria.

Modelo	Cantidad de caminos	Duración de la respuesta impulso	Retardo inicial	Amplitud Máxima	Amplitud Mínima
1	5	0,5 μs	0,2 μs	0,05	0,005
2	10	1,0 μs	0,2 μs	0,01	0,002
3	15	1,5 μs	0,5 μs	0,003	0,0003
4	20	2,0 μs	0,5 μs	0,0005	0,0001

Tabla 2.3: Categorías y características para la respuesta al impulso

- La duración de la respuesta al impulso es la diferencia de tiempo entre el primer y el último impulso individual
- El número de caminos da la cantidad de impulsos individuales que constituyen la respuesta al impulso.
- El retardo inicial antes del primer impulso representa el retardo de propagación de la señal del camino más corto.
- La amplitud máxima y mínima establecen límites entre los cuales deben estar las amplitudes de los impulsos individuales.

2.4.6.2. Procedimiento para la creación de canales en redes internas

Para cada clase de canal in-house, la cantidad de caminos a considerar, la duración de la respuesta al impulso así como el rango de amplitudes de las deltas de Dirac fueron determinados en la tabla 2.3. Sin embargo, la posición en el tiempo de las deltas de Dirac, su amplitud y signo no están definidos. Una vez determinados estos valores, se puede conocer la función de transferencia y por ende las características de atenuación a través de la transformada de Fourier de la respuesta al impulso. Por lo tanto, en referencia a la naturaleza aleatoria de las redes internas, procederemos a definir un método estadístico que puede generar adecuadamente canales de referencia. Adicionalmente, las propiedades del canal cambian continuamente cuando se encienden o apagan dispositivos eléctricos, a diferencia del caso de las redes de acceso.

El procedimiento para la generación de la respuesta al impulso de un canal es el siguiente:

1. La respuesta al impulso comienza con un impulso positivo de máxima amplitud.

2. Los impulsos se posicionan a intervalos regulares hasta llegar a la duración de la respuesta al impulso.

$$t_i = \frac{T_h \cdot i}{N_p - 1} \qquad i = 0, \ldots, N_p - 1 \tag{2.31}$$

   donde t_i la posición temporal del impulso i, T_h es la duración de la respuesta al impulso, y N_p es la cantidad de caminos.

3. Las amplitudes decaen exponencialmente hasta que el último impulso alcance la mínima amplitud según la envolvente $f(t)$.

$$f(t) = b_0 \left(\frac{b_1}{b_0} \right)^{\frac{t}{T_h}} \tag{2.32}$$

   donde b_0 es la amplitud máxima y b_1 la amplitud mínima.

4. Desde el segundo impulso en adelante elegimos las polaridades de los mismos al azar.

5. Las posiciones de los impulsos son elegidas al azar con respecto al tamaño del intervalo regular de espaciamiento. Dichos valores aleatorios se eligen del intervalo

$$-\frac{1}{2}\frac{T_h}{N_p - 1}k_s < \Delta t < \frac{1}{2}\frac{T_h}{N_p - 1}k_s \tag{2.33}$$

   El factor de variación k_s es un valor entre 0 y 1.

6. Finalmente, se agrega el retardo inicial de propagación.

Este procedimiento se observa gráficamente en la figura 2.24.

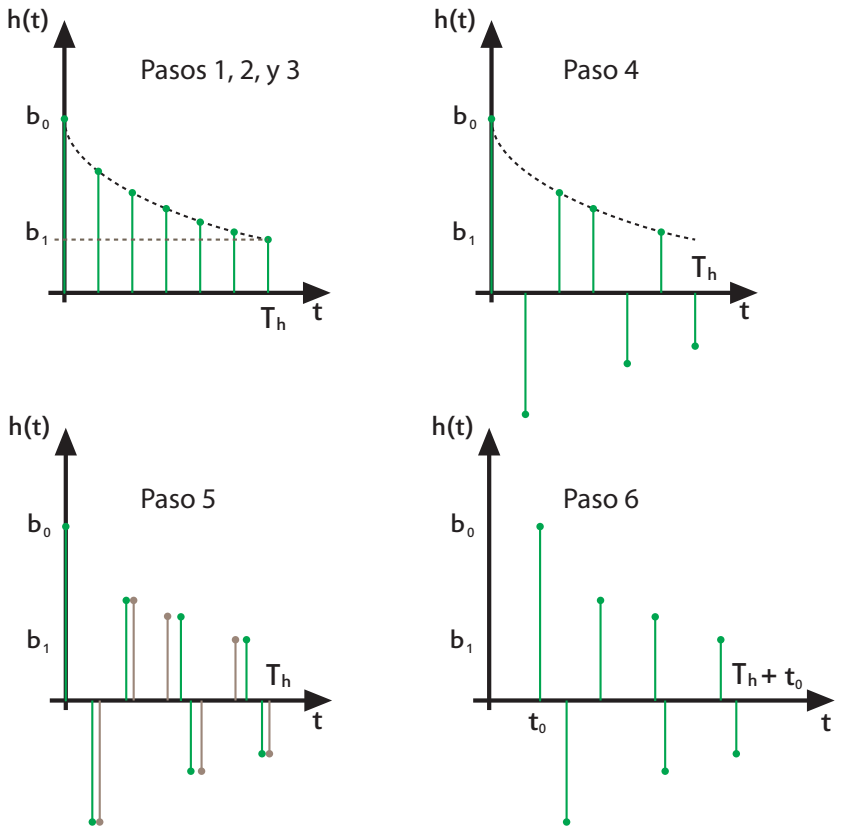
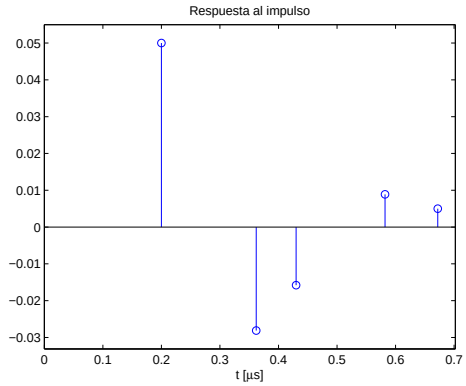
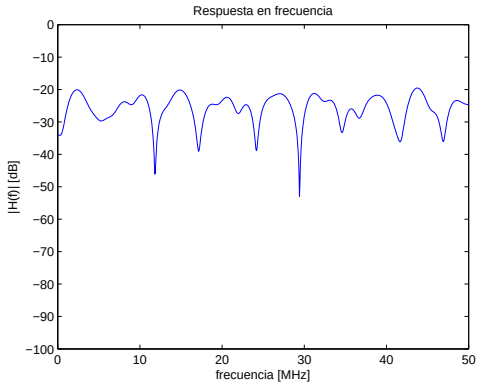


FIGURA 2.24: Procedimiento para la creación de los canales de referencia de redes internas

En las figuras 2.25, 2.26, 2.27 y 2.28 se observan los canales generados para cada modelo (1 a 4) de acuerdo a la tabla 2.3. Se utilizó un factor $k_s = 0{,}9$ para el reposicionamiento aleatorio de los impulsos. En las respuestas al impulso de mayor duración es fácil ver la separación no uniforme y la envolvente exponencial. Con respecto a la característica de atenuación, se evidencia que la irregularidad y la cantidad de *notches* aumentan a medida que la cantidad de caminos se incrementa. Los *notches* observados poseen hasta 40 dB de diferencia respecto del piso de atenuación. El piso de atenuación aumenta a medida que el número de modelo crece. Estos cuatro canales podrían servir como base para la implementación en un emulador de canal pero sin embargo utilizaremos como canales predeterminados los definidos por OPERA para poder hacer pruebas estandarizadas [1] [3] (ver apéndice §A.2).

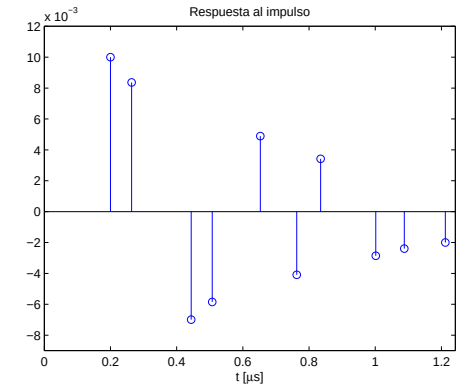


(a) Respuesta al impulso

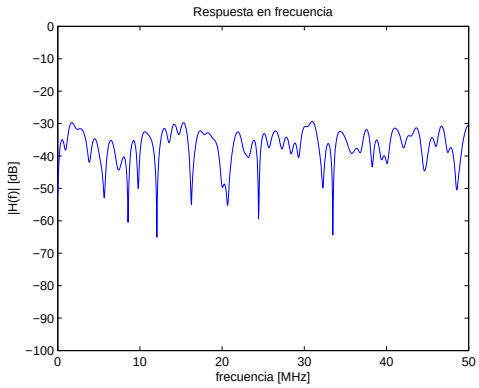


(b) Módulo de la transferencia

FIGURA 2.25: Modelo 1: canal in-house de 5 caminos

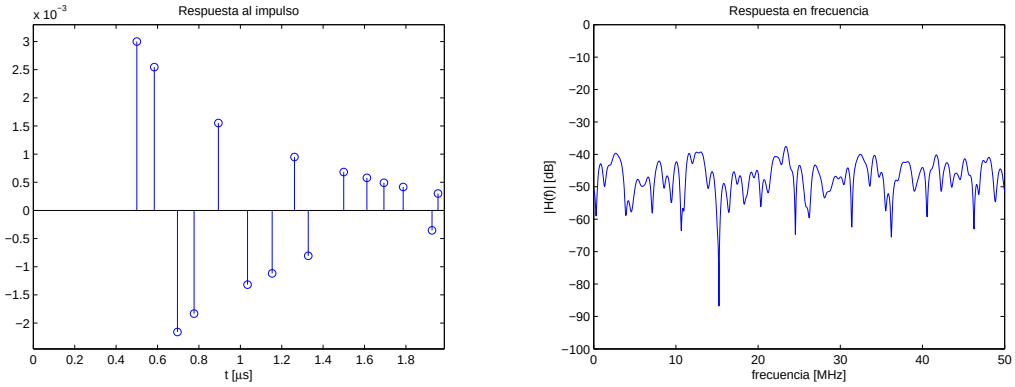


(a) Respuesta al impulso



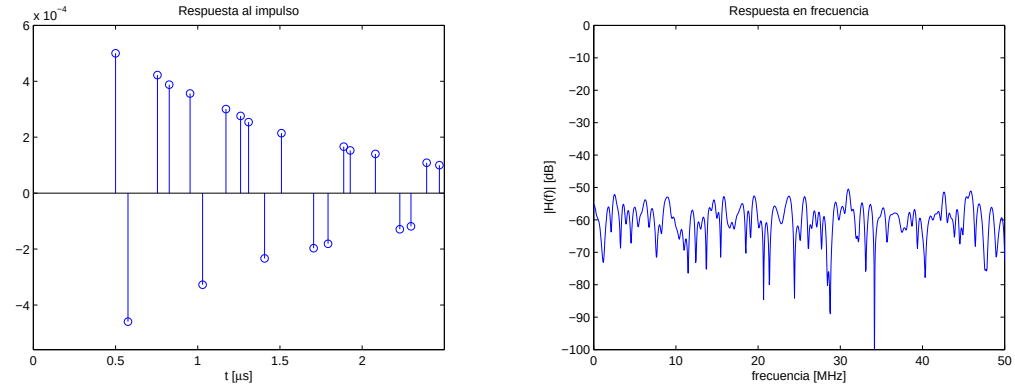
(b) Módulo de la transferencia

FIGURA 2.26: Modelo 2: canal in-house de 10 caminos



(a) Respuesta al impulso (b) Módulo de la transferencia

FIGURA 2.27: Modelo 3: canal in-house de 15 caminos



(a) Respuesta al impulso (b) Módulo de la transferencia

FIGURA 2.28: Modelo 4: canal in-house de 20 caminos

2.4.7. Resumen

En resumen podemos afirmar lo siguiente acerca del modelo presentado

- es un modelo fenomenológico de tipo *top-down*.
- tiene una reducida cantidad de parámetros lo que facilita su simulación.
- permite aumentar su precisión aumentando la cantidad de caminos N.
- no es trivial la estimación de sus parámetros y deben utilizarse técnicas de evolución. Si agrego un nuevo camino $N + 1$ debo recalcular los N anteriores.
- si bien sus parámetros tuvieron un correlato con la realidad en la deducción del modelo, no deben asignárseles ninguna consideración de este tipo a los valores

estimados si no se utilizan coeficientes g_i complejos. Y aún en este último caso tampoco podríamos asegurarnos un correlato físico ya que el modelo es una simple representación de una realidad más compleja que cumple fines ulteriores.

2.5. Resumen

En este capítulo se describió brevemente la topología de las redes eléctricas que constituye el ambiente en el cual se dan las comunicaciones de datos sobre líneas de tensión. Se abordaron desde un punto de vista físico las efectos que sufre la señal durante la propagación. El canal presenta una característica multi-camino y para las redes de accesso también presenta un comportamiento pasabajos. Se expuso un modelo matemático para este canal que sirve para describir tanto canales de la red de acceso como de las redes in-house. El modelo consta de cualidades deseables para una implementación como se verá en el capítulo 4.

3

Escenario de ruido

Una de las características más distintivas del canal BPL es el escenario de ruidos al cual se encuentra sometido. Como mencionamos anteriormente, la principal función de la red eléctrica es el óptimo transporte de energía en bajas frecuencias para la alimentación de máquinas o dispositivos eléctricos. Estos son los principales contribuyentes en la generación de ruidos impulsivos. Además, debido a los factores característicos de las instalaciones de las redes eléctricas, topología y tipos de cable utilizados, también predomina la interferencia de banda angosta. Por estos motivos, mientras que el canal AWGN es el modelo habitual utilizado en telecomunicaciones, en el ámbito PLC éste no es representativo de la realidad [49]. En general podremos decir que el rango de frecuencias que va desde algunos cientos de kiloHertz hasta 30 MHz está dominado por ruidos de banda angosta y por ruidos impulsivos.

3.1. Clasificación de ruidos

Los ruidos observados en comunicaciones de banda ancha sobre líneas de tensión son aditivos y pueden separarse en cinco clases:

1. **Ruido de fondo Coloreado:** tiene una densidad espectral de potencia relativamente pequeña, variando con la frecuencia. Este tipo de ruido es principalmente causado por la suma de numerosas fuentes de ruido de baja potencia. Su densidad espectral de potencia varia en el tiempo en el orden de minutos u horas.

2. **Ruido de banda Angosta:** compuesto por señales sinusoidales con amplitud modulada. Este tipo de ruido está principalmente causado por la interferencia de las emisoras de radiodifusión de onda media y onda corta. El nivel de potencia recibido suele variar con el transcurrir del día. Esto se debe a que la

propagación de las ondas de radio dependen del estadío de la atmósfera, en particular de la ionósfera, el cual varía durante el día.

3. **Ruido Impulsivo periódico, síncrono con la frecuencia de red:** estos impulsos tienen una tasa de repetición de 50 Hz o 100 Hz y están sincronizados con la red. Son de corta duración, incluso algunos microsegundos, y tienen una densidad espectral de potencia de decrece con la frecuencia. Este tipo de ruido es causado en general por dispositivos como los dimmers o por fuentes de alimentación que operan sincrónicamente con la frecuencia de red.

4. **Ruido Impulsivo periódico, asíncrono de la frecuencia de red:** estos impulsos tiene una tasa de repetición entre 50 kHz y 200kHz, que resultan en un espectro de líneas discreto, cuya separación depende de la frecuencia de oscilación. Este tipo de ruido es causado generalmente por las fuentes de alimentación por conmutación.

5. **Ruido Impulsivo (Aperiódico) Asincrónico:** es causado por los transitorios en la red. Estos impulsos tienen duraciones que van desde algunos microsegundos hasta algunos milisegundos con tiempos de ocurrencia aleatorios. La densidad espectral de potencia puede llegar a valores grandes, estando 50 dB por encima del ruido de fondo.

Los ruidos tipo 1 al 3 suelen ser estacionarios en períodos de tiempo del orden de los segundos, minutos o incluso horas. A su vez podrían llegar a ser sumarizados como ruido de fondo. Sin embargo, los ruidos tipo 4 y 5 son variantes en el tiempo en términos de microsegundos o milisegundos. Durante la ocurrencia de dichos impulsos la densidad espectral de potencia es notablemente grande y puede ocasionar errores de bits o ráfagas de errores en una transmisión de datos. Sin embargo, desde el punto de vista de la simulación/emulación es deseable tener el mayor control posible del escenario de ruido en tanto el modelo no se torne muy costoso en términos computacionales. Por lo tanto, se plantearán modelos para cada uno de los cinco tipos de ruido.

3.2. Ruido de fondo Coloreado

El ruido de fondo coloreado es causado por la superposición de múltiples fuentes de ruido de baja potencia y puede modelarse como un proceso aproximadamente Gaussiano [5]. En general, la densidad espectral de potencia se encuentra entre -120 dB (V^2/Hz) y -140 dB (V^2/Hz) con una tendencia creciente hacia las bajas frecuencias (menores a 1 MHz).

En la figura 3.1 se observa una típica medición del ruido de fondo [5] con una densidad espectral de potencia relativamente chica. El ruido de fondo se encuentra parcialmente superpuesto con ruido de banda angosta. El ruido de banda angosta a

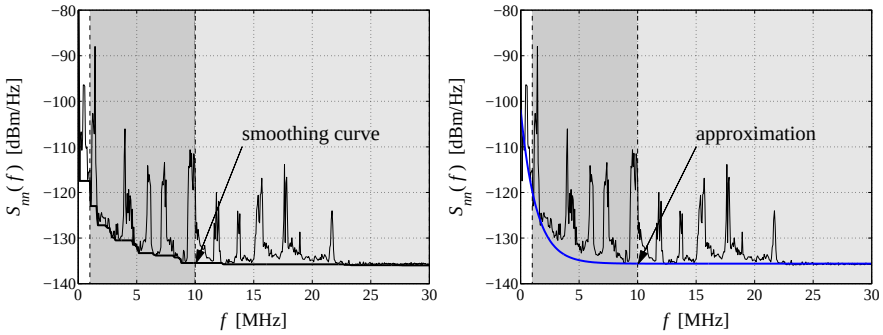


FIGURA 3.1: Extracción de PSD del ruido coloreado

veces es considerado parte del ruido de fondo. Sin embargo, por las razones anteriormente descritas nosotros lo generaremos de manera separada. De aquí en adelante, el ruido de fondo será considerado como ruido sin ninguna interferencia de banda angosta.

Generalmente, se puede observar que la densidad espectral de potencia disminuye a medida que aumenta la frecuencia. A su vez, el análisis de las mediciones de Benyoucef muestra que la para las frecuencias altas del espectro el ruido de fondo puede considerarse de intensidad constante. Adicionalmente, los resultados de múltiples mediciones de ruido en [31] han mostrado que la disminución de la densidad espectral de potencia con la frecuencia puede aproximarse por una exponencial decreciente en la escala logarítmica. El mismo enfoque se encuentra en [5], aunque Benyoucef también considera una expresión más compleja pero de mayor precisión al realizar el ajuste a la *smoothing curve* tal como muestra la figura 3.1. La expresión que utiliza es la superposición de exponenciales tal como muestra la ecuación 3.1.

$$S_{nn}(f) = N_0 + \sum_{k=1}^{M} N_k \cdot e^{-\frac{f}{f_k}} \quad [\text{dBm/Hz}] \tag{3.1}$$

Los órdenes superiores a 1 se utilizan para ajustar con mayor precisión la PSD en el rango de frecuencias inferior hasta los 500 kHz. Sin embargo, estos términos son desestimados debido a que el rango de frecuencias de la PSD que describen no es de importancia para las comunicaciones de alta velocidad (banda ancha) sobre líneas de tensión. Por lo tanto, la densidad espectral de potencia que utilizaremos tiene la siguiente expresión

$$S_{nn}(f) = N_0 + N_1 \cdot e^{-\frac{f}{f_1}} \quad [\text{dBm/Hz}] \tag{3.2}$$

N_0 es el valor de la densidad espectral de potencia para $f \to \infty$. Mientras que N_1 es la diferencia entre $S_{nn}(\infty)$ y $S_{nn}(0)$. El tercer parámetro f_1 sirve para modelar la tasa de decaimiento. Entonces podemos ver la curva exponencial de primer orden graficada en azul junto con la medición en color negro en la parte derecha de la figura 3.1.

Anteriormente mencionamos que los canales BPL exhiben una gran variabilidad debido a la no exclusividad de los mismos para la transmisión de datos. Es por ello que si se quiere tener un modelo matemático, éste tiene que tener componentes estadísticas. El autor de [5] realizó una caracterización estadística de los parámetros de (3.2), cuyos resultados se muestran en la tabla 3.1.

	N_0 [dBm/Hz]	N_1 [dBm/Hz]	f_1 [MHz]
Distribución	Normal	Uniforme	Exponencial desplazada
Oficina	$\sigma_{N_0} = 1,29$	$a_{N_1} = 23,06$	$\lambda_{f_1} = 1,300$
	$\mu_{N_0} = -135,00$	$b_{N_1} = 74,97$	$f_{min} = 0,096$
Residencial	$\sigma_{N_0} = 4,14$	$a_{N_1} = 30,83$	$\lambda_{f_1} = 0,840$
	$\mu_{N_0} = -137,20$	$b_{N_1} = 70,96$	$f_{min} = 0,100$

Tabla 3.1: Modelo estadístico de los parámetros del ruido de fondo coloreado

3.3. Ruido de Banda Angosta

En general, los escenarios de ruido presentes en PLC contienen ruido de banda angosta, cuya intensidad y frecuencia varían a lo largo del tiempo y dependen de la ubicación. Las principales fuentes para este tipo de ruido son las radiodifusoras de onda larga, media y corta así como otros servicios de radio. En consecuencia, casi toda la banda de frecuencias que utiliza PLC se encuentra superpuesta con este tipo de interferencia. A continuación se muestra el espectro de ruido [49] que evidencia la aparición de interferencias de banda angosta en la figura 3.2. La señal tiene una resolución de 8 bits y fue capturada a 50 Msps con un DSO. La densidad espectral de potencia fue estimada usando el método de Welch a partir de una señal almacenada de 20 ms de duración. La resolución espectral obtenida es de 750 Hz. Notar que el gráfico tiene un piso de ruido da -30 dBmV o -105,7 dBmV/Hz [*] , que es un poco superior al promedio de -135 dbm/Hz que suele encontrarse (tabla 3.1). Una causa posible podría ser el ruido de cuantización al tener una resolución de 8 bits, que sólo permite un rango dinámico -42 dB si consideramos el bit de signo.

[*] usando la fórmula dBm = dBmV $- 10 \cdot \log_{10}\left(\frac{R}{10^{-3}}\right)$ suponiendo R= 50Ω, que es la entrada típica de un DSO, tenemos -77 dBm y teniendo en cuenta que dicho valor es sobre 750 Hz entonces tenemos $10 \cdot \log_{10}(1/750) - 77$ dBm $\approx -105,7$ dBm/Hz

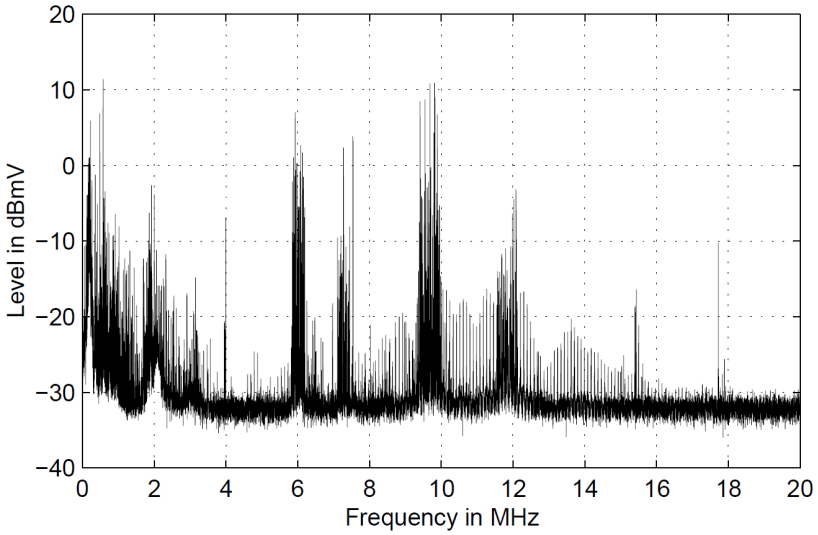


FIGURA 3.2: Estimación de la PSD del ruido en un canal BPL

Podemos ver que la densidad espectral de potencia del ruido de banda angosta es marcadamente superior al ruido de fondo. En especial se observa la interferencia causada por las estaciones de radio con longitud de onda de 49 m (5,95 - 6,2 MHz), 41 m (7,2 - 7,5 MHz), 31 m (9,4 - 10,1 MHz) y 25 m (11,8 - 12,1 MHz). A su vez, en el espectro por debajo de los 5 MHz la mayoría de la interferencia también puede ser caracterizada como ruido de banda angosta. Por lo tanto, el ruido de banda angosta merece ser considerado de manera separada para su síntesis en un emulador. Una consecuencia de que este ruido sea causado por ondas de radio es que su influencia puede ser distinta dentro de un mismo edificio o casa, por ejemplo dependiendo de en cuál piso se está [13, p. 268].

El modelo que le corresponde es el mismo que describe la superposición de señales de amplitud modulada, es decir:

$$n_{NB}(t) = \sum_{i=1}^{N} A_i(t) \sin(2\pi f_i t + \theta_i) \tag{3.3}$$

donde N es la cantidad de portadoras consideradas cuya frecuencia central f_i es distinta para cada una. La señal de información a modular es $A_i(t)$ y la fase es θ_i, la cual suele considerarse constante y aleatoria en el intervalo $[0; 2\pi)$. Por otro lado, la señal $A_i(t)$ permitirá representar interferencias en una sola frecuencia, en el caso de $A_i(t) = \text{cte}$, o en un rango de frecuencias cuyo ancho de banda depende del ancho de banda de $A_i(t)$. Umehara et al. [40] y Skrzypczak et al. [38] plantean utilizar ruido Gaussiano blanco filtrado para generar $A_i(t)$. Si bien los autores mencionados no justifican su elección en fundamentos empíricos o teóricos, es razonable aceptar esta hipótesis para modelar la superposición de múltiples emisoras de AM o canales de

radioaficionados. La ubicación, potencia, cantidad de portadoras y ancho de banda de las mismas depende obviamente del lugar en que se esté por varios motivos. No todos los países tienen las mismas bandas de frecuencia asignadas ni tampoco hay una misma cobertura de las ondas de radio en cuestión. Por ejemplo, en lugares remotos, como por ejemplo en zonas rurales, hay menor o ninguna cobertura de algunos servicios de radio.

Dependiendo de la cantidad de portadoras y la posición de las mismas, convendrá generar la señal $n_{NB}(t)$ por síntesis de múltiples portadoras en el tiempo o estableciendo amplitudes y fases en el dominio de la frecuencia para finalmente transformarse en una señal temporal por medio de la IFFT. Para la reproducción de un gran número de interferencias de banda angosta el método en el dominio de la frecuencia es el más adecuado. En la figura 3.3 podemos observar el modelo de ruido implementado por síntesis de múltiples portadoras. También podemos ver que el ancho de banda de la señal $A_i(t)$ se realiza mediante un filtro. A su vez, en la figura 3.4 vemos el modelo de generación de ruido de banda angosta a través de la IFFT. El bloque central cumple el rol de determinar en qué frecuencias centrales habrá interferencia y que ancho de banda tendrán las mismas. A simple vista podríamos ver que necesitamos menos generadores de ruido aleatorio Gaussiano que en el primer método.

Un aspecto general del ruido es que su intensidad varía muy lentamente con el tiempo, de manera que nos podríamos tomarnos el atrevimiento de cambiar los parámetros sólo cuando cambiemos de escenario de emulación. Normalmente, la máxima intensidad de este tipo de ruido se da en las horas de la noche dado que en ese momento las condiciones de propagación de las ondas de radio en la atmósfera son más favorables [13, pp. 264-265][49].

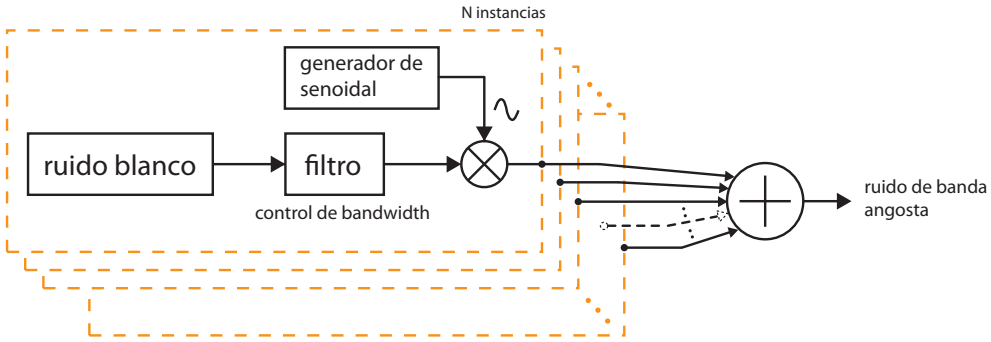


FIGURA 3.3: Modelo del ruido de banda angosta

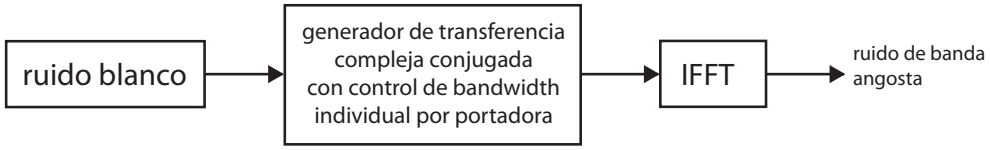


FIGURA 3.4: Modelo del ruido de banda angosta mediante IFFT

3.4. Ruido Impulsivo periódico, síncrono con la red

El ruido impulsivo periódico sincrónico ocurre en frecuencias de 50 Hz o 100Hz en una red de alterna de 50 Hz (60 Hz o 120 Hz en una red de 60 Hz). Estos son causados por convertidores de potencia sincrónicos como ser dimmers y todo tipo de rectificadores que utilicen diodos o SCR.

La figura 3.5 muestra la medición de un típico ruido impulsivo sincrónico con un período de 10 ms que corresponde a una frecuencia de 100 Hz.

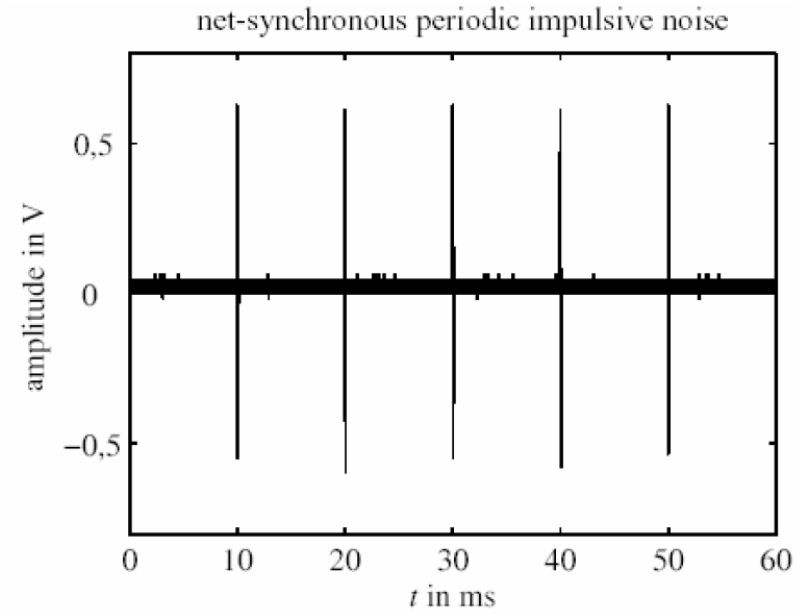


FIGURA 3.5: Ruido impulsivo periódico sincrónico

Por un lado, el ruido impulsivo sincrónico se caracteriza por una envolvente que puede ser descripta por una señal de impulsos rectangulares repetitivos. Esta envolvente determina la amplitud (A), el período (t_P) y la duración (t_B) del impulso. Estos se observan en la figura 3.6. Vale la pena recalcar que $t_P = 10$ ms o $t_P = 20$ ms típicamente. Por otro lado, para conocer el impacto del ruido es importante conocer las características temporales de la señal dentro de la envolvente. El conocimiento del comportamiento temporal de dicho intervalo de perturbación determina las características espectrales. Por lo tanto, este ruido impulsivo puede ser modelado

aproximadamente por una fuente de ruido coloreada donde una envolvente determina cuando la señal debe pasar o no. Dicho modelo es mostrado en la figura 3.7.

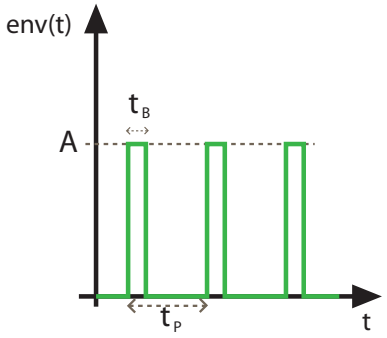


FIGURA 3.6: Envolvente del ruido y sus parámetros

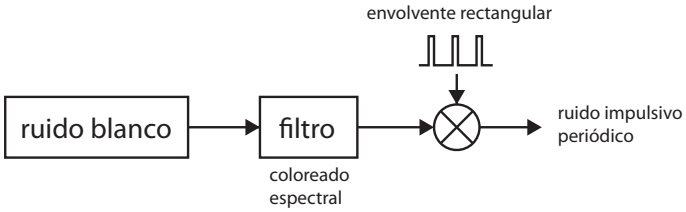


FIGURA 3.7: Modelo del ruido impulsivo periódico sincrónico

Las características espectrales pueden asumirse constantes para una determinada fuente de ruido. Con este modelo se pierde la capacidad de modelar variaciones espectrales en el tiempo, sin embargo para el análisis del impacto del ruido en un sistema de comunicaciones esta aproximación es aceptable.

3.4.1. Análisis espectral

Para poder realizar observaciones sobre las características del espectro, debemos hacer un análisis del mismo. En primer lugar, tenemos la ventana rectangular

$$r(t) = A \cdot \Pi\left(\frac{t}{t_B}\right) \tag{3.4}$$

$$\Pi(t) = \begin{cases} 1 & -1/2 \leqslant t \leqslant 1/2 \\ 0 & \forall \text{ otro } t \end{cases} \tag{3.5}$$

cuya transformada de Fourier es

$$R(f) = A \cdot t_B \cdot \text{sinc}(\pi f t_B) \tag{3.6}$$

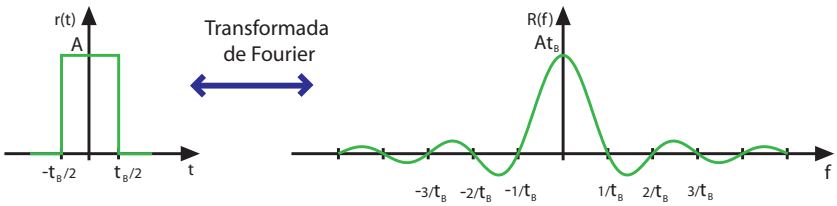


FIGURA 3.8: Pulso rectangular y su transformada de Fourier

Para tener un tren de impulsos rectangulares convolucionamos $r(t)$ con un tren de deltas de Dirac ($p_{t_P}(t)$) distanciadas entre sí t_P.

$$p_{t_P}(t) = \sum_{k=-\infty}^{\infty} \delta\left(t - kt_P\right) \tag{3.7}$$

Dicho tren de deltas se transforma en otro tren de deltas de Dirac en frecuencia. La separación entre deltas es el recíproco de t_P, es decir $1/t_P$.

$$P_{t_P}(f) = \frac{1}{t_P} \sum_{k=-\infty}^{\infty} \delta\left(f - \frac{k}{t_P}\right) \tag{3.8}$$

Por lo tanto, podemos expresar la señal de pulsos rectangulares como

$$r_P(t) = r(t) * p_{t_P}(t) \tag{3.9}$$

que en el dominio de la frecuencia se transforma en una multiplicación de las transformadas. En la figura 3.9 se observa el espectro de $R_P(f)$ que resulta ser un tren de deltas modulado por una función $\text{sinc}(\cdot)$[2].

$$R_P(f) = A \cdot \frac{t_B}{t_P} \cdot \text{sinc}(\pi f t_B) \sum_{k=-\infty}^{\infty} \delta\left(f - \frac{k}{t_P}\right) \tag{3.10}$$

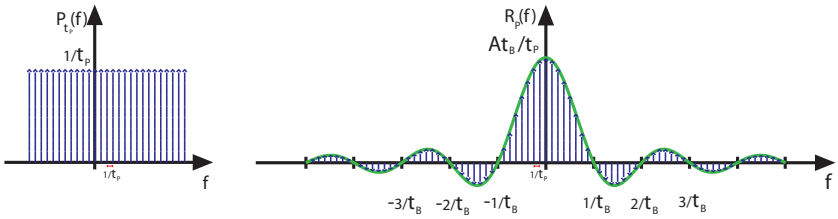


FIGURA 3.9: Transformadas de Fourier de $p_{t_P}(t)$ y $R_P(t)$

[2] $\text{sinc}(x) \triangleq \frac{\sin(x)}{x}$

Finalmente, el ruido periódico impulsivo sincrónico puede expresarse de la siguiente manera

$$n_{PI}(t) = n(t) \cdot r_P(t) \tag{3.11}$$

donde $n(t)$ es el ruido coloreado que será modulado, tal cual se ve en la figura 3.7. Ahora realizaremos un análisis en frecuencia a modo indicativo. Decimos que es indicativo porque nos da una idea de la situación pero no es formalmente correcto porque el espectro de un ruido no está dada por la transformada de Fourier de la señal temporal sino por la densidad espectral de potencia del proceso. Entonces, su característica en frecuencia está dada por el producto de convolución $N(f) * R_P(f)$.

En la práctica t_B se encuentra entre 10 μs y 100 μs, y por tanto sus valores recíprocos son 100 kHz y 10 kHz respectivamente. También, se observa que el ancho de banda equivalente del ruido coloreado es mucho mayor que $1/t_B$. Este hecho se traduce en que el espectro del ruido $N(f)$ es muy plano comparado con la sinc cuyo lóbulo principal tiene un ancho de $2/t_B$, tal como muestra la figura 3.10. Por este motivo, $R_P(f)$ es visto como una especie de "delta" frente a $N(f)$ y esto implica que al ser convolucionado, el espectro resultante $N_{PI}(f)$ tiene un perfil similar que $N(f)$ excepto por un factor de escala.

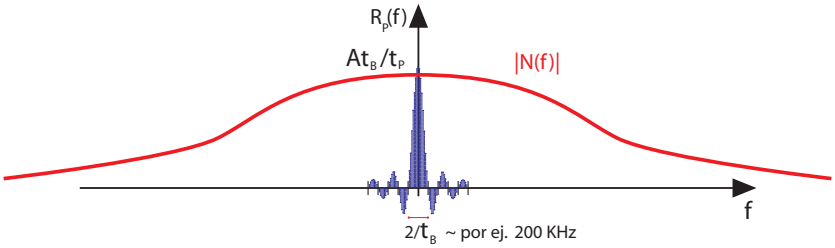


FIGURA 3.10: Comparación entre $|N(f)|$ y $R_P(f)$

3.5. Ruido Impulsivo periódico, asíncrono con la red

El ruido impulsivo periódico asincrónico con la frecuencia de la red está principalmente causado por fuentes de alimentación por conmutación. En general, la frecuencia de repetición, $\frac{1}{t_P}$, se encuentra entre 50 kHz y 2 MHz [1, p. 48]. Según los autores de [1] y [13] estos ruidos ocurren solamente por un breve lapso y tienen un amplitud relativamente baja es muy difícil de medirlos y analizarlos y por ende, su impacto en la medición del espectro de ruido y en los sistemas de comunicaciones puede describirse mejor como un aumento en la densidad espectral de potencia del ruido de fondo. Por lo tanto, ellos afirman que los ruidos de este tipo deben ser considerados como parte del ruido de fondo coloreado. Sin embargo es el mismo autor (K. Dostert) que en [49] muestra que dicho tipo de ruido no es despreciable y de hecho se observa en la medición que publicaron. En la figura 3.2 se observa un espectro de

líneas cuya separación es de 100 kHz correspondiente a ruido impulsivo periódico con período de 10 μs [49]. Dichas características espectrales pueden afectar de manera considerable tonos completos en esquemas de modulación multiportadora como ser OFDM ya que su potencia puede ser más de 10 dB mayor que el ruido de fondo coloreado.

Este error puede explicarse por suponer que el método de síntesis de los impulsos es el mismo que el utilizado para el ruido impulsivo periódico sincrónico (figura 3.7), es decir que su espectro es ruido coloreado de banda ancha. Sin embargo, la medición observada de este ruido puede explicarse si el espectro de los impulsos es ruido de banda angosta cuya frecuencia central se encuentra desplazada en el orden los megahertz o decenas de megahertz tal como observamos en la figura 3.2 (nuevamente usamos $N(f)$ para hacer un análisis a modo indicativo) . La naturaleza de estos ruidos es causada por la existencia de circuitos resonantes con frecuencia f_c tal como se observa en la figura 3.11. Si el ancho de banda del espectro de los impulsos, n_{bw}, es menor que $\frac{1}{t_P}$, luego de la convolución con el espectro de la señal de impulsos rectangulares $r_P(t)$ se obtendrá un espectro con forma de peine, $N_{PI}(f)$.

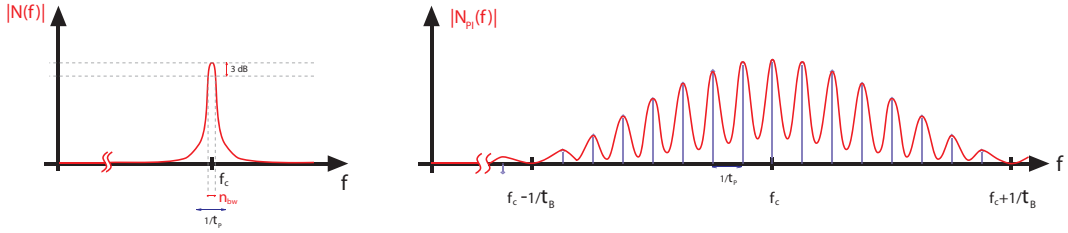


FIGURA 3.11: PSD del ruido impulsivo periódico asincrónico

El modelo de ruido es entonces similar al del ruido impulsivo sincrónico (ecuación 3.11) excepto que ahora el ruido es de banda angosta y con una frecuencia central f_c, y está nomenclado como $n_b(t)$

$$n_{PI_{asinc}}(t) = n_b(t) \cdot r_P(t) \tag{3.12}$$

3.6. Ruido Impulsivo (aperiódico) asincrónico

El ruido impulsivo posee una gran variación en el tiempo por su naturaleza, alejándonos de los modelos estacionarios que se pueden aplicar a los ruidos de fondo. Este ruido está causado por todo tipo de eventos de conmutación, por ejemplo provenientes de electrodomésticos, motores, capacitores de lámparas de descarga, etc. Además, suele ocurrir en ráfagas, lo cual aumenta el impacto disruptivo. Sus propiedades espectrales y temporales son muy variadas ya que puede ser causado por fuentes de distinta índole.

Los impulsos suelen tener una forma similar a una sinusoide amortiguada o subamortiguada, pero también pueden no tener una estructura definida. Esta afirmación es visible en una de las señales adquiridas y presentadas en [49]. En la señal

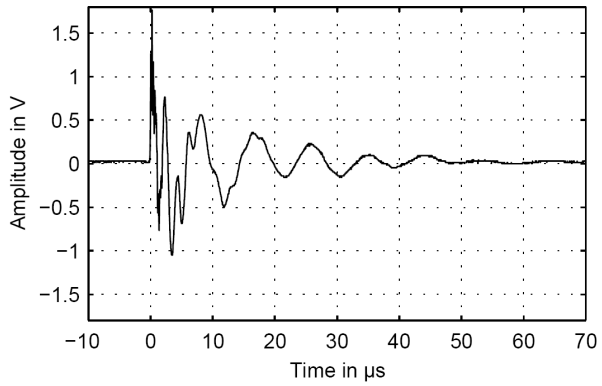


FIGURA 3.12: Impulso 1 de ruido

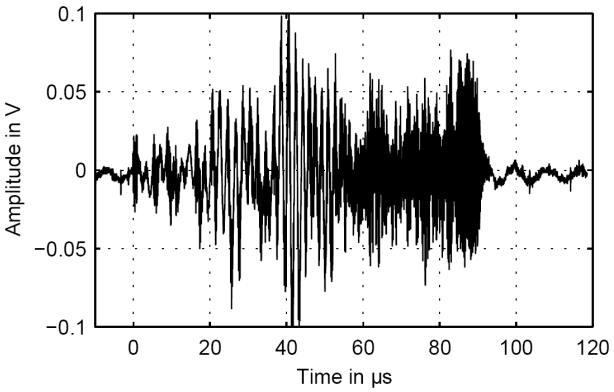


FIGURA 3.13: Impulso 2 de ruido

de la figura 3.12 se aprecia un flanco creciente seguido por una oscilación amortiguada. La duración aproximada de la señal es de 50 μs. En el segundo impulso, figura 3.13, no se observa una estructura clara, su amplitud es de 0,1 V y su duración es aproximadamente 90 μs con un final abrupto. Estos ejemplos muestran la gran diversidad de señales que se pueden encontrar. Es por esto que mencionamos que las características espectrales son muy variadas.

3.6.1. Caracterización de los impulsos

Para caracterizar el impacto de los impulsos en la transmisión de datos se considera la energía y la potencia del impulso. Junto con el tiempo de arrivo t_{arr} y la duración, o ancho, t_w del impulso se puede calcular la energía E_{Imp} a partir de la

señal del impulso de ruido $n_{Imp}(t)$.

$$E_{Imp} = \int_{t_{arr}}^{t_{arr}+t_w} n_{Imp}(t)^2 \, dt \tag{3.13}$$

La energía del impulso depende tanto de la forma de la señal como de su duración. Para poder comparar dichos eventos con el ruido de fondo la potencia promedio es una magnitud más adecuada. La potencia promedio P_{Imp} puede ser calculada mediante:

$$P_{Imp} = \frac{1}{t_w} \int_{t_{arr}}^{t_{arr}+t_w} n_{Imp}(t)^2 \, dt \tag{3.14}$$

La potencia promedio de la señal de ruido de fondo $n_{BG}(t)$ observado durante un tiempo T_B puede ser obtenida de:

$$P_N = \frac{1}{T_B} \int_{0}^{T_B} n_{BG}(t)^2 \, dt \tag{3.15}$$

De esta manera, la relación entre la potencia promedio del ruido de fondo P_N y la potencia del impulso P_{Imp} da una medida del cambio dinámico de escenario de ruido que ocurre durante un evento impulsivo. Esta comparación se puede ver en la tabla 3.2 donde se muestran los parámetros característicos de los impulsos. Mientras que el impulso 2 está 21 dB por sobre el ruido de fondo, el impulso 1 empeora la SNR más de 40 dB durante la ocurrencia del evento. Vale la pena aclarar que los

	Ancho del Impulso t_w	Amplitud del Impulso A	Potencia del Impulso P_{Imp}	Energía del Impulso E_{Imp}
Impulso fig. 3.12	46,1 μs	1,77 V	$-11{,}1$ dBV^2	5,54 dBμV^{2}s
Impulso fig. 3.13	90,6 μs	0,1 V	$-31{,}3$ dBV^2	$-11{,}7$ dBμV^{2}s
Ruido de Fondo			$P_N = -52{,}5$ dBV^2	

Tabla 3.2: Características de los impulsos de la figura 3.12 y 3.13

parámetros fueron computados [49] con un banco de mediciones que contemplaba un rango de frecuencias de 0,2 a 20 MHz. Para un estudio más preciso es necesario trabajar con la densidad espectral de potencia $S_{nn,Imp}(f)$ ya que tiene en cuenta la distribución de la potencia del ruido en el espectro. La estimación de la densidad espectral de potencia de los dos impulsos ejemplos es mostrada en la figura 3.14. Para el cálculo de la misma se basó en una estimación paramétrica sobre un modelo AR. Ambos impulsos exceden al ruido de fondo en la totalidad del rango de frecuencias por al menos 10 o 15 dB. En algunas bandas de frecuencia el impulso 1 supera al ruido de fondo por más de 50 dB, mientras que el impulso 2 sólo llega a exceder en

30 dB. Se puede observar que la energía de los impulsos está concentrada en ciertos rangos de frecuencia. El valor máximo se encuentra por debajo de 1 MHz. Esta es una afirmación que tiene validez general. Su característica de banda ancha está dada por los fuertes flancos, mientras que su concentración en determinadas bandas de frecuencia está dada por las oscilaciones.

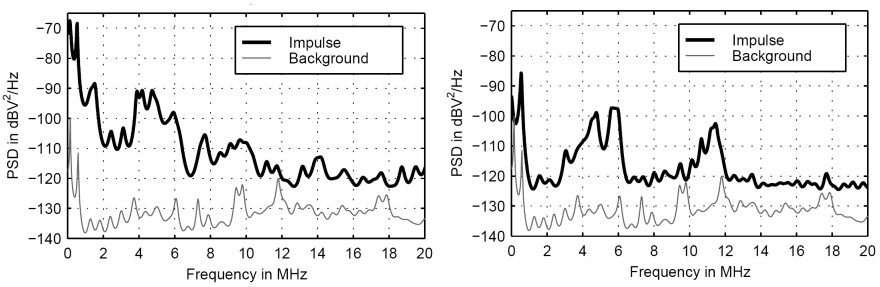


FIGURA 3.14: Densidad Espectral de Potencia de los impulsos

3.6.2. Parámetros característicos del ruido impulsivo

El ruido impulsivo aperiódico asincrónico puede caracterizarse, así como el ruido impulsivo periódico, por la amplitud, duración del impulso, distancia entre impulsos y características espectrales. A diferencia del ruido periódico, estas propiedades no son constantes sino que son diferentes para cada evento singular. El objetivo del modelado es encontrar un modelo estocástico que describa las propiedades estadísticas de dichas variables. La dificultad que se presenta es cubrir toda la variedad de parámetros en un único modelo unitario. Por ello, para el modelado y la generación en un simulador tiene sentido considerar la envolvente del ruido y las caracteríticas espectrales de los impulsos por separado. Zimmermann [47] ha realizado estudios prolongados sobre las propiedades de este ruido y a continuación mencionaremos algunos resultados junto con la presentación de los parámetros característicos. Vale la pena notar que Chan and Donaldson [11] también han realizado estudios sobre parámetros de amplitud, duración y tiempo de interarribo de los impulsos, sin embargo como estos fueron realizados 1988 y con instrumental de limitaciones tecnológicas acordes a la época, sus resultados se corresponden a escenarios particulares aunque la metodologúa de estudio es similar a la utilizada por Zimmermann.

3.6.2.1. Tasa de impulsos

Un criterio importante que se utiliza para caracterizar la intensidad del ruido impulsivo es la tasa de impulsos. La dificultad que se presenta en la medición que es este ruido se encuentra mezclado con el ruido impulsivo periódico. Es por ello que a la hora de modelar el ruido impulsivo aperiódico no deben considerarse estos valores sin un procesamiento adecuado. Además, la tasa de impulsos depende de la sensibilidad de detección. La sensibilidad de detección especifica el valor de tensión

que el máximo absoluto de un impulso debe exceder por para que este sea reconocido como ruido. Si este nivel se aumenta, la tasa de impulsos disminuye dramáticamente. Valores típicos para la tasa de impulsos son 0,1 impulsos por segundo para baja perturbación y 100 impulsos por segundos para ambientes con alta perturbación con un nivel de detección de 100 mV.

3.6.2.2. Duración relativa de la perturbación

La duración relativa de la perturbación es la fracción de tiempo promedio en la que existe perturbación. Asimismo, su valor también depende del nivel de detección. Valores típicos [47] pueden ir desde 0,001 % hasta 1 % usando un nivel de detección de 100mV. En estos valores también se incluyó el ruido impulsivo periódico.

3.6.2.3. Amplitud del impulso

La distribución de amplitudes de los impulsos se aproxima a una distribución exponencial [1, 12]. Pueden ocurrir discrepancias con el modelo si el ruido está dominado por unas pocas fuentes lo que deriva en el predominio de ciertos valores de amplitudes. Los impulsos con amplitudes mayores a 1 V son de rara ocurrencia [1, 49].

3.6.2.4. Ancho y espaciamiento temporal de impulsos

La duración de los impulsos y el espaciamiento temporal, o distancia entre impulsos, también tienen una distribución exponencial. Por lo tanto, podemos ver que hay múltiples distribuciones que se solapan. Valores típicos para la duración de los impulsos son 100 μs, y para la distancia temporal están entre 10 ms y 1 s. La duración del impulso, la distancia entre impulsos, la duración relativa de la perturbación y la tasa de impulsos no son independientes sino que se influencian entre sí, haciendo que la generación de un escenario de ruido de referencia sea de mayor dificultad.

Contexto de validez del modelo Bajo estas premisas es válido modelar los impulsos como un ruido caracterizado por una gran densidad espectral que es modulado en el tiempo por ventanas rectangulares. El tiempo que la ventana permanece abierta es el tiempo que más adelante denominaremos t_{w_i}. Debido a la naturaleza aleatoria del fenómeno se utilizará un proceso estocástico para simular este tipo de interferencia.

3.6.3. Ruido en ráfagas

El ruido en ráfagas en una manifestación particular del ruido impulsivo aperiódico asincrónico. Este suele darse cuando la fuente causante de los impulsos suele producir varios impulsos en un corto lapso de tiempo en un lugar de uno solo. A nivel de modelo matemático puede pensarse que durante un breve tiempo, llamémoslo tiempo de ráfaga, la tasa de impulsos es muy elevada. Las características espectrales de cada

uno de los impulsos dentro de una ráfaga podrían considerarse similares sin cometer demasiado error en el modelado. Esto es así debido a que provienen de la misma fuente. De todas maneras, se busca una emulación estadística de la realidad y no una réplica de la misma, es decir que lo generado por el emulador tenga el mismo impacto estadístico que lo acontecido en la realidad. Los detalles de su generación se verán más adelante en §3.12.

3.7. Modelo de ruido de fondo Coloreado

El ruido de fondo coloreado tendrá la siguiente densidad espectral de potencia

$$S_{nn}(f) = N_0 + N_1 \cdot e^{-\frac{f}{f_1}} \quad [\text{dB V}^2/\text{Hz}] \qquad f \geqslant 0 \tag{3.16}$$

donde N_0 es el valor de la densidad espectral de potencia para $f \to \infty$. Mientras que N_1 es la diferencia entre $S_{nn}(\infty)$ y $S_{nn}(0)$. El tercer parámetro f_1 sirve para modelar la tasa de decaimiento.

Este modelo será generado a través de la siguiente arquitectura de la figura 3.15

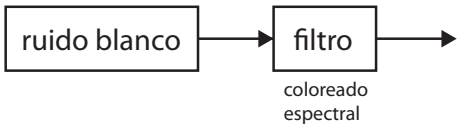


FIGURA 3.15: Arquitectura de generación de ruido coloreado

3.8. Modelo de ruido de Banda Angosta

En esta sección describiremos los dos modelos expuestos en §3.3 para la generación de ruido de banda angosta.

3.8.1. Modelo de superposición de señales moduladas

En este modelo, la generación de ruido de banda angosta queda descripta por la superposición de señales de amplitud modulada, es decir:

$$n_{NB}(t) = \sum_{i=1}^{N} A_i(t) \sin(2\pi f_i t + \theta_i) \tag{3.17}$$

donde N es la cantidad de portadoras consideradas cuya frecuencia central f_i es distinta para cada una. La señal de información a modular es $A_i(t)$ y la fase es θ_i, la cual suele considerarse constante y aleatoria en el intervalo $[0; 2\pi)$.

Por otro lado, la señal $A_i(t)$ permitirá representar interferencias en una sola frecuencia, en el caso de $A_i(t) = \text{cte}$, o en un rango de frecuencias cuyo ancho de banda

depende del ancho de banda de $A_i(t)$. El control del ancho de banda o coloreado se puede realizar con un filtro AR de primer orden (AR 1). En el apéndice §D.1 se describen las características de un proceso AR 1. Sus características espectrales están dadas por la ecuación (D.8) que repetiremos a continuación

$$S_{XX}(f) = \frac{\sigma_N^2}{1 - \varphi \cdot \cos(2\pi f) + \varphi^2} \quad 0 \leqslant f < 1$$

La elección de este tipo de filtro es porque permite controlar el ancho de banda mediante el uso de un sólo parámetro, φ, y el bajo costo de recursos que implica. A continuación mostraremos como éste influencia el ancho de banda. Como nosotros modularemos este ruido necesitamos que sea un espectro de banda base y por lo tanto sólo usaremos valores de φ positivos y menores que 1. Para calcular cual será el ancho de banda para un dado φ deberemos primero definir cómo medimos el ancho de banda y plantear las ecuaciones correspondientes. Nosotros trabajaremos con el ancho de banda de -3 dB y por lo tanto tenemos el siguiente planteo, hallar el valor máximo de $S_{n_b n_b}(f)$ y luego el valor de f al cual vale la mitad.

$$\max_{\varphi>0} S_{n_b n_b}(f) = S_{n_b n_b}(0) = \frac{\sigma_N^2}{1 - 2\varphi + \varphi^2} \tag{3.18}$$

entonces la frecuencia de -3 dB será aquella en cuyo valor $S_{n_b n_b}(f_{1/2})$ sea la mitad de $S_{n_b n_b}(0)$. Si despejamos términos llegamos a (3.19) y de ahí es fácil llegar a la expresión de $f_{1/2}$ (3.20).

$$S_{n_b n_b}(f_{1/2}) = \frac{\sigma_N^2}{1 - 2\varphi \cdot \cos(2\pi f_{1/2}) + \varphi^2} = \frac{1}{2} \frac{\sigma_N^2}{1 - 2\varphi + \varphi^2}$$

$$\cos(2\pi f_{1/2}) = \frac{4\varphi - 1 - \varphi^2}{2\varphi} \tag{3.19}$$

$$f_{1/2} = \frac{1}{2\pi} \operatorname{arc\,cos}\left(\frac{4\varphi - 1 - \varphi^2}{2\varphi}\right) \tag{3.20}$$

Debemos tener en cuenta que este ruido está en banda base y va a ser modulado y en consecuencia el ancho de banda será el doble de $f_{1/2}$ y que además la frecuencia está normalizada y debe multiplicarse por la frecuencia de muestreo f_s.

$$BW = 2 \cdot f_{1/2} \cdot f_s \tag{3.21}$$

A continuación observamos las figuras 3.16 y 3.17 donde se muestra el ancho de banda para algunos valores de φ. Se puede ver que los valores de φ de interés son bastante cercanos a 1. Los valores graficados se detallan en la tabla 3.3 y corresponden para una frecuencia de muestreo de 80 Msps. El último valor de esta tabla es de importancia ya que nuestra representación numérica es de punto fijo. Es importante notar que si se utilizasen otros valores de f_s el valor del ancho de banda variaría de acuerdo a la ecuación (3.21).

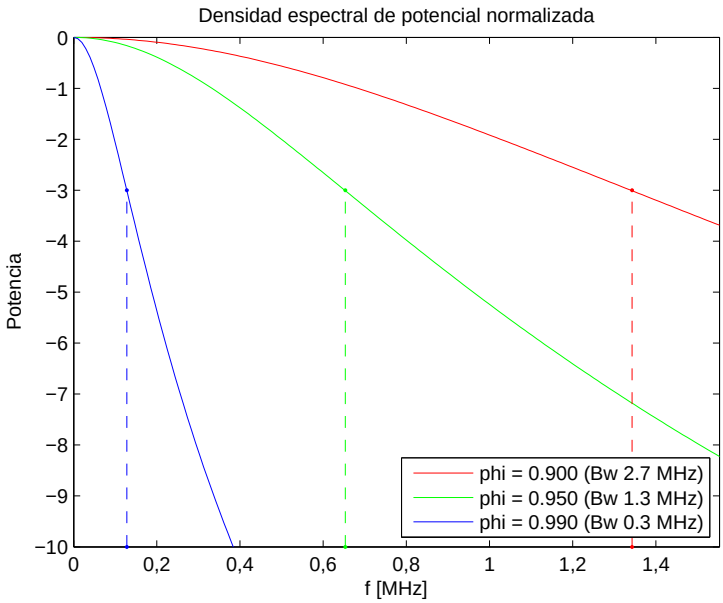


FIGURA 3.16: Ancho de banda en función de φ a 80 Msps ($BW > 0{,}3$ MHz)

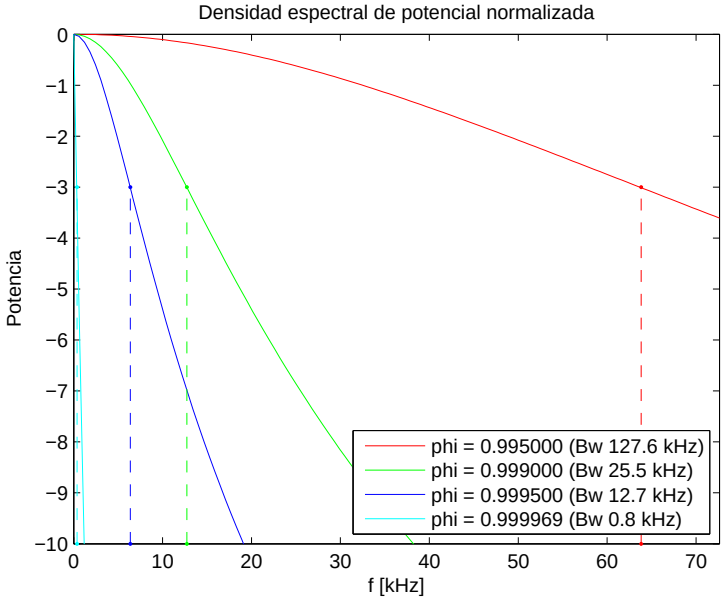


FIGURA 3.17: Ancho de banda en función de φ a 80 Msps ($BW < 0{,}3$ MHz)

La potencia de salida de un proceso AR 1 está dada por la ecuación (D.3) y depende de φ

$$\sigma_X^2 = \frac{\sigma_N^2}{1-\varphi^2}$$

φ ($f_s = 80$ Msps)	BW
0,9	2,7 MHz
0,95	1,3 MHz
0,99	0,3 MHz
0,995	127,6 kHz
0,999	25,5 kHz
0,9995	12,7 kHz
$2^{-1} + \ldots + 2^{-15}$	0,8 kHz

Tabla 3.3: Relación entre φ y el ancho de banda de -3 dB

Por lo tanto, si deseamos tener un ruido con potencia σ^2_{des} debemos multiplicar la salida por una constante c:

$$c = \sigma_{des} \cdot \sqrt{1 - \varphi^2} \tag{3.22}$$

donde el factor $\sqrt{1 - \varphi^2}$ logra que la potencia del proceso sea unitaria y luego σ_{des} la amplifica para tener una potencia deseada de σ^2_{des}.

Filtros IIR de orden 2 Sería desconsiderado no incluir un breve comentario acerca del trabajo que se realizó en este tema pero que decidimos no incluirlo de manera extensiva porque no correspondía al objeto primario de esta tesis. Una pregunta válida que uno puede hacerse es: ¿por qué se eligió un proceso AR 1 para colorear el ruido blanco? Si bien expusimos la simplicidad de este método, la utilización de un filtro de primer orden impone restricciones acerca de cuan abrupta será la caída de la respuesta en frecuencia tras la frecuencia de corte. Entonces planteamos estudiar filtros de orden superior ya que estos son mucho más performantes para la transición a la banda no pasante. Las conclusiones que obtuvimos fueron las siguientes:

- La estabilidad numérica era un problema mayor y llevaba a trabajar con precisiones de 59 bits que eran necesarias para lograr no sólo una estabilidad BIBO sino una buena copia de la respuesta al impulso.
- El hecho de agregar un multiplicador más y de mayor precisión generaba problemas para lograr los objetivos establecidos para la frecuencia de funcionamiento. Entonces se estudiaron técnicas eficientes para la implementación de alta velocidad de filtros digitales [4, 29, 35]. Éstas se basaban en aplicar la técnica de *pipelining*, es decir, subdivir las operaciones para lograr el mismo resultado pero en un mayor número de etapas. Esto permitía insumir mayor tiempo para realizar las multiplicaciones al costo de aumentar la cantidad de recursos necesarios. La técnica estudiada se llama *Look-Ahead Transformation* que aplicada a un filtro IIR de segundo orden de estructura *Direct Form II* se obtenía otra conocida como *Scattered Look-Ahead Form*[3]. Este camino se abandonó al ver

[3] esta forma tiene la estabilidad asegurada si el filtro original es estable.

que los recursos necesarios para esta implementación no tenían sentido en nuestra plataforma y decidimos enfocarnos en la estrategia del AR 1. Aún peor, de haber utilizado un filtro de orden superior y con mayor utilización de recursos, en la comparación contra el método de la IFFT se vería en una posición más desfavorable.

3.8.2. Modelo utilizando la IFFT

Este modelo es más apropiado para generar una mayor cantidad de interferencias de banda angosta debido que dicha cantidad no afecta los recursos involucrados en la generación. En el caso anterior, cada instancia de ruido necesitaba de recursos asignados exclusivos. Por lo que un generador de ruido de 8 interferencias de banda angosta necesita del doble de recursos que uno de 4. En el caso del método de la IFFT lo único que varían son los valores del vector de amplitudes. El vector de amplitudes es el que determina qué potencia tendrá la señal en determinada frecuencia discreta f_k.

La desventaja de este método podría verse en la precisión de la ubicación de las interferencias en el espectro y en la mayor cantidad de parámetros que tiene el modelo. En el caso de superposición de señales moduladas teníamos tres parámetros por instancia (potencia, φ del AR 1 y frecuencia central), si obviamos la fase aleatoria, φ_i, de la portadora. En contraparte, si utilizáramos una IFFT de 16384 puntos necesitaremos de 8193 parámetros de amplitud y 8193 parámetros de control del ancho de banda de cada tono. Aunque esta complejidad se puede resolver fácilmente con una computadora y no comprendería un requisito adicional al sistema de emulación ya que la misma es necesaria poder configurar y controlar el emulador.

Como nosotros trabajamos las señales en banda base se necesita llegar a una señal real para la generación de ruido mediante la IFFT . Esto puede lograrse de las siguientes maneras:

1. construir un vector de valores adecuados que al antitransformar genere una parte imaginaria nula

   a) generar un vector de valores complejos conjugados simétricos, es decir que la parte real es par y la imaginaria es impar[4].

   b) generar un vector de valores reales par, es decir que forzamos la parte imaginaria a cero.

   c) generar un vector de valores imaginarios impar, es decir que forzamos la parte real a cero.

2. construir un vector de valores cualesquiera y descartar la parte imaginaria luego de antitransformar

[4] cuando se hable de par o impar en el contexto de la DFT será siempre en módulo N, donde N es la cantidad de puntos de la DFT.

Sin embargo, si analizamos la opción 2 veremos que ésta no es correcta. Esto se debe a que si no se respeta que el vector sea complejo conjugado simétrico entonces se están especificando amplitudes distintas en una misma frecuencia. Cuando se utiliza la transformada discreta de Fourier con señales reales se debe cumplir que $Y[k] = Y^{\star}[((-k))_N]$ donde $Y[k]$ es la transformada de Fourier de la señal real y $((k))_N$ significa k módulo N. En ese contexto se podría decir que la frecuencia $f_1 = \frac{2\pi k}{T_s}$ y la frecuencia $f_2 = \frac{2\pi(N-k)}{T_s}$ describen un mismo tono y al asignarle amplitudes distintas la especificación sería inconsistente. De todos modos, si realizáramos el método obtendríamos una señal real al descartar la parte imaginaria, entonces la pregunta a hacer es: ¿qué tranformada de Fourier $Y_{eff}[k]$ generaría dicha señal real?. Utilizando las propiedades de la DFT [28] es fácil ver que sería equivalente a haber usado $Y_{eff}[k] = \frac{1}{2}\left(Y[k] + Y^{\star}[((-k))_N]\right)$. Por lo tanto, no lograríamos controlar el espectro de la manera deseada. Notar que si nuestro vector es complejo conjugado simétrico utilizando la ecuación anterior llegamos a $Y_{eff}[k] = Y[k]$.

Por otro lado, los primeros tres métodos ($1a$, $1b$ y $1c$) son válidos con la excepción que para la opción $1a$ tengo que generar tanto la parte real como la imaginaria. Nosotros elegiremos generar sólo la parte real ya que en nuestro contexto de emulación de ruido no se pierde generalidad. Por lo tanto, especificaremos las amplitudes de cada valor de frecuencia en el espectro normalizado de 0 a π y luego construiremos el vector par módulo N.

El procedimiento para generar el vector real par módulo N es el siguiente

1. Especifico los $\frac{N_{FFT}}{2}+1$ valores de amplitudes y_k correspondientes a las frecuencias $f_k = \frac{f_s \cdot k}{N_{FFT}} \quad 0 \leqslant k \leqslant \frac{N_{FFT}}{2}$.
   Entonces tengo: $Y_{half}[k] = y_k, \quad 0 \leqslant k \leqslant \frac{N_{FFT}}{2}$
2. Construyo un vector par módulo N_{FFT} a partir de $Y_{half}[k]$

$$Y[k] = \begin{cases} Y_{half}[k] & 0 \leqslant k \leqslant \frac{N_{FFT}}{2} \\ Y_{half}[N_{FFT} - k] & \frac{N_{FFT}}{2} < k \leqslant N_{FFT} - 1 \end{cases} \tag{3.23}$$

Luego debemos ver como elegir los y_k de manera que generen el perfil de ruido que deseamos. Cada valor estará formado por una parte aleatoria, ya que queremos generar ruido aleatorio, y una parte determinística que se usa para establecer la potencia. Por ejemplo usaremos $y_k = \sigma_k \cdot n_k$ donde n_k es un número aleatorio de *varianza unitaria* y σ_k es la constante que determina la potencia. El control de ancho de banda por tono se realiza jugando con la correlación entre los n_k correspondientes al mismo subíndice k pero de distintas IFFTs. Intuitivamente podemos ver que si n_k fuese siempre constante su aporte a la antistransformada a lo largo del tiempo (varias IFFTs) sería un tono sinusoidal puro continuo. Con esta metodología mantenemos separado el proceso de generación de números aleatorios de la especificación del perfil de ruido.

El posicionamiento de las interferencias en el espectro queda determinado por la cantidad de puntos de la IFFT, N_{FFT}, y por el índice k del vector de amplitudes

$Y[k]$. La fórmula que determina las posibles posiciones es:

$$f_k = \frac{f_s \cdot k}{N_{FFT}} \qquad 0 \leqslant k \leqslant \frac{N_{FFT}}{2} \tag{3.24}$$

Si nuestro emulador utilizara una tasa de 80 Msps y 16384 puntos tendríamos una separación de 4,88 kHz. Esta cantidad de puntos está basada en querer tener la posibilidad de tener buen posicionamiento en frecuencia con respecto a los sistemas BPL actuales que tienen a lo sumo 6144 puntos [19] que representa un espaciamiento entre tonos de 13 kHz. En el caso de superposición de ruidos modulados el posicionamiento es de infinita precisión en un entorno de evaluación ideal. Aunque en la realidad es tan preciso como la resolución numérica del generador sinusoidal lo permita.

3.8.3. Resumen Comparativo

En la siguiente tabla podemos observar una comparativa sintetizada

Característica	**Método por Superposición**	**Método de IFFT**
Cantidad de ruidos de banda angosta	▪ depende de la cantidad de instancias de generación	▪ depende de la cantidad de puntos de la IFFT y es $N_{FFT}/2$ (no cuenta la frecuencia $f_0 = 0$)
Posicionamiento en frecuencia	▪ idealmente infinito ▪ depende de precisión del generador sinusoidal	▪ depende de la cantidad de puntos de la IFFT y de la frecuencia de muestreo
Cantidad de parámetros	▪ 3 por portadora (potencia, φ, frecuencia central)	▪ parámetros de potencia: $N_{FFT}/2 + 1$ ▪ parámetros de ancho de banda: $N_{FFT}/2 + 1$ ▪ NO depende de la cantidad de portadoras, depende de la cantidad de puntos de la IFFT
Recursos utilizados	▪ depende de la cantidad de interferencias a generar ▪ su escalabilidad es lineal	▪ sólo depende de la cantidad de puntos de la IFFT ▪ su escalabilidad con la cantidad de puntos es mejor que lineal pero mayor que $n \log n$

3.9. Modelo de ruido Impulsivo periódico sincrónico

Finalmente, el ruido periódico impulsivo sincrónico puede expresarse de la siguiente manera

$$n_{PI}(t) = n(t) \cdot r_P(t + \varphi) \tag{3.25}$$

donde $n(t)$ es el ruido coloreado que será modulado, $r_P(t + \varphi)$ la señal de pulsos rectangulares de período t_P, ancho de pulso t_B, y con una fase φ aleatoria uniforme $[0; t_P)$ tal como se define en la ecuación 3.9.

Valores típicos de t_P y t_B:

- t_P 10 ms o 20ms
- t_B algunos microsegundos

3.10. Modelo de ruido Impulsivo periódico asincrónico

El modelo de ruido impulsivo periódico asincrónico puede expresarse como

$$n_{PI_{asinc}}(t) = n_b(t) \cdot r_P(t + \varphi) \tag{3.26}$$

donde $n_b(t)$ es ruido de banda angosta modulado a una frecuencia f_c y $r_P(t + \varphi)$ la señal de pulsos rectangulares de período t_P, ancho de pulso t_B, y con una fase φ aleatoria uniforme $[0; t_P)$ tal como se define en la ecuación 3.9.

Valores típicos de t_P y t_B:

- t_P entre 0,5 μs y 20 μs
- t_B menor t_P, depende de *duty cycle* de la fuente de alimentación que a su vez depende de la carga

3.11. Modelo de ruido Impulsivo (aperiódico) asincrónico

El método de análisis espectral de los impulsos que se utilizó para caracterizarlos en frecuencia, se puede utilizar de manera inversa para sintetizar impulsos [1, 49]. Entonces, determinaremos el contenido de un impulso utilizando un ruido coloreado cuya densidad espectral de potencia sea considerablemente superior que la del ruido de fondo. La síntesis de dicho ruido sólo será dirigida hacia la salida sólo durante los tiempos de ocurrencia de impulsos. Dicha modulación temporal del ruido puede ser lograda a través de una envolvente temporal. Este modelo simplificado ha probado tener un impacto similar al de un modelo mucho más complejo, por ejemplo modelos que consideran que cada fuente de ruido tiene su propia función de transferencia [9, 10]. Entonces a continuación procederemos a describir el algoritmo de generación. Dicho método es muy similar al que se planteó para generar impulsos periódicos en la sección 3.9. El modelo se ilustra en la figura 3.18 que es similar al de la figura 3.7.

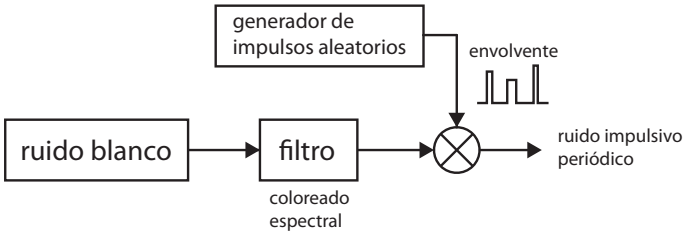


FIGURA 3.18: Modelo del ruido impulsivo (aperiódico) asincrónico

Como en este caso los impulsos son aleatorios debemos tener un modelo estocástico que describa los valores característicos. Estos son:

- t_{w_i} ancho del impulso i

- A_i amplitud de la envolvente del impulso i
- t_{A_i} tiempo entre impulsos, que es el tiempo entre el fin del impulso i y el comienzo del impulso $i+1$.

y se observan en la figura 3.19.

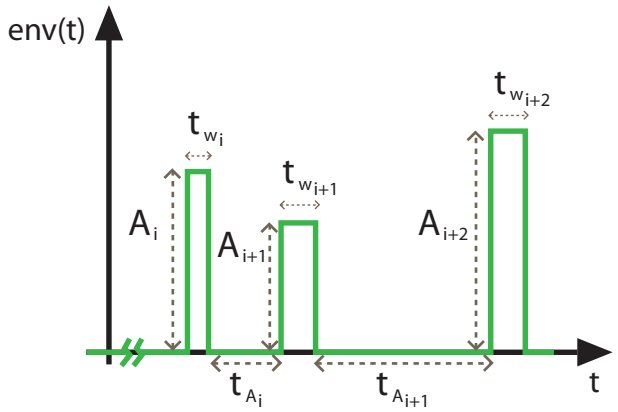


FIGURA 3.19: Envolvente del ruido y sus parámetros

Las propiedades *temporales* estarán determinadas por una cadena de Markov, en particular una cadena de Markov *particionada* [18]. Las propiedades de amplitud pueden ser descriptas con un buen nivel de aproximación por una distribución exponencial [1, 12]. Para ver una breve referencia teórica sobre cadenas de Markov referirse al apéndice D.2.

La cadena particionada de Markov que utilizaremos tiene los n estados z_i particionados en dos grupos. Tenemos el grupo $\mathcal{U} = \{z_1, z_2, \ldots, z_\nu\}$ y el grupo $\mathcal{G} = \{z_{\nu+1}, z_{\nu+2}, \ldots, z_n\}$ y la función de salida $\Phi(k)$

$$\Phi(k) = \Phi(z(k)) = \begin{cases} 0 & z(k) \in \mathcal{U} \\ 1 & z(k) \in \mathcal{G} \end{cases} \tag{3.27}$$

los ν estados de $\mathcal{U}$ representan el caso donde no hay impulso y los $w = n - \nu$ estados de $\mathcal{G}$ representan la ocurrencia de un impulso. En el modelo de Fritchman [18], las transiciones sólo pueden ocurrir de un estado de un grupo a otro estado del otro grupo o al estado en sí mismo. Esta característica resulta en la siguiente estructura para matriz de probabilidades de transición.

$$\mathbf{P} = \begin{bmatrix} \mathbf{P_U} & \mathbf{P_{UG}} \\ \mathbf{P_{GU}} & \mathbf{P_G} \end{bmatrix} = \left[\begin{array}{cccccc} p_{1,1} & & 0 & p_{1,\nu+1} & \cdots & p_{1,n} \\ & \ddots & & \vdots & & \vdots \\ 0 & & p_{\nu,\nu} & p_{\nu,\nu+1} & \cdots & p_{\nu,n} \\ \hline p_{\nu+1,1} & \cdots & p_{\nu+1,\nu} & p_{\nu+1,\nu+1} & & 0 \\ \vdots & & \vdots & & \ddots & \\ p_{n,1} & \cdots & p_{n,\nu} & 0 & & p_{n,n} \end{array}\right] \tag{3.28}$$

Las matrices $\mathbf{P_U}$ y $\mathbf{P_G}$ establecen las probabilidades de permanecer en un estado no perturbado o perturbado respectivamente, mientras que $\mathbf{P_{UG}}$ y $\mathbf{P_{GU}}$ dan las probabilidades de transición de un estado no perturbado hacia uno pertubado y viceversa. La ventaja del modelo particionado es que las matrices $\mathbf{P_U}$ y $\mathbf{P_G}$ son matrices diagonales.

Sin embargo, la extracción de parámetros en base a mediciones y el correspondiente diseño de escenarios de ruido de referencia sigue siendo un problema difícil de manejar. Para simplificar se recurre a la inclusión de dos nuevos estados de transición, sobre los cuales ocurren las transiciones de estados no perturbados hacia estados perturbados y viceversa. Entonces este modelo puede ser descripto por las matrices $\mathbf{U}$ y $\mathbf{G}$, siendo $\mathbf{U}$ la matriz asociada a estados libres de impulsos y $\mathbf{G}$ a la matriz asociada a estados de impulsos.

$$\mathbf{U} = \begin{bmatrix} u_{1,1} & 0 & \dots & 0 & u_{1,\nu+1} \\ 0 & u_{2,2} & \ddots & \vdots & u_{2,\nu+1} \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & u_{\nu,\nu} & u_{\nu,\nu+1} \\ u_{\nu+1,1} & u_{\nu+1,2} & \cdots & u_{\nu+1,\nu} & 0 \end{bmatrix} \tag{3.29}$$

y

$$\mathbf{G} = \begin{bmatrix} g_{1,1} & 0 & \dots & 0 & g_{1,w+1} \\ 0 & g_{2,2} & \ddots & \vdots & g_{2,w+1} \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & g_{w,w} & g_{w,w+1} \\ g_{w+1,1} & g_{w+1,2} & \cdots & g_{w+1,w} & 0 \end{bmatrix} \tag{3.30}$$

Debido a la inserción de estos estados de transición dichas matrices tienen un orden mayor que $\mathbf{P_U}$ y $\mathbf{P_G}$. Pero como no se puede permanecer en el estado de transición la probabilidad del último elemento diagonal es cero en ambas matrices. El modelo simplificado se puede ilustrar en la siguiente figura.

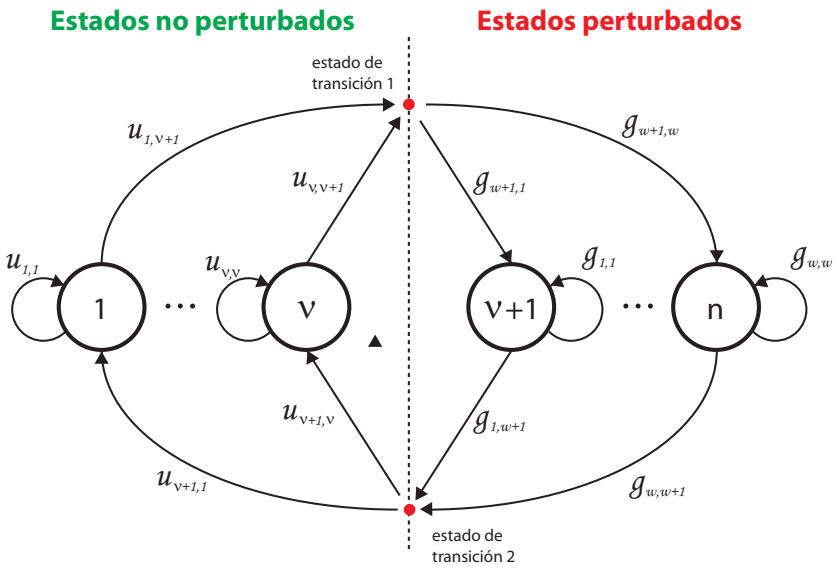


FIGURA 3.20: Modelo de Markov particionado

Esta simplificación presentada es útil pero para el análisis del escenario de ruido y la implementación del modelo de ruido en un simulador es necesario convertir este modelo (ecuaciones 3.29 y 3.30) al de la ecuación 3.28. Para ello se utilizan las siguientes relaciones

$$\mathbf{P_U} = \begin{bmatrix} u_{1,1} & & 0 \\ & \ddots & \\ 0 & & u_{\nu,\nu} \end{bmatrix} \tag{3.31}$$

$$\mathbf{P_G} = \begin{bmatrix} g_{1,1} & & 0 \\ \vdots & \ddots & \\ 0 & & g_{w,w} \end{bmatrix} \tag{3.32}$$

$$\mathbf{P_{UG}} = \begin{bmatrix} u_{1,\nu+1} \\ \vdots \\ u_{\nu,\nu+1} \end{bmatrix} \begin{bmatrix} g_{w+1,1} & \cdots & g_{w+1,w} \end{bmatrix} = \begin{bmatrix} u_{1,\nu+1} \cdot g_{w+1,1} & \cdots & u_{1,\nu+1} \cdot g_{w+1,w} \\ \vdots & & \vdots \\ u_{\nu,\nu+1} \cdot g_{w+1,1} & \cdots & u_{\nu,\nu+1} \cdot g_{w+1,w} \end{bmatrix} \tag{3.33}$$

$$\mathbf{P_{GU}} = \begin{bmatrix} g_{1,w+1} \\ \vdots \\ u_{w,w+1} \end{bmatrix} \begin{bmatrix} u_{\nu+1,1} & \cdots & u_{\nu+1,\nu} \end{bmatrix} = \begin{bmatrix} g_{1,w+1} \cdot u_{\nu+1,1} & \cdots & g_{1,w+1} \cdot u_{\nu+1,\nu} \\ \vdots & & \vdots \\ g_{w,w+1} \cdot u_{\nu+1,1} & \cdots & g_{w,w+1} \cdot u_{\nu+1,\nu} \end{bmatrix} \tag{3.34}$$

Para ilustrar el modelo, podemos suponer que los ν estados no perturbados se corresponden a la suma ponderada de ν distribuciones exponenciales de tiempos entre

impulsos (ecuación (3.57)) y que los w estados perturbados corresponden a la suma ponderada de w distribuciones exponenciales de tiempos de duración de impulsos (ecuación 3.56). Estudios realizados en [47] muestran que se puede reproducir bien el escenario de ruido con cuatro estados no perturbados y dos estados perturbados. La introducción del modelo particionado nos lleva a una representación más prolija y acotada dado que la cantidad posible de transiciones queda notablemente reducida. Esta visualización del modelo es un prerrequisito para poder diseñar escenarios de ruido con múltiples estados de impulso ya que con este enfoque el modelo queda descripto por las matrices **U** y **G** que pueden ser tratadas por separado. Además, esta estructura permite calcular de manera simple las probabilidades estacionarias de los estados, como muestra el apéndice D.4.

3.11.1. Interpretación de las probabilidades de transición

Como vimos en D.2, la cadena de Markov evoluciona en momentos discretos. En este caso dichos momentos discretos son múltiplo del tiempo de muestreo t_s. Por lo tanto, la interpretación básica de $p_{i,i}$ es la probabilidad de permanecer en el estado i dado que estábamos anteriormente en el estado i. En la realidad, el tiempo es continuo pero nosotros lo discretizamos en intervalos de duración t_s, en los cuales puede o no haber un evento de impulso. Por esta razón, se toma como referencia del modelo continuo al proceso Poisson. En éste se tiene un tasa de eventos por unidad de tiempo λ_i. Por lo tanto, la probabilidad de que un intervalo de tiempo t_s este libre de eventos es la probabilidad de que el evento ocurra fuera del intervado, es decir

$$P\left(\begin{array}{c}\text{intervalo libre}\\ \text{de eventos}\end{array}\right) = P\left(T_i > t_s\right) = e^{-\lambda_i t_s} \triangleq p_{i,i} \tag{3.35}$$

Como el proceso carece de memoria, este razonamiento es válido para cualquier iteración, ya que no importa si en el anterior no hubo evento.

$$P\left(T_i + (k+1)t_s > t_s | T_i > kt_s\right) = P\left(T_i > t_s\right) \tag{3.36}$$

Hay que tener en cuenta que el proceso Poisson empieza cuando se entra en el estado i proveniente de un estado del grupo opuesto. A su vez, hay tantas tasas de eventos λ_i como estados tiene el grupo correspondiente.

Entonces en base a este conocimiento podremos obtener valores estadísticos de interés.

- Duración promedio del impulso asociado al estado i
- Tiempo entre impulsos asociado al estado i
- Probabilidades estacionarias
- Tasa de impulsos

Vale la pena recalcar que como trabajamos en tiempo discreto las probabilidades dependen del período de muestreo t_s. Es fácil ver dicha dependencia en (3.35). Las

implicancias de este hecho son importantes ya que convertir las probabilidades para un tiempo de muestreo a otro distinto presenta distintas opciones que veremos más adelante.

3.11.1.1. Duración promedio del impulso asociado al estado i

En tiempo discreto, este proceso se convierte en una serie de experimentos. El experimento tiene dos resultados posibles

1. permanezco en el estado actual
2. salto a un estado distinto del actual

En cada experimento tengo una probabilidad $1 - p$ de permanecer en el mismo estado y una probabilidad p de saltar hacia otro estado. La cantidad de experimentos necesarios para saltar hacia un estado distinto conforma una variable aleatoria de distribución Geométrica descripta por la función de probabilidad

$$P(X = x) = (1 - p)^{x-1} p \qquad x = 0, 1, \cdots \tag{3.37}$$

En nuestro caso, dada las definiciones previas $1 - p = p_{i,i}$ y $p = 1 - p_{i,i}$. Por lo que nuestra variable geométrica queda

$$P(X = x) = p_{i,i}^{x-1}(1 - p_{i,i}) \qquad x = 0, 1, \cdots \tag{3.38}$$

cuya media estadística es

$$E[X] = \sum_{x=1}^{\infty} x \cdot p_{i,i}^{x-1}(1 - p_{i,i}) = \frac{1}{1 - p_{i,i}} \tag{3.39}$$

y representa la cantidad media de intervalos de tiempo de duración t_s que tendremos que esperar para ver un cambio de estado. Como el estado inicial está asociado al impulso, dicho tiempo es la duración media del impulso expresada en intervalos de tiempo. Si multiplicamos dicha cantidad por t_s, obtenemos dicho valor expresado en segundos.

$$\boxed{\bar{t_i} = \frac{t_s}{1 - p_{i,i}}} \tag{3.40}$$

También se suele usar otra medida llamada $t_{50\,\%,i}$ o período mínimo de retención que ocurre con probabilidad 0,5. En estadística se conoce a dicha métrica como la mediana. Es decir, el 50 % de las veces la duración del impulso es al menos $t_{50\,\%,i}$ en el estado i.

$$P(T > t_{50\,\%,i}) = \frac{1}{2} = e^{-\lambda_i t_{50\,\%,i}} \Longrightarrow \ln \frac{1}{2} = -\lambda_i t_{50\,\%,i} \tag{3.41}$$

pero según la ecuación (3.35)

$$\ln p_{i,i} = -\lambda_i t_s \tag{3.42}$$

entonces

$$t_{50\,\%,i} = t_s \frac{\ln \frac{1}{2}}{\ln p_{i,i}} \tag{3.43}$$

En tiempo discreto sería

$$t_{50\,\%} = t_s \tilde{X} \qquad \tilde{X} \in \mathbb{R} / \sum_{k=\lfloor \tilde{X} \rfloor}^{\infty} p_{i,i}^k (1 - p_{i,i}) \geqslant \frac{1}{2}, \quad \sum_{k=1}^{\lceil \tilde{X} \rceil} p_{i,i}^k (1 - p_{i,i}) \geqslant \frac{1}{2} \tag{3.44}$$

[5] sin embargo, por practicidad tomaremos $\tilde{X} \in \mathbb{Z}$. Es fácil probar que el método continuo sirve para obtener $t_{50\,\%}$ por equivalencia probabilística de sucesos. De esta manera, es más tratable matemáticamente hablando.

3.11.1.2. Tiempo entre impulsos asociado al estado i

Las ecuaciones (3.40) y (3.43) pueden ser aplicados a estados no perturbados.

3.11.1.3. Probabilidades estacionarias

Las probabilidades estacionarias de una cadena de Markov nos dan las probabilidades que tiene cada estado en régimen estacionario. Estos datos son importantes ya que podemos obtener la duración relativa de estados de impulso directamente de dichos valores. En una cadena irreducible y aperiódica[6] el vector de probabilidades $\mathbf{p} = [\mathbf{p}_1 \cdots \mathbf{p}_n]$ al cual convergen las probabilidades de transición es independiente de la condición inicial. Este vector se puede hallar resolviendo el siguiente sistema de ecuaciones

$$\mathbf{p} = \mathbf{p} \cdot \mathbf{P} \qquad \text{s. t.} \qquad \sum_j \mathbf{p}_j = 1 \tag{3.45}$$

La duración relativa de la perturbación es igual a

$$\text{dur. rel. pert.} = \frac{\text{tiempo de perturbación}}{\text{tiempo total}} \tag{3.46}$$

que es fácil de calcular utilizando las probabilidades estacionarias.

$$\text{dur. rel. pert.} = \frac{P(\text{estar en un estado perturbado})}{\underbrace{P(\text{estar en un estado perturbado o no perturbado})}_{1}} \tag{3.47}$$

---

[5] $\lceil y \rceil = \min_{x \in \mathbb{Z}} x \geqslant y$ y $\lfloor y \rfloor = \max_{x \in \mathbb{Z}} x \leqslant y$

[6] irreducible (desde cualquier estado es posible llegar a cualquier estado en un algún tiempo) y aperiódica (las transiciones de los estados deben poder ocurrir en cualquier momento y no en períodos regulares mayores a una iteración)

entonces queda

$$\text{dur. rel. pert.} = \sum_{j=\nu+1}^{n} \mathbf{p}_j \tag{3.48}$$

ya que la probabilidad de la suma de sucesos disjuntos es igual a la suma de probabilidades. (recordar que los estados $\nu + 1, \ldots, n$ pertenecían al grupo $\mathcal{G}$ que era el grupo de estados perturbados).

3.11.1.4. Tasa de impulsos

Una vez conocidas las probabilidades estacionarias de los estados y la duración media de cada uno podemos determinar la tasa media de impulsos R_{imp} mediante la siguiente expresión.

$$\mathsf{R}_{imp} = \sum_{i=1}^{\nu} \frac{\mathbf{p}_i}{\bar{t}_i} \tag{3.49}$$

La demostración de dicha ecuación se encuentra en el apéndice D.3.

3.11.1.5. Conversión de probabilidades a otros tiempos de muestreo

Hay dos posibles métodos para convertir las probabilidades de transición correspondientes a un tiempo de muestreo t_s a otro tiempo de muestreo t'_s.

El primer método tiene en cuenta de donde sale la probabilidad $p_{i,i}$ es decir, del proceso Poisson y en particular de la ecuación (3.35). Por lo tanto

$$p'_{i,i} = e^{-\lambda_i t'_s} = p_{i,i}^{\frac{t'_s}{t_s}} \tag{3.50}$$

Como la métrica $t_{50\,\%,i}$, ecuación (3.43), era la mediana de la distribución exponencial negativa y no se ha alterado dicha distribución, entonces conservará su valor. Esto se puede ver a continuación

$$t'_{50\,\%,i} \triangleq t'_s \frac{\ln \frac{1}{2}}{\ln p'_{i,i}} = \cancel{t'_s} \frac{\ln \frac{1}{2}}{\frac{\cancel{t'_s}}{t_s} \ln p_{i,i}} = t_s \frac{\ln \frac{1}{2}}{\ln p_{i,i}} \triangleq t_{50\,\%,i}$$

Sin embargo, para que la nueva matriz sea una matriz de probabilidades de transición debe verificarse la ecuación (D.12). Entonces las demás probabilidades deben ser

$$p'_{i,j} = p_{i,j} \frac{1 - p'_{i,i}}{1 - p_{i,i}} \quad i \neq j \tag{3.51}$$

Es fácil ver que esta conversión de tiempos de muestreo no conserva la duración promedio del impulso asociada al estado i ($\bar{t}_i$) ni la distribución de probabilidad estacionaria.

El segundo método tiene en cuenta la conservación de $\bar{t}_i$, ecuación 3.40. Por lo tanto

$$p'_{i,i} = 1 - \frac{t'_s}{t_s}(1 - p_{i,i}) \tag{3.52}$$

y entonces

$$\bar{t}'_i \triangleq \frac{t'_s}{1-p'_{i,i}} = \frac{\cancel{t'_s}}{\cancel{1-1} + \frac{\cancel{t'_s}}{t_s}(1-p_{i,i})} = \frac{t_s}{1-p_{i,i}} \triangleq \bar{t}_i$$

Además, si consideramos el cálculo de las probabilidades estacionarias (apéndice D.4) tenemos que las variables auxiliares se conservan

$$K'_{U,i} = \frac{u'_{\nu+1,i}}{1-u'_{i,i}} = \frac{\frac{\cancel{t'_s}}{\cancel{t_s}}u_{\nu+1,i}}{\frac{\cancel{t'_s}}{\cancel{t_s}}(1-u_{i,i})} = \frac{u_{\nu+1,i}}{1-u_{i,i}} \triangleq K_{U,i}, \quad i = 1, \cdots, \nu, \tag{3.53}$$

$$K'_{G,i} = \frac{g'_{w+1,i}}{1-g'_{i,i}} = \frac{\frac{\cancel{t'_s}}{\cancel{t_s}}g_{w+1,i}}{\frac{\cancel{t'_s}}{\cancel{t_s}}(1-g_{i,i})} = \frac{g_{w+1,i}}{1-g_{i,i}} \triangleq K_{G,i}, \quad i = 1, \cdots, w \tag{3.54}$$

y por lo tanto se conserva la distribución de probabilidades estacionarias definida en la ecuación D.26. Esto último implica que la duración relativa de la perturbación también se conserva (ecuación (3.48)).

Para conocer cuál era la diferencia entre ambos métodos en la práctica decidimos aplicarlos a matrices con valores propuestos en [3]. En lugar de realizar simulaciones de los procesos de Markov decidimos analizar la matriz calculando las probabilidades estacionarias. También evaluamos los valores $\bar{t}_i$, la duración relativa de la perturbación y la tasa de impulsos. Inicialmente nos sorprendieron los resultados porque había una diferencia muy pequeña entre ambos métodos. Probamos con diversas tasas de conversión de tiempos de muestreo y los resultados seguían siendo muy similares. Por ejemplo, para la duplicación de frecuencia de muestreo las diferencias en la tasa de impulsos eran de 0,01 % . En los $\bar{t}_i$ las diferencias fueron aún menores para aquellos estados con probabilidades muy cercanas a 1 y el peor caso 4,5 % se dió para un $p_{i,i} \approxeq 0{,}828$. Entonces decidimos encontrar la explicación a este comportamiento para ver si podíamos definir si algún método era preferible a otro para alguna situación en particular. Para ello se desarrolló la ecuación (3.50) en una serie de Taylor y se llegó a la conclusión que para $p_{i,i} \longrightarrow 1$ la aplicación de una u otra es indistinta (ver apéndice §C.1). Por este motivo, los cambios de tiempos de muestreo aplicados a nuestras matrices daban diferencias muy pequeñas y la aplicación de los métodos sólo difiere en la duración de los estados cuya $p_{i,i}$ no es tan cercana a 1, que suelen ser estados perturbados cuya probabilidad estacionaria es muy baja.

La única consideración a tener en cuenta es que en el segundo método, ecuación (3.52), para valores extremos es posible que la probabilidad de menor que 0. Con valores extremos nos referimos a $p_{i,i}$ pequeños ($< 0{,}6$) y cocientes $\frac{t'_s}{t_s}$ considerables (> 2). Vale la pena aclarar que $p_{i,i}$ pequeños no tienen sentido en el contexto de nuestro estudio porque representan eventos de muy baja probabilidad y muy corta duración, aproximadamente $2t_s$. Por lo tanto, el impacto en la señal transmitida es despreciable y además la inclusión de este estado en nuestro modelo de Markov sería un desperdicio de recursos.

3.11.2. Estimación de $\mathbf{U}$ *y* $\mathbf{G}$ *a partir de mediciones*

Una manera de determinar los valores de los elementos de las matrices $\mathbf{U}$ y $\mathbf{G}$ es a partir de las mediciones según se propone en [49]. Para ello deberemos realizar estadísticas sobre el ancho de los impulsos (t_w) y sobre el tiempo entre impulsos (t_A). Con esos datos podremos estimar la función de distribución complementaria para el ancho de los impulsos como para el tiempo entre impulsos.

La función de distribución complementaria (FDC) denota la probabilidad de que una variable aleatoria X exceda el valor x.

$$F_C(x) = P(X > x) \tag{3.55}$$

Aplicando este concepto al estudio del ancho de los impulsos tenemos que la probabilidad de que un impulso exceda los $k \cdot t_s$ segundos está dada por:

$$F_{C_w}(k) = \begin{cases} 1 & k = 0 \\ \sum_{j=1}^{w} g_{w+1,j} \cdot g_{j,j}^k & k = 1, 2, \cdots \end{cases} \tag{3.56}$$

y aplicando el mismo razonamiento al tiempo entre impulsos tenemos que la probabilidad de que el tiempo entre impulsos exceda los $k \cdot t_s$ segundos es:

$$F_{C_A}(k) = \begin{cases} 1 & k = 0 \\ \sum_{j=1}^{\nu} u_{\nu+1,j} \cdot u_{j,j}^k & k = 1, 2, \cdots \end{cases} \tag{3.57}$$

es decir que ambas FDCs pueden ser expresadas como elementos de las matrices $\mathbf{U}$ y $\mathbf{G}$. Se puede observar como ambas funciones de distribución complementarias consisten en sumas de exponenciales ponderadas. Por lo tanto, los elementos de las matrices $\mathbf{U}$ y $\mathbf{G}$ pueden ser estimados a partir de mediciones de las distribuciones de ancho de impulsos y de tiempo entre impulsos.

El algoritmo utilizado es el de ajuste a una curva y consiste en los siguientes pasos:

- proceso las mediciones (señales temporales) y obtengo las estimaciones $\hat{F}_{C_w}(k)$ y $\hat{F}_{C_A}(k)$, que serán las curvas a ajustar
- para cada una de ellas ejecuto el alguna variante del algoritmo de Nelder Mead Simplex [27] utilizando la función $\Gamma(k) = \sum_{i=1}^{m} a_i \cdot b_i^k$
- Si usé $\hat{F}_{C_w}(k)$ entonces podré obtener estimaciones de los elementos de la matriz $\mathbf{G}$. Por otro lado, si usé $\hat{F}_{C_A}(k)$ las estimaciones serán los elementos de la matriz $\mathbf{U}$.
- una vez que tengo validada $\hat{\mathbf{U}}$ y $\hat{\mathbf{G}}$, es decir que cumple con los requisitos para ser una matriz de probabilidades, usaré las ecuaciones (3.33) y (3.34) para construir la matriz de probabilidades de transición (3.28).

3.12. Modelo de ruido Impulsivo en ráfagas

Este tipo de ruido aparece enmarcado dentro del ruido aperiódico asincrónico. Los efectos del mismo son críticos para los sistemas de comunicaciones ya que la tasa de impulsos y la duración relativa del ruido aumenta significativamente durante el transcurso del mismo. Por estos motivos, es necesario poder reproducir escenarios donde se presenten ráfagas, característica que el modelo propuesto anteriormente no podía reproducir. Por lo tanto, la generación del ruido en ráfagas se planteará por separado.

Para el modelado del ruido de ráfaga se utiliza un modelo jerárquico de dos niveles. El nivel superior describe la ocurrencia de intervalos de ráfaga. Un intervalo de ráfaga contiene varios impulsos de ruido consecutivos. El nivel inferior determina la ocurrencia de los impulsos individuales dentro del intervalo de ráfaga. Por lo tanto, el nivel superior define la envolvente para los intervalos de ráfagas, mientras que el nivel inferior define la envolvente para los impulsos individuales dentro del intervalo de ráfaga. La señal temporal se genera a partir de ruido blanco filtrado como en los modelos anteriores.

La figura 3.21 muestra los parámetros característicos del modelo de nivel superior. Para poder describir esta envolvente necesitamos tres parámetros: la amplitud $A_{B,i}$,

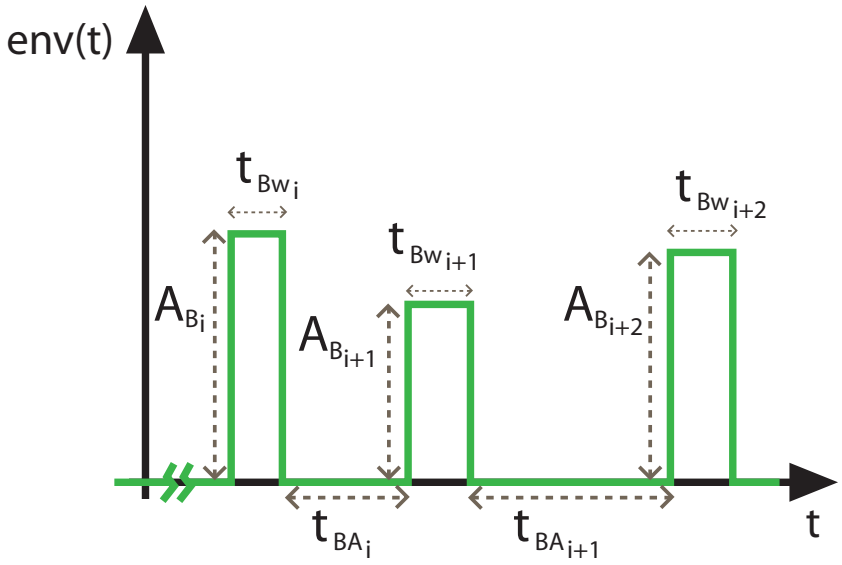


FIGURA 3.21: Parámetros del modelo de nivel superior

la duración del intervalo de ráfaga $t_{Bw,i}$ y el tiempo entre intervalos de ráfaga $t_{BA,i}$. Un intervalo de ráfaga está definido por una cantidad mínima (ej: 3) de impulsos consecutivos que no deben espaciarse más allá que un tiempo máximo (ej: 4 ms). Las posiciones temporales de los impulsos individuales quedan definidas por el modelo de nivel inferior, cuyos parámetros se ven en la figura 3.22. Aquí se puede apreciar el estado del modelo de ruido de ráfaga en una franja de colores debajo de las formas

de onda de las envolventes. El nivel inferior se encarga de refinar detalles del modelo

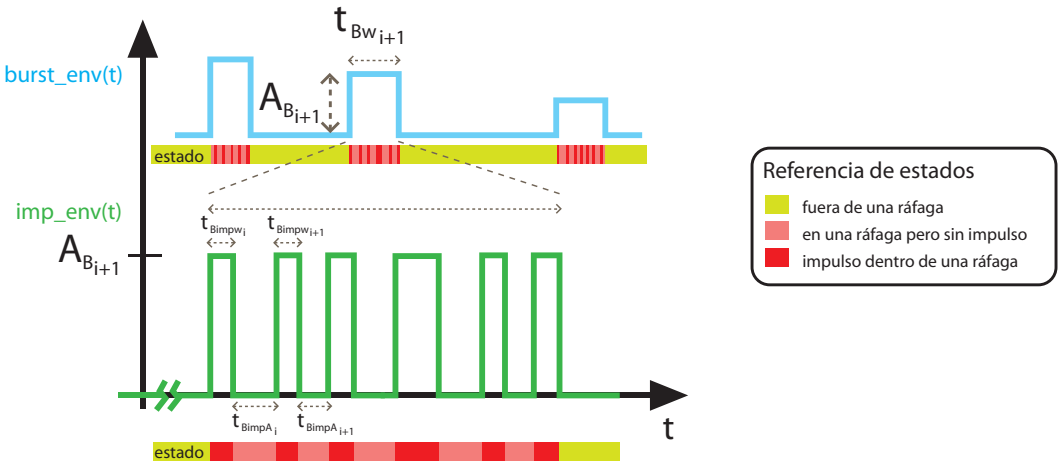


FIGURA 3.22: Parámetros del modelo de nivel inferior y estados del modelo

superior. Para simplificar el modelo se supone que dentro de un intervalo de ráfaga la amplitud A_B es constante. Esta suposición es razonable ya que los impulsos de una misma ráfaga suelen estar causados por la misma fuente. El agregar la capacidad de modelar distintas amplitudes complica innecesariamente el modelo. El impacto del ruido de ráfaga es tal que se asume que la funcionalidad del sistema de comunicaciones queda completamente perturbada durante este tiempo. Los impulsos rectangulares individuales quedan caracterizados por los siguientes tres parámetros: la amplitud constante $A_{B,i}$, el ancho del pulso $t_{Bimpw,i}$ y el tiempo entre impulsos de la misma ráfaga $t_{BimpA,i}$. Un modelo de baja complejidad sería considerar que dentro de una ráfaga los impulsos tienen todos el mismo ancho y la misma separación. Sin embargo, este escenario no es apropiado para una emulación realista del ruido. Es posible que algunos sistemas puedan ser optimizados para la periodicidad de estos impulsos y den buenos resultados en el laboratorio pero tengan mala performance en la realidad. Por ello, necesitamos implementar un modelo estocástico que describa los parámetros anteriormente mencionados para los impulsos dentro de una ráfaga.

Como dijimos antes, el ruido de ráfaga es un caso particular del ruido asincrónico y aperiódico. Sin embargo, el modelo particionado de Markov que aplicaremos aquí puede simplificarse desestimando los múltiples estados perturbados y no perturbados. Entonces, en correspondencia con el modelo jerárquico utilizaremos dos cadenas de Markov independientes como se muestra en la figura 3.23. En esta figura se ilustra el estado del modelo con tres colores: verde (sin ráfaga), rosa (en una ráfaga pero sin impulso), y rojo (en un impulso). Esta referencia de estados se utiliza también en la figura 3.22.

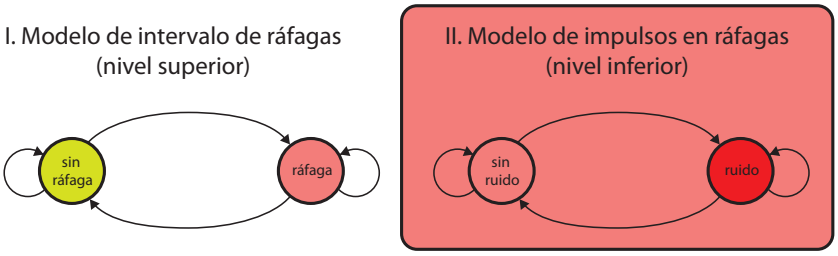


FIGURA 3.23: Modelo de Markov para el ruido de ráfaga

El modelo de nivel superior puede tener una resolución temporal más gruesa ya que modela grupos de impulsos. Este hecho podría aprovecharse para ahorrar recursos si la emulación se realiza con procesadores ya que se trabajaría con un tiempo de muestreo mayor a t_s. Por otro lado, el modelo de nivel inferior debe poseer la máxima resolución temporal para determinar la posición y duración de los impulsos individuales dentro de la ráfaga. Cuando el modelo de nivel superior se encuentra en el estado "sin ráfaga" el estado de la otra cadena de Markov es irrelevante.

3.12.1. Descripción paramétrica del modelo de ráfaga

El modelo de Markov para el ruido de ráfaga puede describirse utilizando pocos parámetros: la matriz de probabilidad de transiciones para el nivel superior e inferior. En ambas cadenas hay dos estados, uno que corresponde al estado no perturbado y otro al perturbado. Esta cantidad de estados es suficiente y simplifica la representación. Entonces, para el modelo de nivel superior la matriz de probabilidades de transición es

$$\mathbf{P_I} = \begin{bmatrix} p_{I,11} & p_{I,12} \\ p_{I,21} & p_{I,22} \end{bmatrix} \tag{3.58}$$

para el modelo de nivel inferior es

$$\mathbf{P_{II}} = \begin{bmatrix} p_{II,11} & p_{II,12} \\ p_{II,21} & p_{II,22} \end{bmatrix} \tag{3.59}$$

En ambas cadenas el primer estado es el no perturbado y el segundo el perturbado. Junto con las matrices de probabilidad de transición tenemos las probabilidades en estado estacionario

$$\mathbf{p_I} = \begin{bmatrix} \mathbf{p_{I,1}} & \mathbf{p_{I,2}} \end{bmatrix} \tag{3.60}$$

y

$$\mathbf{p_{II}} = \begin{bmatrix} \mathbf{p_{II,1}} & \mathbf{p_{II,2}} \end{bmatrix} \tag{3.61}$$

Entonces, de acuerdo a las definiciones anteriores $\mathbf{p_{I,2}}$ determina la duración relativa de los intervalos de ráfaga y $\mathbf{p_{II,2}}$ la duración relativa de la perturbación en una ráfaga. Es de esperar que $\mathbf{p_{I,1}} \gg \mathbf{p_{I,2}}$ ya que la mayoría del tiempo no hay ráfagas.

Por otro lado, la duración relativa total de la perturbación es el producto de ambas probabilidades estacionarias

$$\mathbf{p}_G = \mathbf{p}_{\mathbf{I},2}\mathbf{p}_{\mathbf{II},2} \tag{3.62}$$

Además, podemos conocer la tasa de ráfagas mediante la ecuación

$$\mathsf{R}_{imp_I} = \frac{\mathbf{p}_{\mathbf{I},1}}{\bar{t}_{I,1}} \simeq \frac{1}{\bar{t}_{I,1}} = \frac{1 - p_{I,11}}{t_s} \tag{3.63}$$

así como podemos conocer la tasa de impulsos dentro de una ráfaga utilizando la fórmula obtenida en la sección D.3

$$\mathsf{R}_{imp_{II}} = \frac{\mathbf{p}_{\mathbf{II},1}}{\bar{t}_{II,1}} \tag{3.64}$$

Por otro lado, podemos conocer la tasa total de impulsos R_{imp_B} causados por ráfagas multiplicando la tasa de impulsos durante una ráfaga $\mathsf{R}_{imp_{II}}$ por la probabilidad de ocurrencia de una ráfaga $\mathbf{p}_{\mathbf{I},2}$.

$$\mathsf{R}_{imp_B} = \mathsf{R}_{imp_{II}}\mathbf{p}_{\mathbf{I},2} \tag{3.65}$$

3.12.2. Consideraciones de diseño de la matriz de probabilidades de transición

En la sección §3.11.1 expusimos cómo obtener valores estadísticos que nos dan una idea del escenario de ruido. Recapitulando, estos eran:

1. duración relativa de la perturbación: cantidad entre 0 y 1 que indica la proporción de tiempo que se encuentra en un estado perturbado. Se calcula según la ecuación (3.48)

2. tasa de impulsos/ráfagas: es la cantidad media de eventos por unidad de tiempo que se espera ver. Se habla de impulsos en el modelo de ruido asincrónico aperiódico y se habla de ráfagas en la descripción del modelo de nivel superior, que no es otra cosa que un caso particular del primero desde el punto de vista estrictamente matemático. Se calcula según las ecuación (D.22) o (3.63) según corresponda.

3. tiempo medio de duración de un impulso asociado a un estado i. Ver ecuación (3.40).

4. tiempo medio entre arribo de impulsos asociado a un estado i. Se utiliza la misma ecuación que en caso anterior, excepto que se calculan usando las probabilidades asociadas a estados no perturbados.

La principal dificultad que presenta el problema de diseñar una matriz de probabilidad de transiciones es que si me baso en los valores anteriormente descriptos, tengo que tener en cuenta que no son independientes. Es decir, la imposición de uno afecta

al resto. Por ejemplo, la duración relativa de la perturbación aumenta si se aumenta el ancho promedio del pulso manteniendo la misma tasa de impulsos ó si se aumenta la distancia entre impulsos disminuye la tasa de impulsos y la duración relativa de la perturbación.

Se podría hablar de dos enfoques para generar las matrices para la simulación.

- Enfoque basado en mediciones (Curve Fitting Methods), ver §3.11.2.
- Enfoque basado en el conocimiento a priori de parámetros característicos del ruido impulsivo.

Para el segundo enfoque, el principal parámetro de diseño es dar la probabilidad estacionaria $\mathbf{p}$ de la distribución de estados. Ésta define la duración relativa de la perturbación, que influencia considerablemente en la potencia del ruido. Además, la duración promedio de los estados también es un parámetro importante ya que junto con la probabilidad estacionaria determina la tasa de impulsos. El grado de complejidad del problema está directamente asociado a la cantidad de estados de nuestro modelo. El análisis de mediciones [47] ha mostrado que es posible una buena reproducción del escenario de ruido impulsivo aperiódico asincrónico utilizando cuatro estados no perturbados y dos estados perturbados. Por lo tanto, no tiene sentido utilizar seis o más estados ya que algunos de estos se utilizarían con una probabilidad extremadamente baja. Para el caso del ruido impulsivo en ráfagas las matrices suelen tener dos estados, uno perturbado y otro no perturbado. La matriz de 2×2 es muy fácil de tratar matemáticamente y no impone mayor problema. Sin embargo en matrices de 6×6 la tarea es iterativa. Me tengo que valer de hipótesis iniciales y controlar todos los valores estadísticos para ver si mi matriz va rumbo al escenario de ruido que quiero diseñar y seguir iterando hasta obtener un resultado aceptable. Para lograr esta tarea es fundamental entender las relaciones entre los valores, algunas de las cuales no son tan intuitivas. Por ejemplo, el caso de la relación entre las probabilidades estacionarias y los elementos de las matrices $\mathbf{U}$ y $\mathbf{G}$ que se muestra en el apéndice §D.4.

3.13. Resumen

En este capítulo se describió el escenario de ruido presente en el ambiente BPL. Este canal no fue diseñado para comunicaciones sino que su principal función es proveer energía a dispositivos eléctricos. Estos últimos son la principal causa del ruido. Una de las características más distintivas de este canal es la presencia de ruidos impulsivos, en particular el ruido impulsivo aperiódico asincrónico y el ruido impulsivo en ráfagas. Los efectos disruptivos de ambos son muy importantes y deben ser un factor de peso a la hora de diseñar un sistema de comunicaciones. Por lo tanto no es el típico canal AWGN. También se abordó la caracterización de los ruidos mediante un enfoque espectral y/o temporal según fuera más conveniente. Luego

se presentaron modelos para cada uno de los ruidos. En el caso de ruido de banda angosta se analizaron dos alternativas.

4

Arquitectura de un emulador por hardware

En el capítulo 2 se presentó un modelo general para la transferencia del canal PLC y en el capítulo 3 se expusieron modelos para los ruidos que afectan al mismo. En este capítulo se explicará el diseño de la arquitectura del sistema digital para el canal PLC.

4.1. Contexto de diseño de la arquitectura

En esta sección procederemos a describir el contexto en que se diseñó la arquitectura de implementación de los distintos módulos del emulador. Vale la pena recordar la diferencia entre modelo y arquitectura. Un modelo es una representación imperfecta de la realidad, en particular, en ingeniería utilizamos modelos matemáticos. Mientras que la arquitectura es la manera concreta en que se implementa un modelo. Para poder diseñar la arquitectura es necesario conocer con qué tipo de recursos tecnológicos contamos. Por lo tanto, a continuación explicaremos el contexto que tenemos para definir la arquitectura.

También presentaremos un diagrama en bloques de alto nivel donde se ve claramente la modularización de nuestro diseño.

4.1.1. Justificación del uso de la tecnología FPGA

La implementación de un emulador de canal BPL impone claros requisitos de performance al ser un sistema de tiempo real. Si se desea poder emular un canal de 40 MHz de ancho de banda la frecuencia de muestreo mínima debe ser mayor a 80 MHz. En consecuencia, los modelos planteados en el capítulo 3 y en §2.4 deben funcionar a dicha frecuencia y deben hacerlo en *simultáneo*. Las restricciones de tiempo real de esta aplicación establecen requisitos considerables. Por ejemplo,

- Si quisiéramos implementar una respuesta al impulso de 10 μs en un filtro FIR de 800 coeficientes, eso implicaría $6{,}4 \cdot 10^{10}$ multiplicaciones y adiciones por segundo, 64 GMACs. Los FPGAs pueden explotar su elevado nivel de paralelismo.
- Se necesitan elevadas tasas de I/O. En sistemas que comparten *buses* de datos esto puede ser un problema, ya que la transferencia de datos de 16 bits de ancho a 80 Msps implica 320 MB/s por cada sentido de emulación. En un FPGA se tiene lógica y caminos de datos dedicados para esta tarea.
- Se requieren latencias de entrada-salida constantes y menores a 1 μs. Esto se puede lograr fácilmente con un FPGA a diferencia de un procesador de propósito general o un DSP donde el procesador puede atender interrupciones y arbitrar uso de recursos que hagan más difícil de predecir y controlar estos retardos.

También existen otros factores que no están relacionados a la performance

- Los sistemas de procesamiento de señales suelen cobrar vida a partir de un diagrama en bloques. Muchas veces es más fácil traducir el diagrama en bloques al FPGA que convertirlo a un código que se ejecuta en un DSP.
- Si nuestro sistema no tiene muchas operaciones condicionales el uso de FPGAs es recomendado. Si tuviese muchos una implementación por software sería más conveniente.
- Si el sistema se puede descomponerse en muchas tareas paralelas y repetitivas el uso de FPGAs es más adecuado.
- Si el sistema procesa de manera repetitiva bloques de información de tamaño fijo y no adaptativos es más conveniente el uso de FPGAs.
- El consumo de energía es mucho menor en los FPGAs en comparación con los microprocesadores que ejecutan tareas equivalentes.
- Si el sistema requiere manejar datos de punto flotante es más adecuado utilizar un microprocesador, un DSP o una GPU. Si bien los FPGAs pueden trabajar con punto flotante se especializan en trabajar con punto fijo.
- Los FPGAs ofrecen mayor flexibilidad de conexión con dispositivos externos en comparación cone los microprocesadores que utilizan interfases estandarizadas como PCI Express, Serial Rapid IO, GigabitEthernet, etc.
- Los FPGAs son reconfigurables y así puede variarse el diseño para que cumpla una función distinta. Los ASICs no ofrecen esta flexibilidad, además de necesitar un gran volumen para que éstos sean económicamente convenientes.

Un factor de peso importante para nosotros fue el hecho de que hoy en día las empresas fabricantes de FPGAs ofrecen herramientas de software que permiten utilizar técnicas *Model-Based Design*. En la técnica de diseño tradicional el proceso de desarrollo está dividido en dos fases: la fase de ingeniería de sistema y la fase de ingeniería de hardware. En la técnica tradicional existen dos modelos, el de sistema,

hecho en MATLAB, y el de hardware que está hecho en VHDL. En la transferencia de información de uno a otro intervienen documentos, vectores de pruebas y personas, por lo tanto puede dar lugar a errores y pérdida de información. En *Model-Based Design* las dos fases están fusionadas en una única fase donde el modelado del problema se utiliza para hacer el desarrollo y análisis del modelo y también para la implementación en VHDL. El modelo tiene que representar fielmente las señales que en última instancia van a ser implementadas. Ésto permite detectar errores de diseño en una etapa temprana. En el proceso computarizado de conversión a VHDL idealmente no hay pérdida de información ni introducción de errores. En resumen, las técnicas *Model-Based Design* permiten acelerar el desarrollo de manera considerable ahorrando un 50 % o más de horas de ingeniería en promedio.

4.1.2. Diagrama en bloques del emulador

El emulador está descripto en términos generales por el diagrama en bloques que se observa en la figura 4.1. Nosotros describiremos un sólo sentido del canal ya que el diagrama en bloques del otro sentido sería idéntico excepto que comparten el módulo de configuración. En la izquierda se ve que ingresa la señal del transmisor, módem 1 TX. Ésta es alterada por la respuesta en frecuencia del canal y luego perturbada por los distintos ruidos típicos del ambiente BPL. La señal de salida se conecta con el receptor, módem 2 RX. El bloque de la transferencia y los seis bloques de ruido son configurados a través un bloque de configuración. Este bloque se comunica con un software GUI en una PC y es el encargado de controlar la lógica de procesamiento de señales de los otros bloques. En este capítulo no desarrollaremos sobre este bloque ya que por su importancia tiene un capítulo aparte[1]. Vale la pena destacar la simple modularización de la arquitectura que es una característica que permitirá el fácil reuso de bloques para nuevos diseños.

[1] Ver capítulo 6 Interfaz de control del emulador

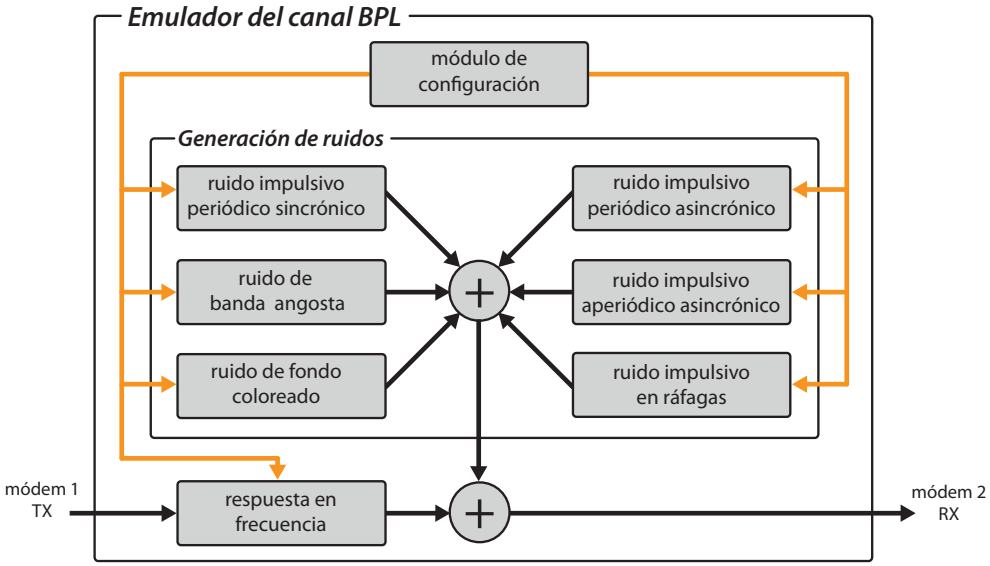


FIGURA 4.1: Diagrama en bloques del emulador (en un sólo sentido)

4.2. Requerimientos del emulador

A continuación se plantearán los requerimientos que debe cumplir el emulador

4.2.1. Requerimientos funcionales

El emulador debe

RQ_01 poder emular todos los canales de referencia planteado en la recomendación emitida por el consorcio OPERA [3].

RQ_02 permitir cambiar la configuración del canal sin necesidad de reprogramar el FPGA.

RQ_03 permitir cambiar la configuración del canal a través de una interfaz gráfica que se ejecuta en una computadora personal.

RQ_04 permitir cambiar la transferencia del canal sin detener el proceso de emulación para poder representar una variación lenta y progresiva del canal.

RQ_05 poder permitir la emulación de otros canales cualesquiera mientras estos puedan ser representados por la arquitectura utilizada.

RQ_06 permitir cambiar cada uno de los parámetros de manera independiente. Por independencia de parámetros nos referimos a que para cambiar el valor, vecto-

rial o escalar, de un parámetro no tengamos que reconfigurar el valor de otro parámetro que cumple otra función[2].

RQ_07 poder emular cada uno de los seis tipos de ruidos descritos en el capítulo 3:

1. ruido de fondo coloreado
2. ruido de banda angosta
3. ruido impulsivo periódico sincrónico
4. ruido impulsivo periódico asincrónico
5. ruido impulsivo aperiódico asincrónico
6. ruido impulsivo en ráfaga

RQ_08 poder emular ruido de fondo coloreado Gaussiano con un rango dinámico de -140 dBm/Hz a -64 dBm/Hz

RQ_09 poder emular ruido impulsivo periódico sincrónico con t_P tomando valores igual a 20 ms ó a 10 ms

RQ_10 poder emular al menos 4 interferencias de banda angosta con un ancho de banda controlable que va desde los 2 kHz hasta los 1000 kHz.

RQ_11 poder emular ruido impulsivo aperiódico asincrónico pudiendo elegir entre varios escenarios predeterminados o poder establecer arbitrariamente la matriz de probabilidades de transición de la cadena de Markov.

RQ_12 poder emular ruido impulsivo en ráfaga pudiendo elegir entre varios escenarios predeterminados o pudiendo establecer arbitrariamente las matrices de probabilidades de transición del modelo jerárquico.

RQ_13 poder calibrar la intensidad de los distintos ruidos generados de manera independiente.

RQ_14 poder comunicarse con la PC a través de una interfaz serie RS232. Esta interfaz será el medio a través del cual un software controlará al emulador.

RQ_15 El ruido de los módulos de ruidos impulsivos sincrónico, asincrónico, aperiódico asincrónico y en ráfagas debe ser configurable en términos de coloración espectral.

RQ_16 El ruido uniforme debe:

- tener al menos 16 bits de resolución

[2] Un contraejemplo sería que sólo se puede cargar la configuración de un módulo como si fuese un todo. Es decir, recargando todos los parámetros a la vez.

- una secuencia cuya período sea mayor a una hora funcionando a una tasa de 80 Msps. Por ejemplo, una longitud de $2^{40} - 1$ palabras que implica un período de 3 horas 49 minutos a 80 Msps.
- debe tener un buen diagrama en el test serial[3] [15].

RQ_17 El ruido Gaussiano debe tener una densidad de probabilidad que se desvíe menos de 0,2 % de una distribución Gaussiana para $|x| < 4{,}8\sigma$

4.2.2. Requerimientos no funcionales

El emulador debe

RQNF_01 poder implementarse utilizando el kit de FPGA Digilent XUPV5-LX110T, que es el kit del que dispone el Laboratorio de Procesamiento de Señales de las Comunicaciones (LPSC).

RQNF_02 realizarse utilizando el software *Xilinx System Generator* para MATLAB *Simulink*©

4.3. Arquitectura de la transferencia del canal

Uno de los desafíos tecnológicos que plantea la emulación es la implementación de la transferencia. Si quisiéramos poder implementar respuestas al impulso de al menos 10 μs de duración, suponiendo a una frecuencia de muestreo de 80 Msps, y usásemos una estructura FIR, eso implicaría un filtro de 800 coeficientes. A su vez, la tasa de salida es la misma que la de entrada. Por lo tanto, ello se traduce en $6{,}4 \cdot 10^{10}$ multiplicaciones y adiciones, ó 64 GMACs, por cada segundo emulado. En un ambiente de simulación, donde no hay restricciones de tiempo real, es un costo que se podría pagar por obtener la flexibilidad que una estructura FIR ofrece. Sin embargo, no deja de ser una implementación ineficiente si sólo queremos simular respuestas al impulsos de canales BPL. Para el caso de la emulación, dicha cantidad de operaciones resulta ser impráctica o ineconómica para su implementación con múltiples procesadores de propósito general o con múltiples DSPs. Aún utilizando tecnologías que permiten paralelizar las operaciones e implementarlas en un solo chip, como los FPGAs, el costo del mismo y el excesivo uso de recursos lógicos hace que la alternativa del FIR sea poco eficiente en términos económicos[4]. Hay que mencionar que hoy en día existen tecnologías basadas en los GPUs donde se aprovecha el alto grado de paralelización de su arquitectura y de especialización en operaciones de punto flotante u operaciones trascendentales. Si bien hoy su poder

[3] para más información ver §4.4.1.2.

[4] Los FPGAs con centenas de multiplicadores por hardware cuestan alrededor de 10.000 USD o más.

de procesamiento alcanza algunos TeraFLOPs, y su complejidad de programación de software está disminuyendo gracias a la existencia de *frameworks* más amigables como OpenCL o nVidia CUDA, estas plataformas plantean un desafío extra dado que se encuentran dentro de una computadora de arquitectura x86. Eso implica que debemos tener que lidiar con otra serie de desafíos para resolver restricciones de tiempo real, entrada y salida de datos, manejo de la información en los buses, latencias, etc. Además, a nivel de eficiencia energética el sistema completo que contiene al GPU consume mayor potencia que el caso del FPGA.

Por estos motivos es que también se estudió el canal para luego plantear un modelo en §2.4.1. Obviamente, este modelo es muy útil en el contexto de nuestro estudio pero carece de la flexibilidad que posee el FIR a la hora de poder representar respuestas al impulso arbitrarias. Desde el punto de vista de la implementación, este modelo debe simplificarse aún más para lograr una implementación eficiente.

4.3.1. Implementación eficiente de la transferencia

Si observamos la figura 4.2 vemos que para el modelo de Zimmermann-Dostert necesitamos un filtro pasabajos para cada camino y que además dichos filtros son distintos ya que las distancias d_i son distintas. Implementar dicha cantidad de filtros hace que la complejidad de este modelo sea comparable o superior a la de un filtro FIR aproximadamente. Por lo tanto, se debe plantear una alternativa donde se reduzca la cantidad de recursos mientras se conserve lo más fielmente posible la representación de la respuesta impulsiva. Este planteo nos lleva a la subfigura derecha de la figura 4.2 donde vemos que se han resumido todos los diferentes filtros en un solo filtro.

De esta manera, el modelo queda descompuesto en dos etapas: la etapa multi-camino y la etapa pasabajos. En la etapa multi-camino se generan los desvanecimientos selectivos de frecuencias mientras que en la etapa pasabajos se le otorga al canal la característica de atenuación. Esta división fue pensada con el objetivo en mente de poder reproducir los canales in-house, que se caracterizan por no tener una característica pasabajos, en cuyo caso el filtro utilizado es la identidad o delta de Dirac unitaria.

La etapa multi-camino consta de una serie de 20 retardos programables tal como se ve en la figura 4.3. Cada retardo programable es capaz de retardar entre 1 y 64 clocks la muestra. Se eligió una cantidad de 20 retardos acorde a lo propuesto por el consorcio OPERA para poder reproducir canales in-house muy ramificados. A su vez, cada camino cuenta con peso asignable, que en la figura se ve como a_i para $i = 1, \ldots, 20$. Por lo tanto, podemos ver que en esta etapa tenemos 40 parámetros escalares programables en un solo sentido de la comunicación.

La etapa pasabajos consta de un filtro FIR de 29 coeficientes. OPERA [3] propone una implementación con 32 coeficientes pero en sus canales de referencia hace un uso máximo de 29 coeficientes.

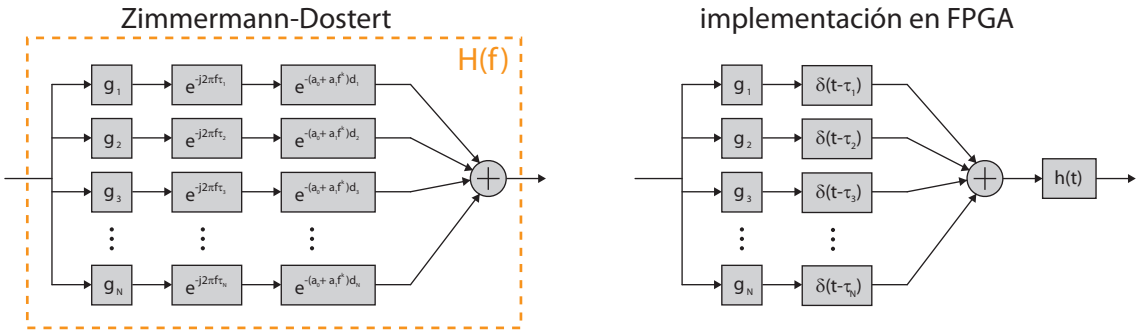


FIGURA 4.2: Comparación de modelos de la transferencia

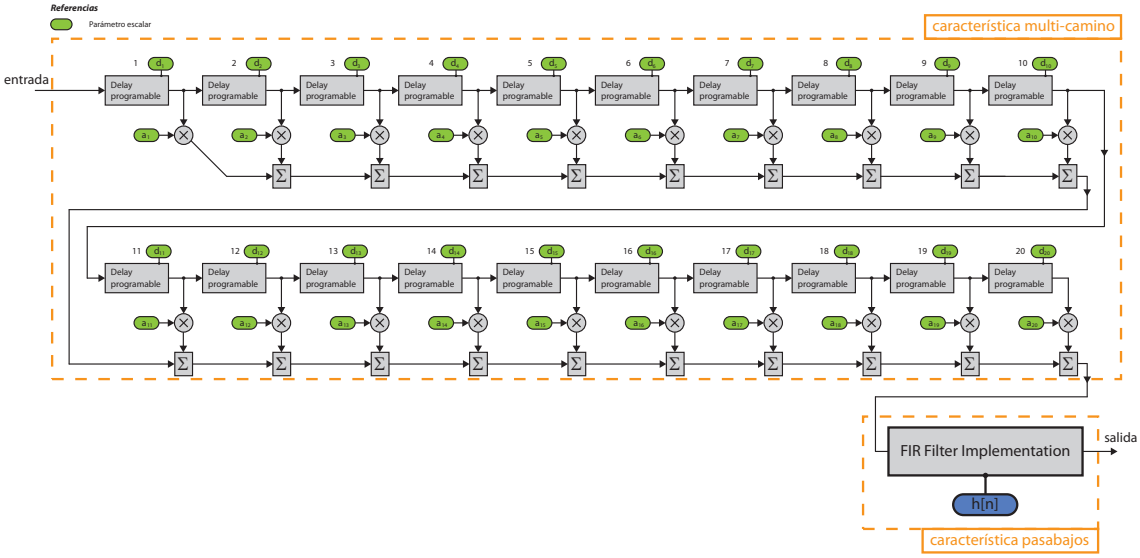


FIGURA 4.3: Arquitectura de la transferencia

4.4. Generación de ruido

La generación de ruido aleatorio requiere de una verdadera fuente aleatoria. Ésta puede ser la medición de un fenómeno natural cuyos valores sigan una distribución conocida que luego sería adquirida para adecuarse a un sistema digital. Sin embargo, en nuestro caso no contamos con dichos recursos por lo que deberemos utilizar mecanismos de generación de ruido pseudoaleatorio. Existen una gran variedad de métodos de generación, de los cuales presentaremos algunos destacando sus puntos fuertes y débiles.

4.4.1. Ruido Uniforme

Para generar secuencias pseudoaleatorias de ruido uniforme entre 0 y 1 existen varios métodos conocidos. En nuestro caso necesitamos una salida de al menos 16 bits de resolución y deseamos una secuencia cuyo período sea mayor a una hora a 80

Msps. A su vez, tenemos el requerimiento de generar muestras a la misma tasa que la señal de entrada del emulador. Otro punto muy importante a tener en cuenta es que la calidad de los números pseudoaleatorios sea buena, es decir, que la secuencia generada se comporte de manera similar a una verdadera secuencia aleatoria.

A continuación se hará una breve evaluación de los siguientes métodos de generación de números pseudoaleatorios de 16 bits. A su vez, también haremos una breve introducción del tema de manera de tener una base teórica del contexto.

- 1 LFSR
- Múltiples LFSR
- Otros. Cellular Automata (CA), Multiple CA generator

4.4.1.1. Linear Feedback Shift Register

El nombre Linear Feedback Shift Register se debe a que se utiliza un registro de desplazamiento que está retroalimentado a través de una operación lineal. Existen dos tipos de LFSRs: el de Fibonacci y el de Galois, que a su vez pueden ser implementados usando compuertas XOR o XNOR, aunque en este último caso la función no es lineal sino que es una función afín. Los LFSRs están definidos por su polinomio característico $G(x)$. Éste determina cuales son las etapas que deben retroalimentarse. Hay que mencionar que siempre se retroalimenta el bit de la salida. En las figuras 4.4 y 4.5 se observan un LFSR de Fibonacci y uno de Galois, respectivamente, con un polinomio característico $G(x) = x^{16} + x^{14} + x^{13} + x^{11} + 1$. A su vez es fácil ver que si se utilizan XORs existe un estado prohibido que es el todos '0', ya que la operación $0 \oplus 0 = 0$ y entonces el LFSR se quedaría estancado en dicho estado. Si se utilizan XNORs el estado prohibido es el de todos '1'.

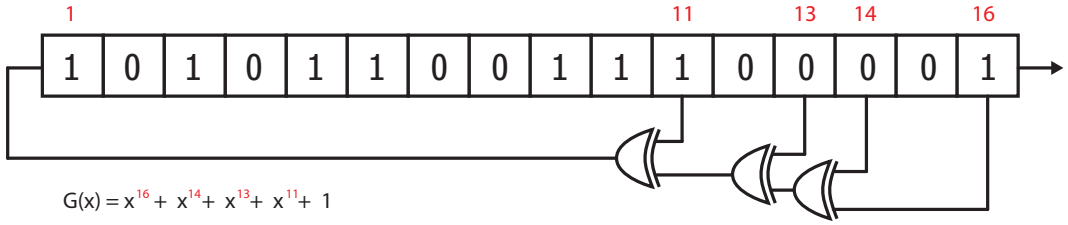


FIGURA 4.4: LFSR de Fibonacci

Si bien el LFSR de Fibonacci y el de Galois tienen una arquitectura distinta, estos producen la misma secuencia de salida. Sin embargo, ante una misma semilla inicial las secuencias de ambos no están sincronizadas. Por otro lado, una ventaja del LFSR de Galois es que su lógica tiene sólo una etapa y por este motivo puede funcionar con un clock de mayor frecuencia que el de Fibonacci.

El polinomio característico $G(x)$ determina la secuencia de salida que tendrá el LFSR. La semilla determina en que parte de la secuencia empezará a producir la

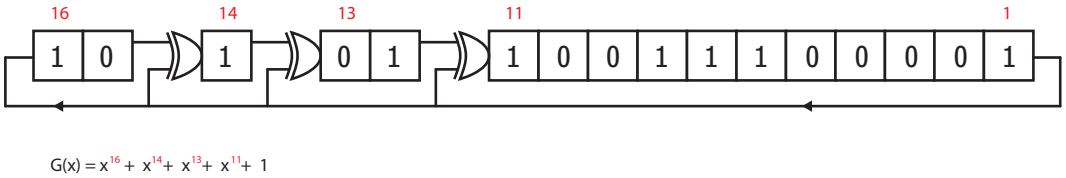


FIGURA 4.5: LFSR de Galois

salida. Se dice que un LFSR genera una secuencia máxima cuando el período de la secuencia es igual a $2^n - 1$ siendo n el número de etapas. Para que un polinomio genere secuencias máximas este debe satisfacer ciertas condiciones y además tiene ciertas propiedades

- la cantidad de retroalimentaciones debe ser par (sin contar la de la salida).
- el conjunto de retroalimentaciones, identificados por su exponente, tomados en su totalidad deben ser relativamente primos. Es decir que no debe haber un divisor común para todos.
- puede haber más de un polinomio que genere una secuencia máxima.
- si se conoce un polinomio de secuencia máxima, $x^n + x^A + x^B + x^C + 1$, inmediatamente se conoce el polinomio "espejo" de máxima $x^n + x^{n-C} + x^{n-B} + x^{n-A} + 1$.

4.4.1.2. Método 1: 1 LFSR

La utilización de un solo LFSR para generar una secuencia pseudoaleatoria de 16 bits se realiza mediante la deserealización de la secuencia de 1 bit. Es decir que se toma como salida a etapas intermedias del LFSR tal como puede verse en la figura 4.6. Vale la pena notar que en este último gráfico aprovechamos la arquitectura interna del LFSR de Fibonacci para extraer la salida paralela. Para el caso de un LFSR de Galois es necesario agregar un buffer de salida (figura 4.7). Esta configuración presenta una debilidad que es que falla el test serial[5] descripto por Donald Knuth [15]. En cualquier instante discreto t hay un 50 % de probabilidad que el valor en $t + 1$ sea correctamente predicho. Esto se debe a que para un LFSR de longitud n si en el instante t su valor es v en el instante $t + 1$ su valor será $v/2$ ó $v/2 + 2^{n-1}$. Esto se puede ver en la figura 4.8 donde para cada valor $v(t)$ existen sólo 2 posibles valores $v(t + 1)$.

[5] Este test examina si la distribución de pasar de un número q a otro r es uniforme para todos los pares (q, r) posibles. Para realizar este test se utilizan los pares de números consecutivos (sin superposición), es decir $(v(2k), v(2k+1))$ para $0 \leqslant k < N$ ya que deben ser muestras independientes.

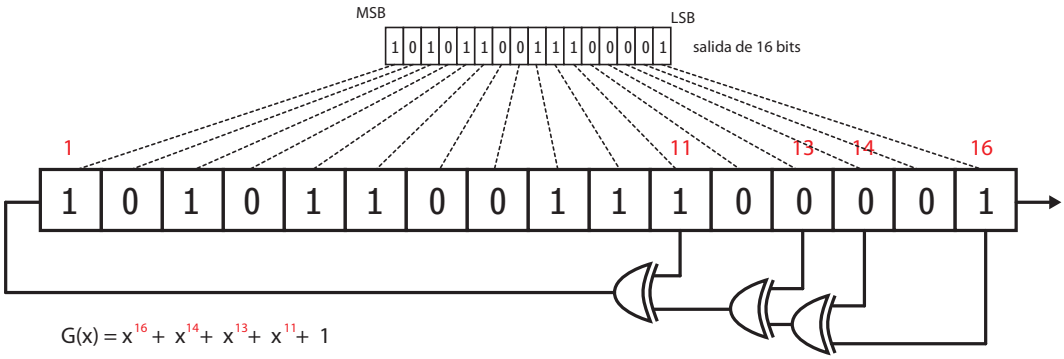


FIGURA 4.6: LFSR de Fibonacci con salida paralela

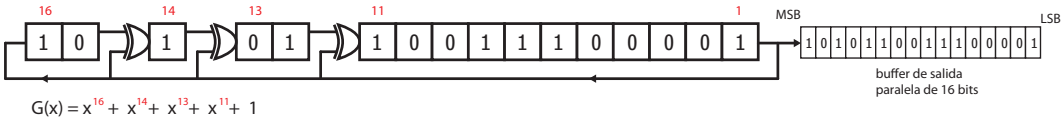


FIGURA 4.7: LFSR de Galois con salida paralela

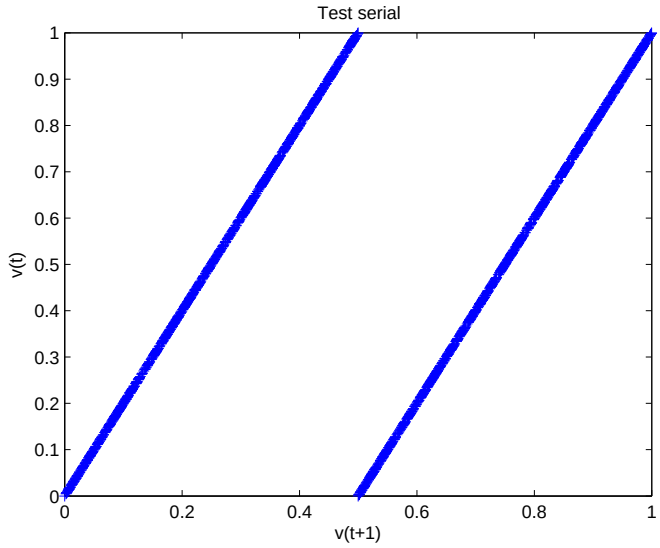


FIGURA 4.8: Test Serial 1 LFSR (valores normalizados)

Este método para generar secuencias pseudoaleatorias presentó problemas al simular la cadena de Markov. Primero se utilizó un LFSR de $n = 16$ con salida de $m = 16$ bits y se pensó que su secuencia de 65535 bits podía no ser suficiente para las longitudes de secuencias necesarias para estimar probabilidades muy pequeñas. Por lo tanto se aumentó la longitud a $n = 40$ manteniendo el ancho de la palabra de salida. En este último caso, luego de 2 millones de iteraciones la estimación de la distribución de probabilidad de estados seguía difiriendo muy significativamente de

la distribución estacionaria (ver tabla 4.1).

Distribución	p_1	p_2	p_3	p_4	p_5
Teórica	0,6997	0,0353	0,2549	0,0080	0,0019
1 LFSR	0,00002	0	0,9384	0,0579	0,0035
MATLAB random	0,6864	0,0508	0,2524	0,0082	0,0019

Tabla 4.1: Estimación de probabilidades estacionarias con $2 \cdot 10^6$ iteraciones

4.4.1.3. Método 2: Múltiples LFSR

Otro método para generar secuencias pseudoaleatorias sería leer la salida paralela cada m ciclos, es decir que las palabras de salida no compartan bits en común. El problema es que la tasa de generación de datos se reduce m veces, lo cual es un limitante importante en nuestra aplicación. Sin embargo, se puede obtener un equivalente implementando m LFSRs en paralelo. Vale la pena distinguir m, el ancho de palabra, de n, la cantidad de etapas. Nosotros hemos elegido $n = 40$ para que la secuencia generada tuviera una duración de horas a una tasa de varias decenas de Msps.

El efecto de utilizar 16 LFSRs de 40 etapas iniciados cada uno con una semilla distinta puede verse que genera un mejor resultado en el test serial (figura 4.9).

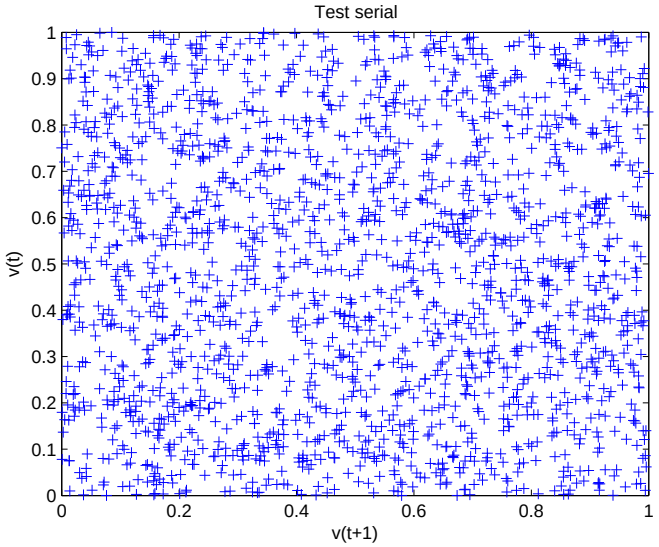


FIGURA 4.9: Test Serial 16 LFSR (valores normalizados)

4.4.1.4. Método 3: Cellular Automata

Si bien este método no lo utilizaremos en el diseño final vale la pena presentarlo ya que suele ser usado para la generación de números pseudoaleatorios [26] en diversas aplicaciones. La razón de la no utilización se debe a que, como veremos más adelante, produce una peor calidad de secuencias pseudoaleatorias ante un mayor uso de recursos de FPGA.

Un autómata celular es un modelo matemático para un sistema dinámico que evoluciona a pasos discretos. Para la generación de secuencias pseudoaleatorias se utilizan autómatas de una dimensión, que están constituidos por una cadena de células. Cada célula puede tener dos valores, viva o muerta y a su vez tiene dos vecinos: izquierdo y derecho. En cada iteración el valor de una célula c está dado por una regla. Para esta implementación se usa la regla 30, que establece que para cualquier célula c en un instante t su valor en $t+1$ será $c_{t+1} = (left + c_t) \oplus right$, donde $\oplus$ es la operación XOR. Para los extremos de esta cadena, se considera como si fuese circular, es decir que el vecino de la célula 1 es la célula n (para un autómata unidimensional de n células).

A continuación se observan las figuras 4.10 y 4.11 que muestran el test serial para el caso de un CA de 32 bits y de 32 CA de donde se extrae 1 bit de cada CA. Comparativamente, el un 1D CA produce mejor números pseudoaleatorios que un solo LFSR.

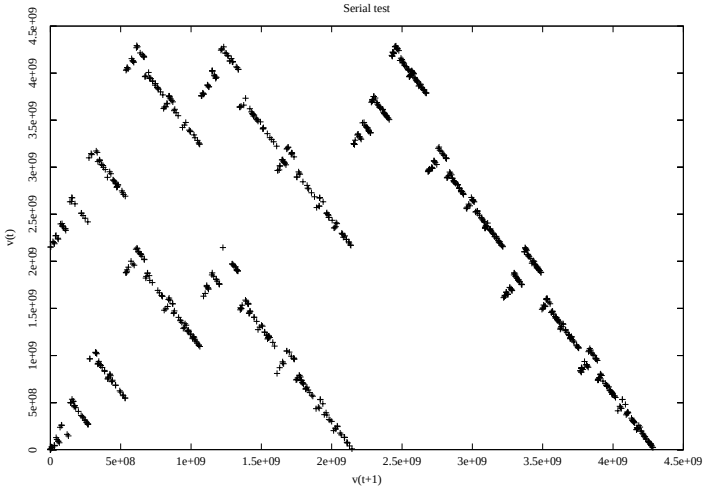


FIGURA 4.10: Test Serial CA unidimensional

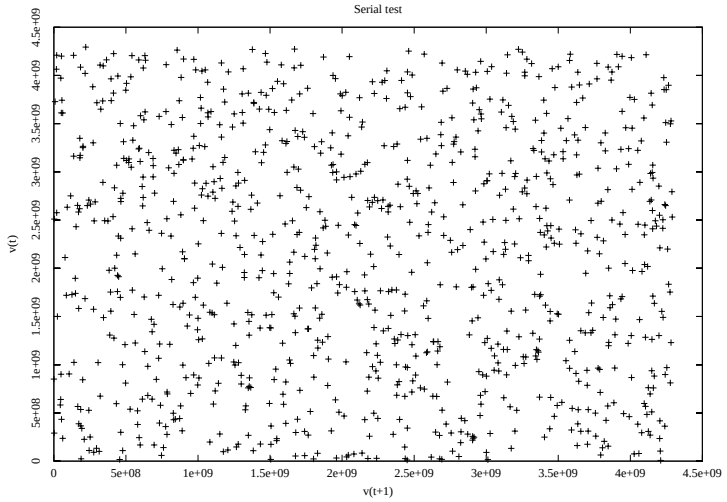


FIGURA 4.11: Test Serial 32 CA unidimensionales

4.4.1.5. Comparación entre LFSRs y CAs

Aquí expondremos los resultados hallados en [26]. Para evaluar la calidad de los ruidos pseudoaleatorios se los expone a un conjunto de tests llamado *Diehard test suite*, mantenidos por George Marsaglia [25]. Cada uno de los tests produce uno o más $p-$values. Un $p-$value puede considerarse bueno, malo o sospechoso. El criterio del test es que si $p \geqslant 0{,}998$ entonces es clasificado como malo. Si $0{,}95 \leqslant p < 0{,}998$ entonces es considerado sospechoso y el resto de los valores son dados como buenos. Cada $p-$value malo cuenta 4 puntos, cada $p-$value sospechoso suma 2 y los $p-$values buenos no suman a la cuenta. Entonces el puntaje total está dado por la suma de los puntajes de cada tests. Usando este esquema, puntajes altos representan una mala calidad de generación de secuencias pseudoaleatorias mientras que valores bajos se asocian a buenos generadores. En la tabla 4.2 vemos el puntaje obtenido en el *Diehard test suite* y los recursos de FPGA requeridos en un FPGA Xilinx Virtex-E XCV2000E.

Generador	**Ranking**	**Puntaje**	**FPGA Slices**	**Máx. Clock [MHz]**
32 LFSR	1	162	130	134
32 1DCA	2	640	284	105
1 1DCA	3	676	22	125
1 LFSR	4	756	18	188
Peor resultado posible	-	876	-	

Tabla 4.2: Resultado general del *Diehard test suite*

De estos datos podemos concluir que la estrategia de uso de LFSRs en paralelo da como resultado un mejor puntaje que los otros métodos. Además utiliza una

cantidad moderada de recursos de FPGA y lograr alcanzar buenas velocidades de clock en comparación con el resto. Por lo tanto utilizaremos dicho enfoque a la hora de generar secuencias pseudoaleatorias uniformes en nuestro emulador.

4.4.2. Ruido Gaussiano

El ruido Gaussiano ocupa un rol especial en la modelización de muchos fenómenos físicos además de contar con una fácil tratabilidad matemática. Su aparición en la naturaleza es recurrente ya que su causa puede ser justificada por la suma de muchas fuentes, cada una con una pequeña contribución pero independientes entre sí y con algunas características similares. En nuestro caso debemos generar ruido Gaussiano para poder alimentar algunos de los modelos descriptos anteriormente.

La generación de ruido Gaussiano plantea en principio mayor dificultad de implementación que la generación de ruido uniforme. Sin embargo, es un tema que ha sido ampliamente estudiado debido a que es un elemento clave para la simulación en muchas disciplinas [39]. Existen cuatro categorías de algoritmos de generación de números aleatorios Gaussianos:

1. *por inversión de la densidad de probabilidad acumulada*: simplemente invierten la densidad de probabilidad acumulada para producir un número de una distribución deseada,

2. *por transformación*: implica la transformación directa de numéros aleatorios uniformes a una distribución Gaussiana,

3. *por rechazo*: también contienen una generador de números aleatorios uniformes y una transformación pero agregan un paso adicional de rechazar condicionalmente algunos de los valores transformados,

4. *métodos recursivos*: utilizan combinaciones lineales de números aleatorios Gaussianos previamente generados para producir nuevos valores.

A su vez se podría hablar de otra clasificación que separa en dos grupos a los algoritmos:

- *métodos exactos*: se llaman así a los métodos que evaluados en un entorno ideal produzcan números aleatorios Gaussianos perfectos. Un ejemplo de esta categoría es el método de Box-Muller que si fuese implementado usando infinita precisión produciría números Gaussianos puros.

- *métodos aproximados*: se llaman así a los métodos que por más de ser evaluados en condiciones ideales sólo produzcan una aproximación a un número aleatorio Gaussiano. Un ejemplo es el teorema Central del Límite, ya que si bien puedo sumar verdaderos números aleatorios uniformes necesitaría una cantidad infinita, por lo que resultaría un valor aproximado en cualquier implementación práctica.

De todos los diversos métodos existentes nosotros nos enfocaremos en el de Box-Muller que será explicado a continuación. La razón del mismo es que la herramienta que utilizaremos para implementar el emulador nos brinda un bloque que implementa eficientemente Box-Muller para la generación de ruido Gaussiano.

4.4.2.1. Box-Muller

La transformada Box-Muller [7, 32] es uno de los primeros métodos exactos de transformación. Éste produce un par números aleatorios Gaussianos a partir de un par de números aleatorios uniformes[6] $]0;1[$ (U_1, U_2). Este método utiliza el hecho que la distribución bidimensional de dos Gaussianas independientes es radialmente simétrica si éstas tienen la misma varianza. Esto puede verse fácilmente por simple multiplicación de dos distribuciones unidimensionales $e^{-x^2}e^{-y^2} = e^{-(x^2+y^2)} = e^{-r^2}$. El algoritmo de Box-Muller puede entenderse como un método en que la salida representa coordenadas en el plano bidimensional. La magnitud del correspondiente vector puede obtenerse transformando un número aleatorio uniforme, y la fase consta de otro número aleatorio que se obtiene escalando el número por 2π. En la figura 4.12 se observa la transformación de coordenadas. Luego se realizan las proyecciones sobre los ejes coordenados para obtener los valores Gaussianos. El algoritmo 1 da el pseudocódigo para implementar dicho método. Debido a que el algoritmo produce dos números aleatorios cada vez que es ejecutado, es común devolver el primer número y luego almacenar en un *cache* el otro en el próximo llamado. El cálculo de seno

Algorithm 1 Box-Muller

```
function BOXMULLER(U1,U2)
    a ← √(−2 ln U1), b ← 2πU2
    return (a sin b, a cos b)        ▷ Devuelve un par de números independientes
end function
```

y el coseno puede usualmente realizarse en un paso y además se han publicado algoritmos altamente optimizados, y adecuados para precisión de punto fijo, basados en evaluación de funciones [6, 24, 44].

4.4.2.2. Suma de Uniformes (Teorema Central del Límite)

Otro método de transformación es la aplicación del teorema central del límite. Éste nos dice que la suma de K variables aleatorias uniformes $]-0{,}5;0{,}5[$ independientes tiende a una variable aleatoria con distribución Gaussiana de media nula y desvío estándar $\sqrt{\frac{K}{12}}$. La aproximación de esta distribución resultante es mejor cuanto mayor sea K. Sin embargo, la principal desventaja de este método es que la convergencia es lenta con el aumento de K. Esto se podría intuir al notar que la suma está acotada al intervalo $]-K/2; K/2[$ y que la función de densidad está compuesta

[6] Notar que debemos excluir el 0 del conjunto de resultados posibles

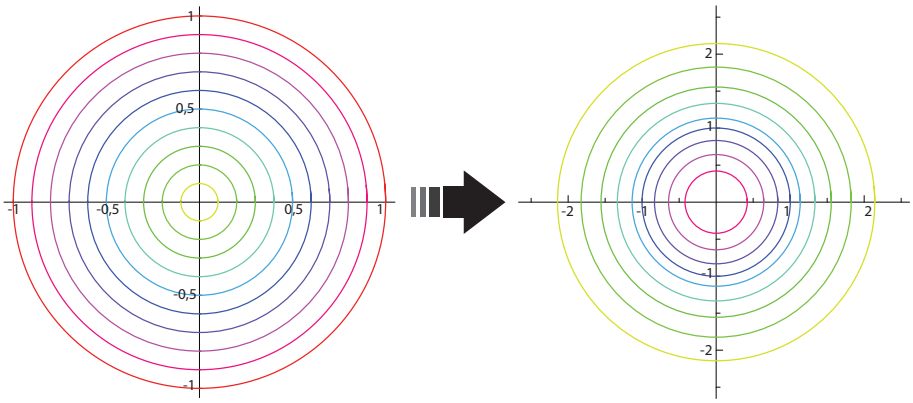


FIGURA 4.12: Ilustración de la transformación de coordenadas

por segmentos polinomiales de orden $K-1$, que surgen de la convolución de densidades de probabilidad uniformes. Por lo tanto, la aproximación en las colas de la distribución es muy mala. Se han desarrollado métodos para mitigar este problema "estirando" la función de densidad en las colas usando un polinomio de interpolación de Chebychev. Mientras que esta técnica mejora el resultado, las desviaciones de una verdadera Gaussiana son significativas para valores prácticos de K. Además no hemos expuesto el desafío que implica en términos computacionales generar grandes cantidades de números aleatorios uniformes y sumarlos. Estos hechos hacen que la generación de números aleatorios Gaussianos a través del teorema central de límite rara vez sea utilizada. Sin embargo, este enfoque se ha utilizado en la implementación por hardware como una manera de combinar dos o más números Gaussianos de menor calidad para producir uno de buena calidad [20, 24, 44].

4.5. Arquitectura del ruido de fondo coloreado

El modelo de ruido de fondo coloreado expuesto en §3.7 propone colorear una fuente de ruido blanco. Por este motivo tenemos que implementar un filtro que cumpla con dicho objetivo.

La arquitectura planteada se muestra en la figura 4.13 y se compone de un generador de ruido Gaussiano blanco y un filtro FIR. El generador de ruido utiliza el método Box-Muller y el filtro FIR tendrá 17 coeficientes. La cantidad de coeficientes fue determinada de manera de adherir a la recomendación de OPERA [3].

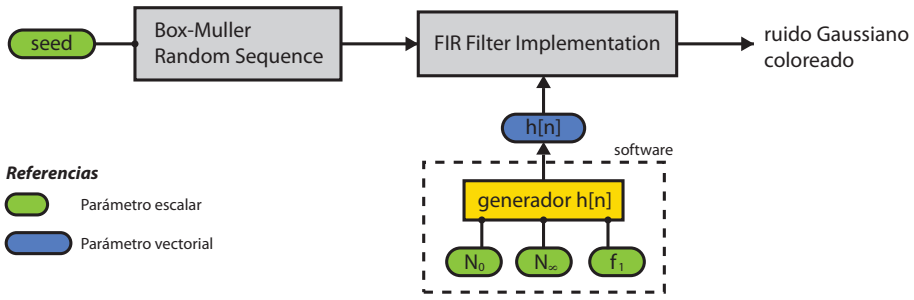


FIGURA 4.13: Diagrama en bloques de la arquitectura de la generación de ruido coloreado

El cálculo de los coeficientes del filtro en base a los parámetros N_0, N_∞ y f_1 se realizará en un software que corra en la PC del usuario. Por ejemplo, podría ser implementado en un script de MATLAB. Es conveniente que los distintos escenarios se precalculen para luego almacenarse en una base de datos o archivo de configuración y así poder prescindir de esa etapa para la operación del emulador.

4.6. Arquitectura del ruido de banda angosta

En la generación del ruido de banda angosta existen dos alternativas. La primera está basada en el modelo (3.17) y constituye la generación de ruido banda limitada para luego ser modulado en frecuencia por una portadora. La segunda se basa en la utilización de la IFFT §3.8.2.

4.6.1. Generación por superposición de señales moduladas

La primer alternativa consiste en sintetizar ruido de banda limitada y luego modularlo por una portadora. El ruido que utilizaremos tendrá distribución Gaussiana tal como el utilizado por Umehara et al. [40] y Skrzypczak et al. [38]. La limitación del ancho de banda o coloración del ruido blanco generado por el bloque Box-Muller se hará utilizando un proceso AR 1. La motivación del mismo se basa en que permite controlar eficientemente el ancho de banda a un costo muy reducido. Tanto el ancho de banda equivalente del ruido como la frecuencia portadora son parámetros configurables por el usuario. Tendremos varias instancias de este ruido modulado que luego se sumarán por principio de superposición. El diagrama en bloques de la arquitectura lo podemos ver en la figura 4.14

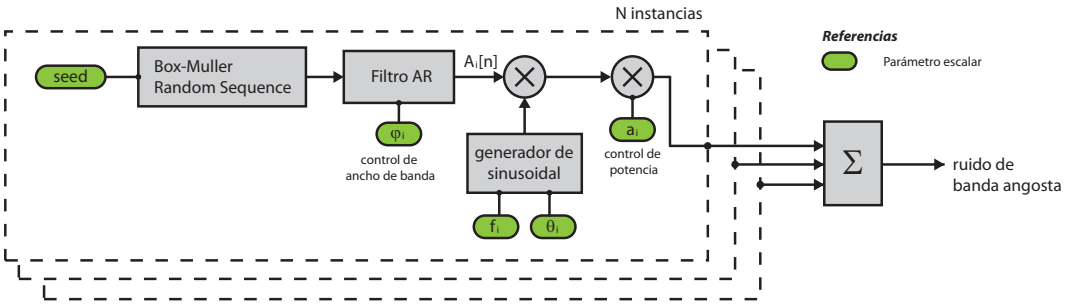


FIGURA 4.14: Diagrama en bloques de la arquitectura de la generación de ruido de banda angosta

4.6.2. Generación por IFFT

Este método es conveniente si la cantidad de portadoras que queremos generar es muy grande. Para generar el espectro deseado debemos tener en cuenta el procedimiento descripto en §3.8.2 para construir nuestro vector a antitransformar. Es decir que el vector de amplitudes generado debe respetar en última instancia la ecuación (3.23).

El tamaño de la IFFT fue determinado en base a un criterio que pretende generar interferencias con buena capacidad de posicionamiento en el espectro. Recordemos que las frecuencias centrales de los tonos están dadas por la ecuación

$$f_k = \frac{f_s \cdot k}{N_{FFT}} \qquad 0 \leqslant k \leqslant \frac{N_{FFT}}{2}$$

Además, si tenemos en cuenta que los sistemas BPL actuales tienen a lo sumo una FFT de 6144 puntos [19] que a 80 Msps representa un espaciamiento entre tonos de 13 kHz, consideramos apropiado sugerir un espaciamiento entre tonos de un tercio de dicho valor ó 4,3 kHz. Por lo tanto, debemos utilizar una IFFT de 16384 puntos.

$$\Delta_f = \frac{f_s}{N_{FFT}} \approx 4{,}3 \text{ kHz s. t. } N_{FFT} = 2^N \Rightarrow N = 14 \Rightarrow N_{FFT} = 16384 \Rightarrow \Delta_f \approx 4{,}88 \text{ kHz} \tag{4.1}$$

Entonces, acorde al procedimiento debemos generar $\frac{N_{FFT}}{2} + 1 = 8193$ valores que formarán el vector de 16384 valores

Nosotros decidimos agregar un control de ancho de banda para cada tono para tener la posibilidad de implementar interferencias que se asemejan a tonos puros. Por ejemplo, provenientes de algún oscilador en algún equipo. Dicho control de ancho de banda se logra con un proceso AR 1 aplicado a cada uno de los 8193 tonos que controlamos. La relación entre el parámetro φ y el ancho de la interferencia no es fácil de determinar analíticamente y por ello se recurrió a simulaciones. También se evidenció que un valor de φ superior a 0,5 no tenía mucho sentido práctico. Este resultado es muy distinto al observado para el método de generación por modulación de ruidos, donde el valor de φ juega un rol vital. La razón por la cual utilizamos

un proceso AR 1 fue para correlacionar los valores de dicho tono entre sucesivas antitransformaciones ya que un tono puro tendría un valor constante para todas las antitransformaciones. La correlación aquí causa que los valores de la salida sean menos variables y por lo tanto su ancho de banda se vaya afinando.

En la figura 4.15 se puede ver el diagrama en bloques de la arquitectura propuesta. Allí se observan los parámetros correspondientes a los valores φ y a la regulación de potencia de cada tono. Vale la pena recordar que nosotros hemos elegido generar sólo la parte real ya que de esta manera nos ahorraremos recursos. El bloque central toma el ruido de entrada, lo multiplica por la amplitud correspondiente y aplica el proceso AR 1 para calcular $x[n]$[**] de cada tono. Para ello se vale de una memoria para recordar el valor anterior, $x[n-1]$. Para entender mejor como funciona dividiremos el tiempo en intervalos de 16384 muestras. En el comienzo de un intervalo primero se calculan los 8193 valores $x[n]$ que se van almacenando en la memoria a medida que son calculados. En los 8191 clocks restantes se extrae la información de la memoria para poder armar una señal par. En el próximo intervalo los valores $x[n]$ almacenados pasan a ser tener el rol de $x[n-1]$ y comienza el nuevo ciclo de cálculo. Este proceso se ve gráficamente en la figura 4.16. Debido a que el sistema funciona a la velocidad del clock este proceso no es por lotes (o *burst I/O*) sino que es continuo (o *streaming*). El multiplexor se utiliza para representar el *pass-through* de los primeros 8193 valores.

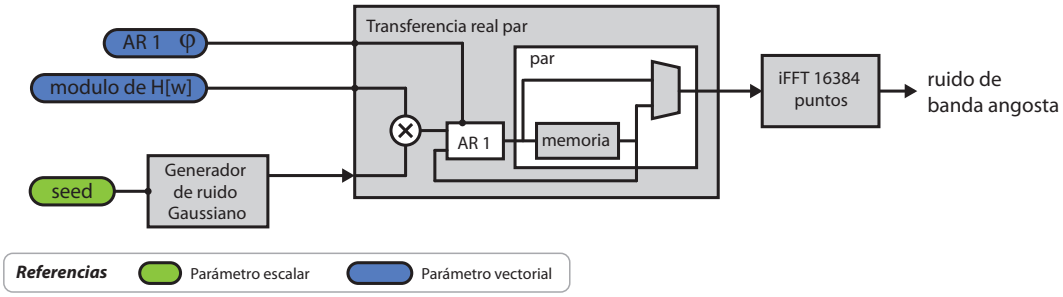


FIGURA 4.15: Diagrama en bloques de la arquitectura de la generación de ruido de banda angosta basada en la IFFT

[**] de la ecuación $x[n] = \varphi \cdot x[n-1] + e[n]$, donde en este caso $e[n] = a \cdot \eta[n]$ y $\eta[n]$ es ruido Gaussiano blanco de potencia unitaria.

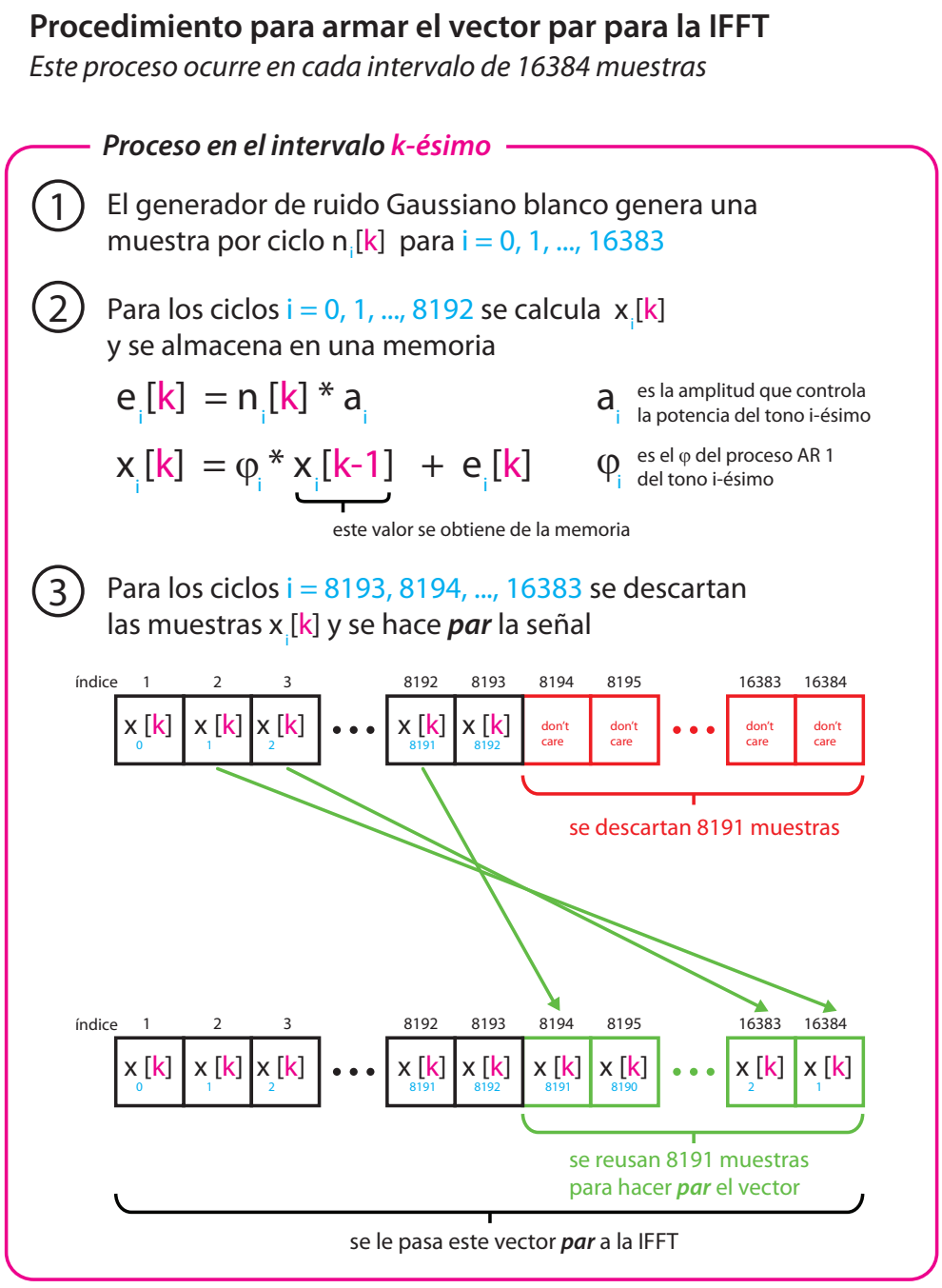


FIGURA 4.16: Proceso para construir el vector de entrada *par* para la IFFT

Control de ancho de banda En §3.8.1 se utiliza un proceso AR 1 para colorear un ruido Gaussiano blanco. Este proceso es fácil de entender. Sin embago, el rol del proceso AR 1 a la entrada de la IFFT es más heurístico. Se lo utiliza para correlacionar los valores de un determinado tono entre transformadas, ya que si el valor fuese constante eso se antitransformaría en un tono puro continuo. Es decir que al estar

correlacionados, el tono tiene menos fluctuaciones de amplitud entre transformadas, asemejándose a una senoidal. En la figura 4.17 se observa el tono puro de referencia y dos trazos: uno para $\varphi = 0$ y para con $\varphi = 0{,}5$. En el caso $\varphi = 0$ el tono presenta grandes fluctuaciones de amplitud de entre bloques de IFFTs. Para el caso de $\varphi = 0{,}5$ se ve que las amplitudes de los distintos segmentos senoidales son parecidas, es decir están correlacionadas.

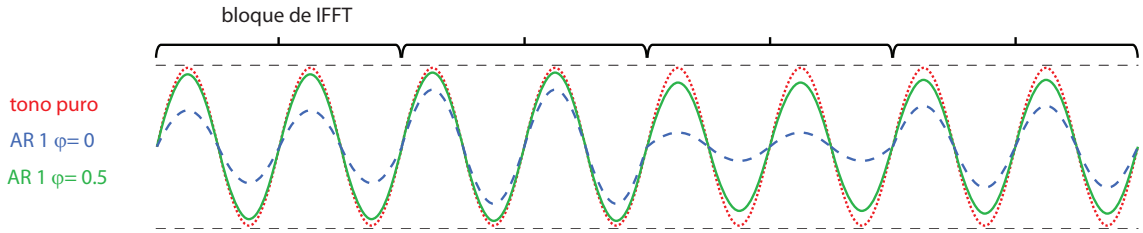


FIGURA 4.17: Interpretación del control de ancho de banda en la entrada de la IFFT.

Como el análisis teórico no es simple hemos decidido hacer simulaciones para establecer la relación entre el valor φ y el ancho de banda. Las simulaciones tampoco fueron convencionales ya que para lograr valores estables y precisos hubo que trabajar con $2{\cdot}10^9$ iteraciones y un estimador de Welch de $2^{20} = 1048576$ puntos para disminuir cualquier efecto de ventaneo. Los resultados se encuentran expuestos en la tabla 4.3.

φ	**Ancho de banda**
0,0	4,2 kHz
0,3	1,7 kHz
0,5	1,1 kHz

Tabla 4.3: Relación entre φ y el ancho de banda de un tono.

4.7. Arquitectura del ruido impulsivo periódico sincrónico

La arquitectura del ruido impulsivo periódico sincrónico puede derivarse de la arquitectura del ruido de fondo coloreado §4.5 ya que este ruido es un ruido coloreado cuya salida es comandada por una envolvente periódica. Por lo tanto, nos concentraremos en explicar como generar la envolvente periódica.

Para generar la envolvente periódica debemos tener un elemento que implemente la cuenta de dicho intervalo de tiempo. Además, dicho intervalo debe ser configurable. Por lo tanto, utilizamos un contador con un *reset* sincrónico junto con un comparador. Cuando el contador llegue a la cuenta especificada se reseteará generando el intervalo de tiempo deseado. Para implementar el ancho del pulso utilizamos otro comparador donde en este caso la salida será "1" cuando la cuenta sea menor al parámetro especificado t_B expresado en ciclos de clock. Luego dicha señal periódica será utilizada

para comandar un multiplexor. El multiplexor determina si a la salida hay ruido o no.

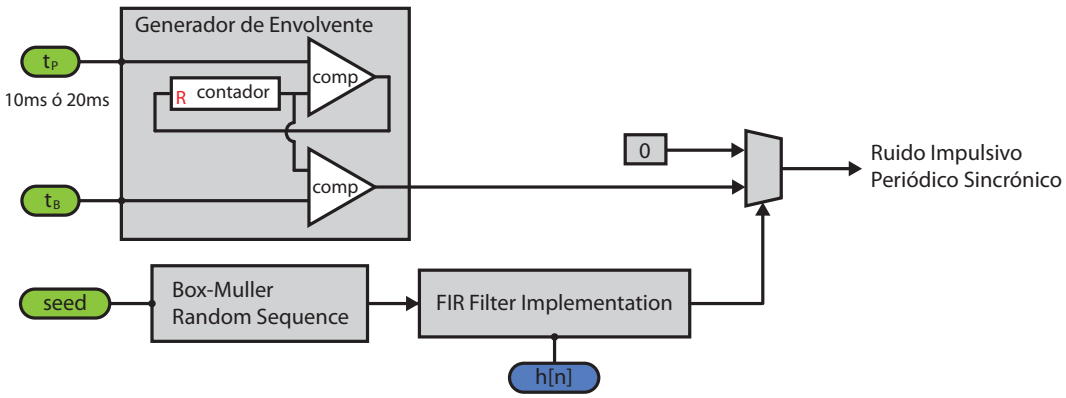


FIGURA 4.18: Diagrama en bloques de la arquitectura de generación de ruido impulsivo periódico sincrónico

4.8. Ruido impulsivo periódico asincrónico

El ruido impulsivo periódico asincrónico tiene la misma arquitectura que el ruido impulsivo periódico sincrónico §4.7. La única diferencia está dada en los valores de t_P y t_B.

4.9. Arquitectura del ruido impulsivo (aperiódico) asincrónico

Para poder generar este ruido necesitamos implementar un cadena de Markov. Y para poder implementar una cadena de Markov necesitamos una variable aleatoria con distribución uniforme entre 0 y 1 o cualquier otro rango que sería ser transformado al intervalo [0,1). A continuación se muestra el diagrama en bloques de la implementación de la cadena de Markov en la figura 4.19.

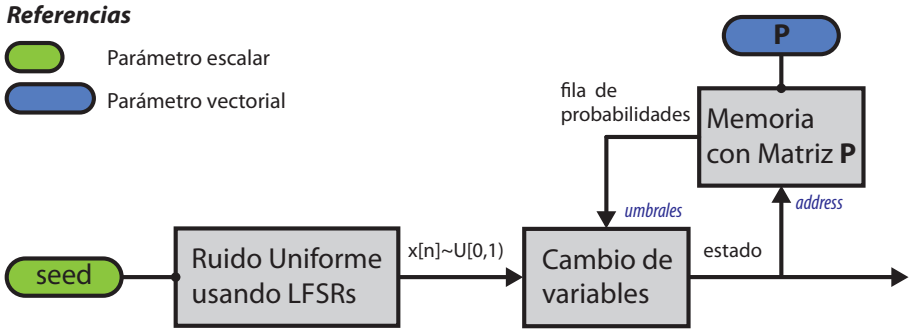


FIGURA 4.19: Diagrama en bloques de la arquitectura de cadena de Markov

Funcionamiento El generador de ruido uniforme entre 0 y 1 alimenta el bloque que realiza un *cambio de variables*[7] cuya salida es el estado actual de la cadena de Markov. La otra entrada del bloque es un vector de probabilidades compuesto por una de las filas de la matriz de transición $\mathbf{P}$. El bloque *"Memoria con Matriz* $\mathbf{P}$*"* son varias memorias que comparten el puerto de direccionamiento, el cual es a su vez el estado actual de la cadena de Markov. Con la salida de este bloque se arman umbrales para decidir a cuál estado se transicionará. El algoritmo 2 muestra el funcionamiento del bloque *"Cambio de variables"*. Sin embargo, este algoritmo cuenta con una desventaja así como está planteado. La síntesis en hardware de un código derivado dará como resultado que haya sumadores. Si observamos con detenimiento notaremos que los elementos que se suman son constantes, por lo tanto es posible trasladar esa suma a los valores almacenados en las memorias. Eso daría lugar al algoritmo 3. Entonces tendríamos almacenadas las probabilidades acumuladas. Y por lo tanto el bloque llamado *"Memoria con Matriz* $\mathbf{P}$*"* deberíamos llamarlo *"Memoria con Matriz* $\mathbf{P}_{\mathbf{acum}}$*"*.

Algorithm 2 Cambio de variables

```
/* Se asume que hay una p6 tal que p6 = 1 − (p1 + p2 + p3 + p4 + p5) */
if x < p1 then
    estado ← 0
else if x < (p1 + p2) then
    estado ← 1
else if x < (p1 + p2 + p3) then
    estado ← 2
else if x < (p1 + p2 + p3 + p4) then
    estado ← 3
else if x < (p1 + p2 + p3 + p4 + p5) then
    estado ← 4
else
    estado ← 5
end if
```

Como vimos en el modelo (§3.11), hay estados perturbados y estados no perturbados. Los análisis de las mediciones de Zimmermann [47] consideran que se puede obtener una buena reproducción utilizando 4 estados no perturbados y 2 estados perturbados. Agregar más estados no tiene sentido porque estos serían de muy baja probabilidad haciendo el diseño más costoso para el aporte que estos estados podrían ofrecer.

La cadena de Markov genera como salida una secuencia de estados. Si los estados 0 al 3 son estados no perturbados, entonces se puede generar la envolvente que modula al ruido coloreado con un simple comparador, tal como muestra la figura 4.20. Si el

[7] hablando en términos probabilísticos

Algorithm 3 Cambio de variables. Versión Mejorada

if $x < \Pr(X \leqslant 0)$ **then**
    $estado \leftarrow 0$
else if $x < \Pr(X \leqslant 1)$ **then**
    $estado \leftarrow 1$
else if $x < (\Pr(X \leqslant 2)$ **then**
    $estado \leftarrow 2$
else if $x < \Pr(X \leqslant 3)$ **then**
    $estado \leftarrow 3$
else if $x < \Pr(X \leqslant 4)$ **then**
    $estado \leftarrow 4$
else
    $estado \leftarrow 5$
end if

valor del estado es menor o igual a 3 la salida del comparador será cero y en otro caso será uno. Cada pulso rectangular de la envolvente tiene una amplitud dada por un generador de amplitudes aleatorias que mantiene su valor constante durante el pulso y que cambia de valor ante un nuevo pulso gracias a un detector de flancos positivo. Luego, la envolvente final se utiliza para modular el ruido Gaussiano coloreado que se genera como es descripto anteriormente.

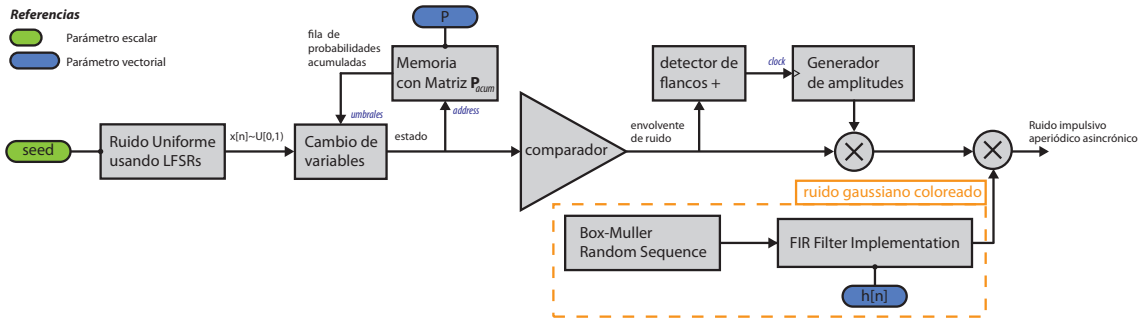


FIGURA 4.20: Diagrama en bloques del ruido impulsivo (aperiódico) asincrónico

4.10. Arquitectura del ruido impulsivo en ráfagas

El ruido impulsivo en ráfagas tiene un modelo similar al del ruido impulsivo aperiódico asincrónico. La diferencia está en que el ruido en ráfaga tiene dos cadenas de Markov pero de menor cantidad de estados. Por estos motivos, la arquitectura tendrá partes muy similares y los bloques en común no serán explicados.

En la figura 4.21 observamos el diagrama en bloques de la arquitectura y se puede apreciar claramente ambas cadenas de Markov. La de primer nivel es la encargada de generar las envolventes de ráfaga. Mientras que la segunda se encarga de describir

los impulsos individuales dentro de una ráfaga. En la segunda cadena se observa que a través de un multiplexor se fuerza al estado "1" cuando comienza una ráfaga, la cual es detectada con el detector de flancos positivos. La razón por la que se fuerza el estado es para asegurar que un intervalo de ráfaga comience con un impulso. Cada cadena tiene sólo dos estados que son representados por "0" y "1". La salida de ambas cadenas pasan por una operación `AND` que manifiesta la jerarquía de la primera sobre la segunda en dejar pasar o no los impulsos individuales creados por la cadena de segundo nivel. Por último vemos el generador de amplitudes que multiplica las envolventes de impulsos individuales. Notar que el generador de amplitudes sólo cambia de valor cuando hay una nueva ráfaga. Por este motivo, los impulsos de una misma ráfaga son multiplicados por el mismo valor. Esto refleja la correlación que tienen los impulsos de una ráfaga ya que en la realidad suelen originarse en la misma fuente. Los componentes restantes ya fueron explicados para el ruido impulsivo aperiódico asincrónico §4.9.

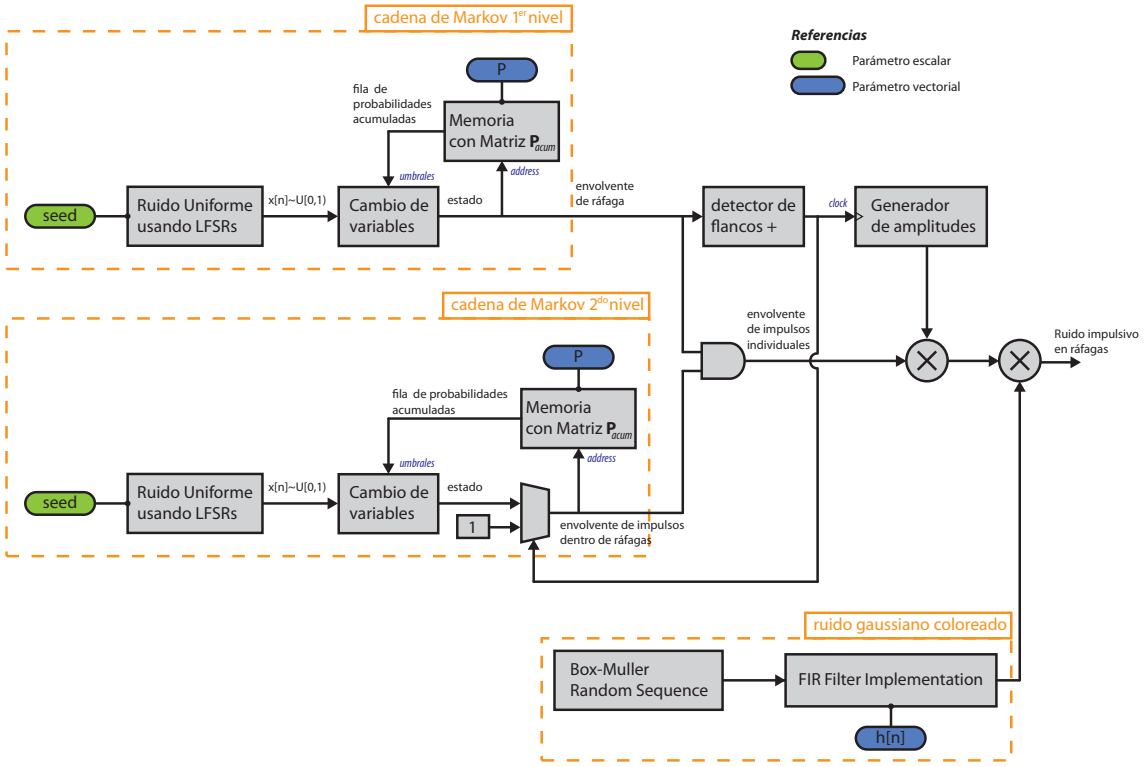


FIGURA 4.21: Diagrama en bloques del ruido impulsivo en ráfagas

4.11. Resumen

En este capítulo se describió en qué contexto se diseñó la arquitectura y por qué se decidió utilizar la tecnología FPGA. La arquitectura planteada se caracteriza por su simple estructura en módulos que permite el reuso o intercambio de bloques para

emular otros tipos de canales. También se especificaron los requerimientos que debía cumplir el emulador. Luego se plantearon las arquitecturas para la transferencia y para cada tipo de ruido. Nuevamente se expusieron dos alternativas para el ruido de banda angosta.

5

Implementación sobre FPGA

Este capítulo tratará sobre la implementación en el FPGA de la arquitectura diseñada en el capítulo 4. Se describirá el proceso de diseño, las herramientas utilizadas y las decisiones que se tuvieron que tomar ante las relaciones de compromiso para poder satisfacer los requerimientos.

5.1. Banco de trabajo

En la figura 5.1 se presenta el banco de trabajo utilizado. Este se compone de:

- una computadora personal con conexión serial RS232 y puerto USB para la conexión de la interfaz JTAG.
- el kit de FPGA Digilent XUPV5-LX110T, que es el kit del que dispone el Laboratorio de Procesamiento de Señales de las Comunicaciones (LPSC). Este kit contiene:
  - 1 FPGA Xilinx Virtex-5 XC5VLX110, más especificamente *xc5vlx110t-1ff1136*. Las características de este FPGA están en la tabla de la figura 5.4.
  - interfaz de programación JTAG
  - Dos Xilinx XCF32P Platform Flash PROMs (32 Mbyte cada una) para almacenamiento de extensas configuraciones.
  - Un controlador de configuración Xilinx SystemACE Compact Flash.
  - Un módulo 64-bit wide 256Mbyte DDR2 small outline DIMM (SODIMM) compatible con IP de EDK y drivers de software.
  - On-board 32-bit ZBT SRAM sincrónica e Intel P30 StrataFlash.
  - 10/100/1000 tri-speed Ethernet PHY que soportan interfases MII, GMII, RGMII, y SGMII.

- Controlador para USB host y USB peripheral.
- Un generador de clocks de sistema programable.
- Codec AC97 estéreo con line in, line out, headphone y audio digital SP-DIF.
- puerto RS232, Display LCD 16x2 y otros dispositivos de I/O y puertos

- los cables correspondientes: Xilinx Platform cable USB II y el cable RS232 que debe ser *null-modem*.

Vale la pena notar que este kit no está pensado para aplicaciones de DSP sino para aplicaciones de lógica digital como implementar procesadores para su estudio académico.

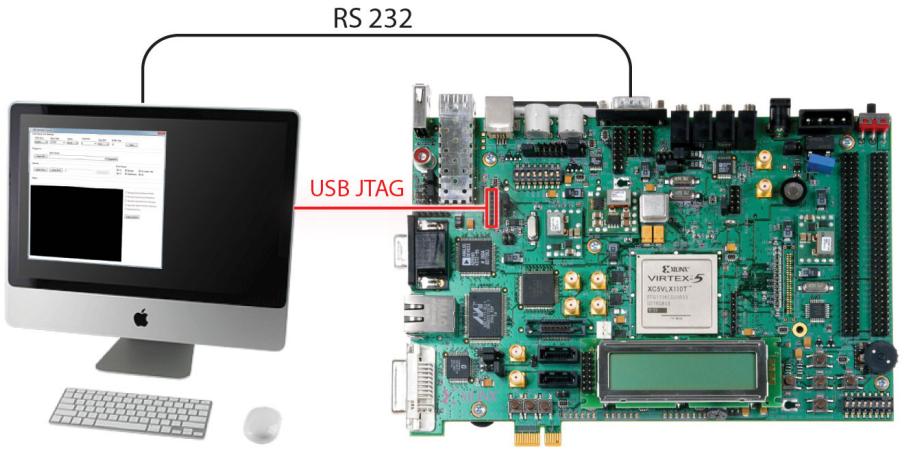


FIGURA 5.1: Banco de trabajo

5.2. Herramientas de diseño

Para poder implementar la arquitectura que diseñamos en el capítulo 4 en el FPGA necesitamos utilizar herramientas provistas por *Xilinx*. Éstas fueron las siguientes:

- *System Generator* es una herramienta de *Xilinx* que permite el uso del entorno *Simulink* de MATLAB para el diseño de sistemas de procesamiento de señales. Esta herramienta permite que el usuario no tenga experiencia previa en FPGAs de *Xilinx* o en metodologías de diseño RTL. Más importante aún es que esta herramienta permite acelerar el tiempo de desarrollo considerablemente. Mathworks, creadores de MATLAB, muestran en su página[1] un caso de estudio real en que un experto en diseño RTL y un experto en *System Generator* realizaron el mismo proyecto de manera independiente. Ambos disponían de

[1] `http://www.mathworks.com/fpga-design/simulink-with-xilinx-system-generator-for-dsp.html`

los mismos recursos[2] y como resultado final se obtuvo una relación 4 a 1 en las horas de ingeniería dedicadas. Es decir, que utilizando la técnica conocida como *Model-Based Design* el ingeniero de *Xilinx* utilizó un cuarto del tiempo.

  - *Wavescope* es un bloque de *System Generator* que permite capturar las señales dentro de un modelo de *System Generator* para luego poder visualizarlas y analizarlas. Su presentación es similar a la de los típicos simuladores como *Questasim*.
  - *Hardware Co-Simulation* es una manera simular el diseño realizado utilizando la técnica de *hardware-in-the-loop*. Su motivación principal es la aceleración por hardware. Sin embargo, hay *cores* que no se puede simular íntegramente en la simulación por software. Por ejemplo, este es el caso del *soft-processor MicroBlaze* y la razón por la cual nosotros utilizamos *Hardware Co-Simulation*.

- *Xilinx Platform Studio* es una herramienta que permite diseñar plataformas de procesadores embebidos. Con este software diseñamos la plataforma del *soft-processor MicroBlaze*.
- *Xilinx Software Development Kit* es un IDE que se utiliza para escribir y desarrollar software para la plataforma de *MicroBlaze*.
- *Chipscope Pro Analyzer* es un software que se conecta con el FPGA a través de la interfaz JTAG y permite emular un analizador lógico. Para ello debemos introducir unos *cores* de *Chipscope Pro* en nuestro diseño. En *System Generator* la tarea de inserción de los *cores* se encuentra abstraída y se trabaja a alto nivel.
- *Timing Analyzer* es una herramienta que permite analizar los tiempos de las señales una vez que ya fueron ruteadas dentro del FPGA. Permite identificar los caminos críticos que limitan la velocidad de nuestro diseño. También brinda información sobre la composición del valor de retardo, es decir qué fracción se debe al ruteo y a la lógica, y a su vez puede identificar qué elementos atraviesa ese camino.

5.3. Generalidades de la implementación

En el proceso de implementación constantemente se toman decisiones de compromiso. Este hecho se debe a que la realidad choca con las "buenas" condiciones del ambiente de diseño o de laboratorio. La cantidad de recursos ahora es limitada y además se suman restricciones que antes eran inexistentes o irrelevantes al objeto del estudio. Es por ello que el proceso de implementación constituye uno de los grandes desafíos de la ingeniería.

A continuación se tratarán una serie de temas que surgen a la hora de la implementación.

[2] por ejemplo, misma disponbilidad de *cores* de *LogiCORE*

5.3.1. Consideraciones de punto fijo

Una de las primeras restricciones con que uno se encuentra al pasar de las simulaciones numéricas hacia el hardware es el tipo de datos a utilizar. Las simulaciones, para validar una teoría, en muchos casos utilizan la máxima precisión numérica práctica posible. Y esto es acertado porque el objetivo es verificar el modelo matemático. El tipo de dato típicamente utilizado es el punto flotante de doble precisión o como se conoce en la jerga, *double*. Sin embargo, construir una computadora que trabaje con dicho tipo de datos es realmente costoso, complejo y también podría ser tecnológicamente irrealizable si las condiciones de ejecución son muy estrictas respecto a performance. Entonces, es ahí donde aparece una de las primeras relaciones de compromiso ya que en lugar de punto flotante se utiliza representación y aritmética de punto fijo. El pasaje de una a otra es una ciencia en si mismo ya que surgen problemas numéricos de diversa índole. Por ejemplo, es vital definir el ancho de la palabra y donde estará situada su coma fraccionaria de manera que no haya[3] *overflow* y que se tenga la máxima resolución para dicho ancho para que los algoritmos funcionen correctamente. También aparecen nuevos actores como el ruido de cuantización y los problemas de estabilidad numérica en algoritmos que poseen realimentación. Además, como la precisión es finita, la capacidad o no de hacer redondeo toma un rol más importante que en el caso de punto flotante. Todos estos problemas de matemáticas "finitas" son parte de los nuevos problemas de la matemática y que son de difícil análisis y que muchas veces terminan resolviéndose de maneras pragmáticas y no tan formales.

Los módems BPL de hoy en día cuentan con una densidad espectral de potencia máxima de transmisión de entre -50 y -55 dBm/Hz y un rango dinámico de hasta 90 dB [37]. La limitación de potencia se debe al cumplimiento de la regulación de EMI impuesta para cada país. Por ejemplo, en la figura 5.2 se observan las máscaras de emisión de radiación para dos países y la propuesta de la RegTP[4] en Alemania. La máscara del Reino Unido[5] es tan restrictiva que no permite ningún tipo de comunicaciones. Por otro lado, el rango dinámico también se encuentra limitado por una conjunción de causas que se reflejan en costos económicos y límites físicos.

[3] o se minimice la ocurrencia de los mismos

[4] sería el equivalente a la CNC en Argentina.

[5] del año 2001.

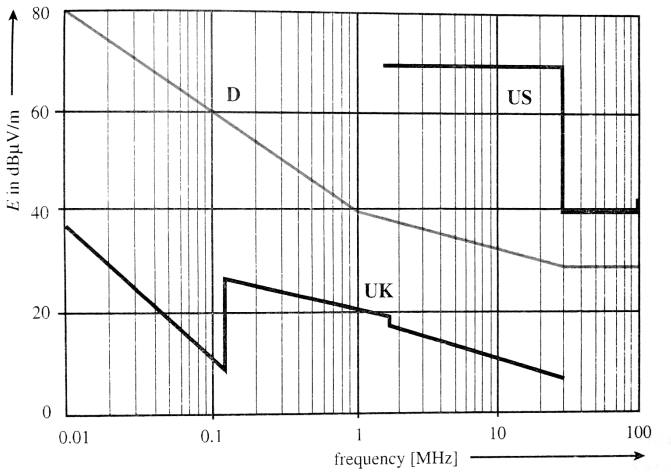


FIGURA 5.2: Límites máximos de radiación para dispositivos de comunicaciones *powerline*. curva D: propuesta por RegTP en Alemania.

Costos tecnológicos Los costos asociados a ADCs y DACs se vuelven considerables cuando estos son de una muy elevada tasa de muestro y una alta resolución. Esto es así ya que este tipo de conversores no es de uso masivo, su complejidad tecnológica puede ser alta y sus aplicaciones son restringidas.

Límite de potencia, atenuación y piso de ruido La potencia de transmisión se encuentra limitada por regulación y en el mejor de los casos la PSD puede tener un valor de -50 dBm/Hz. El piso de ruido típico en un receptor suele ser de -140 dBm/Hz. Allí tendríamos 90 dB de rango dinámico pero si también tenemos en cuenta la atenuación del canal que suele ser de 40 dB, se puede decir que se está más que cubierto con 90 dB de rango dinámico en el receptor. En la figura 5.3 [37] se observa una medición de un canal en un edificio y se ven magnitudes similares a las mencionadas. En rosa se indica la atenuación que sufre la señal inyectada (línea roja superior, -55 dBm/Hz). También vemos la densidad espectral de ruido en azul y en el medio, en verde, se encuentra la relación señal ruido disponible. A partir de esta última se podría tener un estimado de la capacidad del canal aplicando el teorema de Shannon. Decimos que es un estimado ya que no se están respetando las hipótesis del teorema[6].

$$C = \int_{f_1}^{f_2} \log_2 \left(1 + \frac{S_{rr}(f)}{S_{nn}(f)}\right) df \qquad \text{con } B_w = f_2 - f_1 \tag{5.1}$$

donde $S_{rr}(f)$ es la densidad espectral de potencia de la señal del transmisor recibida en el receptor y $S_{nn}(f)$ es la densidad espectral de potencial del ruido.

[6] el canal no es AWGN.

Los 90 dB de rango dinámico pueden lograrse utilizando una representación numérica de punto fijo de 16 bits con signo. Un bit se destina al signo y los restantes 15 bits se utilizan para representar las posibles amplitudes. En consecuencia, tenemos

$$20 \log_{10}(2^{15}) = 90,3 \text{ dB}$$

Por este motivo decidimos trabajar con una resolución de 16 bits.

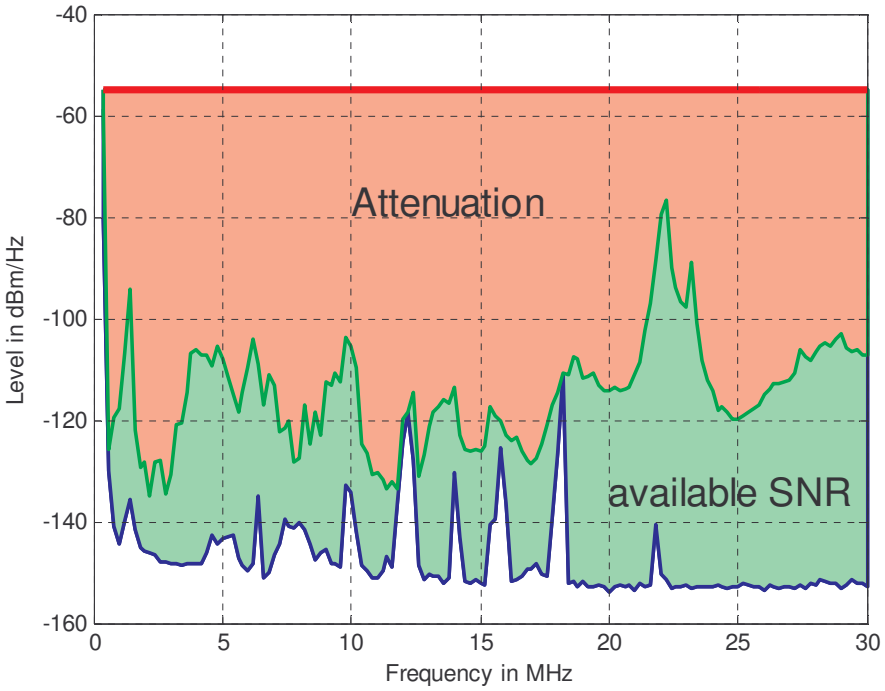


FIGURA 5.3: Relación Señal Ruido disponible

5.3.2. Tipos de dato de punto fijo en System Generator

Los tipos de dato de punto fijo se describen con la siguiente nomenclatura

- *Bool* no es estrictamente un tipo de dato de punto fijo sino que es el tipo booleano que puede tomar los valores *true* or *false*, representados por "1" y "0" respectivamente. Vale la pena mencionar que no es equivalente a `UFix_1_0`.
- *Unsigned* se expresan como `UFix_X_Y`, donde `X` es la cantidad total de bits e `Y` es la cantidad de bits fraccionarios. Por ejemplo, `UFix_5_3` puede representar valores desde 0 hasta 3,875 = $(2^4 + 2^3 + 2^2 + 2^1 + 2^0) \cdot 2^{-3}$.
- *Signed (2's complement)* se expresan como `Fix_X_Y`, donde `X` es la cantidad total de bits e `Y` es la cantidad de bits fraccionarios. Por ejemplo, `Fix_5_3` puede representar valores desde $-2 = (-1 \cdot 2^{-4}) \cdot 2^{-3}$ hasta 1,875 = $(-1 \cdot 0 \cdot 2^4 + 2^3 + 2^2 + 2^1 + 2^0) \cdot 2^{-3}$. Notar que el rango se ha reducido porque el bit más

significativo representa $-a_{m-1} \cdot 2^{m-1} \cdot 2^{-f}$ para una palabra de m bits con f bits fraccionarios, donde a_{m-1} es el valor del bit más significativo.

En el desarrollo de cada módulo se han estudiado los tipos de dato a utilizar en cada una de las señales. Explicar la justificación del tipo de datos de cada señal para cada módulo haría difícil la lectura. Por lo tanto se ha realizado este ejercicio solamente en el módulo de generación de ruido de banda angosta por superposición de portadoras §5.6.1. También nos pareció el caso más adecuado porque se tuvo que realizar un estudio numérico para el lazo del AR 1. En los gráficos correspondientes a ese módulo se observarán los tipos de dato de cada señal. Para el resto de los módulos de generación de ruidos o de la transferencia sólo hemos mencionado el tipo de dato de la salida. Para determinar el tipo de dato de la salida hemos supuesto que el sistema está calibrado de manera tal que los valores están expresados en Volts, es decir que un uno numérico representa 1 Volt. Éste sería un punto a tener en cuenta cuando se agregasen al sistemas los módulos ADC/DAC ya que probablemente se necesitaría "recalibrar" la interpretación entre el valor numérico y la unidad Volt.

5.3.3. Parametrización

Las implementaciones aquí presentadas son parametrizadas a través de recursos compartidos. Los recursos compartidos son accedidos desde un *soft-processor MicroBlaze* integrado en el FPGA que corre un software que permite configurar los distintos módulos implementados. Los recursos compartidos puede ser de distintos tipos: memorias compartidas, registros compartidos y FIFOs. Si se quiere una explicación más detallada el lector debe referirse al capítulo 6 que está dedicado a explicar cómo se controla el emulador utilizando recursos compartidos.

5.3.4. Implementación de filtros FIR

Una limitación de nuestro kit XUPV5 es que cuenta con una pequeña cantidad de *slices DSP48E*. En la figura 5.4 se observa una tabla perteneciente a una hoja de datos con las especificaciones generales de la familia *Virtex-5*. En dicha tabla se observa que la serie *Virtex-5 SXT*, que está especializada en procesamiento de señales con capacidad de interconexión serial avanzada, tiene un mayor número de *DSP48E slices* llegando en su extremo a 1056. Sin embargo, la serie *Virtex-5 LXT*, la de nuestro kit, fue pensada para lógica avanzada y por ello no cuenta con abundantes recursos para el procesamiento de señales.

Device	Configurable Logic Blocks (CLBs)			DSP48E Slices[2]	Block RAM Blocks			CMTs[4]	PowerPC Processor Blocks	Endpoint Blocks for PCI Express	Ethernet MACs[5]	Max RocketIO Transceivers[6]		Total I/O Banks[8]	Max User I/O[7]
	Array (Row x Col)	Virtex-5 Slices[1]	Max Distributed RAM (Kb)		18 Kb[3]	36 Kb	Max (Kb)					GTP	GTX		
XC5VLX30	80 x 30	4,800	320	32	64	32	1,152	2	N/A	N/A	N/A	N/A	N/A	13	400
XC5VLX50	120 x 30	7,200	480	48	96	48	1,728	6	N/A	N/A	N/A	N/A	N/A	17	560
XC5VLX85	120 x 54	12,960	840	48	192	96	3,456	6	N/A	N/A	N/A	N/A	N/A	17	560
XC5VLX110	160 x 54	17,280	1,120	64	256	128	4,608	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX155	160 x 76	24,320	1,640	128	384	192	6,912	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX220	160 x 108	34,560	2,280	128	384	192	6,912	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX330	240 x 108	51,840	3,420	192	576	288	10,368	6	N/A	N/A	N/A	N/A	N/A	33	1,200
XC5VLX20T	60 x 26	3,120	210	24	52	26	936	1	N/A	1	2	4	N/A	7	172
XC5VLX30T	80 x 30	4,800	320	32	72	36	1,296	2	N/A	1	4	8	N/A	12	360
XC5VLX50T	120 x 30	7,200	480	48	120	60	2,160	6	N/A	1	4	12	N/A	15	480
XC5VLX85T	120 x 54	12,960	840	48	216	108	3,888	6	N/A	1	4	12	N/A	15	480
XC5VLX110T	160 x 54	17,280	1,120	64	296	148	5,328	6	N/A	1	4	16	N/A	20	680
XC5VLX155T	160 x 76	24,320	1,640	128	424	212	7,632	6	N/A	1	4	16	N/A	20	680
XC5VLX220T	160 x 108	34,560	2,280	128	424	212	7,632	6	N/A	1	4	16	N/A	20	680
XC5VLX330T	240 x 108	51,840	3,420	192	648	324	11,664	6	N/A	1	4	24	N/A	27	960
XC5VSX35T	80 x 34	5,440	520	192	168	84	3,024	2	N/A	1	4	8	N/A	12	360
XC5VSX50T	120 x 34	8,160	780	288	264	132	4,752	6	N/A	1	4	12	N/A	15	480
XC5VSX95T	160 x 46	14,720	1,520	640	488	244	8,784	6	N/A	1	4	16	N/A	19	640
XC5VSX240T	240 x 78	37,440	4,200	1,056	1,032	516	18,576	6	N/A	1	4	24	N/A	27	960
XC5VTX150T	200 x 58	23,200	1,500	80	456	228	8,208	6	N/A	1	4	N/A	40	20	680
XC5VTX240T	240 x 78	37,440	2,400	96	648	324	11,664	6	N/A	1	4	N/A	48	20	680
XC5VFX30T	80 x 38	5,120	380	64	136	68	2,448	2	1	1	4	N/A	8	12	360
XC5VFX70T	160 x 38	11,200	820	128	296	148	5,328	6	1	3	4	N/A	16	19	640
XC5VFX100T	160 x 56	16,000	1,240	256	456	228	8,208	6	2	3	4	N/A	16	20	680
XC5VFX130T	200 x 56	20,480	1,580	320	596	298	10,728	6	2	3	6	N/A	20	24	840
XC5VFX200T	240 x 68	30,720	2,280	384	912	456	16,416	6	2	4	8	N/A	24	27	960

FIGURA 5.4: Familia de FPGA *Virtex-5*

Por otro lado, la naturaleza de nuestro diseño iba a requerir la implementación de varios filtros FIR. En una etapa temprana se investigó como realizar filtros FIR de alta performance en un FPGA sin utilizar multiplicadores por hardware y se llegaron a publicaciones que hablaban sobre aritmética distribuida, de las cuales destacamos *Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review* [41]. En esta técnica las operaciones aritméticas que aparecen en el procesamiento de señales (suma, multiplicación, etc.) se realizan de una manera diferente. Aún mejor, las operaciones más típicas que encontramos en el procesamiento de señales (sumas de productos) son las que la aritmética distribuida calcula de manera más eficiente. Básicamente, podría decirse que DA es un método de cómputo *serial* de bits que calcula el producto interno de dos vectores en único paso directo[7].

Xilinx contaba con una propiedad intelectual llamada *Distributed Arithmetic FIR Filter* que fue discontinuada pero antes fue incorporada a otra llamada *IP LogiCORE FIR Compiler*. En *System Generator* esta última se instancia a través del bloque *"FIR Compiler 5.0"*. Cuando se elige la opción de implementación de aritmética distribuida también nos da la posibilidad de que el filtro sea reconfigurable. Esta

[7] Al ser un único paso y procesar bit por bit, las operaciones no están "agrupadas" y ya no se puede distinguir "ahí está el multiplicador, etc.". Por este motivo es que se llama aritmética distribuida.

funcionalidad parece trivial en aritmética convencional, pero si vemos un ejemplo de diagrama en bloques de un filtro DA FIR (figura 5.5) observamos que la "inteligencia" está en la LUT. Entonces si decidimos cambiar los coeficientes del filtro, estos no sólo deben ser transferidos a una memoria interna sino que debe *recalcularse* el contenido de la LUT. Por lo tanto, debe tener una lógica de recálculo tal como se observa en la figura 5.6.

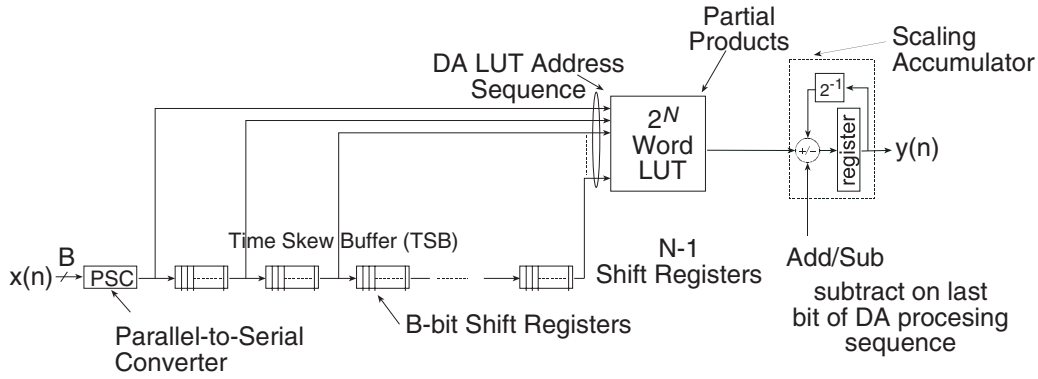


FIGURA 5.5: Filtro FIR serial de aritmética distribuida

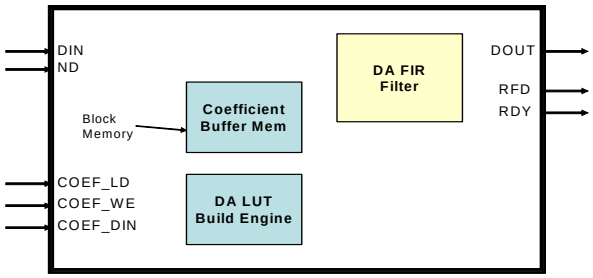


FIGURA 5.6: Filtro FIR de aritmética distribuida reconfigurable

Para darnos cuenta del impacto a nivel recursos que tiene la implementación en aritmética distribuida armamos dos diseños simples que contenían un filtro FIR de 29 coeficientes reconfigurables. En un caso utilizamos aritmética distribuida y en el otro elegimos *Transpose Multiply-Acumulate* que utiliza multiplicadores por hardware. En la tabla 5.1 se resumen los resultados. Podemos observar que la implementación con *slices* DSP48E ocupa casi una cuarta parte, un 26 %, en *slices*, no ocupa BRAM/FIFO pero sí consume 29 *slices* DSP48E. Por otro lado, la implementación en aritmética distribuida consume 1 BRAM/FIFO pero no utiliza *slices* DSP48E. Además, la versión *Transpose Multiply-Acumulate* funciona un 58 % más rápido. Estos diseños se implementaron para un FPGA XC5VLX110T-1.

Tipo de implementación	Slices	LUTRAM	BRAM/FIFO	DSP48E	Máx. Frec.
Transpose Multiply FIR 29 coef.	**402**	**936**	**0**	**29**	178,4 **MHz**
Distributed Arithmetic FIR 29 coef.	**1559**	**2236**	**1**	**0**	281,1 **MHz**

Tabla 5.1: Comparación de utilización de recursos en el FPGA. DA FIR *vs* Transpose Multiply DSP48E FIR. (XC5VLX110T-1)

5.3.5. Implementación de ruido Gaussiano

En §4.4.2 explicamos los distintos métodos que existían para generar números pseudoaleatorios con distribución Gaussiana. En particular hicimos foco en el método de Box-Muller (ver §4.4.2.1). También mencionamos la posibilidad de combinar algunos métodos para mejorar la calidad de los números. Por ejemplo, aplicar el Teorema Central del Límite a números generados por el método Box-Muller.

Xilinx System Generator provee un *Reference Blockset* que resuelve problemas típicos. Estos bloques abarcan implementaciones de filtros, CORDICs, Line Buffers, máquinas de estado y bloques de comunicaciones. Dentro de estos últimos se encuentra el bloque *"White Gaussian Noise Generator"* que se ve en la figura 5.7. Este bloque está basado en un *core* llamado *Additive White Gaussian Noise* que fue discontinuado[8]. La implementación del mismo se observa en la figura 5.8 y se basa en combinar cuatro bloques Box-Muller y sumarlos para aplicar el Teorema Central de Límite [20]. Los valores de la función de Box-Muller se almacenan en memorias ROM y el ruido uniforme está producido por *multiple-bit-leap-forward* LFSRs[9]. Dentro de sus especificaciones se destacan las siguientes:

- Período de la sequencia: aproximadamente $2^{190} \approx 1{,}57 \cdot 10^{57}$ muestras
- Función de densidad de probabilidad: se desvía menos del 0,2 % para $|x| < 4\sigma$ y se puede obtener un expresión cerrada de la misma.
- Densidad espectral de potencial plana

Por lo tanto, este generador de ruido Gaussiano blanco cumple con el requerimiento **RQ_17**.

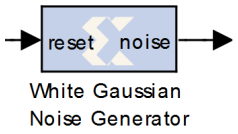


FIGURA 5.7: Bloque *"White Gaussian Noise Generator"*

[8] `http://www.xilinx.com/products/intellectual-property/DO-DI-AWGN.htm`

[9] este tipo de LFSRs genera más de una muestra por ciclo.

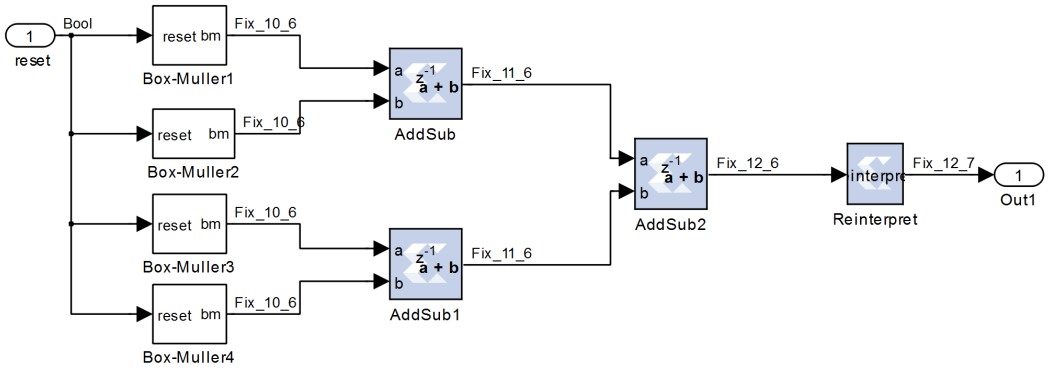


FIGURA 5.8: Interior del bloque *"White Gaussian Noise Generator"*

5.4. Implementación de la transferencia

La implementación de la transferencia acorde al diseño expuesto en §4.3.1 necesita de los siguientes componentes:

- un bloque que emule la característica multi-camino del canal BPL.
- un filtro pasabajos para emular la atenuación en función de la frecuencia que tienen los canales de las redes de acceso.

El bloque que implementa la transferencia se llama *"Forward Channel"*[10] y cuenta con una entrada y tres salidas tal como se ve en la figura 5.9. La entrada `in` es la señal que transmite el módem que está bajo prueba. Las salidas son:

- `delayed`: es la señal de entrada retrasada por todos los retardos programables (*debugging*).
- `out`: es la señal de entrada afectada por la característica multi-camino y el filtro FIR pasabajos.
- `multipath`: es la señal de entrada afectada solamente por la característica multi-camino (*debugging*).

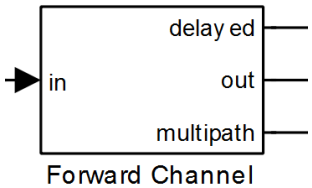


FIGURA 5.9: Bloque *"Forward Channel"*

[10] porque en este caso analizaremos el sentido *forward*.

En la figura 5.10 podemos observar como está compuesto el bloque que emula la transferencia. Allí se observa el bloque *"Multipath Characteristic"* y el bloque *"MicroBlaze Reloadable FIR Channel LP"*.

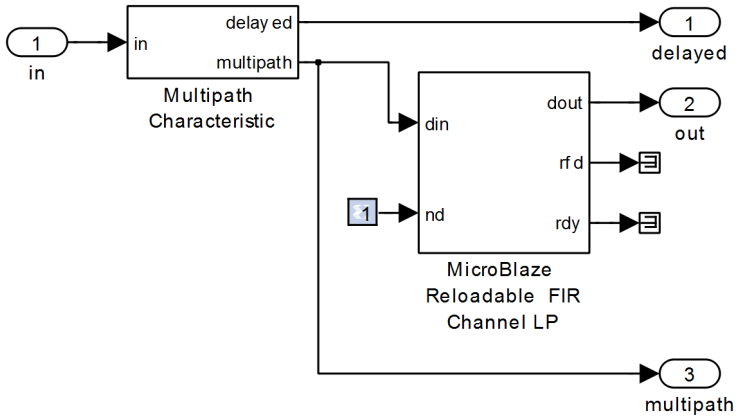


FIGURA 5.10: Subsistema *"Forward Channel"*

El bloque "Multipath Characteristic" es el encargado de aplicarle la característica multi-camino a la señal de entrada. Éste cuenta con una entrada, llamada `in`, y dos salidas llamadas `delayed` y `multipath` que son las mismas que las descriptas anteriormente. En las figuras 5.11 y 5.12 se puede observar la implementación de este bloque que consta de los siguientes componentes:

- 20 bloques que retrasan y multiplican la señal. Estos bloques también rutean la señal de entrada retrasada para que la procese el próximo bloque de la cadena.
- 2 memorias compartidas que se utilizan para almacenar los retardos y coeficientes de cada una de las 20 etapas.
- un *tree-adder* que suma la salida de cada una de las 20 etapas. Vale la pena mencionar que se adoptó esta estructura porque posee menor retardo que una cadena lineal como se observa en la figura 4.3.

En esta figura 5.12 se ha hecho uso de los puntos suspensivos "..." para no dibujar las 20 etapas. Las salidas `delay_value_o` y `coeff_value_o` de cada bloque *"delay mult subsystem"* son ruteadas hacia *Mux*s de MATLAB que combinan las señales en un vector. Dicho ruteo se realizó para poder armar un visualizador de los valores de los retardos programables y coeficientes durante una etapa de diseño en que se utilizó *Hardware Co-Simulation*.

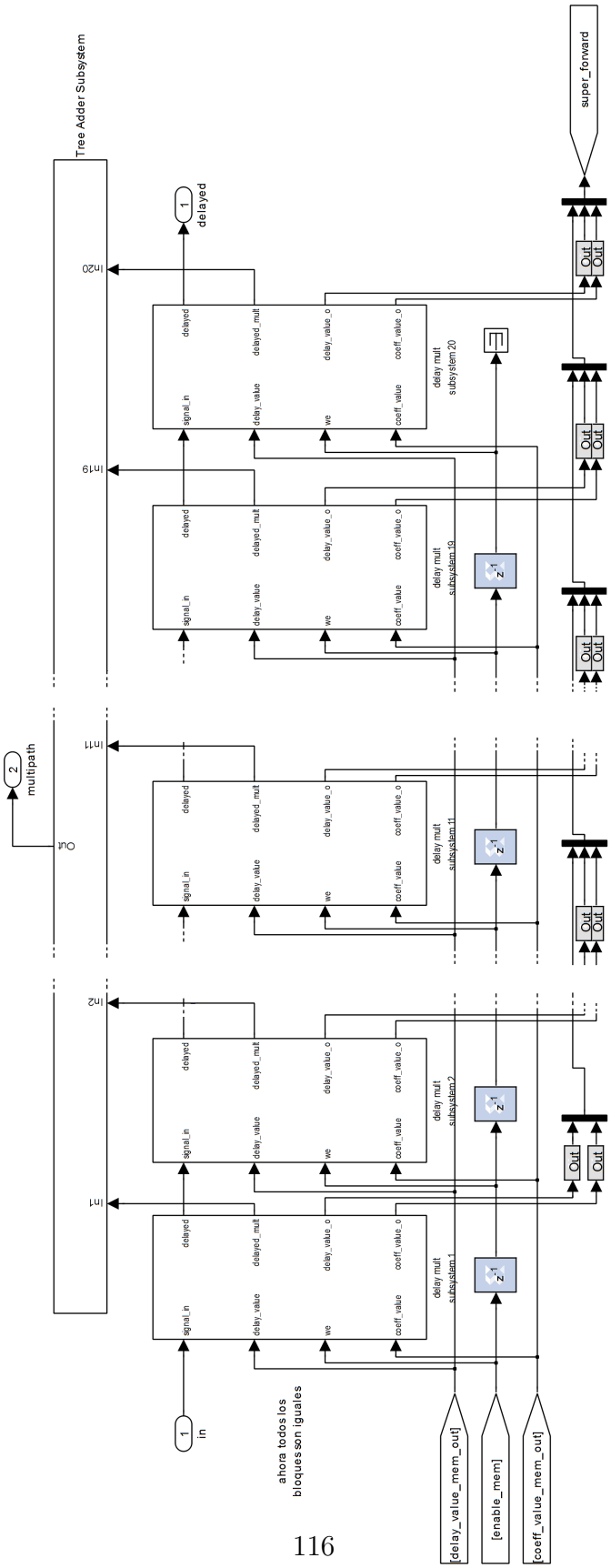


FIGURA 5.11: Subsistema *"Multipath Characteristic"*. Parte 1.

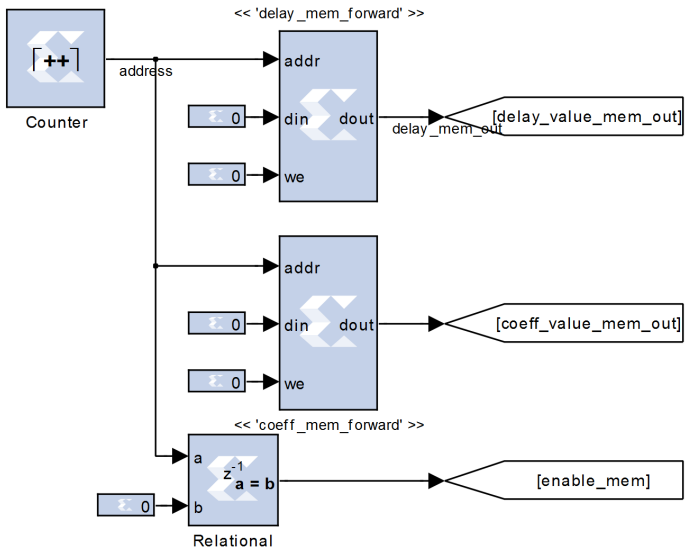


FIGURA 5.12: Subsistema *"Multipath Characteristic"*. Parte 2.

El bloque "MicroBlaze Reloadable FIR Channel LP" es un filtro FIR reconfigurable y cuenta con dos entradas y tres salidas. La entradas son:

- `din`: data in, es la señal a ser filtrada.
- `nd`: new data, es una señal que indica cuando hay datos en la entrada para procesar. Ésta es una señal del bloque de *Xilinx "FIR Compiler 5.0"* y la fijamos en "1" porque siempre vamos a estar procesando la señal de entrada. Si bien la hoja de datos dice que no debería ser "1" cuando `rfd` esté en "0", también dice que ignorará la entrada. Por lo tanto, no hay nignún problema.

Las salidas de dicho bloque son:

- `dout`: data out, es la señal filtrada.
- `rfd`: ready for data, nos indica cuando el bloque *"FIR Compiler 5.0"* está listo para aceptar datos de entrada.
- `rdy`: ready, nos indica cuando una nueva muestra de salida está disponible en `dout`.

En la figura 5.13 se puede observar la implementación del bloque *"MicroBlaze Reloadable FIR Channel LP"*. Este bloque se integra por los siguientes componentes:

- un bloque *"FIR Compiler 5.0"*. Este bloque es provisto por *System Generator* y es un *wrapper* del *core LogiCORE FIR Compiler 5.0*. Nosotros hemos configurado el filtro para que tenga 29 coeficientes y que su implementación sea de aritmética distribuida debido a la limitada cantidad de *slices* DSP48E. Estas *slices* son las que contienen los multiplicadores por hardware. También

podemos ver un bloque de tipo *cast* para cambiar el tipo de datos de la salida a `Fix_16_15` ya que el filtro sólo tiene precisión completa cuando su implementación es de aritmética distribuida.

- un bloque *From FIFO* que es una FIFO de 16 palabras que puede ser escrita desde el software de *MicroBlaze*. Este bloque es muy práctico debido al mecanismo interno de recarga de coeficientes del *"FIR Compiler 5.0"*. Entonces la lógica de recarga se reduce a lo que muestra la figura. Por ejemplo, si la FIFO se vacía, la señal `coef_we` toma el valor "0" y se suspende el proceso de carga. Cuando desde el *MicroBlaze* se escriben nuevas palabras y vuelve a haber más datos en la FIFO, se retoma la carga de coeficientes hasta la cantidad esperada.
- un *shared register* y un detector de flancos positivo que sirven para señalizar cuando reconfigurar los coeficientes del filtro.

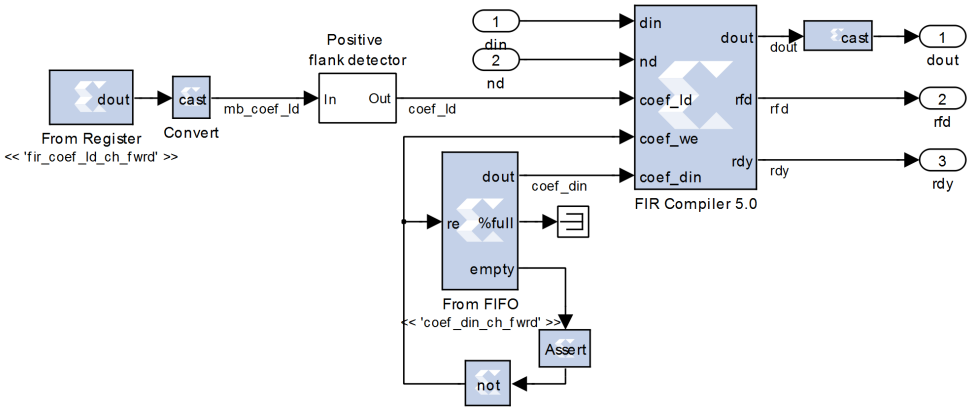


FIGURA 5.13: Subsistema *"MicroBlaze Reloadable FIR Channel LP"*

El tipo de datos de la salida es `Fix_16_15` debido a que la entrada es del mismo tipo y la atenuación promedio presente en cualquier canal da una salida con una potencia muy inferior. Los 16 bits se utilizan para conservar el rango dinámico elegido.

5.5. Implementación del ruido de fondo coloreado

En esta sección se mostrará la implementación de este ruido según el diseño expuesto en §4.5, pero la inclusión de ésta en la arquitectura final quedará definida según cómo se implemente el ruido de banda angosta.

Para la implementación de la arquitectura propuesta en el capítulo 4 se necesitan los siguientes componentes:

- un filtro FIR reconfigurable de 17 coeficientes
- una fuente de ruido Gaussiano blanco

El bloque que implementa este ruido tiene una salida, llamada `bg_noise`, que es el ruido de fondo coloreado tal como se ve en la figura 5.14. Por otro lado, en la figura 5.15 podemos ver como está compuesto dicho bloque. Allí se puede reconocer el generador de ruido Gaussiano blanco y el filtro FIR reconfigurable. El bloque *"MicroBlaze Realoadable FIR"* es similar al visto para la transferencia con la salvedad de que tiene 17 coeficientes. Este filtro también se implementa utilizando aritmética distribuida. También podemos ver un registro que sirve para cortar la cadena combinacional lógica y para sólo dejar pasar la salida cuando ésta sea válida.

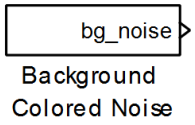


FIGURA 5.14: Bloque *"Background Colored Noise"*

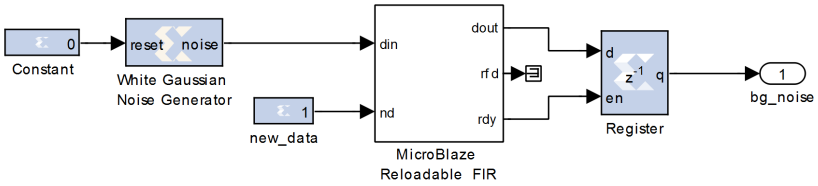


FIGURA 5.15: Subsistema *"Background Colored Noise"*

Recursos utilizados Los recursos utilizados para generar este ruido son

- 1479 Slices
- 6 BRAM/FIFO
- 4 DSP48E

El tipo de datos de la salida es `Fix_16_15` debido a que la potencia de salida es muy baja. Los 16 bits se utilizan para conservar el rango dinámico elegido.

5.6. Implementación del ruido de banda angosta

Para la implementación del generador de ruido de banda angosta se diseñaron ambas arquitecturas para su posterior comparación desde el punto de vista de la utilización de recursos. En el proceso de diseño se fueron realizando optimizaciones y se fueron comparando.

Para la generación de este tipo de ruido los documentos de OPERA [1, 3] a los cuales se tuvo acceso no explican por qué se eligió implementar este ruido utilizando la opción de superposición de señales de amplitud modulada. Además, dicen no

necesitar fuentes de ruido blanco Gaussiano y sólo hacen énfasis en la utilización de DDSs. Si bien dicen utilizar el modelo de señales sinusoidales moduladas con amplitudes y anchos de banda variables no especifican cómo implementan esto. Una posible explicación es que sólo utilicen DDSs e intenten controlar el ancho de banda de la señal a través de la entrada de fase, y la amplitud podría implementarse con algún multiplicador o mecanismo de *bit-shifting* si quieren ahorrar recursos a costa de flexibilidad. Aún si ese fuese el caso estarían diciendo que la señal de la entrada de fase no tendría naturaleza aleatoria. Nosotros atacaremos estas restricciones y nos impondremos como objetivo lograr una máxima funcionalidad a un costo razonable.

5.6.1. Generación por superposición de señales moduladas

El diseño planteado se observa en la figura 5.16 y consta de 4 generadores de interferencia de banda angosta. Cada generador es idéntico y sus salidas se superponen a través de la adición en un *tree adder*.

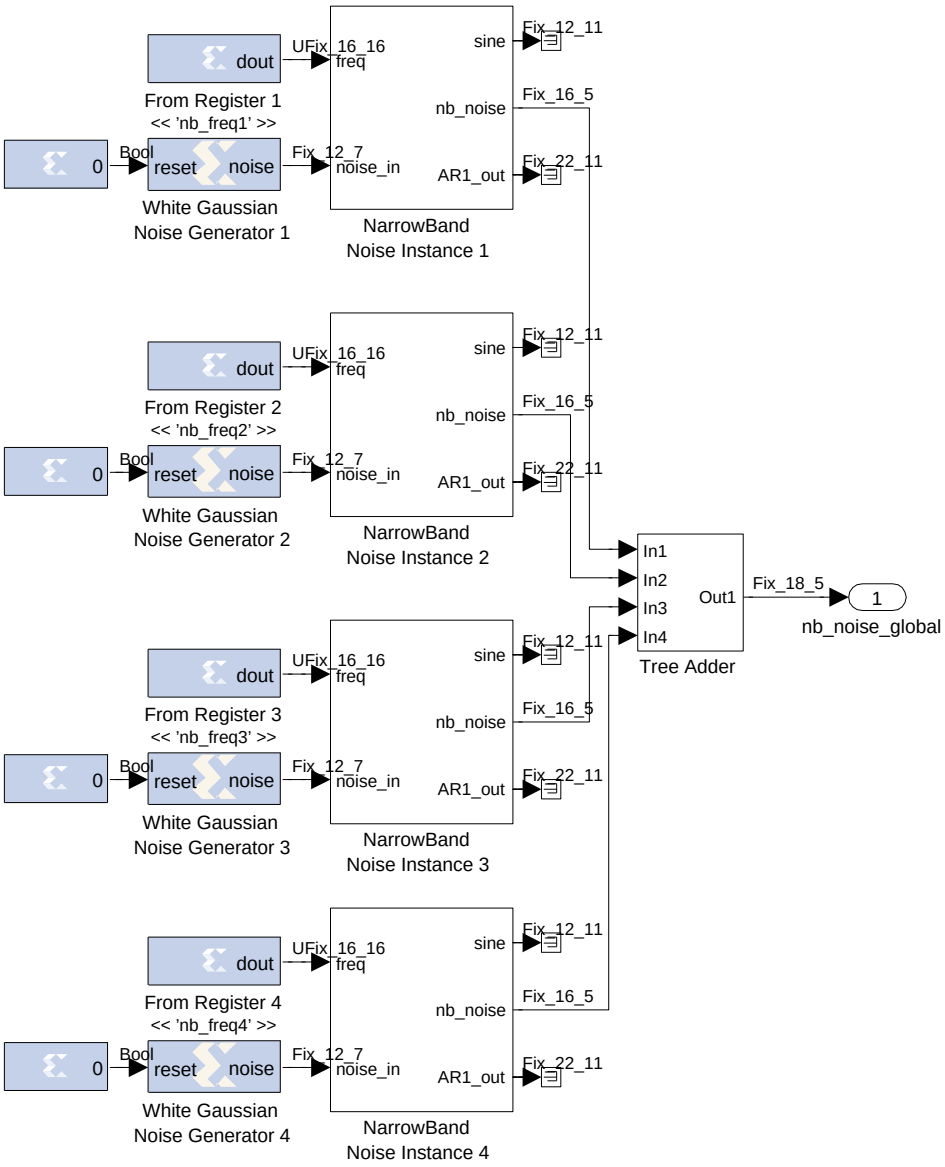


FIGURA 5.16: Implementación del ruido de banda angosta por modulación de señales

El bloque "NarrowBand Noise Instance" junto con el bloque *"White Gaussian Noise Generator"* forman un generador de ruido Gaussiano de banda angosta y cuenta con dos entradas y tres salidas, dos de las cuales se utilizan para debugging[11] y que al rutearlas hacia un terminador terminarán siendo *excluidas automáticamente* en el proceso de síntesis de hardware de la FPGA.

Las entradas de dicho bloque son:

[11] usos comunes son: la generación de gráficos a través de *Simulink Scopes* o la grabación de las señales en el *workspace* para poder trabajarlas offline.

- `freq`: toma una constante `UFix_16_16` que a través de un acumulador su resultado excitará un bloque de DDS. El rango posible va desde 2^{-16} hasta 0,5, donde para hallar la frecuencia en Hz debemos multiplicar dicha constante por la frecuencia de muestreo f_s. Entonces, para `freq`= 0,5 tendremos un valor de $\frac{f_s}{2}$ y será la frecuencia más alta representable, de acuerdo a lo esperado por el teorema de Nyquist. Y para 2^{-16} tendremos la frecuencia mínima que depende de un parámetro del DDS. En nuestro caso será $\frac{f_s}{2^{16}}$.
- `noise_in`: toma una señal de ruido Gaussiano aleatorio blanco de media nula y varianza unitaria.

Las salidas de dicho bloque son:

- `sine`: es la salida del bloque DDS. Debido a que la excitación de fase es lineal obtendremos como resultado una señal sinusoidal.
- `nb_noise`: ruido de banda angosta modulado.
- `AR1_out`: salida del proceso AR 1 que es modulado.

En la figura 5.17 se puede observar la implementación del bloque *"NarrowBand Noise Instance"*. Este bloque se integra por los siguientes componentes:

- un DDS para la generación de la portadora.
- el bloque *"AR1 Subsystem"*.
- *shared registers* que serán utilizados para la parametrización desde el software a través de un *soft-processor MicroBlaze*.
  - el registro `ar1_phi_1` controla el parámetro φ del proceso AR 1 (§D.1).
  - el registro `ar1_power_phi_1` es el encargado de regular la potencia del ruido de banda angosta. Este número se calcula a partir de dos valores: σ_{nb_1} y el factor de corrección $\sqrt{1-\varphi^2}$ cuya función es lograr que la potencia del proceso sea unitaria (D.3). De esta manera, el valor del registro será

    $$\texttt{ar1\_power\_phi\_1} = \sigma_{nb_1} \cdot \sqrt{1-\varphi^2}$$

    y la potencia de ruido será $\sigma^2_{nb_1}$. Vale la pena mencionar que este valor será precalculado fuera del software de *MicroBlaze*.
- multiplicadores implicados en el proceso de modulación y control de la potencia. Los retardos que tiene cada multiplicador son expresados en la notación de la transformada Z y se deben una implementación óptima desde el punto de vista de *pipelining* cuando se utilizan los multiplicadores por hardware. Para verificar si la cantidad de etapas es óptima, la interfaz del bloque de multiplicación nos permite configurar un parámetro para que haga este chequeo (figura 5.18). De haber configurado una cantidad de ciclos inferior a la óptima, la simulación no correrá y nos indicará los valores mínimos para el funcionamiento óptimo. Con estos valores elegidos se maximiza la velocidad a la que pueden funcionar los multiplicadores.

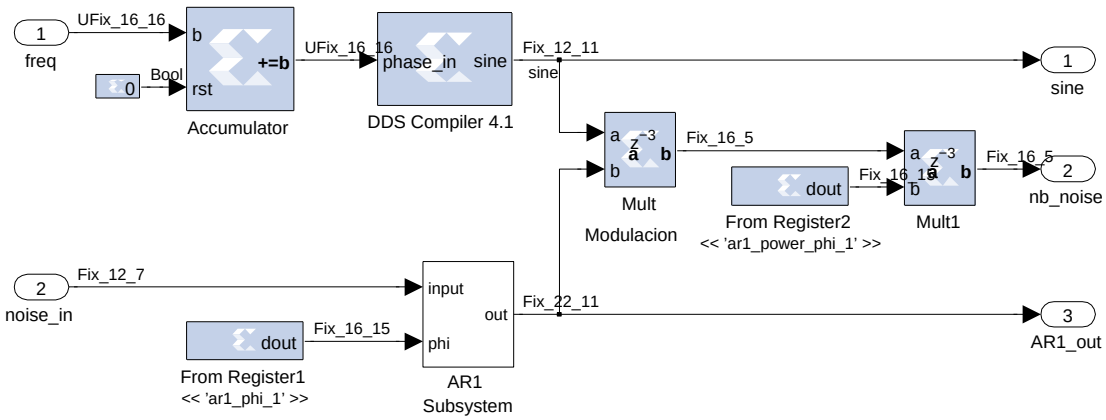


FIGURA 5.17: Subsistema "Narrowband Noise Instance"

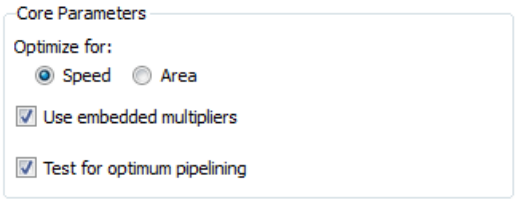


FIGURA 5.18: Opción de optimización de los multiplicadores por hardware

El bloque "AR1 Subsystem" implementa el proceso AR 1 y posee dos entradas y una salida. Las entradas de dicho bloque son:

- `input`: toma una señal de ruido Gaussiano aleatorio blanco de media nula y varianza unitaria.
- `phi`: es el valor del parámetro φ del proceso. Éste es controlado a través de un *shared register*.

Las salida de dicho bloque es:

- `out`: es la salida del proceso AR 1.

En la figura 5.19 se puede observar la implementación del bloque *"AR1 Subsystem"*. Este bloque se integra por los siguientes componentes:

- un sumador y un multiplicador que implementan $x[n] = \varphi \cdot x[n-1] + e[n]$
- dos bloques *convert*, uno para reducir la precisión de la salida y otro para controlar la precisión en el lazo de realimentación.

El estudio de por qué se utilizaron dichas precisiones numéricas se expondrá en el apéndice §E.1.

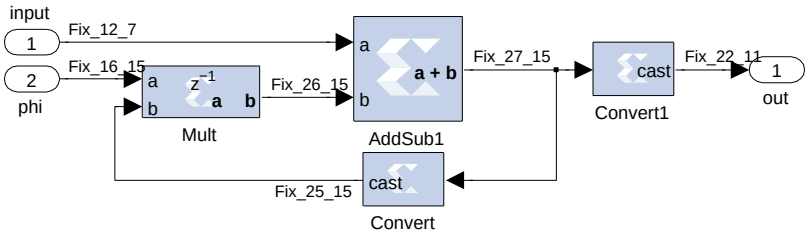


FIGURA 5.19: Subsistema "AR1 Subsystem"

Recursos utilizados Los recursos utilizados para generar 4 instancias de ruido de banda angosta son

- 3222 Slices
- 23 BRAM/FIFOs
- 36 DSP48Es. La descomposición de dicho valor es la siguiente, 9 por cada generador de ruido de banda angosta y cuyo detalle es:
  - 1 por cada "AR1 subsystem"
  - 2 por cada DDS
  - 2 por modulación y escalado
  - 4 por cada generador de ruido Gaussiano

5.6.2. Generación por IFFT

La otra alternativa propuesta para la generación del ruido de banda angosta es a través de la IFFT. Este diseño es adecuado para la generación de una gran cantidad de instancias de ruido de banda angosta. Una ventaja inicial es que requiere de una sola fuente de ruido Gaussiano blanco. Sin embargo, suele emplear mayores recursos para su implementación. Pero la única manera de saberlo es hacer el diseño completo y comparar la utilización de recursos.

El bloque *"FFT Narrow Band Noise Subsystem'* tiene dos salidas tal como se ve en la figura 5.20:

- `nb_noise`: es el ruido de banda angosta generado. El atractivo de esta implementación es que también podemos generar el ruido de fondo coloreado.
- `par_input`: es una salida de *debugging* que es la entrada de la parte real de la IFFT que fue convertida en una señal par por las razones expresadas en §3.8.2.

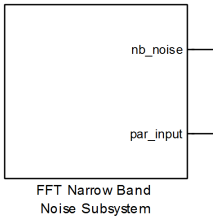


FIGURA 5.20: Bloque *"FFT Narrow Band Noise Subsystem"*

En la figura 5.21 podemos observar como está compuesto el bloque que genera el ruido de banda angosta y de fondo coloreado. Sus componentes son:

- una fuente de ruido Gaussiano blanco.
- el bloque *"AR1 Vector 8K Par with PowerScaling Subsystem"* es el encargado de adecuar la señal para la entrada de la IFFT.
- el bloque de *Xilinx "Fast Fourier Transform 7.1"* es un *wrapper* del *core LogiCORE IP Fast Fourier Transform v7.1*. Para ver consideraciones especiales de los parámetros referirse al apéndice §E.2.
- lógica para aplicar un *workaround* a un problema de numérico de truncamiento causado por usar un bloque *convert*.
- un multiplexor y un registro compartido que se utilizan para determinar si el bloque produce o no salida de ruido.

El bloque *reinterpret* antes de la IFFT se utiliza para llevar la entrada a un tipo de datos `Fix_(X+1)_X` que es requerido por el bloque *"Fast Fourier Transform 7.1"*[45]. El bloque *convert* después de la transformada se usa para que la potencia sea acorde al rango de nuestra especificación.

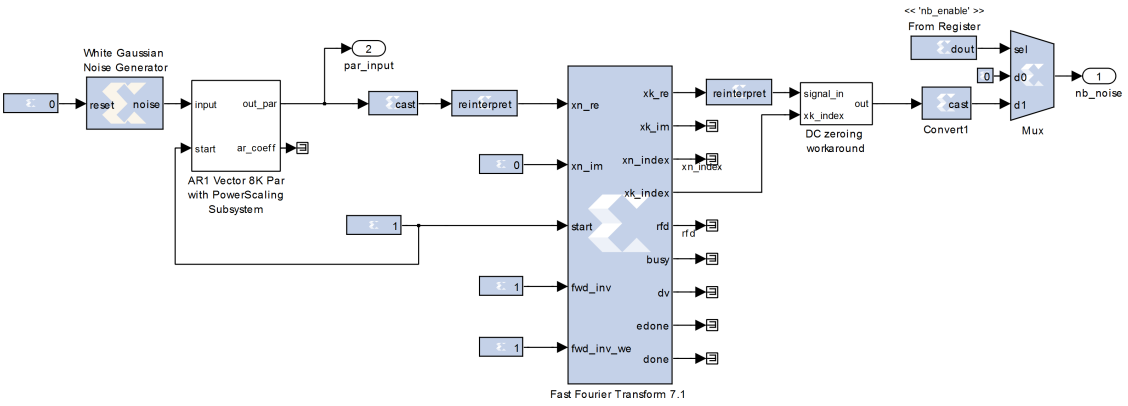


FIGURA 5.21: Subsistema *"FFT Narrow Band Noise Subsystem"*

El bloque "AR1 Vector 8K Par with PowerScaling Subsystem" construye la señal de entrada de la IFFT para que a la salida tengamos la densidad espectral de potencia deseada. Este bloque tiene dos entradas y dos salidas. Las entradas son:

- `input`: es la señal de entrada a los procesos AR 1. Esta debe ser ruido Gaussiano blanco.
- `start`: indica cuando empezar a armar la señal par. Se usa para estar en sincronismo con la entrada de la IFFT.

Las salidas son:

- `out_par`: es la señal par que antitransformada nos dará la PSD deseada.
- `ar_coeff`: esta es una señal ded *debugging* que muestra cúales son los coeficientes de cada proceso AR 1.

En la figura 5.22 se observa como está compuesto este bloque. El bloque toma la señal de entrada `input` que es un ruido Gaussiano blanco y de cada 16384 muestras sólo considera las primeras 8193 desde que la señal de `start` toma el valor "1"[12]. Esto ocurre cada bloques de 16384 muestras. Cada una de esas 8193 muestras constituyen la entrada de 8193 procesos AR 1 que funcionan a una tasa $\frac{f_s}{16384}$. A la entrada de cada uno de esos procesos se les controla la potencia mediante un factor de multiplicación ya que el ruido de entrada tiene potencia unitaria. Luego el bloque *"Par Output Maker 8K Subsystem"* se encarga de construir una señal par de 16384 muestras para la entrada de la IFFT. También es este bloque el que ignorará las últimas 8191 muestras de cada grupo de 16384 ya que la lógica del AR 1 en realidad procesa la entrada de manera continua.

En el bloque se observan dos memorias compartidas. Una para los coeficientes φ de los procesos AR 1 y otra para los coeficientes de multiplicación que controlan la potencia.

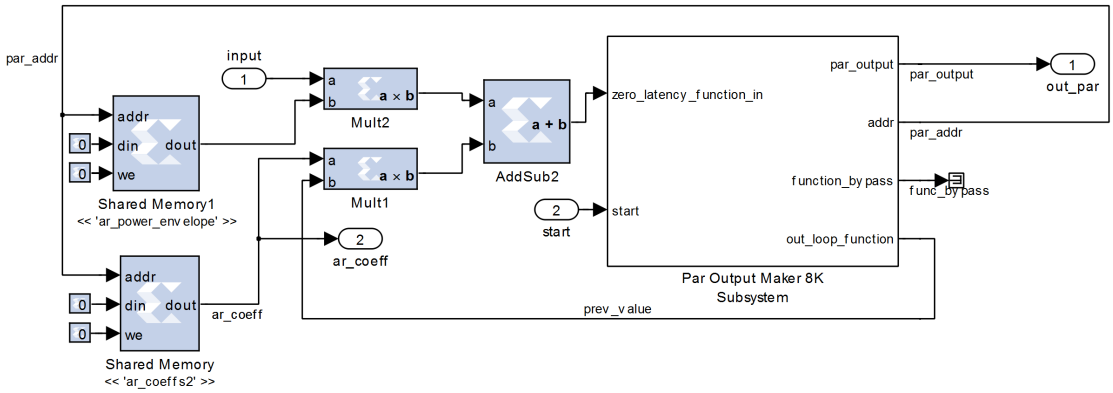


FIGURA 5.22: Subsistema *"AR1 Vector 8K Par with PowerScaling Subsystem"*

[12] esta señal debe mantenerse constante para el funcionamiento de este bloque

En la figura 5.23 se observa cómo está implementado el bloque *"Par Output Maker 8K Subsystem"*. Allí se ve la *Dual Port RAM* que se encarga de almacenar los 8193 valores. Estos valores se utilizarán para poder construir la señal par y para la próxima iteración de cada proceso AR 1. También se puede ver la lógica complementaria involucrada que maneja la memoria y que arma la señal par decidiendo si la señal de entrada pasa directo o se utiliza la información almacenada en la memoria.

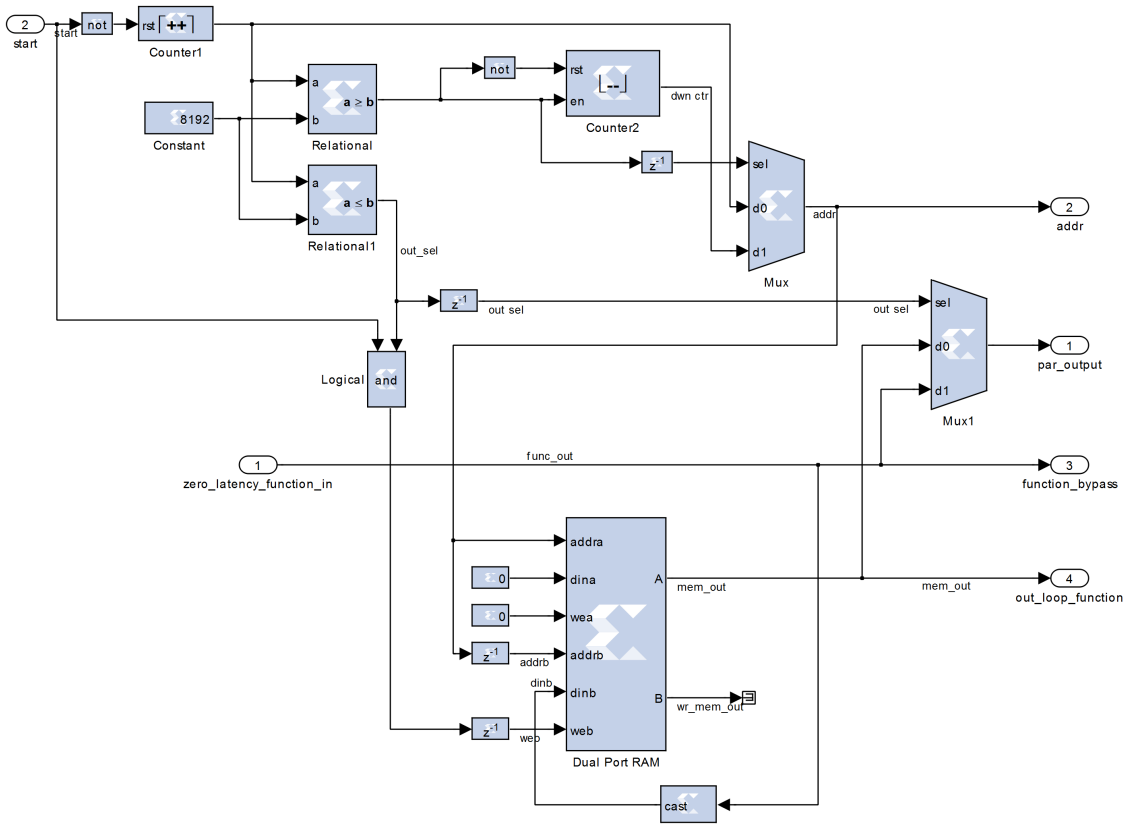


FIGURA 5.23: Subsistema *"Par Output Maker 8K Subsystem"*

El bloque "DC zeroing workaround" se utiliza para forzar a 0 el primer valor de la IFFT de 16384 puntos. El origen del problema es el bloque *convert*(o *cast*) entre la salida `out_par` y el bloque *reinterpret* (ver figura 5.21). Este bloque descarta bits menos significativos, es decir trunca por derecha, y por lo tanto la señal aleatoria pasa a tener una pequeña componente de continua que luego ésta es reflejada en la salida de la IFFT como un pico negativo en el primer valor tal como se observa en la figura 5.24. Este problema sólo se hace evidente cuando se quieren generar ruidos de muy baja potencia, de otra manera el fenómeno queda enmascarado. En la figura se compara contra la IFFT de MATLAB que utiliza la entrada sin truncar y vemos que no presenta dicho comportamiento. En la subfigura inferior vemos la diferencia entre

ambas IFFTs. La diferencia que se percibe como un ruido a lo largo de las 16384 muestras se debe a temas de precisión numérica. En primer lugar, la entrada de la IFFT de MATLAB no está truncada y tiene mejor resolución, y en segundo lugar las operaciones se calculan en punto flotante de doble precisión.

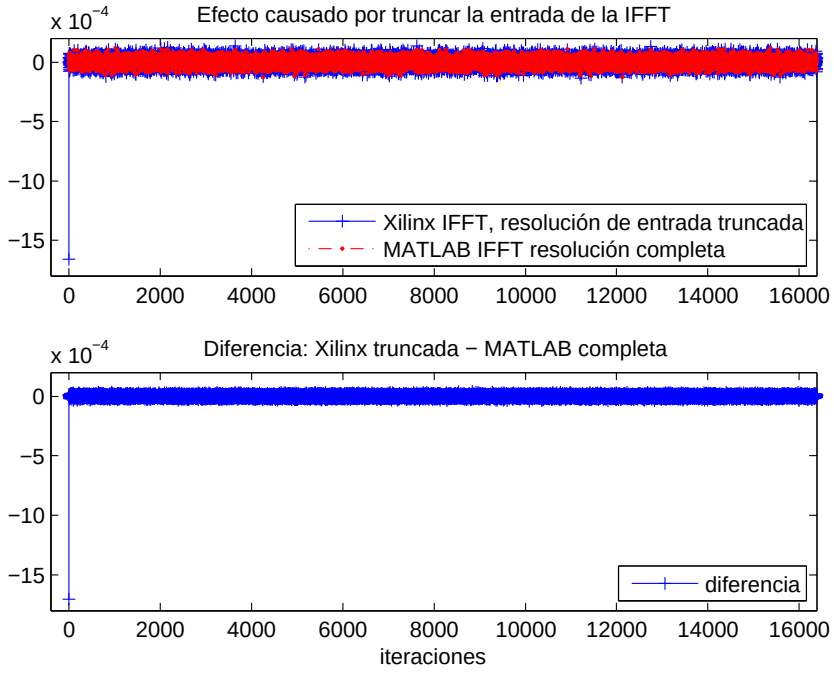


FIGURA 5.24: Problema númerico por truncamiento en la entrada de la IFFT.

El tipo de datos de la salida es `Fix_16_15`, ya que estando configurado a la potencia máxima de -64 dBm/Hz ninguna de las simulaciones mostró que se alcanzó la amplitud unitaria. Por lo tanto, la probabilidad de que ello ocurra es extremadamente baja y concluimos que `Fix_16_15` es el tipo de datos adecuado.

5.6.3. Comparación de recursos

En la tabla 5.2 presentamos un resumen de los recursos utilizados para cada una de las distintas alternativas. La primer alternativa es implementar el ruido de fondo coloreado según §5.5 y el de banda angosta por separado según §5.6.1. La segunda alternativa es implementar ambos ruidos utilizando la arquitectura de la IFFT.

Ruido	Slices	BRAM/FIFO	DSP48E
de fondo coloreado	1479	6	4
de banda angosta por superposición de portadoras	3222	23	36
subtotal	**4701**	**29**	**40**
de fondo coloreado y de banda angosta por IFFT	**3991**	**59**	**40**

Tabla 5.2: Utilización de recursos para el ruido de fondo coloreado y de banda angosta para las distintas alternativas

Si comparamos los valores vemos que son similares. Una opción consume más *slices* pero otra más BRAM/FIFOs. Sin embargo, la alternativa de la IFFT permite mucha mayor funcionalidad a un costo de 30 BRAM/FIFOs más. Nuestro chip posee 148 BRAM/FIFOs y por lo tanto decidimos que es un costo razonable por toda la funcionalidad extra que nos brinda.

5.7. Implementación del ruido impulsivo periódico sincrónico

La implementación de este ruido acorde al diseño expuesto en §4.7 necesita de los siguientes componentes:

- un generador de pulsos con período y ancho del pulso programables capaz de llegar a frecuencias de repetición tan bajas como 50 Hz.
- una fuente de ruido Gaussiano coloreado programable.

Ambos componentes son parametrizados a través de recursos compartidos.

El bloque que implementa este ruido tiene tres salidas tal como se ve en la figura 5.25

- `int rst`: internal reset, es una señal usada para *debugging*.
- `envelope`: es la envolvente que determina si hay o no hay impulso de ruido (*debugging*).
- `noise`: es el ruido impulsivo periódico sincrónico

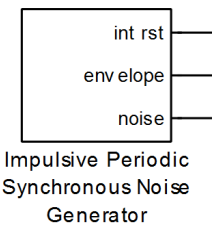


FIGURA 5.25: Bloque *"Impulsive Periodic Synchronous Noise Generator"*

En la figura 5.26 podemos ver como está compuesto el bloque que genera ruido impulsivo periódico sincrónico. Allí se puede observar el bloque *"Periodic Pulse Generator 24 bit"*, el bloque *"Colored Noise"* y la lógica complementaria que determina si la salida es ruido coloreado o es cero. También vemos los registros compartidos que permiten parametrizar la envolvente y determinar si el bloque está activado.

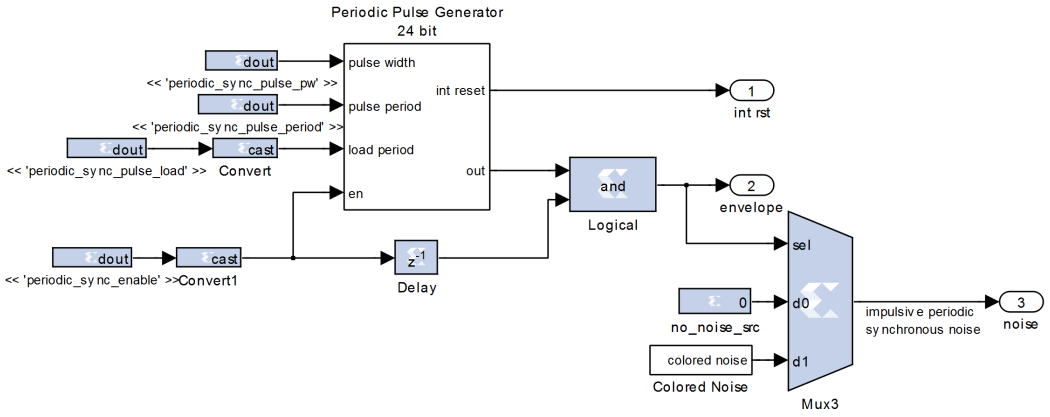


FIGURA 5.26: Subsistema *"Impulsive Periodic Synchronous Noise Generator"*

El bloque "Periodic Pulse Generator 24 bit" es el encargado de generar la envolvente periódica que determina cuando hay o no impulso y cuenta con cuatro entradas y dos salidas, una de las cuales se utiliza para *debugging* y que al rutearla hacia un terminador no tendrá relevancia en el proceso de síntesis de hardware de la FPGA. Las entradas de dicho bloque son:

- `pulse width`: se espera una señal constante cuyo valor es el parámetro que establece la duración del impulso. Si el valor en esta entrada vale N el impulso tendrá $N + 1$ ciclos de clock de duración.
- `pulse period`: se espera una señal constante cuyo valor es el parámetro que establece período de repetición de los impulsos. Si el valor en esta entrada vale N el período durará $N + 1$ ciclos de clocks.
- `load period`: cuando esta señal está en "1" se toman en cuenta los valores que están presentes en las dos entradas anteriormente descritas.
- `en`: enable, cuando esta señal está en "1" empieza a funcionar el contador interno.

Las salidas de dicho bloque son:

- `int reset`: internal reset, es una señal usada para *debugging* que indica cuando se resetean ambos contadores.
- `out`: es la envolvente periódica que se usa para generar impulsos.

En la figura 5.27 se puede observar la implementación del bloque *"Periodic Pulse Generator 24 bit"*. Este bloque está formado por un contador de 24 bits con reset sincrónico. A través de la utilización de comparadores y el reset es que se puede generar la envolvente con las características deseadas. Es importante que el contador sea de 24 bits para poder generar bajas frecuencias de repetición. Por ejemplo, si llegáramos a tener un clock de 100 MHz podríamos generar frecuencias de 6 Hz. Por lo tanto, estamos cubiertos para frecuencias bastante superiores sí así se necesitara en un futuro. La entrada enable `en` también sirve para poder darle una fase aleatoria de comienzo.

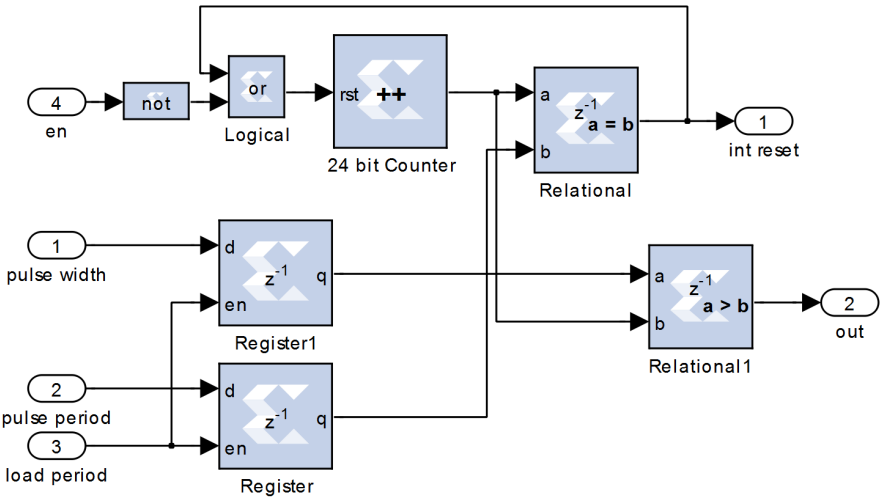


FIGURA 5.27: Subsistema *"Periodic Pulse Generator 24 bit"*

El bloque "Colored Noise" es el generador de ruido Gaussiano coloreado y cuenta con una sola salida llamada `colored noise`. En la figura 5.28 vemos que el bloque está compuesto por un generador de ruido Gaussiano blanco y un bloque que provee un filtro FIR reconfigurable, similar al utilizado para implementar la característica pasabajos de la transferencia en §5.4. Por lo tanto, no entraremos en detalles. La única diferencia es que éste posee 21 coeficientes en vez de 29. También está implementado utilizando aritmética distribuida. Podemos observar un registro a la salida que sirve para cortar el camino combinacional lógico. La potencia de este ruido se puede regular a través de los coeficientes del FIR ya que el generador de ruido Gaussiano blanco tiene potencia unitaria.

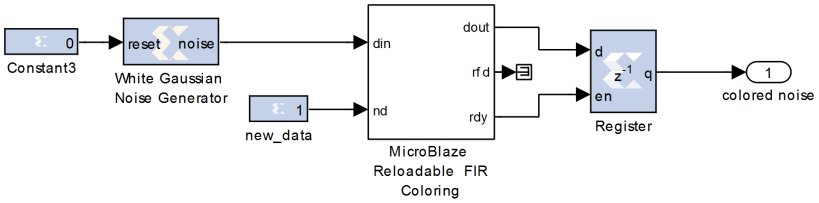


FIGURA 5.28: Subsistema *"Colored Noise"*

El tipo de datos de la salida es `Fix_16_15` tomando en cuenta que la amplitud está expresada en Volts y que los valores que suele manejar este tipo de ruido se encuentran en las decenas de miliVolts.

5.8. Implementación del ruido impulsivo periódico asincrónico

La implementación de este ruido acorde al diseño expuesto en §4.8 necesita de los mismos componentes que el ruido impulsivo periódico sincrónico en §5.7. La única diferencia es que el bloque que genera la envolvente aquí se llama *"Periodic Pulse Generator"* y utiliza un contador de 16 bits ya que no tiene el requerimiento de llegar a frecuencias tan bajas. De hecho, suponiendo un clock de 100 MHz la frecuencia más baja sería aproximadamente de 1526 Hz.

El tipo de datos de la salida es `Fix_16_15` tomando en cuenta que la amplitud está expresada en Volts y que los valores que suele manejar este tipo de ruido se encuentran en las decenas de miliVolts.

5.9. Implementación del ruido impulsivo aperiódico asincrónico

Para la implementación de este ruido según el diseño en §4.9 se necesitan los siguientes componentes:

- un generador de ruido uniforme para alimentar la cadena de Markov.
- una cadena de Markov conformada por las memorias que almacenan la matriz de probabilidades acumuladas de transición y una lógica que determina el estado y qué probabilidades condicionales utilizar en función del estado en que se está.
- una fuente de ruido Gaussiano coloreado programable.
- un generador de números con distribución exponencial para determinar la amplitud del impulso.
- un decisor que determina si la salida es ruido o no.

El bloque que implementa este ruido tiene cinco salidas tal como se ve en la figura 5.29:

- `random`: es el ruido uniforme [0,1) que alimenta la cadena de Markov.
- `envelope_o`: es la envolvente que determina si hay o no hay impulso de ruido.
- `state_o`: es el estado de la cadena de Markov, que es un valor numérico que va de 0 a 5.
- `imp_noise`: es el ruido impulsivo aperiódico asincrónico
- `amplitude`: es el multiplicador de amplitud correspondiente a cada impulso. Esta señal sólo cambia cuando hay una nueva envolvente de impulso.

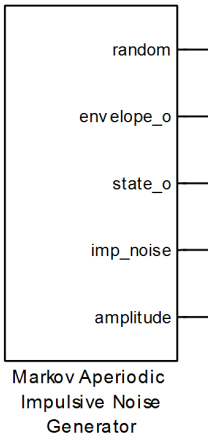


FIGURA 5.29: Bloque *"Markov Aperiodic Impulsive Noise Generator"*

En la figura 5.30 podemos observar como está compuesto el bloque que genera el ruido impulsivo aperiódico asincrónico. Allí se puede reconocer el generador de ruido uniforme, el bloque que implementa la lógica de la cadena de Markov, el decisor, el generador de ruido Gaussiano coloreado y el generador de amplitudes con distribución exponencial.

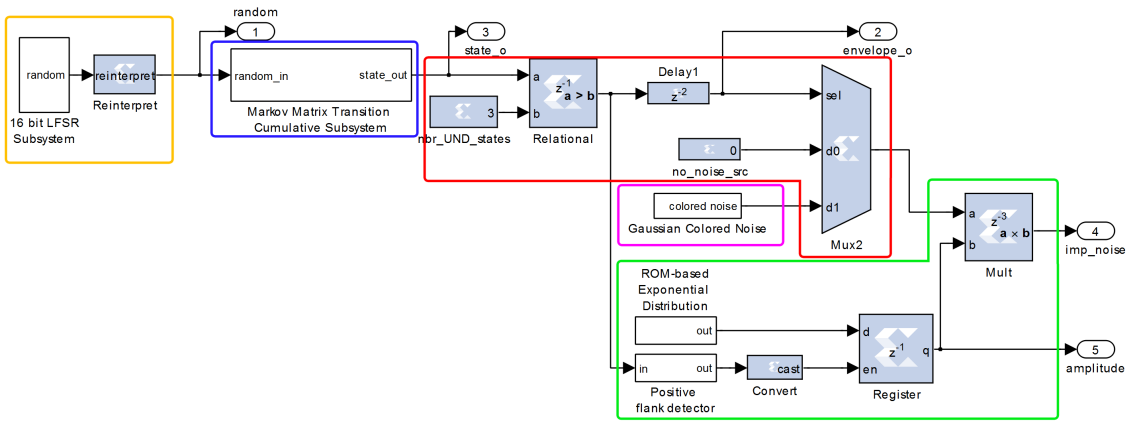
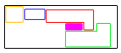
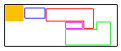


FIGURA 5.30: Subsistema *"Markov Aperiodic Impulsive Noise Generator"*



El bloque "Colored Noise" es idéntico al expuesto en §5.7 y por lo tanto no será explicado nuevamente.



El bloque"16 bit LFSR Subsystem" se compone de otros dos bloques tal como se ve en la figura 5.31

- *"LFSR Load Subsystem"*: este bloque se encarga de la lógica de programación de los 16 LFSR.
- *"16 programmable LFSR of 40 bits"*: está compuesto por 16 LFSR de Galois de 40 bits de longitud y concatena la salida para formar una palabra de 16 bits de ancho. Por lo tanto, este bloque generador de ruido verifica el requerimiento **RQ_16**.

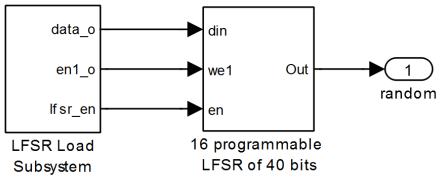


FIGURA 5.31: Subsistema *"16 bit LFSR Subsystem"*

En la figura 5.32 se observa el bloque *"LFSR Load Subsystem"* y sus salidas son:

- `data_o`: palabra de 40 bits a cargar como semilla.
- `en1_o`: pulso de enable para el primer LFSR.
- `lfsr_en`: habilita o no que los LFSRs funcionen.

Este bloque está formado por los siguientes componentes:

- dos memorias compartidas llamadas `lfsr_seed_1_upper` y `lfsr_seed_1_lower` que almacenan las semillas a programar. El motivo por el cual hay dos es para superar la limitación de 32 bits que tiene la API de *System Generator* que se intercomunica con el *soft-processor MicroBlaze*. Entonces una memoria almacena palabras de 8 bits mientras la otra tiene 32 bits.
- el *shared register* `lfsr_rst_1` que controla cuando recargar la semilla de los LFSRs.
- la lógica *one-shot* que se encarga de disparar y detener el contador que barre todas las direcciones de memoria posibles.
- lógica complementaria para generar las señales `en1_o` y `lfsr_en`.

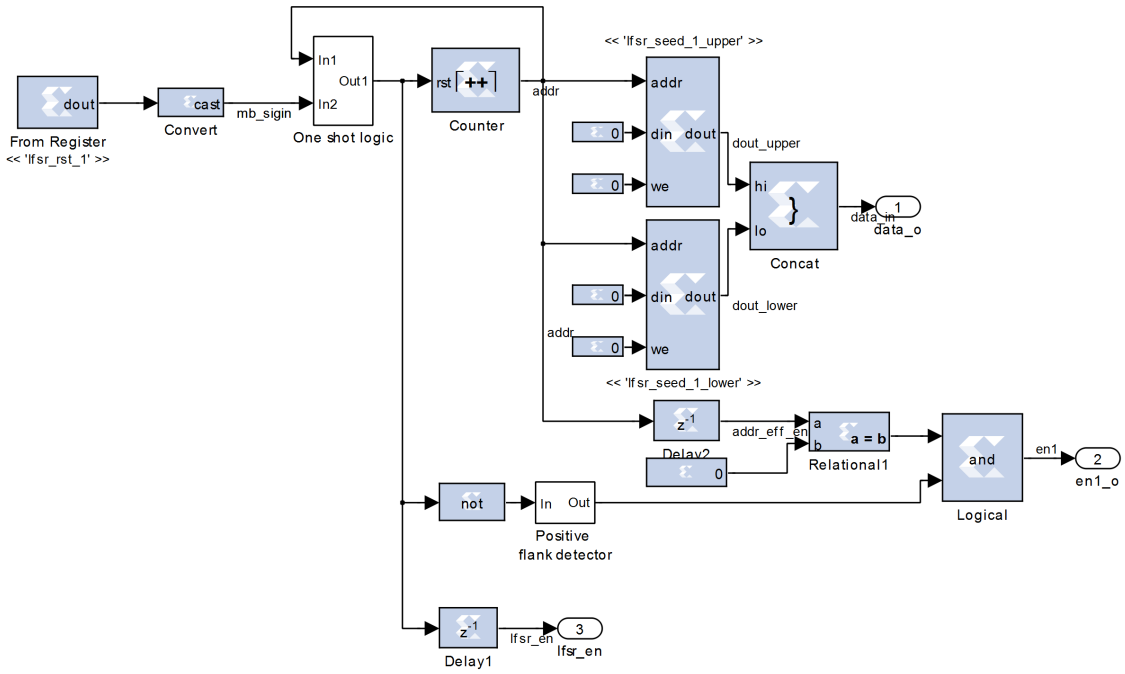


FIGURA 5.32: Subsistema *"LFSR Load Subsystem"*

El bloque "16 programmable LFSR of 40 bits" está compuesto por los 16 LFSRs y la concatenación de sus salidas para generar una palabra de 16 bits. Este bloque tiene 3 entradas:

- `din` palabra de 40 bit que será programada en un LFSR.
- `we1` entrada de pulso de un ciclo de duración que indica cuando programar el primer LFSR.
- `en` entrada que determina si los LFSRs pueden avanzar.

En la figura 5.33 se puede observar la implementación del bloque *"16 programmable LFSR of 40 bits"*. Allí se puede ver como el pulso que entra por `we1` se va propagando a través de una cadena de delays y habilita la programación sucesiva de cada uno de los LFSRs a medida que también vamos cambiando la palabra en la entrada `din`. El polinimio característico de los LFSRs de 40 bits que da una secuencia maximal que utilizamos es $G(x) = x^{40} + x^{38} + x^{21} + x^{19} + 1$ [43].

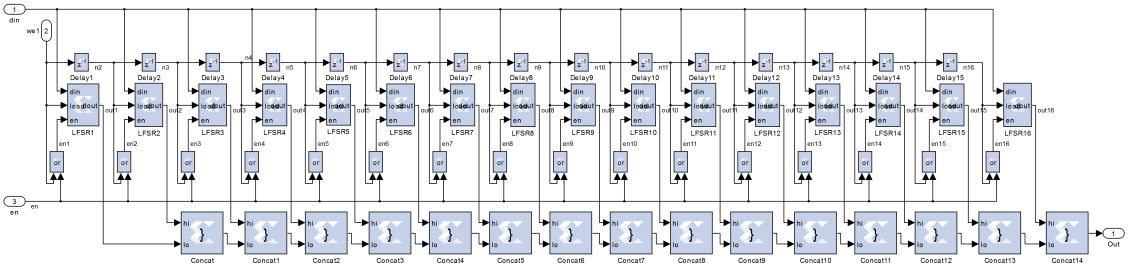
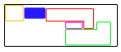


FIGURA 5.33: Subsistema *"16 programmable LFSR of 40 bits"*



El bloque "Markov Matrix Transition Cumulative Subsystem" implementa la lógica de la cadena de Markov y cuenta con una entrada y una salida. La entrada `random_in` espera ruido uniforme `UFix_16_16` y la salida `state_out` es el valor del estado en que se encuentra la cadena.

En la figura 5.34 podemos ver como está integrado el bloque *"Markov Matrix Transition Cumulative Subsystem"*. Sus componentes son:

- 5 memorias compartidas que se utilizan para almacenar las probabilidades acumuladas. Cada memoria contiene una columna de la matriz, de manera que para una determinada dirección de memoria la salida conjunta es una fila. Recordemos que cada fila de la matriz contiene las probabilidades condicionales acumuladas para transicionar al próximo estado. Vale la pena notar que se utilizan 5 memorias para 6 estados porque la última columna está compuesta por todos unos y se puede obviar[13]. También es importante mencionar el porqué del tipo de dato utilizado `UFix_17_16` en comparación contra `UFix_16_16` del ruido uniforme de entrada. La razón es que necesitamos distinguir al 1 de $(1 - 2^{-16})$ para representar eventos de probabilidad nula.

- el bloque de decisión "decidercumulative" compara la entrada de ruido con las probabilidades de transición condicionales acumuladas y determina el próximo estado.

- El multiplexor y los dos registros compartidos se utilizan como control. Con ellos se puede forzar la salida de este bloque a cualquier estado. Este control puede utilizarse para apagar este ruido si forzamos la salida a un valor de estado no perturbado.

[13] sería el caso del último **else** en el algoritmo 3 de la sección §4.9.

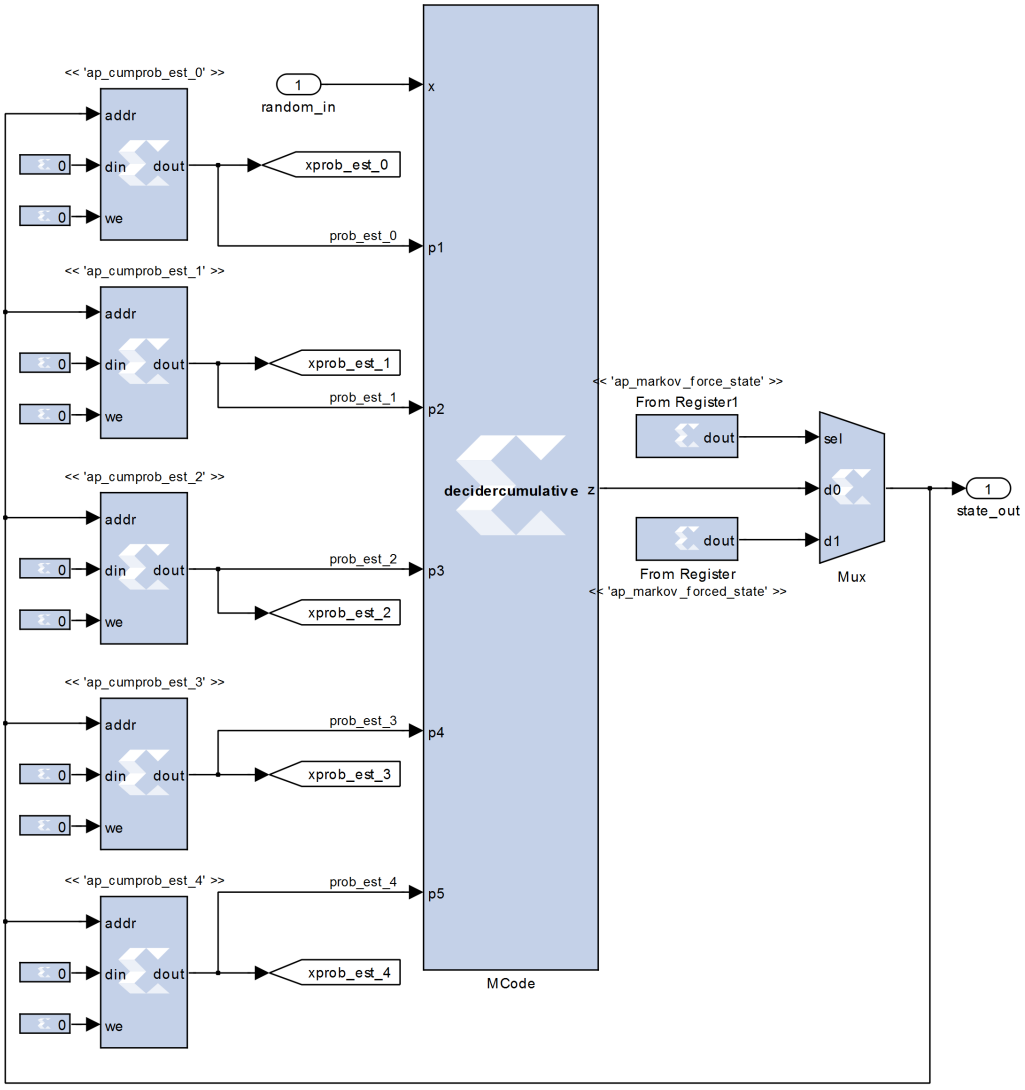
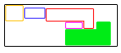


FIGURA 5.34: Subsistema *"Markov Matrix Transition Cumulative Subsystem"*



El bloque "ROM-based Exponential Distribution" se compone de un LFSR de 8 bits con semilla programable y una *lookup table* ROM que convierte la distribución uniforme en una distribución exponencial negativa. El subsistema del LFSR de 8 bits es similar al de 16 bits. Este bloque va acompañado de la lógica que fija un valor constante de amplitud durante la duración de un impulso. Esta última está implementada con un detector de flancos positivo y un registro con *enable* que almacena el valor sólo al principio de una envolvente de impulso. También se puede ver el multiplicador que toma como entradas al valor de amplitud calculado y a la señal de salida del multiplexor.

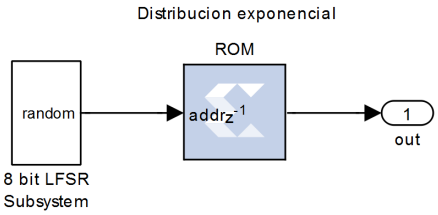
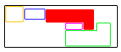


FIGURA 5.35: Subsistema *"ROM-based Exponential Distribution"*



La lógica de decisión que establece si hay ruido o no a la salida está compuesta por un comparador y un multiplexor. El multiplexor elige una de sus entradas en función de si el número de estado corresponde al grupo de estados perturbados o no.

El tipo de datos de la salida es `Fix_16_13`. Esto significa que podemos representar valores desde -4 a casi 4. Tomando que este valor está expresado en Volts y dada la estadística de amplitudes estudiada [1, 49] la probabilidad de superar 4 Volts es extremadamente baja. Sólo el 2 % de los impulsos superan 1 Volt.

5.10. Implementación del ruido impulsivo en ráfagas

Para la implementación de este ruido según el diseño en §4.10 se necesitan los siguientes componentes:

- dos generadores de ruido uniforme para alimentar a las dos cadenas de Markov.
- dos cadenas de Markov conformadas por las memorias que almacenan la matriz de probabilidades acumuladas de transición y una lógica que determina el estado y qué probabilidades condicionales utilizar en función del estado en que se está.
- una fuente de ruido Gaussiano coloreado programable.
- un generador de números con distribución exponencial para determinar la amplitud de la envolvente de la ráfaga

El bloque que implementa el ruido impulsivo en ráfagas tiene ocho salidas tal como se ve en la figura 5.36:

- `random01_1`: es el ruido uniforme [0,1) que alimenta la cadena de Markov de primer nivel.
- `random01_2`: es el ruido uniforme [0,1) que alimenta la cadena de Markov de segundo nivel.
- `state_out`: es la salida de la primer cadena de Markov, es decir la envolvente que determina si hay o no ráfaga.
- `state_out_hier`: es la salida de la segunda cadena de Markov, que se utiliza para describir los impulsos individuales dentro de una ráfaga.

- `unscaled_burst_noise`: es el ruido impulsivo en ráfagas sin ser multiplicado por la amplitud que determina la potencia de la ráfaga.
- `burst_envelope`: es la envolvente de los impulsos individuales de amplitud unitaria.
- `burst_amplitude`: es el multiplicador de amplitud correspondiente a cada ráfaga. Esta señal sólo cambia cuando hay una nueva ráfaga.
- `imp_burst_noise`: es el ruido impulsivo en ráfagas

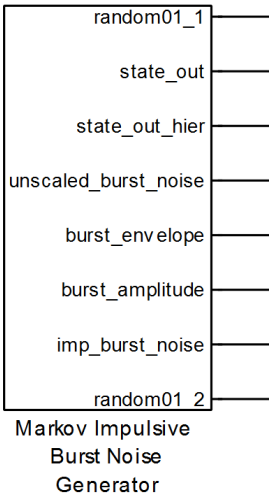


FIGURA 5.36: Bloque *"Markov Impulsive Burst Noise Generator"*

En la figura 5.37 podemos observar como está compuesto el bloque que genera el ruido impulsivo en ráfagas. Allí se pueden reconocer los dos generadores de ruido uniforme, el bloque que implementa la lógica de ráfaga e impulsos individuales y el generador de amplitudes con distribución exponencial. También se ve el generador de ruido Gaussiano coloreado y el multiplexor que determina si hay o no impulso de ruido.

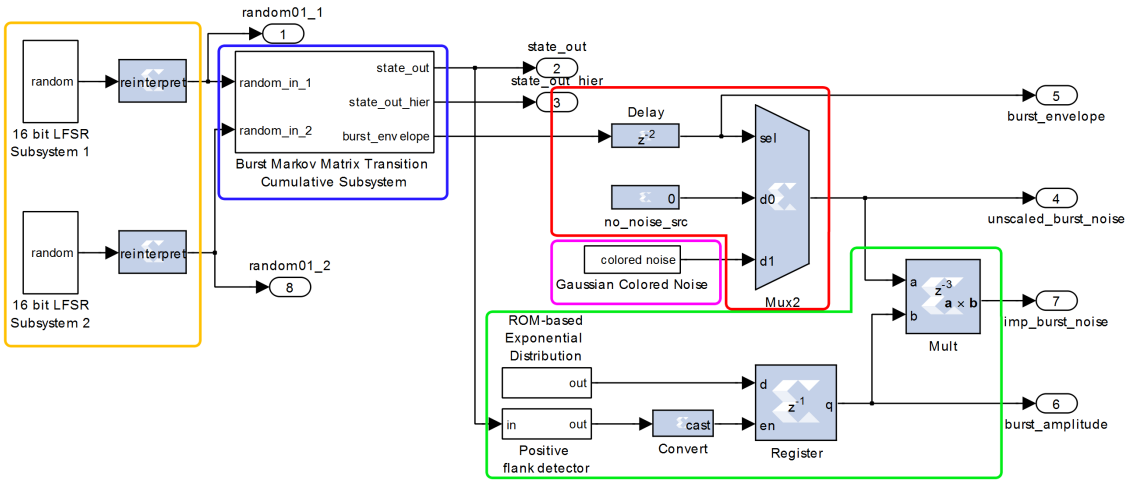
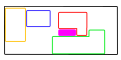
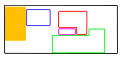


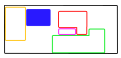
FIGURA 5.37: Subsistema *"Markov Impulsive Burst Noise Generator"*



El bloque "Colored Noise" es idéntico al expuesto en §5.7 y por lo tanto no será explicado nuevamente.



Los bloques "16 bit LFSR Subsystem 1" y "16 bit LFSR Subsystem 2" son idénticos al descripto para el ruido impulsivo aperiódico.



El bloque "Burst Markov Matrix Transition Cumulative Subsystem" implementa la lógica de las dos cadenas de Markov jerárquicas y cuenta con dos entradas y tres salidas. Las entradas de dicho bloque son:

- `random_in_1`: es el ruido uniforme que alimenta la cadena de Markov de primer nivel
- `random_in_2`: es el ruido uniforme que alimenta la cadena de Markov de segundo nivel

y las salidas son:

- `state_out`: es la envolvente de ráfaga, es el estado de la cadena de Markov de primer nivel
- `state_out_hier`: es el estado de la cadena de Markov de segundo nivel (*debugging*).
- `burst_envelope`: es la envolvente de los impulsos individuales

En la figura 5.38 podemos ver como está integrado el bloque *"Burst Markov Matrix Transition Cumulative Subsystem"*. Este bloque está formado por los siguientes componentes:

- una memoria compartida para cada una de las cadenas. Se utiliza una memoria ya que sólo hay dos estados[14]. En este caso no es necesario utilizar el tipo de

[14] ver explicación de porqué se utiliza una menos en §5.9

dato `UFix_17_16` ya que al ser una matriz de 2×2 no puede haber probabilidades nulas.

- los bloques que comparan las probabilidades con las entradas de ruido uniforme y deciden el estado de las cadenas de Markov. Este bloque es más simple debido a que sólo hay dos estados posibles.
- el multiplexor superior y los dos registros compartidos se utilizan como control, y con ellos se puede forzar un estado de ráfaga. A su vez, puede utilizarse para apagar este ruido si el estado forzado toma el valor cero.
- un detector de flancos positivos que está conectado a la envolvente de ráfaga. Éste junto con el multiplexor inferior se utilizan para forzar que haya un impulso individual al comienzo de cada ráfaga. De otra manera, tendríamos dos procesos de Markov independientes y en el comienzo de la envolvente de ráfaga podría no haber un impulso. En §7.7.1 veremos que el forzar el impulso no afecta prácticamente las características estadísticas de este proceso, debido a que las intervenciones ocurren con muy baja frecuencia y además es bastante probable que ya se encuentre en el estado de impulso.
- una compuerta `AND` que relaciona a los procesos de ambos niveles. El proceso que describe los impulsos individuales sólo produce salida si el proceso de primer nivel se lo permite.

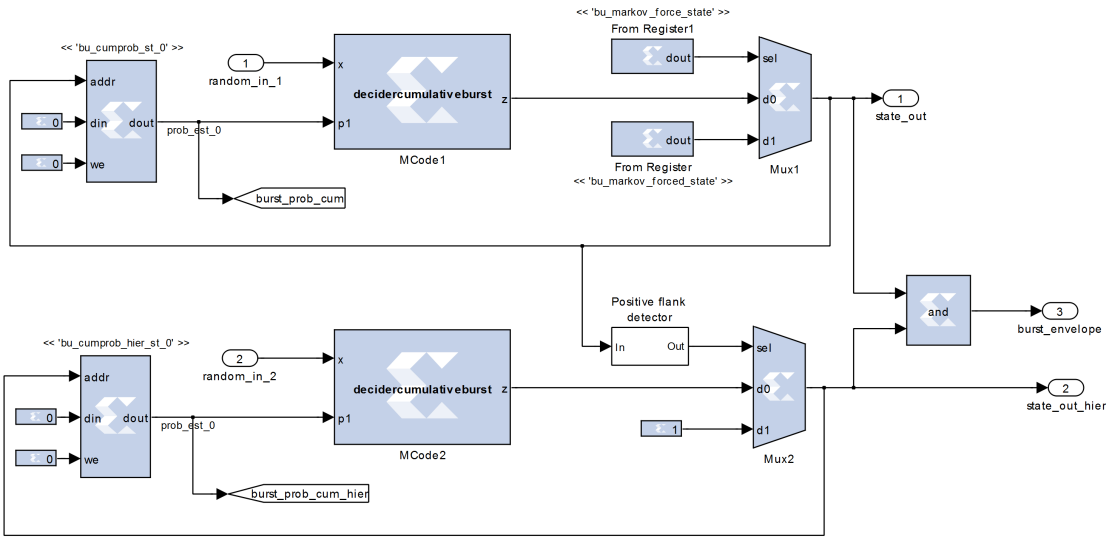
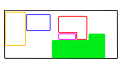
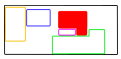


FIGURA 5.38: Bloque *"Burst Markov Matrix Transition Cumulative Subsystem"*



El bloque "ROM-based Exponential Distribution" y la lógica que retiene el valor de amplitud constante son idénticos al caso anterior, con la única salvedad que la amplitud sólo cambia al comienzo de una ráfaga y no de un impulso.



El multiplexor determina si hay o no impulso en función de la salida del bloque *"Burst Markov Matrix Transition Cumulative Subsystem"*. También podemos ver un delay que sirve para compensar por el retardo del detector de flancos y el registro que almacena el valor de amplitud. Una función adicional de éste es cortar el camino combinacional lógico.

El tipo de datos de la salida es `Fix_16_13` por las mismas razones expuestas en el ruido impulsivo aperiódico asincrónico.

5.11. Recursos utilizados en el FPGA

A medida que avanzábamos en la implementación de los distintos módulos nos dimos cuenta que la cantidad de recursos de nuestro FPGA no iba a permitir una implementación en ambos sentidos. Por lo tanto, se decidió implementar un solo sentido. Los parámetros para la síntesis especificaron un esfuerzo de *place-and-route* alto como se ve en las líneas del reporte.

```
Overall effort level (-ol):   High
Router effort level (-rl):    High
```

En la tabla 5.3 observamos un extracto del reporte de utilización jerárquica. Vale la pena aclarar que a dichas cifras hay que tomarlas como aproximaciones ya que si distintos módulos o submódulos comparten *slices* éstas son contabilizadas dos veces. Por ejemplo, en este caso la suma de *slices* de nuestra tabla da 18967 mientras que el dispositivo sólo contiene 17280. Hay que dejar en claro que lo que mostramos es sólo una parte del reporte, por lo que la suma total sería un número aún más grande[15]. Sin embargo, estos números sirven para darnos una idea de la utilización de recursos por módulo. Vale la pena mencionar que algunos de los recursos compartidos están contabilizados en la jerarquía de cada módulo mientras que otros no lo están y se encuentran fuera de ella. Este comportamiento es ineludible a la herramienta.

Los módulos que más recursos consumen son la transferencia, el ruido implementado con la IFFT y el ruido en ráfagas con un promedio de 3800 *slices*. Luego viene el ruido impulsivo aperiódico asincrónico con alrededor 2700 *slices* y los tres restantes con un promedio de 1600 *slices*. Un factor de peso en el consumo de los recursos es la generación de las fuentes de ruido. El generador de ruido Gaussiano blanco y el LFSR de 16 bits de ancho consumen alrededor de 750 *slices* por instancia. Si consideramos que tenemos ocho y sumamos los LFSRs de 8 bits de la distribución exponencial eso representa el 35,3 % del tamaño de nuestro diseño. Otro factor de peso son los filtros FIR que a causa de no contar con multiplicadores por hardware debimos implementarlos en aritmética distribuida. Todos estos juntos representan un 27,9 %. En las figuras 5.39 y 5.40 se puede ver gráficamente como se distribuye la utilización según dos criterios distintos: por módulo y por función.

[15] de hecho da 19866 *slices*.

Módulo	Slices	LUTRAM	BRAM/FIFO	DSP48E
MicroBlaze	**1514**	**129**	**16**	**3**
Transferencia	**3891**	**2914**	**1**	**0**
FIR 29 coeficientes	1686	2258	1	0
Multipath	2205	656	0	0
Ruido de fondo coloreado y de banda angosta por IFFT	**3937**	**3646**	**53**	**40**
FFT	3121	3274	32	34
White Gaussian Noise Generator	760	372	7	4
Ruido de impulsivo periódico sincrónico	**1692**	**1536**	**8**	**4**
FIR 21 coeficientes	880	1164	1	0
White Gaussian Noise Generator	771	372	7	4
Ruido de impulsivo periódico asincrónico	**1655**	**1536**	**6**	**4**
FIR 21 coeficientes	888	1164	1	0
White Gaussian Noise Generator	781	372	5	4
Ruido de impulsivo aperiódico asincrónico	**2719**	**1599**	**7**	**5**
FIR 21 coeficientes	904	1226	1	0
White Gaussian Noise Generator	770	372	5	4
ROM-Based Exponential	367	0	1	0
16 bits wide Progammable LFSR	648	0	0	0
Ruido de impulsivo en ráfagas	**3559**	**1599**	**7**	**5**
FIR 21 coeficientes	931	1226	1	0
White Gaussian Noise Generator	764	372	6	4
ROM-Based Exponential	349	0	0	0
16 bits wide Progammable LFSR 1	723	0	0	0
16 bits wide Progammable LFSR 2	764	0	0	0

Tabla 5.3: Utilización de recursos en el FPGA por módulo

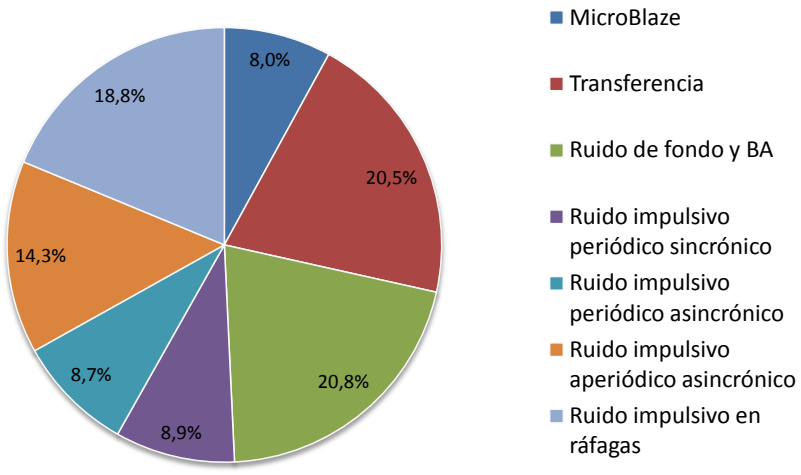


FIGURA 5.39: Utilización de recursos por módulo

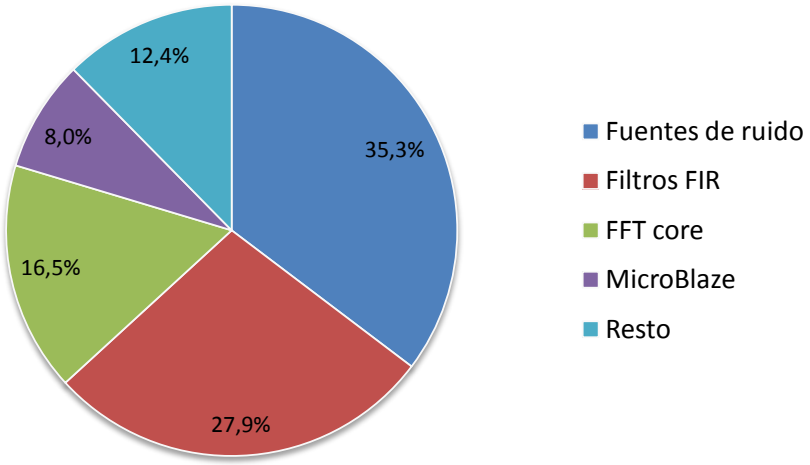


FIGURA 5.40: Utilización de recursos por función

En el reporte general aparece la verdadera utilización de recursos en el dispositivo que es lo que realmente importa. Esta información la hemos volcado en la tabla 5.4. Allí se ve el factor limitante de las *slices* DSP48E utilizado al 95 %, causa por la cual debimos implementar los filtros FIR sin multiplicadores por hardware. Otro factor limitante es la cantidad de BlockRAM/FIFO, donde el *core* de la FFT es el consumidor más importante. Debido a las métricas observadas y al elevado costo de los filtros FIR de aritmética distribuida[16] no se pudo vislumbrar una optimización para que puedan entrar ambos sentidos.

Descripción	**Usadas**	**Disponibles**	**Utilización**
Slice Registers	**41800**	**69120**	**60 %**
Slice LUTs	**39724**	**69120**	**57 %**
usadas como lógica	25647	69120	37 %
usadas como memoria	13187	17920	73 %
route-through exclusivas	890		
***Slices* ocupadas**	**14035**	**17280**	**81 %**
BlockRAM/FIFO	**128**	**148**	**86 %**
DSP48Es	**61**	**64**	**95 %**

Tabla 5.4: Utilización de recursos en el FPGA

[16] ver §5.3.4.

5.12. Diagrama en bloques

En la figura 5.41 observamos como queda el diagrama en bloques general. Allí se puede ver el bloque correspondiente a *Chipscope Pro*, los contadores para estadísticas de los ruidos impulsivos y el detector de flancos positivos que permite generar impulsos al presionar el botón `GPIO_SW_E` de la placa del kit XUPV5.

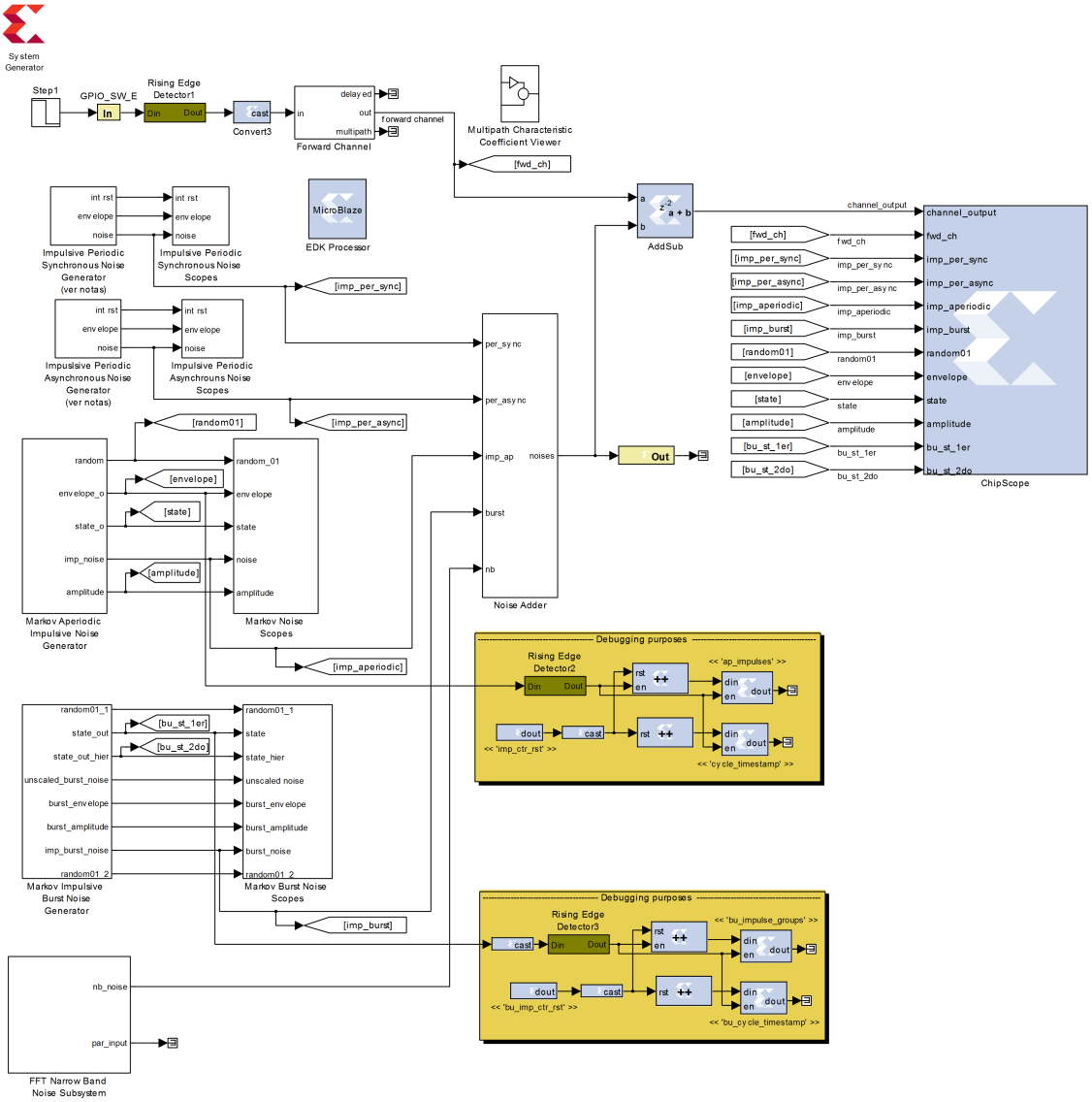


FIGURA 5.41: Utilización de recursos por función

5.13. Especificaciones y comparación contra el emulador de OPERA

Las especificaciones comparadas del emulador son:

- **Transferencia**: permite representar hasta 20 caminos y una longitud máxima de respuesta al impulso de 16 μs a 80 Msps. El filtro FIR programable de 29 coeficientes tiene 16 bits en lugar de los 8 que tiene el propuesto por OPERA.
- **Ruido de fondo coloreado**: se puede representar cualquier forma de densidad espectral de potencia gracias a que se implementa a través de la IFFT. Sus valores de potencia van desde los -140 dBm/Hz a -64 dBm/Hz considerando un ancho de banda de 40 MHz. Supera ampliamente la capacidad de emulación propuesta por OPERA en [3].
- **Ruido de banda angosta**: se implementa a través de una IFFTde 16384 puntos. Por lo tanto se tiene el control de 8192 tonos (excluimos la componente de continua) separados por 4,88 kHz a 80 Msps. La potencia de cada tono se controla entre -140 dBm/Hz a -64 dBm/Hz. También se puede agregar un proceso AR 1 a cada tono para reducir su ancho de banda a valores de aproximadamente 1 kHz. Supera ampliamente la capacidad de emulación propuesta por OPERA que utilizaba DDS y decía no requerir fuentes de ruido blanco Gaussiano. La implementación de OPERA no estaba claramente especificada por falta de información en [3].
- **Ruido impulsivo periódico sincrónico**: la frecuencia de repetición programable permite adaptarse a redes de 50 Hz ó 60 Hz y poder elegir el armónico que deseemos. La fuente de ruido Gaussiano permite ser coloreado mediante un filtro FIR programable de 21 coeficientes. OPERA decide no implementar este ruido por considerarlo despreciable al realizar un análisis espectral que suponemos que está basado en la autocorrelación promediada en un período[17] [33]. La razón por la cual lo consideran despreciable es que al promediar la autocorrelación, el tiempo en que no hay ruido disminuye el valor promedio significativamente y entonces la densidad espectral logra magnitudes comparables al ruido de fondo. Sin embargo, nosotros consideramos que el efecto disruptivo durante su ocurrencia no es despreciable y por este motivo decidimos implementarlo.
- **Ruido impulsivo periódico asincrónico**: la frecuencia de repetición programable permite valores desde los 1221 Hz hasta 10 MHz. La fuente de ruido Gaussiano permite ser coloreado mediante un filtro FIR programable de 21 coeficientes. En cambio OPERA establece frecuencias fijas 50 kHz, 100 kHz y 200 kHz. Además el ruido que utilizan es blanco. Por lo tanto, superamos las capacidades de emulación de [3].
- **Ruido impulsivo aperiódico asincrónico**: permite la carga de cualquier matriz de transición de probabilidades acumulada. Por lo tanto, se puede representar cualquier escenario contemplado por el modelo §3.11. El ruido de los

[17] debido a que es un proceso cicloestacionario y debemos independizarnos de la variable temporal para aplicar la transformada de Fourier.

impulsos proviene de un ruido Gaussiano blanco que es coloreador por un filtro FIR programable de 21 coeficientes. A su vez, las amplitudes de cada envolvente de impulso son gobernadas por una distribución exponencial. Además, las semillas de los generadores de ruido uniforme también son programables. El emulador de OPERA sólo tiene 3 escenarios *Moderate*, *Medium* y *Strong* y además el ruido es blanco [3]. Nuevamente nuestro emulador tiene mayor funcionalidad.

- **Ruido impulsivo en ráfagas**: permite la carga de cualquier matriz de transición de probabilidades acumulada tanto en el primer como en el segundo nivel. Por lo tanto, se puede representar cualquier escenario contemplado por el modelo §3.12. El ruido de los impulsos proviene de un ruido Gaussiano blanco que es coloreado por un filtro FIR programable de 21 coeficientes. A su vez, las amplitudes de cada envolvente de ráfaga son gobernadas por una distribución exponencial, poniendo en evidencia la correlación que existe entre los impulsos de una ráfaga. Además, las semillas de los generadores de ruido uniforme también son programables. En [3] se da a entender que para el proceso de segundo nivel se reutiliza la envolvente periódica del ruido periódico impulsivo asincrónico o se reutiliza el proceso de Markov del ruido impulsivo aperiódico asincrónico. Sin embargo, esto último parecería incompatible ya que el proceso de segundo nivel se caracteriza por tener una muy elevada tasa de impulsos y el anterior mencionado no es tal. De todas maneras, en nuestro emulador se implementan ambos procesos de Markov independientemente y por lo tanto tenemos una mayor funcionalidad que lo propuesto en [3].

- **Coloración espectral variable**: el hecho de que los filtros FIR que colorean el ruido Gaussiano blanco sean programables da la posibilidad de emular la variación espectral que sufren algunos ruidos en el tiempo.

- **Velocidad**: se ha logrado el objetivo de que el diseño implementado del emulador corra a 80 MHz. El camino crítico se presenta en el lazo que implementa el proceso AR 1.

5.14. Resumen

En este capítulo se describieron las herramientas de trabajo que utilizamos para desarrollar las implementaciones. También abordamos el tema de las restricciones que surgen a la hora de la implementación. Luego se presentaron las implementaciones en *System Generator* de las arquitecturas diseñadas en el capítulo 4. En algunos casos se planteó más de una alternativa de implementación para luego decidir por la más apropiada. Y finalmente mostramos la utilización de recursos de cada módulo. Debido al tamaño del diseño pudimos sintetizar un solo sentido del canal.

6

Interfaz de control del emulador

6.1. Motivación

Nosotros tuvimos dos motivaciones principales para el desarrollo de una interfaz de control. La primera era contar con una interfaz que permita que un usuario no interiorizado en el contenido de esta tesis pudiese usar y controlar el emulador que aquí desarrollamos. Y la segunda fue establecer una metodología general de control que sea extensible a otros diseños y que de esta manera facilitase dicha parte del diseño para los próximos proyectos del Laboratorio de Procesamiento de Señales de las Comunicaciones.

Para lograr el primer objetivo desarrollaremos un software con interfaz GUI que correrá en Windows y se comunicará a través del puerto serie RS232 con el kit XUPV5. De esta manera, el usuario se verá abstraído de lo que ocurre en el *back-end* y no tendrá que interactuar con ninguna herramienta específica de FPGAs, para cambiar de escenarios, sino con una simple aplicación de Windows.

El segundo objetivo se alcanza a través de un diseño general y la documentación de la implementación que explica cómo aplicar la metodología a futuros diseños.

La tercer motivación que siempre está presente es la de creación de una interfaz de control para abstraer a las personas de la ingeniería de detalle realizada. Cuando se habla de personas también se incluye a los diseñadores, ya que siempre es necesario ir creando capas de abstracción para visualizar los conceptos e ideas de manera más ordenada y manejable. Y la capa de abstracción más externa es la interfaz de control del emulador.

6.2. Diseño

El diseño planteado para poder controlar lógica de emulación del canal BPL es utilizar un microprocesador que se interconecte con la misma. De esta manera, el

diseño alcanza una gran flexibilidad debido a que las funcionalidades quedan definidas por el software y no por lógica de hardware. A su vez, hace más robusto al diseño debido a que los errores son fácilmente corregibles, algo que si hubiese ocurrido con una implementación de hardware implicaría realizar nuevamente la síntesis del diseño *completo*. También se podrían implementar funcionalidades más complejas impensadas para una implementación únicamente de hardware. En conclusión, la decisión estratégica de haber realizado esta parte del diseño del emulador con un microprocesador y un software es vital para la acelerar el desarrollo, mejorar la mantenibilidad, aumentar la capacidad funcional y toda otra mejora que surja de la flexibilidad de utilizar un software.

6.2.1. MicroBlaze

En el mercado existen circuitos integrados FPGAs que cuentan con microprocesadores físicamente embebidos, es decir que ya están en el silicio del chip. En nuestro caso, nuestro kit XUPV5 no contaba con dicha funcionalidad[1]. Sin embargo, Xilinx provee dos alternativas para implementar un *soft-processor*: *PicoBlaze* y *MicroBlaze*. Ambas alternativas son completamente implementadas en el FPGA y no requieren de componentes externos. Vale la pena mencionar que existen otros *cores*, ya sea comerciales o abiertos, que permiten implementar otras arquitecturas de *soft-processors*.

PicoBlaze es un microcontrolador de 8 bits RISC que es ofrecido de manera gratuita. La programación se realiza en assembler. Este microcontrolador está pensado para consumir pocos recursos y para ejecutar programas de baja complejidad debido a su simple arquitectura. Por lo tanto, tiene importantes limitaciones a nivel de funcionalidad de instrucciones. Además, su PROM sólo soporta un máximo de 18 mil instrucciones.

MicroBlaze es un microprocesador de 32 bits RISC de arquitectura Harvard. Este es ofrecido por Xilinx con su software con la opción Embedded Edition. Este cuenta con arquitecturas opcionales avanzadas como interfases PLB o AXI, Memory Management Units (MMU), caché de instrucciones y datos, profundidad de pipeline configurable, unidad de punto flotante y otras. Una de las características muy buenas a la hora de implementar un microprocesador dentro de un FPGA es que puedo excluir aquellas componentes que no vaya a utilizar y de esta manera se consumirán menos recursos. Para tener una idea de la capacidad de *MicroBlaze*, éste es capaz de correr Linux si está adecuadamente configurado.

[1] esto se debe al modelo de FPGA es el XC5VLX110T. Existen otros modelos de Xilinx como el XC5VFX100T que cuentan con un microprocesador PowerPC

6.2.2. Metodología General

Uno de nuestros objetivos principales era establecer una metodología general para el control de lógica de procesamiento de señales realizada en *System Generator*.

La metodología podría resumirse en las siguientes tareas:

1. Construir una plataforma *MicroBlaze* en la herramienta *Xilinx Platform Studio*. Ver §F.1.

2. Diseñar nuestro sistema de procesamiento de señales en *System Generator* considerando que será controlado por un *soft-processor MicroBlaze*. Eso implicará tener alguno de los siguientes componentes:

   - Shared Memory
   - Shared Register
   - FIFOs

   Ver §6.3 y para un ejemplo ver §6.3.1.

3. Importar la plataforma de *MicroBlaze* en *System Generator* a través de un bloque *"EDK Processor"*. Ver §F.2.

4. Desarrollar el software de control de *MicroBlaze* en C mediante el software *Xilinx SDK* y utilizando la API de interfaz con *System Generator* (ver apéndice §F.5). Para el debugging conjunto del diseño se puede utilizar la salida de compilación *Hardware Co-Simulation* ya que no es posible simular todas las funcionalidades del *MicroBlaze*. Ver apéndice §F.3. También es posible utilizar la salida de compilación *Bitstream* pero la diferencia es que toda la lógica está sintetizada en el FPGA y sigue corriendo. Ver apéndice §F.4.

6.3. Arquitectura

La arquitectura para el control de la lógica de procesamiento se basa en el uso de recursos compartidos en *System Generator* que permiten intercambiar información con el procesador *MicroBlaze*, que es representado por el bloque *"EDK Processor"*. Estos recursos compartidos, representados por bloques, también sirven para intercambiar información entre distintos modelos de *System Generator*, de ahí la razón por la que están apareados. El apareamiento se realiza a través de un nombre que debe ser común a ambos tipos de bloque *From-To*. Estos recursos son leídos desde el *MicroBlaze* a través un periférico especial que hace de interfaz entre el *soft-processor* y el diseño en *System Generator*.

Los bloques que intercambian información son

From Register implementa la salida de un flip-flop D. El valor que se escribe desde el software de *MicroBlaze*[2] llega al modelo de *System Generator*. Por ejemplo, así implementamos la parametrización de los bloques de ruido impulsivo periódico sincrónico y asincrónico.

To Register implementa la entrada de un flip-flop D. El valor que existe en el modelo de *System Generator* puede ser leído por el software del *MicroBlaze*.

Shared Memory implementa una memoria RAM compartida. Nosotros la utilizamos para almacenar datos correlacionados, como por ejemplo los elementos de un vector. En nuestro caso podrían ser los coeficientes de los delays programables de la transferencia o los factores multiplicadores en el bloque de ruido de fondo coloreado y de banda angosta. De esta manera evitamos tener muchos registros, que además sería ineficiente, y nos permite trabajar más fácilmente con información vectorial a través de punteros a nivel software.

From FIFO implementa la parte de salida de una cola FIFO. Los datos que se escriben desde el software de *MicroBlaze* llegan al modelo de *System Generator*. Estos son descargados a través de la señal `re`(*read enable*). Dicha señal puede activada en cualquier momento, por lo que existe la salida `empty` que nos permite saber si hay datos por leer. Por ejemplo, con este tipo de recurso se configuran los coeficientes de los filtros FIR reconfigurables.

To FIFO implementa la parte de entrada de una cola FIFO. Los datos que el modelo de *System Generator* escribe van a parar a la próxima ubicación de memoria disponible. Estos datos se leen desde el software de *MicroBlaze*. Este bloque no se utiliza en nuestro diseño.

Cruzando dominios de clock Al utilizar registros o memorias compartidas[3] existen dos posibilidades: que el par se encuentre en el mismo dominio de clock o en distintos. En el primer caso no habría problemas. En el segundo caso, como los clocks son asincrónicos y no tienen ninguna relación de fase hay que tomar medidas para evitar la corrupción de datos y la metaestabilidad. Estos inconvenientes ocurren porque no se respetan los *setup times* y *hold times*. Para el caso de las memorias se cuenta con la posibilidad de sintetizarla con mecanismos de bloqueo para luego utilizar circuitos de sincronismo que manejen estos. Y para el caso de los registros se pueden tomar medidas para reducir la probabilidad de metaestabilidad. Una acción mitigante sería agregar una cadena de registros en el dominio de clock destino (ver figura 6.1). Estos registros proveen tiempo adicional para que la metaestabilidad se resuelva.

[2] o desde un software corriendo en la PC o desde otro modelo de *System Generator*. En las próximas descripciones estas alternativas serán obviadas.

[3] Se excluye la FIFO ya que ésta fue diseñada para trabajar con clock asincrónicos y cuenta con mecanismos de sincronización.

También se suelen especificar *constraints* de tiempos para reducir la probabilidad de metaestabilidad en los archivos `.ucf`.

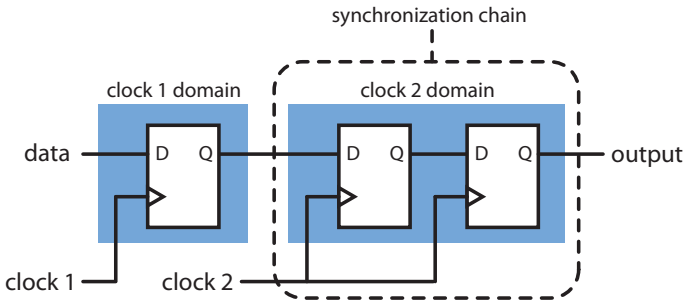


FIGURA 6.1: Cadena de sincronización.

En nuestro diseño la memoria compartida no es un problema ya que en la mayoría de los casos de uso ésta se lee luego de haberse escrito mediante una señalización aparte. Cuando se lee continuamente para programar los retardos y pesos de los caminos de la transferencia el valor inválido duraría sólo 20 ciclos lo que podría interpretarse como un *glitch* o un ruido espurio. Además, teniendo en cuenta la cantidad de ruidos que caracterizan este canal sumado a la baja variabilidad de la transferencia hacen que este problema sea despreciable.

6.3.1. Recursos para el control del emulador

El control del emulador está implementado a través de los siguientes recursos compartidos. A continuación describiremos los mismos agrupados por módulo funcional y cuál es el propósito de cada uno. Sólo se describirá el diseño en el sentido *forward* ya que sino sería redundante.

6.3.1.1. Módulo de la transferencia

Utiliza los siguientes recursos compartidos

From Registers

- `fir_coef_ld_ch_fwrd`: se utiliza para señalizar cuando se deben recargar los coeficientes del filtro FIR.

Shared Memory

- `delay_mem_forward`: almacena los 20 valores de delay programables.
- `coeff_mem_fowrward`: almacena los 20 coeficientes de multiplicación programables.

From FIFO

- `coef_din_ch_fwrd`: se utiliza para pasar los valores de los coeficientes en el proceso de recarga. Su tamaño, de 16 palabras, es menor que la cantidad de coeficientes del filtro. Además, es más conveniente utilizar una FIFO por cómo funciona el mecanismo de recarga.

6.3.1.2. Módulo de ruido de fondo coloreado y de banda angosta

Utiliza los siguientes recursos compartidos

From Registers

- `nb_enable`: determina si está o no activado esté ruido.

Shared Memory

- `ar_power_envelope`: almacena los 8193 valores de la envolvente de potencia del ruido blanco para cada tono.
- `ar_coeffs2`:almacena los 8193 valores de φ del proceso AR 1 para cada tono.

6.3.1.3. Módulo de ruido impulsivo periódico sincrónico

Utiliza los siguientes recursos compartidos

From Registers

- `periodic_sync_pulse_pw`: determina el ancho de la envolvente del impulso.
- `periodic_sync_pulse_period`: determina el período de la envolvente del impulso.
- `periodic_sync_pulse_load`: determina cuando se va a hacer un *latch* de los valores de los anteriores registros.
- `periodic_sync_enable`: determina si está o no activado esté ruido.
- `fir_coef_ld_per_sync_fwrd`: se utiliza para señalizar cuando se deben recargar los coeficientes del filtro FIR para colorear el ruido.

From FIFO

- `coef_din_per_sync_fwrd`: se utiliza para pasar los valores de los coeficientes en el proceso de recarga. Su tamaño, de 16 palabras, es menor que la cantidad de coeficientes del filtro (21).

6.3.1.4. Módulo de ruido impulsivo periódico asincrónico

Utiliza los siguientes recursos compartidos

From Registers

- `periodic_async_pulse_pw`: determina el ancho de la envolvente del impulso.
- `periodic_async_pulse_period`: determina el período de la envolvente del impulso.
- `periodic_async_pulse_load`: determina cuando se va a hacer un *latch* de los valores de los anteriores registros.
- `periodic_async_enable`: determina si está o no activado esté ruido.
- `coef_din_per_async_fwrd`: se utiliza para señalizar cuando se deben recargar los coeficientes del filtro FIR para colorear el ruido.

From FIFO

- `coef_din_per_async_fwrd`: se utiliza para pasar los valores de los coeficientes en el proceso de recarga. Su tamaño, de 16 palabras, es menor que la cantidad de coeficientes del filtro (21).

6.3.1.5. Módulo de ruido impulsivo aperiódico asincrónico

Utiliza los siguientes recursos compartidos

From Registers

- `ap_markov_force_state`: se utiliza para forzar el estado de la cadena de Markov al valor determinado por el registro `ap_markov_forced_state`.
- `ap_markov_forced_state`: es el valor del estado a forzar.
- `lfsr_rst_1`: se utiliza para señalizar cuando se deben recargar las semillas de cada LFSR.
- `lfsr8_rst_1`: se utiliza para señalizar cuando se deben recargar las semillas de cada LFSR que controlan la amplitud.
- `imp_ctr_rst`: se utiliza para señalizar cuando se deben resetear los contadores asociados con las estadísticas de impulsos.
- `fir_coef_ld_ap_fwrd`: se utiliza para señalizar cuando se deben recargar los coeficientes del filtro FIR para colorear el ruido.

To Registers

- `ap_cycle_timestamp`: cantidad de ciclos desde el último reset del contador o desde el encendido del emulador.
- `ap_impulses`: cantidad de impulsos desde el último reset del contador o desde el encendido del emulador.

Shared Memory

- `ap_cumprob_est_0`, `ap_cumprob_est_1`, `ap_cumprob_est_2`, `ap_cumprob_est_3`, `ap_cumprob_est_4`: almacenan la matriz de transición de probabilidades acumulada. Sólo se necesitan 5 memorias para usar 6 estados porque la última columna de matriz siempre son todos unos.
- `lfsr_seed_1_upper`: almacena los 8 bits más significativos de la semilla de 40 bits para cada uno de los 16 LFSRs. El hecho de que haya dos memorias se debe a una limitación de 32 bits que tiene para interactuar con el *MicroBlaze*.
- `lfsr_seed_1_lower`: almacena los 32 bits menos significativos de la semilla de 40 bits para cada uno de los 16 LFSRs.
- `lfsr8_seed_1_upper`: almacena los 8 bits más significativos de la semilla de 40 bits para cada uno de los 8 LFSRs que controlan la amplitud.
- `lfsr8_seed_1_lower`: almacena los 32 bits menos significativos de la semilla de 40 bits para cada uno de los 8 LFSRs que controlan la amplitud.

From FIFO

- `coef_din_ap_fwrd`: se utiliza para pasar los valores de los coeficientes en el proceso de recarga. Su tamaño, de 16 palabras, es menor que la cantidad de coeficientes del filtro (21).

Vale la pena notar que este módulo no posee un registro de *enable* porque dicha funcionalidad puede lograrse forzando a un estado de no ruido.

6.3.1.6. Módulo de ruido impulsivo en ráfagas

Utiliza los siguientes recursos compartidos

From Registers

- `bu_markov_force_state`: se utiliza para forzar el estado de la cadena de Markov del primer nivel al valor determinado por el registro `ap_markov_forced_state`.
- `bu_markov_forced_state`: es el valor del estado a forzar de la cadena de Markov del primer nivel.
- `lfsr_rst_burst_1`: se utiliza para señalizar cuando se deben recargar las semillas de cada LFSR del primer nivel.
- `lfsr_rst_burst_2`: se utiliza para señalizar cuando se deben recargar las semillas de cada LFSR del segundo nivel.
- `lfsr8_rst_2`: se utiliza para señalizar cuando se deben recargar las semillas de cada LFSR que controlan la amplitud.
- `bu_imp_ctr_rst`: se utiliza para señalizar cuando se deben resetear los contadores asociados con las estadísticas de ráfagas.

- `fir_coef_ld_bu_fwrd`: se utiliza para señalizar cuando se deben recargar los coeficientes del filtro FIR para colorear el ruido.

To Registers

- `bu_cycle_timestamp`: cantidad de ciclos desde el último reset del contador o desde el encendido del emulador.
- `bu_impulse_groups`: cantidad de ráfagas desde el último reset del contador o desde el encendido del emulador.

Shared Memory

- `bu_cumprob_st_0`: almacena la matriz de transición de probabilidades acumulada del primer nivel. Sólo se necesita 1 memoria para usar 2 estados porque la última columna de matriz siempre son todos unos.
- `bu_cumprob_hier_st_0`: almacena la matriz de transición de probabilidades acumulada del segundo nivel. Sólo se necesita 1 memoria para usar 2 estados porque la última columna de matriz siempre son todos unos.
- `lfsr_seed_burst_1_upper`: almacena los 8 bits más significativos de la semilla de 40 bits para cada uno de los 16 LFSRs del primer nivel. El hecho de que haya dos memorias se debe a una limitación de 32 bits que tiene para interactuar con el *MicroBlaze*.
- `lfsr_seed_burst_1_lower`: almacena los 32 bits menos significativos de la semilla de 40 bits para cada uno de los 16 LFSRs del primer nivel.
- `lfsr_seed_burst_2_upper`: almacena los 8 bits más significativos de la semilla de 40 bits para cada uno de los 16 LFSRs del primer nivel.
- `lfsr_seed_burst_2_lower`: almacena los 32 bits menos significativos de la semilla de 40 bits para cada uno de los 16 LFSRs del primer nivel.
- `lfsr8_seed_2_upper`: almacena los 8 bits más significativos de la semilla de 40 bits para cada uno de los 8 LFSRs que controlan la amplitud.
- `lfsr8_seed_2_lower`: almacena los 32 bits menos significativos de la semilla de 40 bits para cada uno de los 8 LFSRs que controlan la amplitud.

From FIFO

- `coef_din_bu_fwrd`: se utiliza para pasar los valores de los coeficientes en el proceso de recarga. Su tamaño, de 16 palabras, es menor que la cantidad de coeficientes del filtro (21).

Vale la pena notar que este módulo no posee un registro de *enable* porque dicha funcionalidad puede lograrse forzando a un estado de no ruido.

6.4. Software

En el diseño de un producto o de un prototipo electrónico, como es nuestro caso, es fundamental decidir en etapas tempranas qué funcionalidades se implementarán en software y cuáles en hardware. Estas decisiones no son triviales ya que no son puramente técnicas y entran otros factores como por ejemplo los económicos que las influencian. Un ejemplo es el costo de oportunidad que impone restricciones sobre los tiempos de desarrollo. Las implementaciones en hardware suelen ser más eficientes pero su diseño es de mayor complejidad ya que no se pueden permitir errores debido al muy elevado costo de los mismos. Por otro lado, el software puede permitirse tener *bugs* ya que estos pueden ser fácilmente resueltos con una actualización, y de esta manera poder salir antes al mercado con una versión no tan madura[4].

En nuestro proyecto, se podría decir que las decisiones no fueron tan difíciles. La simplificación no fue sólo por carecer de factores del mundo comercial sino que los requerimientos del sistema y las herramientas con que contábamos fueron la principal causa. En primer lugar, dados los requerimientos de cálculo y complejidad de las tareas, el procesamiento de las señales debía hacerse sí o sí en hardware. En segundo lugar, el control de esa lógica de procesamiento también planteaba dos alternativas de control, por hardware o por software. Dadas las facilidades que nos proveen herramientas de trabajo con las cuales trabajamos, la decisión más adecuada fue hacerlo por software incorporando un *soft-processor*. Si las condiciones hubiesen sido distintas y hubiésemos tenido que incorporar manualmente un *IP core* de algún procesador y haberlo "cableado" internamente con el diseño de *System Generator* trabajando con herramientas de más bajo nivel, y haber tenido que incurrir en el uso de otras herramientas externas para trabajar el software y el estudio de otros conocimientos, la decisión hubiese sido más compleja. Sin embargo, nosotros contamos con un grupo de herramientas que redujeron la complejidad de la decisión porque las mismas redujeron en primer lugar la complejidad de la tarea.

El uso de software disminuye la complejidad de la tarea y nos provee

- flexibilidad (diseño, expansión de funcionalidad, etc)
- capacidad de seguimiento y depuración
- capacidad de corregir errores a un muy bajo costo

6.4.1. Software en MicroBlaze

El diseño del software en *MicroBlaze* se realiza a través del *Xilinx SDK*, que es un IDE Eclipse[5] con plugins de Xilinx. Este entorno de desarrollo tiene muchas funcionalidades al ser un conjunto integrado de herramientas. Nosotros sólo comentaremos algunas de ellas a medida que las vayamos necesitando.

[4] Hoy en día esta práctica es tan frecuente que ya está socialmente aceptada. De hecho, todos saben que la primera generación es la que paga el precio.

[5] `http://www.eclipse.org`

Para trabajar con el SDK debemos tener sintetizado el *MicroBlaze* en el FPGA. Hay dos maneras de hacer esto:

1. Trabajando en modo *Hardware Co-Simulation*. Ver apéndice §F.3.
2. Sintetizando todo el diseño de *System Generator* utilizando el *compilation target Bitstream*. Ver apéndice §F.4.

El primero es muy útil en etapas tempranas ya que permite que parte del modelo se simule en MATLAB y otra parte corra en el FPGA. En nuestro caso sólo corríamos el *MicroBlaze* y todo el diseño restante en Simulink/*System Generator*. De esta manera podíamos, modificar partes del modelo sin tener que resintetizar[6]. Si bien esta modalidad de simulación con *hardware in-the-loop* fue inicialmente pensada para tener una aceleración de hardware hoy ofrece, en versión Beta, la posibilidad de hacer *co-debugging*, es decir permitir depurar el software en la co-simulación. Sin embargo, notamos que a medida que el diseño se hacía más complejo el software SDK se colgaba con mayor frecuencia y por lo tanto en etapas finales nos vimos obligados a utilizar la segunda opción.

La segunda opción es tener el diseño cargado en la FPGA y utilizar el SDK para desarrollar y depurar el código. Hay que tener en cuenta que ahora se está corriendo la lógica de procesamiento de señales a la frecuencia de clock, es decir a decenas de MHz en comparación con la simulación de MATLAB de algunos miles de ciclos por segundo (en el mejor de los casos). Sin embargo, en este modo perdemos las funcionalidades que nos da MATLAB para analizar y visualizar las salidas. Por este motivo, es que se usa en etapas finales donde el diseño de hardware ya fue verificado. En realidad, existe una manera de visualizar secuencias de algunas señales. Esto se logra a través de la tecnología *Chipscope Pro* que permite integrar un analizador lógico en el diseño. Sin embargo, en la práctica surgen problemas para correr el software en modo depuración porque comparten el mismo cable de JTAG y ambas herramientas se van "robando" la propiedad del cable.

Un hecho no menor a mencionar sobre la depuración es que ésta es bastante limitada, ya que pierde referencia del marco de memoria cuando entra a funciones y por lo tanto no se puede conocer el valor de las variables dentro del llamado de una función. Por lo tanto, para depurar hay que programar linealmente y luego que sabemos que funciona bien lo podemos encapsular en una función.

6.4.1.1. Software de control

El software de control tiene las siguientes funciones:

1. leer/escribir el valor de los recursos compartidos
2. comunicarse con el software GUI en la PC a través de RS232

[6] si se agregan o sacan recursos compartidos hay que reconfigurar la plataforma del *MicroBlaze* y por ende hay que resintetizar.

Cuando configuramos la plataforma elegimos no utilizar una memoria externa. Por lo tanto sólo tenemos disponible la memoria local que es de 64 KB. Esto implica que nuestro programa debe tener un tamaño reducido. Este hecho impide que la programación pueda realizarse en C++ debido al tamaño de sus bibliotecas. Por este motivo, la progamación no puede ser orientada a objetos. El programa se terminó desarrollando en C .

Interfaz del software La única interfaz que tiene este software es a través de la UART RS232. RS232 es un estándar consolidado y ubicuo en computadoras de escritorio. Para trabajar con notebooks existen los dispositivos RS232 via USB que son muy genéricos y económicos. Por lo tanto, la elección del puerto serie como medio de comunicación es razonable. La velocidad de transmisión es de 115200 bps. En el diseño de la interfaz se partió de la base de que pueda ser manejada directamente por una persona a través de una consola TTY. Esto implica que sólo se transmitirán carácteres ASCII tanto para las solicitudes como para las respuestas. Por lo tanto hemos decidido codificar la información en hexadecimal. A su vez, se muestran mensajes de menú de ayuda a medida que se van enviando datos.

La filosofía de la aplicación es la siguiente. La aplicación arranca y ejecuta código de inicialización. Una vez terminada esta etapa imprime[7] un menú. El usuario ejecutará comandos, estos constan de varios dígitos hexadecimales. La longitud de los comandos va a depender de la acción y lo que se quiera controlar. Una vez que el comando finaliza se muestra nuevamente el menú y este ciclo se repite infinitamente. En la figura 6.2 podemos ver el diagrama de flujo de la aplicación.

[7] de acá en adelante cuando hable de imprimir estoy hablando de visualizarse en la pantalla de una terminal conectada via RS232

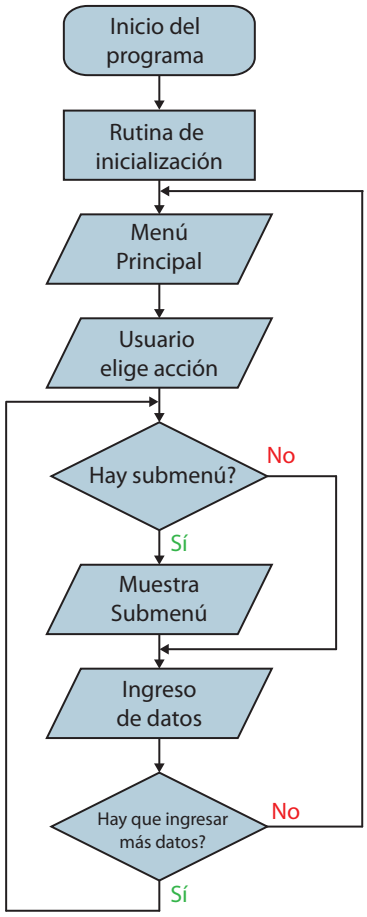


FIGURA 6.2: Diagrama de flujo del software en *MicroBlaze*

La lógica de los comandos es:

1. indicar una función
2. indicar un destino o identificador
3. (opcional) indicar un valor

A medida que la persona va ingresando los dígitos hexadecimales del comando podrán o no aparecer mensajes para guiarlo. Una vez que ingrese la cantidad de dígitos esperados para ese tipo de comando se vuelve al menú principal.

Por ejemplo, un comando para configurar el delay programble número 15 en el sentido forward con un valor de 21 es el siguiente.

```
10f0015
```

donde: `1` indica que voy a programar un delay configurable en el sentido forward
`0f` indica que es el delay número 15, `0f` en hexadecimal
`0015` indica que el valor a programar es 21, `15` en hexadecimal

La sintaxis de comandos se resume en el apéndice §G.1.

Optimización de performance Durante el desarrollo conjunto del software de *MicroBlaze* y el software GUI se percibió que cuando se querían enviar los datos de manera rápida se perdían dígitos hexadecimales y por ende se corrompían los comandos. Este comportamiento no fue evidente cuando se ingresaban los comandos a mano. La causa de este problema era que no llegaban a procesarse los datos lo suficientemente rápido y la cola FIFO de la UART (implementación *uartlite*[8]) perdía datos. Tras hacer una depuración de código se determinó que la lentitud en el procesamienta era causada por la impresión de menúes o mensajes de feedback. Como antes mencionamos, estos mensajes iban destinados al usuario que interactuaba directamente a través de una consola TTY serial. Este problema potencial se disparó luego de decidir implementar el ruido de fondo coloreado y de banda angosta utilizando la IFFT. En ese caso necesitábamos configurar 16386 valores[9]. Por lo tanto se necesitaba optimizar el software. Una posible solución a la que se arribó era reducir la cantidad de feedback ya que el cliente ahora sería el software GUI. Entonces se agregó la funcionalidad de poder reducir la salida a través unos comandos. De esta manera, seguimos contando con la funcionalidad de usuario a través de una consola.

6.4.2. software GUI

El desarrollo de una interfaz gráfica para controlar el emulador fue un objetivo estratégico que nos impusimos para permitir su uso sin necesitar conocimientos detallados de cómo está implementado el diseño. También permite expandir las fronteras de este trabajo del ámbito puramente académico.

Para programar dicha interfaz se utilizó el framework de Microsoft *.NET 3.5* y el lenguaje C#. La justificación de elección se basó casi exclusivamente en los conocimientos previos del autor de este trabajo. De hecho, en un principio se empezó a desarrollar en *.NET 2.0* pero la performance de un componente nos obligó a transformar el proyecto a Microsoft WPF[10]. Estos frameworks facilitan el rápido desarrollo de aplicaciones gráficas.

La filosofía del software se basa en los siguiente puntos:

- Leer un archivo de configuración que contenga las posibles configuraciones de los distintos componentes para que luego el usuario pueda elegir que configuración cargar.
- Mostrar en un recuadro la salida serie del software del *MicroBlaze*. Este cuadro de texto emularía a una terminal pero que sólo es *read-only*.
- Permitir filtrar los mensajes que nos transmite el *MicroBlaze*.
- Permitir enviar comandos manuales.

[8] según la documentación `http://www.xilinx.com/support/documentation/ip_documentation/xps_uartlite.pdf` el tamaño de las FIFOs es de 16 Bytes.

[9] 8193 valores para la potencia y 8193 valores para los procesos AR 1.

[10] Windows Presentation Foundation

- Permitir configurar los parámetros de la UART por si se llegan a cambiar en otra síntesis del diseño.

Concurrencia El manejo del puerto serie en el framework es bloqueante. Por lo tanto, si queríamos que el cuadro de texto se actualice a medida que le vayamos enviando los comandos tuvimos que implementar las funciones del puerto serie en un *thread* separado. Si no se utilizaba esta estrategia, debido a que el puerto serie trabaja a 115200 bps y que las configuraciones[11] pueden tener bastantes Bytes el bloqueo era demasiado perceptible por el usuario y daba la impresión que la aplicación estaba colgada.

Construcción del archivo de configuración Si bien el usuario final interactúa de una manera simple con el emulador en el caso que éste quiera generar un escenario nuevo tendría que modificar el archivo de configuración. El archivo de configuración es un conjunto de configuraciones, las cuales a su vez son conjuntos de comandos. Los comandos son dígitos hexadecimales, por lo tanto, hay que realizar una conversión para poder representar números fraccionarios. Esta tarea sería muy engorrosa si fuese hecha manualmente y además sería faraónica para el caso de los 16386 parámetros del ruido de fondo coloreado y de banda angosta. Por estos motivos se desarrolló un script de MATLAB que genera el archivo de configuración en función de variables numéricas típicas de MATLAB.

6.5. Resumen

En este capítulo presentamos la interfaz de control del emulador que utiliza un *soft-processor MicroBlaze*. Este procesador controla la lógica de procesamiento de señales a través de recursos compartidos (Memorias, registros y FIFOs). El software desarollado para *MicroBlaze* se puede usar tanto por un usuario con una terminal TTY serie o a través de una software GUI desarrollado para que usuarios no interiorizados puedan utilizar el emulador. Al diseño se le incorporó la tecnología *Chipscope Pro* para visualizar las señales internas del FPGA en condiciones operativas mientras se lo operaba con el software GUI.

[11] las configuraciones son conjuntos de comandos.

7

Evaluación del diseño

En el capítulo 5 se expusieron los diseños de implementación para FPGA de los distintos componentes que constituyen el emulador de canal. En este capítulo se evaluarán las salidas de cada bloque conforme a los requisitos planteados en la subsección §4.2.1.

Las evaluaciones serán realizadas en base a simulaciones en System Generator de Xilinx, que es una extensión al Simulink de MATLAB. La validez de éstas últimas está garantizada ya que Xilinx nos especifica que las simulaciones son *bit-accurate* y *cycle-accurate* [46, p. 17].

7.1. Evaluación de la transferencia

El diseño planteado en §4.3.1 cumple con los requisitos planteados en §4.2.1. En esta sección podremos corroborar que se cumplen los requisitos **RQ_01** y **RQ_05**, ya que el resto de los requisitos son verificables en la operación de tiempo real. El cumplimiento de primer requisito se basó en utilizar la arquitectura propuesta por OPERA [3]. Para demostrar la afirmación expondremos algunos canales generados por este bloque.

7.1.1. Canales de acceso

En este caso se ha configurado el emulador para representar el canal de referencia 1 para redes de acceso que corresponde a la clase de 150 metros y calidad buena. Vale la pena mencionar que los canales de referencia correspondientes a redes de acceso especificados en [3][1] tienen errores en los valores de *delay*. En consecuencia, se han

[1] Además no especifica los nueve canales (ver sección §A.1), sino que sólo cinco, de los cuáles uno no se corresponde con ninguno de los nueve.

adaptado dichos valores para que su error cuadrático medio sea mínimo utilizando como referencia a los canales especificados en [1].

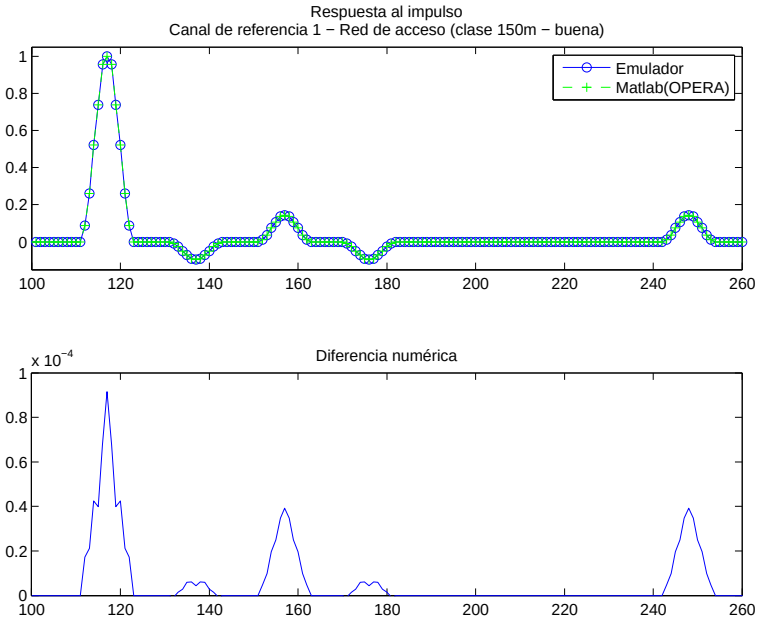


FIGURA 7.1: Comparación de respuesta al impulso implementada vs Matlab *double*

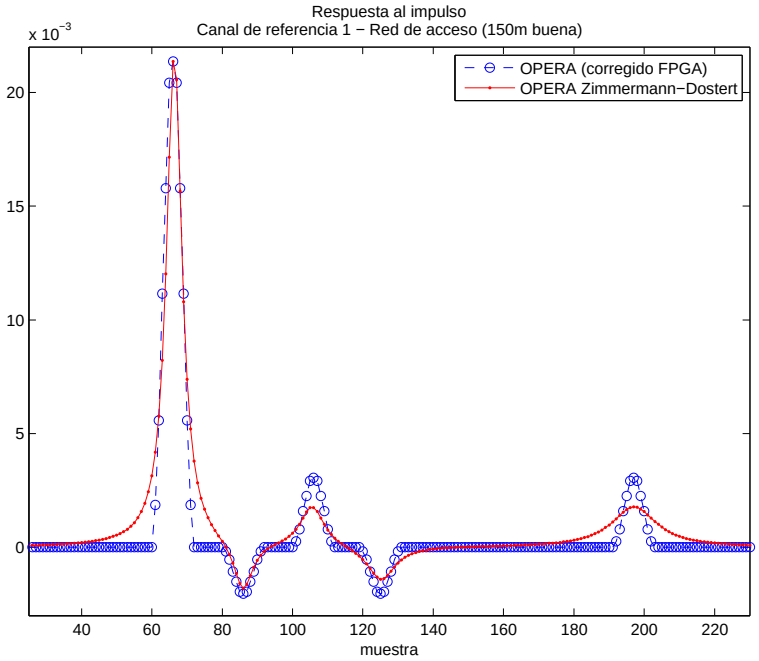


FIGURA 7.2: Comparación de respuesta al impulso para FPGA vs OPERA modelo Zimmermann-Dostert

En la figura 7.1 observamos una comparación entre la respuesta al impulso implementada en el FPGA versus la respuesta al impulso implementada en MATLAB. La diferencia que se observa se debe a la precisión numérica. Nosotros hemos decidido implementar las respuestas al impulso normalizadas de manera que el pico máximo tenga un valor absoluto de 1 para poder realizar una fácil comparación. Sin embargo, debido a que se utiliza un tipo de dato `Fix_16_15` existirán errores ocasionados por la cuantización. Por ejemplo, el valor 1 se representa como 0,999969482421875. A pesar de esto vemos que el error en norma infinito es de cuatro órdenes de magnitud menor, por lo que podemos decir que el error numérico es despreciable. Por otro lado, en la figura 7.2 vemos la comparación entre la respuesta al impulso que utiliza el modelo de Zimmermann-Dostert (§A.1.1.1) y la versión adaptada para el modelo de canal implementado en la FPGA. Las características generales están bien representadas y por lo tanto podemos decir que es una emulación fiel. Si se quisiese un mejor ajuste a la respuesta impulsiva habría que utilizar una mayor cantidad de caminos, ya que en este caso sólo se utilizaron cinco.

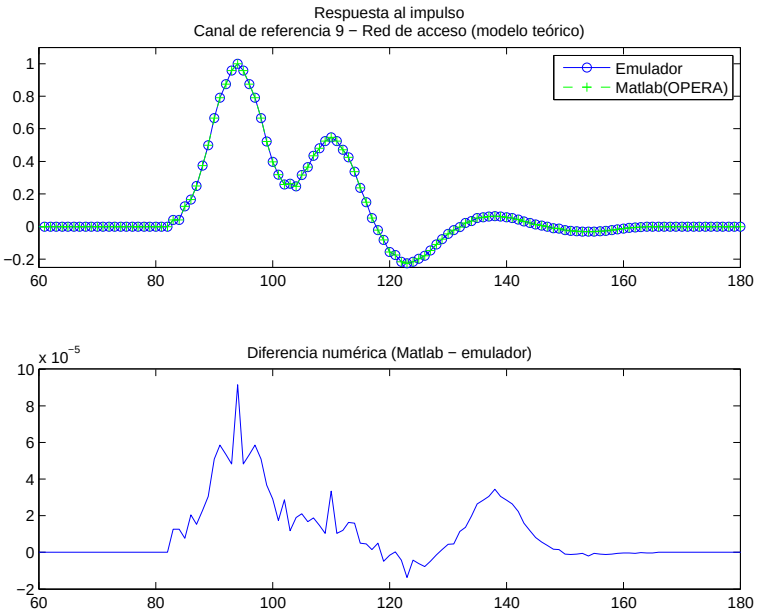


FIGURA 7.3: Comparación de respuesta al impulso implementada vs Matlab *double*

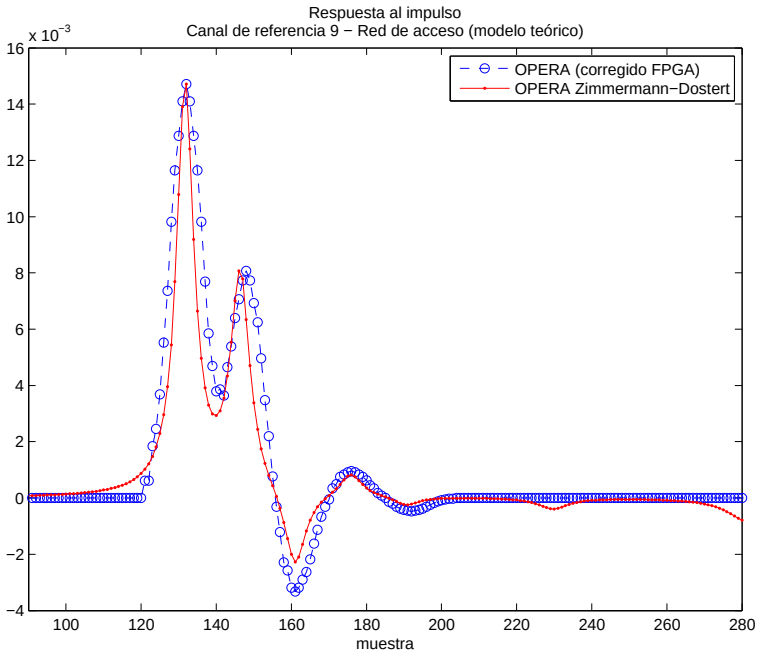


FIGURA 7.4: Comparación de respuesta al impulso para FPGA vs OPERA modelo Zimmermann-Dostert

En este segundo caso de estudio se ha configurado el emulador para modelar al canal de referencia 9, que es el modelo teórico (§A.1.4) que Zimmermann usó para validar el modelo [50] (ver §2.4.4). En la figura 7.3 se puede ver la comparación entre la respuesta al impulso implementada en el FPGA y la implementada en MATLAB. Las diferencias se deben a errores de cuantización y también son cuatro órdenes de magnitud inferior. Por otro lado, en la figura 7.4 observamos la adaptación de la respuesta al impulso para poder implementarla en la FPGA. En este caso es fácil ver que se podría mejorar la representación si se agregan dos caminos más para las últimas reflexiones. Sin embargo, el objetivo en esta sección es mostrar la capacidad de emulación de los modelos de referencia sugeridos por OPERA y no mejorar los parámetros para obtener una representación más precisa.

7.1.2. Canales in-house

El modelo para canales in-house es el modelo de ecos Philipps [30], que coincide con el modelo de Zimmermann-Dostert sin factores de atenuación. Por lo tanto, las respuestas al impulso serán simples deltas de Dirac con diferentes amplitudes y por esta razón el canal no tendrá característica pasabajos.

En el primer caso configuramos el emulador para representar el primer tipo de canal in-house, el modelo 1 que sólo tiene 5 caminos. En la figura 7.5 se ve la comparación entre la respuesta al impulso implementada en el FPGA y la implementada en MATLAB. La diferencia que se observa es debido a errores de cuantización. Sin

embargo, ésta es cuatro órdenes de magnitud menor que la máxima amplitud y por lo tanto es despreciable. Por otro lado, en la figura 7.6 se observa la comparación entre la respuesta al impulso que utiliza el modelo de Zimmermann-Dostert y la versión adaptada para la implementación en FPGA. En el caso de canales sin factor de atenuación, la representación del modelo adaptado para FPGA se corresponde con el modelo teórico de Zimmermann-Dostert y por ello podemos ver una correspondencia exacta si despreciamos errores de cuantización.

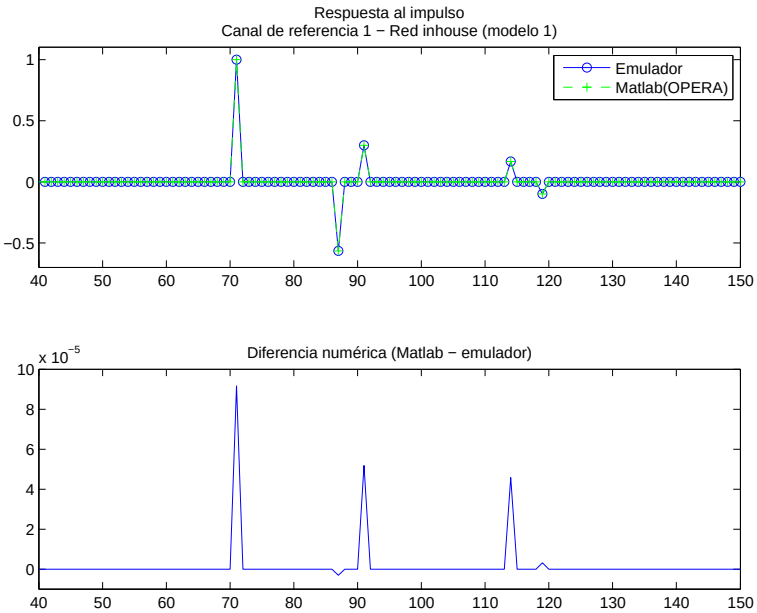


FIGURA 7.5: Comparación de respuesta al impulso implementada vs Matlab *double*

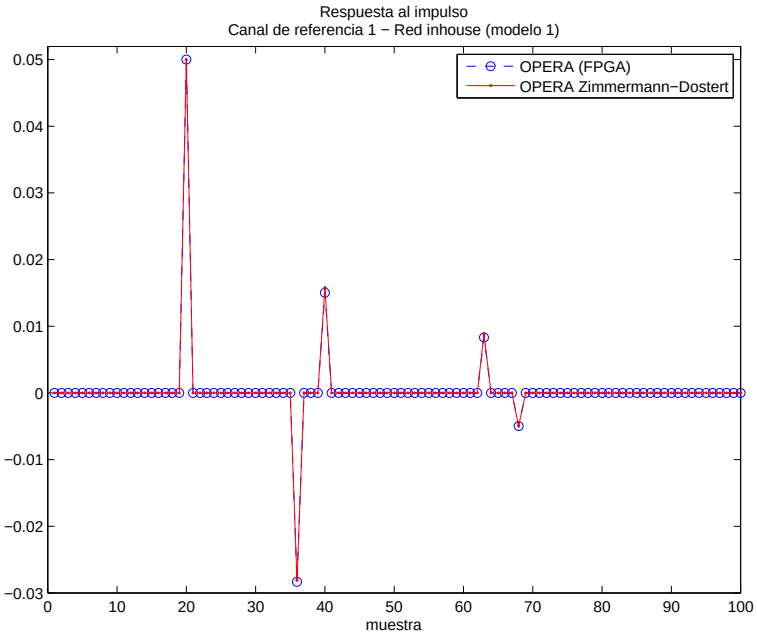


FIGURA 7.6: Comparación de respuesta al impulso para FPGA vs OPERA modelo Zimmermann-Dostert

En un segundo caso configuramos el emulador para representar el cuarto canal de referencia para redes in-house. Éste se caracteriza por tener 20 caminos y se corresponde con una red que tiene múltiples ramificaciones. En vista de lo mencionado para el caso anterior, en las figuras 7.7 y 7.8 veremos una correspondencia casi exacta entre las tres representaciones: la implementada en el FPGA, la implementada en MATLAB en base a [3] y la definida en [1]. Las pequeñas diferencias se generan en el proceso de cuantización pero por ser cuatro órdenes de magnitud menor se pueden despreciar.

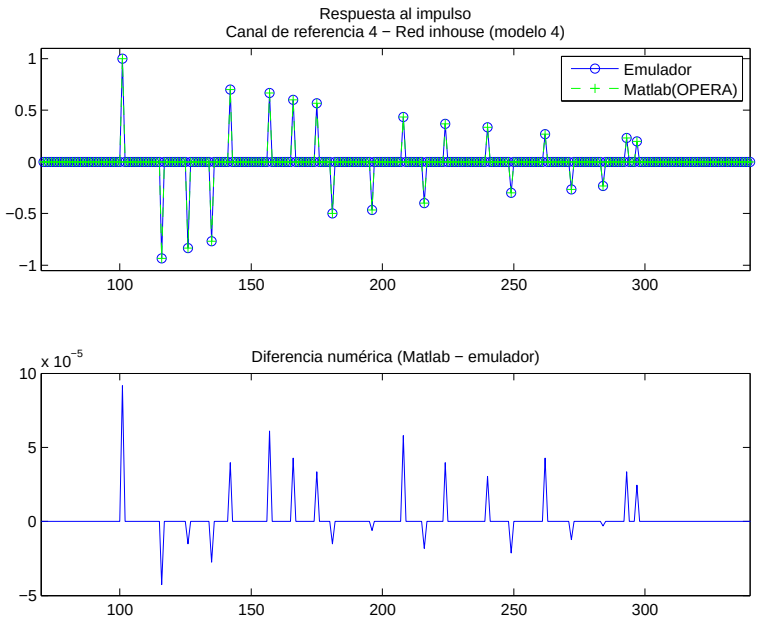


FIGURA 7.7: Comparación de respuesta al impulso implementada vs Matlab *double*

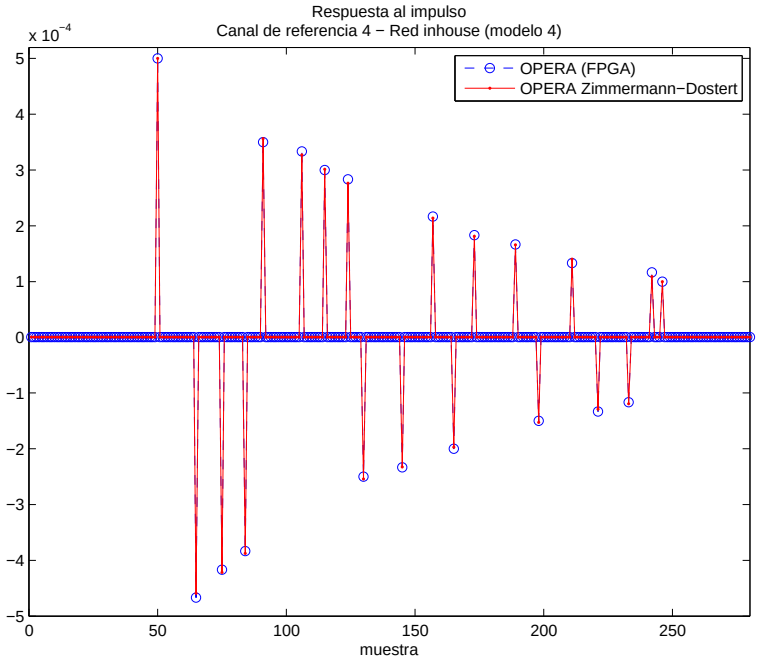


FIGURA 7.8: Comparación de respuesta al impulso para FPGA vs OPERA modelo Zimmermann-Dostert

7.2. Evaluación del ruido de fondo coloreado

En esta sección evaluaremos la generación de ruido coloreado según el modelo expuesto en §3.7 y podremos ver que se cumple con los requisitos **RQ_07** parte 1 y **RQ_08**. La implementación de este ruido ha sido integrada junto con la del ruido de banda angosta. La arquitectura planteada utiliza la IFFT y por ello permite construir un perfil arbitrario de ruido de fondo. Por lo tanto excede ampliamente las especificaciones propuestas por OPERA. Además, la implementación sugerida en [3] proponía el uso de un filtro FIR de orden 16 con un *set* de coeficientes que sólo lograba una diferencia de 17 dB entre N_0 y N_1 contra valores típicos de 50 dB[2].

A continuación mostramos en la figura 7.9 la estimación de la PSD del ruido de fondo para los siguientes parámetros que representan un ambiente de oficinas

- $N_0 = -135$ dBm/Hz
- $N_1 = 49$ dBm/Hz
- $f_1 = 0{,}865$ MHz

En base a estos parámetros se generaron los 8193 coeficientes que generan la envolvente utilizando un script desarrollado en MATLAB conforme a la ecuación (3.16). Se puede apreciar claramente la forma exponencial del ruido de fondo coloreado.

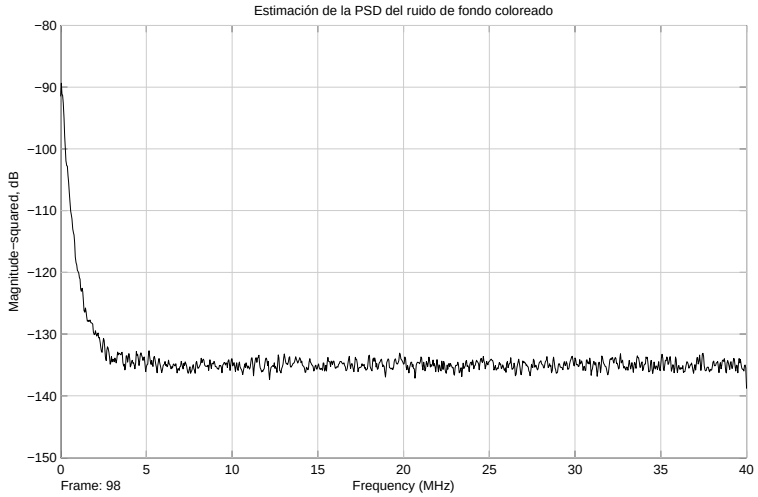


FIGURA 7.9: Estimación de la PSD del ruido de fondo coloreado

En un segundo caso configuramos los parámetros de manera que tengamos -64 dBm/Hz a 20 MHz y -118 dBm/Hz en 0 y 40 MHz cuyo resultado podemos ver en la figura 7.10. Más especificamente se usó el siguiente vector en MATLAB

```
[1:4097 4096:-1:1]/4097
```

[2] Ver tabla 3.1.

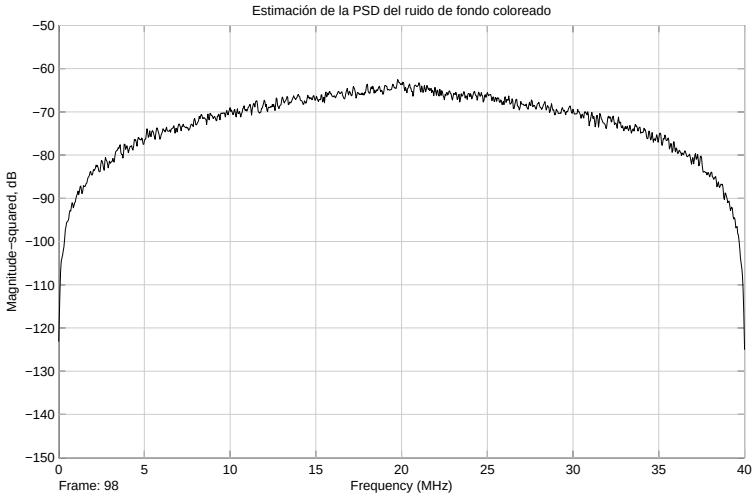


FIGURA 7.10: Estimación de la PSD del ruido de fondo coloreado. Prueba de rango dinámico

Por lo tanto vemos que se satisface la máxima potencia de ruido del requerimiento **RQ_08**.

7.3. Evaluación del ruido de banda angosta

El ruido de banda angosta es un ruido que está presente continuamente y afecta la capacidad del canal. Su fuente son las transmisiones de radio frecuencia. En las instalaciones eléctricas el mismo fenómeno que genera radiación electromagnética también funciona en el sentido opuesto, es decir de recepción. Por lo tanto, los cables constituyen una antena, que es mala respecto desde el punto de vista del diseño de antenas pero es una antena al fin.

En nuestra especificación nos planteamos originalmente simular 8 interferencias de banda angosta (**RQ_10**). OPERA [3] no sólo plantea 3 interferencias sino que fija las frecuencias en 3, 7 y 11 MHz y no especifica nada sobre el ancho de banda de las mismas. Nuestro diseño, al utilizar la IFFT, tiene una enorme flexibilidad y permite simular alrededor de 8 mil interferencias cada una con un ancho de banda independiente.

A continuación generaremos 3 interferencias que serán posicionadas en los mismos lugares especificados por OPERA. La configuración de los parámetros busca generar las siguientes interferencias:

- $f_c \approxeq 3$ MHz, potencia -64 dBm/Hz, ancho de banda $Bw \approxeq 102$ kHz
- $f_c \approxeq 7$ MHz, potencia -84 dBm/Hz, ancho de banda $Bw \approxeq 53$ kHz
- $f_c \approxeq 11$ MHz, potencia -72 dBm/Hz, ancho de banda $Bw \approxeq 200$ kHz

En la figura 7.11 vemos la estimación de la PSD para las 3 interferencias especificadas. Un hecho importante que puede observarse es que el piso de intereferencia impuesto

por las sumas de senoidales (que es la IFFT) es mayor que por ejemplo el valor elegido como piso de ruido, -135 dBm/Hz, en la sección anterior. Vale la pena comentar que este comportamiento también se observa en la generación del ruido por señales moduladas. Por lo tanto, no es una desventaja de este método. En las figuras 7.12, 7.13, y 7.12 se ve el detalle de las interferencias de 3, 7 y 11 MHz respectivamente. Allí se puede apreciar que los parámetros configurados se reflejan en la estimación de la PSD.

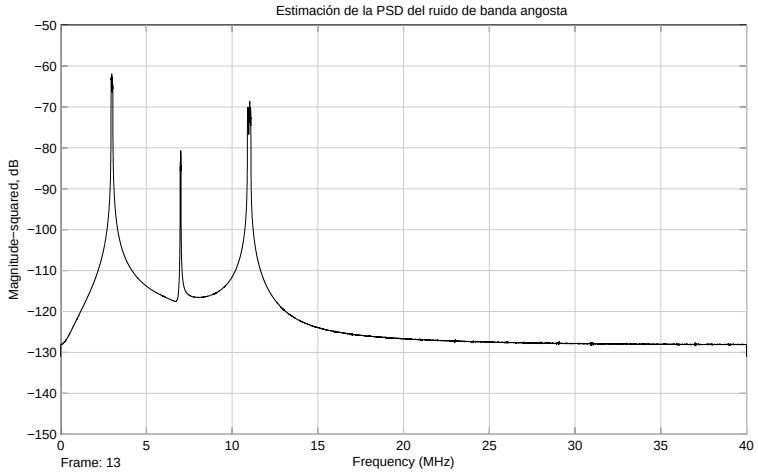


FIGURA 7.11: Estimación de la PSD del ruido de banda angosta

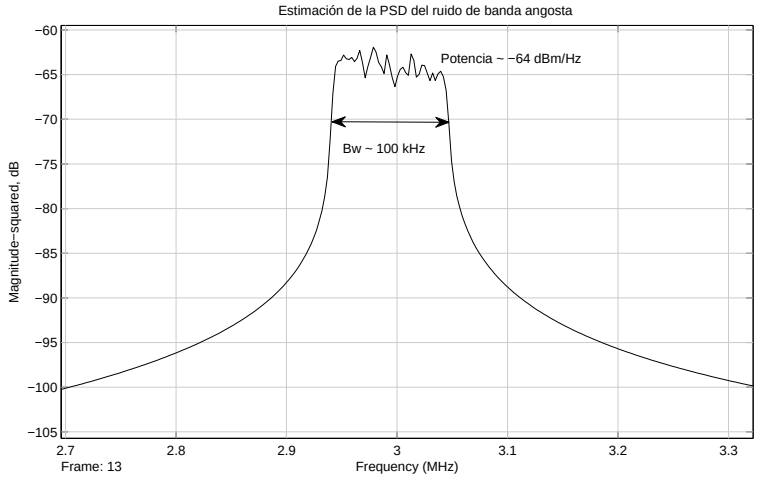


FIGURA 7.12: Estimación de la PSD del ruido de banda angosta. Detalle para interferencia 1 en 3 MHz

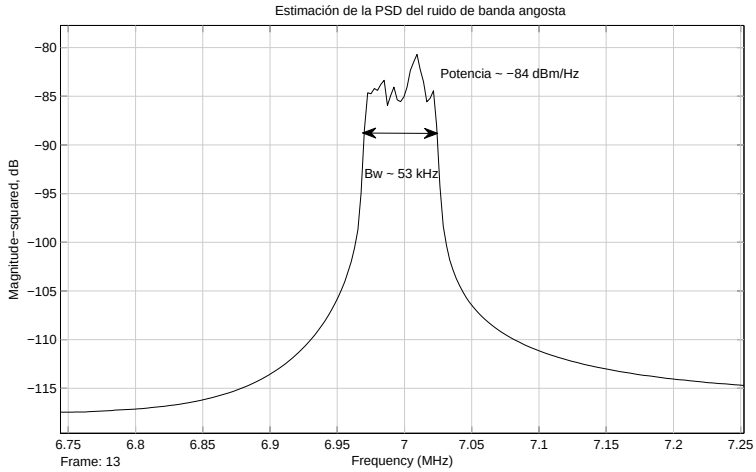


FIGURA 7.13: Estimación de la PSD del ruido de banda angosta. Detalle para interferencia 1 en 7 MHz

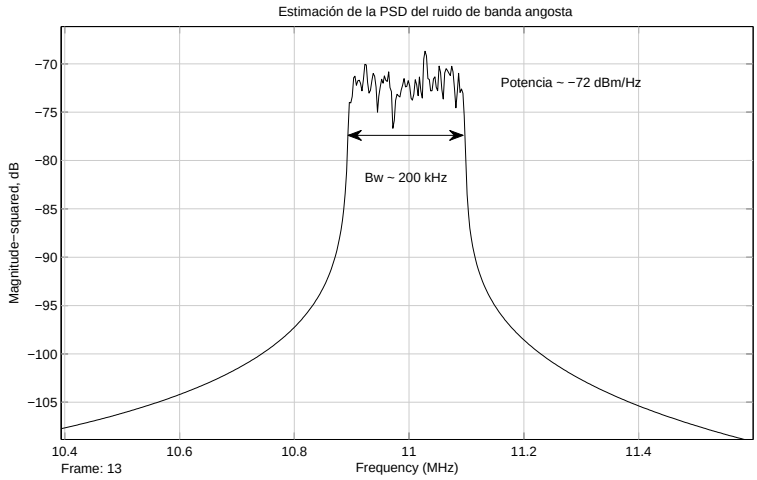


FIGURA 7.14: Estimación de la PSD del ruido de banda angosta. Detalle para interferencia 1 en 11 MHz

7.3.1. Comentarios

Colas de las interferencias de banda angosta En las figuras anteriores podemos ver que cada interferencia tiene colas en el espectro. Un efecto no deseado de las mismas es poner un piso que pudiera ser mayor al piso de ruido que deseásemos establecer. Por ejemplo, si en el caso de la figura 7.11 quisiérmos tener un ruido de fondo coloreado con un piso de ruido de -135 dBm/Hz no lo podríamos lograr. Para cuantizar el efecto de las colas realizamos una simulación con una sola interferencia situada en 11 MHz, con un ancho de banda de 200 kHz y una potencia de -64 dBm/Hz, es decir la máxima potencia que podemos generar. Esta simulación da el espectro observado

en la figura 7.15 cuyo espectro tiene un piso de -126 dBm/Hz en el mejor caso. Esto representaría un rango dinámico de 62 dB entre el máximo y el piso. Sin embargo, los valores de estas interferencias no suelen superar los 30 dB respecto del piso de ruido [38, 40]. Por lo tanto, estamos cubiertos con 32 dB en el mejor caso y 26 dB en un caso muy malo. Un aspecto muy importante a destacar es que si la estimación espectral no se hace adecuadamente, las colas se visualizan incorrectamente. Para ello adjuntamos la figura 7.16 donde se estima el espectro, también por Welch, pero con un tamaño de FFT de 2048 puntos contra los 32768 que usamos para evaluar el ruido de banda angosta. En esa gráfica vemos un piso muy inferior al real. La razón de esto es que se está muestreando la señal en determinados posiciones[3] que luego tiene como consecuencia que las colas parezcan más chicas de lo que son.

El efecto de levantamiento del piso de ruido también ocurre con el método de generación por señales moduladas. En ambos casos la diferencia entre el piso, impuesto por las colas, y el máximo es función del ancho de banda de la interferencia de banda angosta.

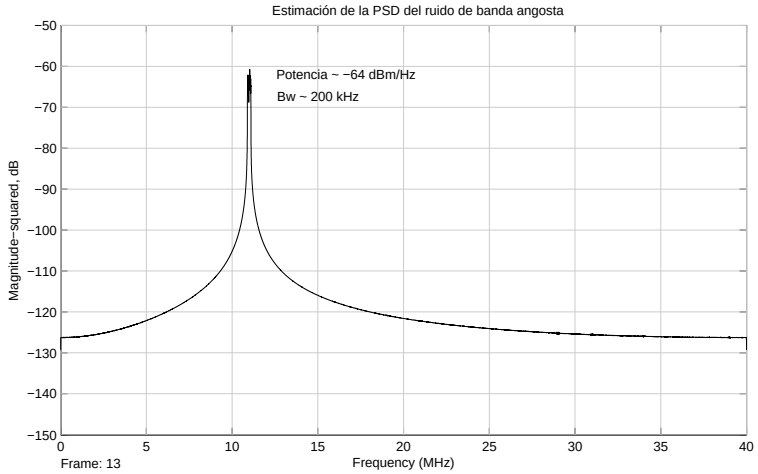


FIGURA 7.15: Estimación de la PSD del ruido de banda angosta. Colas espectrales de la interferencia.

[3] por ejemplo, algunas son ceros.

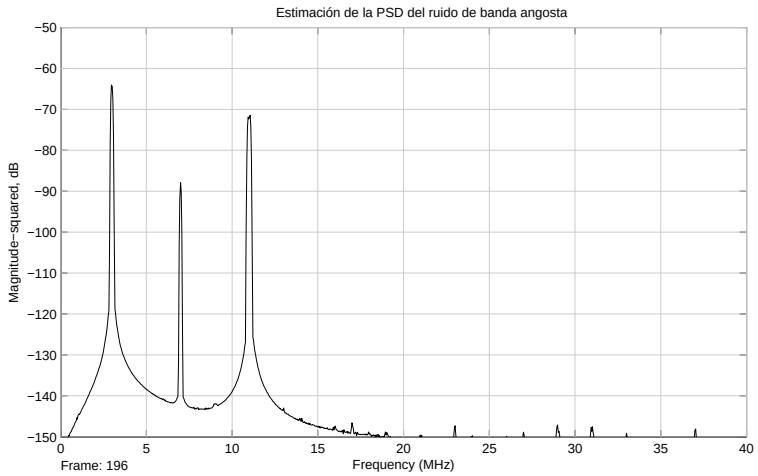


FIGURA 7.16: Mala estimación de la PSD del ruido de banda angosta. Colas espectrales de la interferencia.

Escenario de ruido de fondo coloreado y de banda angosta con picos En este último caso de verificación que mostraremos hemos configurado un escenario más real en el cual se combinan el ruido de fondo coloreado junto con el ruido de banda angosta que a su vez contiene picos en frecuencias específicas. La configuración del ruido de banda angosta fue elegida para parecerse a la de la figura 3.2 y por ello la diferencia entre el piso de ruido nos quedó superior a los 30 dB antes mencionados debido a que elegimos un ruido de fondo de -135 dBm/Hz y no los $-105,7$ dBm/Hz de la mencionada figura. En resumen la configuración es la siguiente:

- $f_c \approxeq 6$ MHz, potencia -85,7 dBm/Hz, ancho de banda $Bw \approxeq 300$ kHz
- $f_c \approxeq 7$,5 MHz, potencia -93,7 dBm/Hz, ancho de banda $Bw \approxeq 250$ kHz
- $f_c \approxeq 9$,6 MHz, potencia -87,7 dBm/Hz, ancho de banda $Bw \approxeq 600$ kHz
- $f_c \approxeq 11$,7 MHz, potencia -97,7 dBm/Hz, ancho de banda $Bw \approxeq 500$ kHz
- piso de ruido -135 dBm/Hz
- picos de ruidos en las distintas bandas

Vale la pena mencionar que nos valimos de un script de MATLAB para generar los 8193 parámetros. Los otros 8193 correspondiente al control de ancho de banda de cada tono se han fijado en 0. En la figura 7.17 vemos la estimación de la PSD del escenario descripto. Debido a la existencia de picos y a la sumatoria de distintas bandas de interferencias notamos que el piso de ruido se elevó a $-133,5$ dBm/Hz. De todas maneras esto podría ser corregido si los parámetros se eligiesen con un criterio más realista, es decir que la diferencia entre el ruido de fondo y las interferencias de banda angosta esté alrededor de 30 dB. Se utilizaron 720 mil muestras para esta simulación.

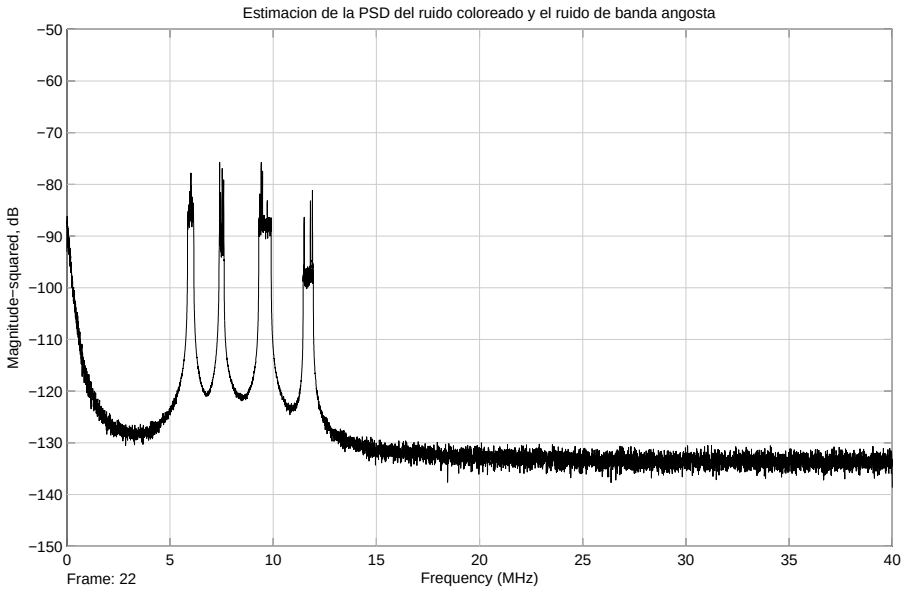


FIGURA 7.17: Estimación de la PSD del ruido de fondo coloreado y de banda angosta

En la figura 7.18 se observa en detalle las interferencias de banda angosta que verifican la potencia configurada y también se pueden ver los picos. El ancho de banda de estos picos es de alrededor de 4,2 kHz. Dicho valor se obtuvo de simulaciones hechas en Matlab con los mismos parámetros de precisión finita. No se realizaron en System Generator porque la resolución de la estimación de la PSD por el método de Welch precisaba una FFT de 2^{20} puntos y simulaciones de $2 \cdot 10^9$ iteraciones, para tener una varianza aceptable, que no eran factibles de realizarse en dicho entorno.

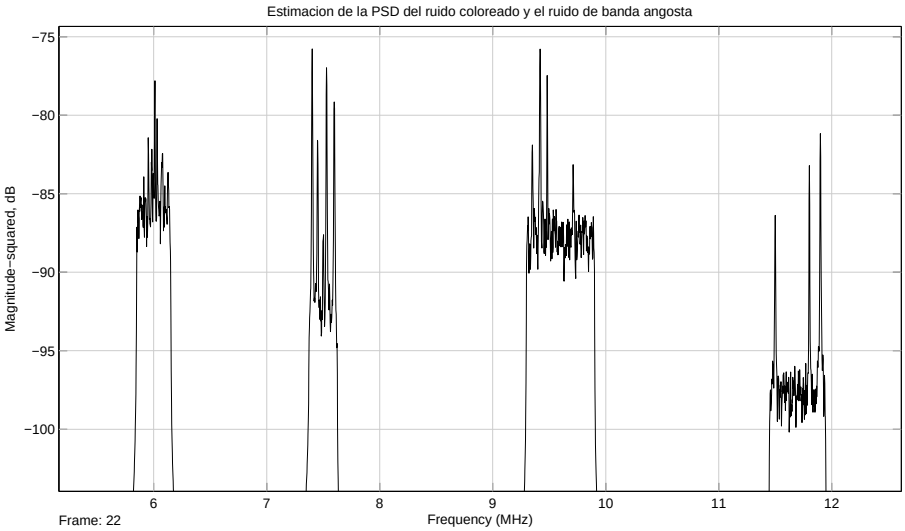


FIGURA 7.18: Detalle de la estimación de la PSD del ruido de fondo coloreado y de banda angosta

7.4. Evaluación del ruido impulsivo periódico sincrónico

El ruido impulsivo periódico sincrónico con la frecuencia de red se origina en equipos que utilicen SCR o diodos rectificadores. Dentro de este tipo de dispositivos entran los convertidores sincrónicos de potencia como por ejemplo un dimmer para lámparas incandescentes. En esta sección podremos ver que se cumple con el requisito **RQ_07** parte 3.

OPERA [1] dice que el efecto de este ruido es despreciable tras realizar un análisis espectral y obtener un espectro parecido al ruido de fondo coloreado. Por ello deciden no implementar este ruido en su emulador. Sin embargo, ese análisis tiene como base teórica el teorema de Wiener-Kinchin que tiene aplicación en procesos aleatorios estacionarios en sentido amplido, pero en nuestro caso el proceso es cicloestacionario. Por lo tanto, este espectro no describe la realidad adecuadamente y no debería ser utilizado para tomar dicha decisión. Desde el punto de vista de un diseñador de sistemas de comunicaciones, éste verá este ruido como algo no despreciable ya que constituye una interferencia de potencia considerable que perturba la señal. Si bien la duración del período es muy grande comparado con la extensión de las PPDUs de hoy en día, los impulsos pueden tener duraciones típicas de algunas decenas de microsegundos. Y por lo tanto pueden llegar a afectar múltiples PPDUs consecutivas y causar el descarte de algunas por ser irrecuperables ante una situación de relación señal-ruido muy desfavorable. Por este motivo, este ruido tiene efectos considerables y para nada despreciables y debe ser tenido en cuenta para construir un emulador.

Para poder evaluar este ruido necesitamos establecer parámetros típicos ya que no existe referencia de estos parámetros en [3]. En este caso elegiremos una frecuencia de repetición de 120 Hz y una duración de impulsos de 16 μs. Estos valores se traducen en 666.667 y 1280 muestras respectivamente a una tasa de muestreo de 80 Msps. En la figura 7.19 podemos ver dos señales características de este bloque:

- *Envolvente*: esta señal controla si pasa o no pasa el ruido coloreado que conforma los impulsos.
- *Ruido impulsivo periódico sincrónico*: es lo que queremos generar.

En la gráfica podemos distinguir el período de aproximadamente 666 mil muestras. Por otro lado, en la figura 7.20 vemos el detalle de una porción de la misma simulación donde se ve un impulso que tiene una duración considerable comparado con los impulsos que veremos para otros tipos de ruidos más adelante.

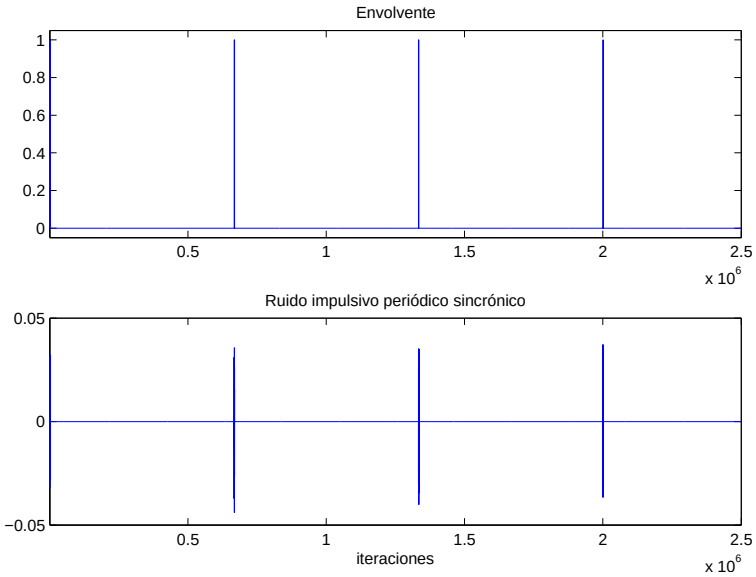


FIGURA 7.19: Emulación del ruido periódico sincrónico

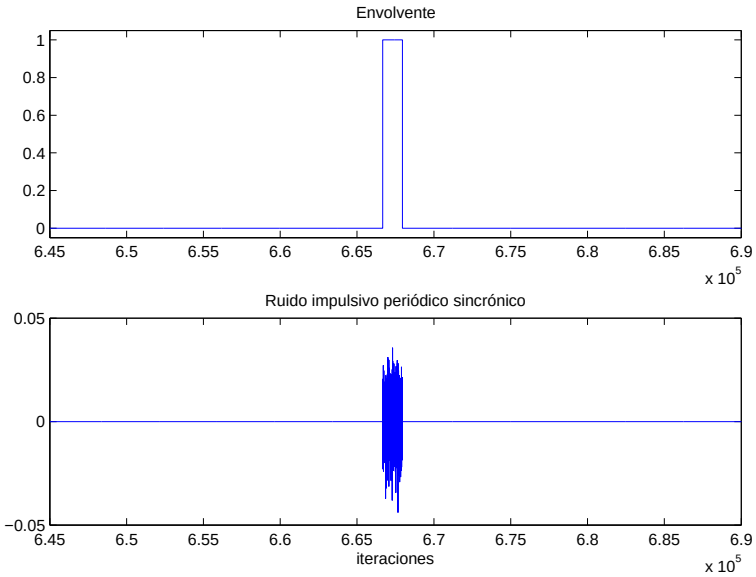


FIGURA 7.20: Detalle de la emulación del ruido periódico sincrónico

Por último, en la figura 7.21 observamos la estimación del espectro de un impulso. Para ello utilizamos el estimador de Welch. Notar que las unidades están en dB/Hz y por lo tanto debemos sumar 30 dB para pasarlo a dBm/Hz. La forma de la PSD se aproxima a la respuesta en frecuencia del filtro que colorea el ruido blanco Gaussiano de la figura 7.22.

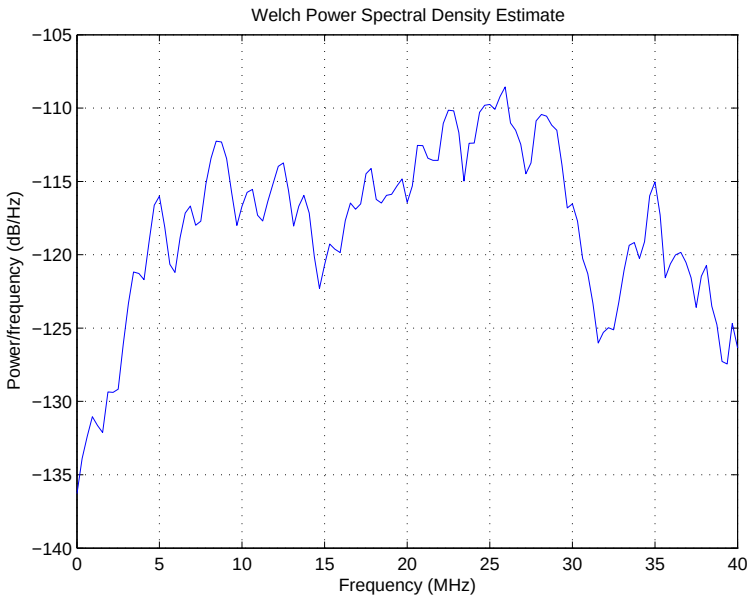


FIGURA 7.21: Estimación de la PSD del ruido impulsivo periódico sincrónico

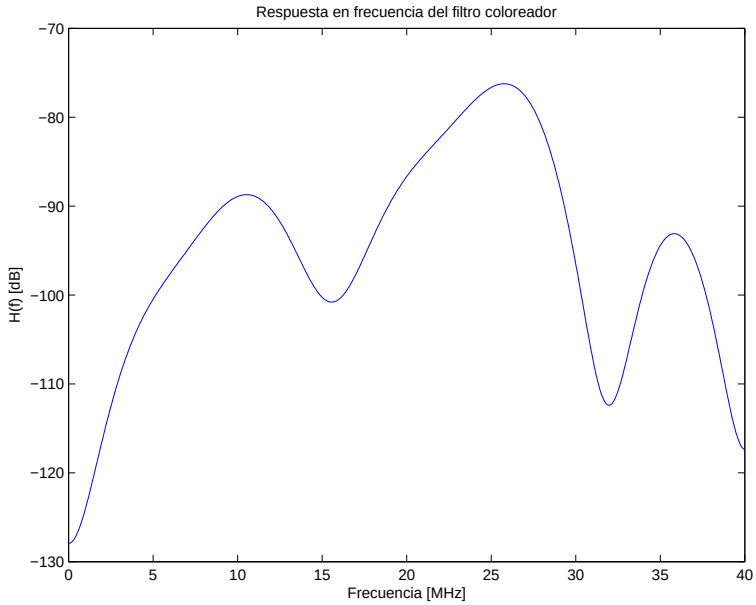


FIGURA 7.22: Respuesta en frecuencia del filtro coloreador de ruido

7.5. Evaluación del ruido impulsivo periódico asincrónico

El ruido impulsivo periódico asincrónico con la frecuencia de red es característico de dispositivos de conmutación cuya frecuencia de trabajo es independiente de la

frecuencia de alimentación. Un dispositivo muy común que entra en esta categoría es la fuente de alimentación de una computadora personal. En esta sección podremos ver que se cumple con el requisito **RQ_07** parte 4.

OPERA [3] sólo especifica dos frecuencias posibles para este tipo de ruido: 50 kHz o 100 kHz. Además, no establece el ancho del impulso y sólo utiliza ruido blanco para el contenido de los impulsos. Nuestro diseño supera con creces estas características y permite especificar la frecuencia y el ancho del impulso. Además permite colorear el ruido que constituye la señal temporal del impulso como se mostrará en una gráfica más adelante.

En la figura 7.23 podemos ver dos señales características de este bloque:

- *Envolvente*: esta señal controla si pasa o no pasa el ruido coloreado que conforma los impulsos.
- *Ruido impulsivo periódico asincrónico*: es lo que queremos generar.

La frecuencia del ruido periódico elegida para esta simulación es de 100 kHz suponiendo una frecuencia de muestreo de 80 Msps. Por lo tanto el período es de 800 muestras. Para el ancho del impulso se eligió un valor típico de 0,2 μs o 16 muestras. También podemos observar las distintas amplitudes positivas y negativas de los impulsos que se deben a que su fuente es Gaussiana.

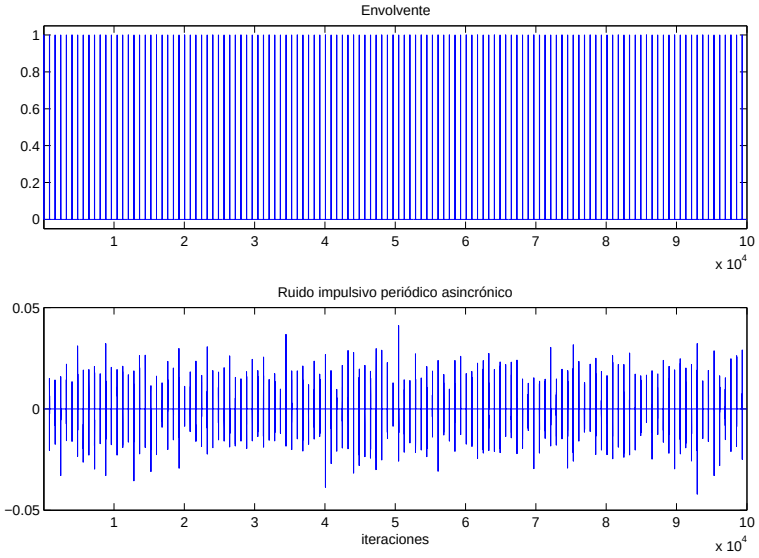


FIGURA 7.23: Emulación del ruido periódico asincrónico

En la figura 7.24 se observa con mayor detalle una porción de la misma simulación. Ahí se aprecia el espaciamiento entre impulsos de 800 muestras y el *duty cycle* de la envolvente. También se llega a distinguir el detalle temporal de cada impulso.

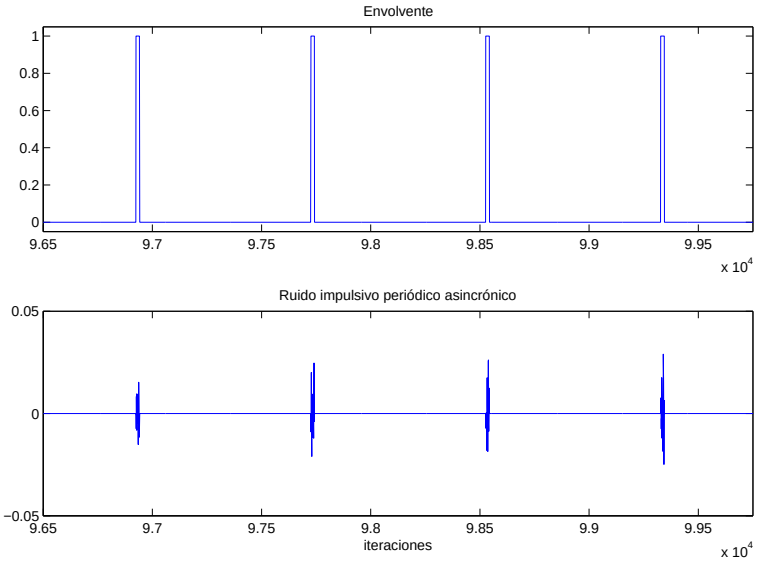


FIGURA 7.24: Detalle de la emulación del ruido periódico asincrónico

A continuación analizaremos el contenido espectral del ruido generado estimando su densidad espectral de potencia mediante el método de Welch. En la figura 7.25 se observa la estimación de la PSD del ruido sobre 300.000 muestras. La característica promedio corresponde a la respuesta en frecuencia del filtro que colorea el ruido Gaussiano blanco, mientras que los múltiples picos que le dan una apariencia de "peine" están relacionados con la frecuencia del ruido periódico.

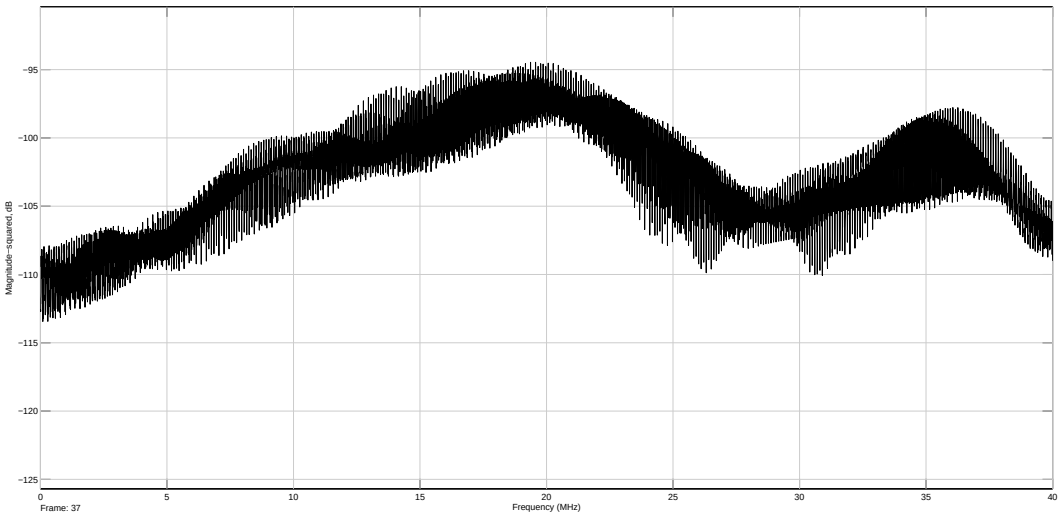


FIGURA 7.25: Detalle de la estimación de la PSD del ruido impulsivo periódico asincrónico

En la figura 7.26 vemos una ampliación de la figura anterior. En ella se puede observar que la separación de los picos se corresponde con la frecuencia de 100

kHz. Esta característica "peine" del espectro se corresponde con lo observado en la densidad espectral medida que mostramos en la figura 3.2.

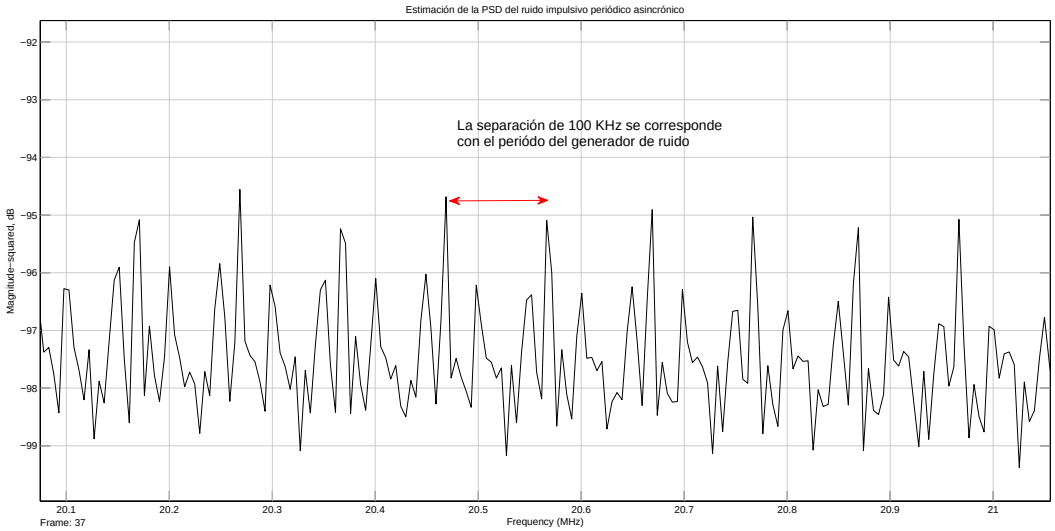


FIGURA 7.26: Estimación de la PSD del ruido impulsivo periódico asincrónico

En base a lo expuesto podemos afirmar que la emulación de este tipo de ruido no sólo queda validada sino que supera ampliamente el diseño propuesto por OPERA.

7.6. Evaluación del ruido impulsivo (aperiódico) asincrónico

Este ruido es una importante característica del emulador y por lo tanto es un componente de gran importancia en el emulador. Aquí podremos ver que se cumple con los requisitos **RQ_07** parte 5 y **RQ_11**.

7.6.1. Comparación estadística

Para la evaluación de este bloque hemos configurado el emulador con la matriz clase *Moderate* según lo establecido en el apéndice de [3]. La matriz que utilizamos ha sido convertida al tipo de dato `UFix_17_16`, mientras que OPERA la especifica con 24 bits de resolución. Sin embargo el análisis de dicha matriz no refleja una variación significativa respecto de nuestra versión cuantizada como veremos más adelante.

A continuación mostraremos la matriz de probabilidades de transición (7.1) y la matriz de probabilidades de transición acumuladada (7.2). En (7.1) podemos observar que hay valores en la diagonal muy cercanos a 1, lo que implica que si se entra en ese estado la probabilidad de seguir en el mismo en la próxima iteración es muy alta y por ende la cantidad promedio de iteraciones que permanecerá en dicho estado tendrá un valor elevado. La matriz que se carga en la memoria es la de la ecuación (7.2). Vale la pena notar que es fundamental elegir buenas semillas para los LFSRs

para la generación de valores pseudoaleatorios entre 0 y 1. De no ser así, los resultados pueden verse altamente comprometidos debido a una baja calidad de números pseudoaleatorios.

$$\mathbf{P} = \begin{bmatrix} 0{,}9867706298828125 & 0 & 0 & 0 & 0{,}0002136230468750 & 0{,}0130157470703125 \\ 0 & 0{,}9999847412109375 & 0 & 0 & 0 & 0{,}0000152587890625 \\ 0 & 0 & 0{,}9999237060546875 & 0 & 0 & 0{,}0000762939453125 \\ 0 & 0 & 0 & 0{,}9997863769531250 & 0 & 0{,}0002136230468750 \\ 0{,}0030517578125000 & 0{,}0000915527343750 & 0{,}0003509521484375 & 0{,}0039215087890625 & 0{,}9925842285156250 & 0 \\ 0{,}0707244873046875 & 0{,}0019989013671875 & 0{,}0082397460937500 & 0{,}0908203125000000 & 0 & 0{,}8282165527343750 \end{bmatrix} \tag{7.1}$$

$$\mathbf{P}_{acum} = \begin{bmatrix} 0{,}9867706298828125 & 0{,}9867706298828125 & 0{,}9867706298828125 & 0{,}9867706298828125 & 0{,}9869842529296875 & 1 \\ 0 & 0{,}9999847412109375 & 0{,}9999847412109375 & 0{,}9999847412109375 & 0{,}9999847412109375 & 1 \\ 0 & 0 & 0{,}9999237060546875 & 0{,}9999237060546875 & 0{,}9999237060546875 & 1 \\ 0 & 0 & 0 & 0{,}9997863769531250 & 0{,}9997863769531250 & 1 \\ 0{,}0030517578125000 & 0{,}0031433105468750 & 0{,}0034942626953125 & 0{,}0074157714843750 & 1 & 1 \\ 0{,}0707244873046875 & 0{,}0727233886718750 & 0{,}0809631347656250 & 0{,}1717834472656250 & 0{,}1717834472656250 & 1 \end{bmatrix} \tag{7.2}$$

Analizando la matriz (7.1) podemos obtener las siguientes características

- La siguiente distribución de probabilidad estacionaria

Nro de Estado	probabilidad estacionaria	$\bar{t_i}$ [muestras]	$\bar{t_i}$ [seg] a 80 Msps
0	0,007971017597185	75,59	0,944867 μs
1	0,195403225526703	65536,00	819,200000 μs
2	0,161016211513567	13107,20	163,840000 μs
3	0,633898816741348	4681,14	58,514286 μs
4	0,000229617790864	134,85	1,685597 μs
5	0,001481110830332	5,82	0,072766 μs

- Tasa de impulsos R_{imp}: 256,13 impulsos por cada millón de muestras, que a 80 Msps es 20490 impulsos por segundo.
- Duración relativa de la perturbación: 0.171073 %.

La simulación realizada constó de 10^6 iteraciones y arrojó los siguientes resultados que serán expuestos en dos tablas para mayor facilidad de comparación. En la tabla 7.1 se observan las comparaciones lado a lado de las probabilidades estacionarias de estado y de la duración promedio de permanencia en cada estado. Podemos ver que las probabilidades estacionarias van tendiendo a las teóricas. Es necesario comprender que para estimar adecuadamente valores pequeños de probabilidad sería necesario simular varios millones de iteraciones. Por otro lado, las duraciones promedio también se van aproximando a sus valores aunque debido a que hay estados con muy pocas muestras eso implica que el promedio muestral no es estadísticamente significativo[4].

[4] el caso extremo se ve para los estados 1 y 4 que tienen sólo una muestra.

Nro de Estado	Probabilidad estacionaria (teórica)	Probabilidad estacionaria (estimación)	Duración promedio en el estado i (teórica)	Duración promedio en el estado i (estimación) $[N$ de $\frac{1}{N}\sum x_i]$
0	0,007971017597185	0,009922990077010	75,59	81,33 [122]
1	0,195403225526703	0,143077856922143	65536,00	143078,00 [1]
2	0,161016211513567	0,180857819142181	13107,20	12918,43 [14]
3	0,633898816741348	0,664156335843664	4681,14	4538,55 [147]
4	0,000229617790864	0,000319999680000	134,85	320,00 [1]
5	0,001481110830332	0,001664998335002	5,82	5,88 [283]

Tabla 7.1: Comparación 1 entre valores teóricos y simulados del proceso particionado de Markov

En la segunda tabla (ver 7.2) se compara la tasa de impulsos y la duración relativa de la perturbación, es decir que fracción del tiempo observado se encuentra ocupada por ruidos impulsivos. Los valores son muy similares y deberían converger a medida que la cantidad de iteraciones simuladas aumente.

	Tasa de impulsos por millón de muestras	Duración relativa de la perturbación
Teórico	256,1	0,171073 %
Simulado	284,0	0,198500 %

Tabla 7.2: Comparación 2 entre valores teóricos y simulados del proceso particionado de Markov

7.6.2. Gráficos

En esta subsección observaremos algunas señales que caracterizan el bloque de generación de ruido impulsivo aperiódico asincrónico. En la primer figura, la 7.27, se observan todas las 10^6 iteraciones para las siguientes señales:

- *Números pseudoaleatorios uniformes*: estos son los que alimentan a la cadena de Markov.
- *Estado de la cadena de Markov*: son los estados, del 0 al 5, que va tomando la cadena de Markov.
- *Envolvente de amplitud constante*: es la señal que determina si va a haber salida de ruido o no. No contempla la potencia del impulso.
- *Multiplicador de amplitud*: esta señal es la salida de un proceso exponencial que mantiene fijo su valor durante la duración del impulso y se utiliza para multiplicar al impulso y cambiar su potencia.
- *Ruido impulsivo aperiódico asincrónico*: es lo que queremos generar.

En particular se puede distinguir claramente la ocurrencia del estado 1 cuya duración es de aproximadamente 143.000 iteraciones y ocurre alrededor de las 120 mil iteraciones. También se ve claramente la variación de las amplitudes del proceso exponencial que se mantienen constantes hasta el próximo impulso.

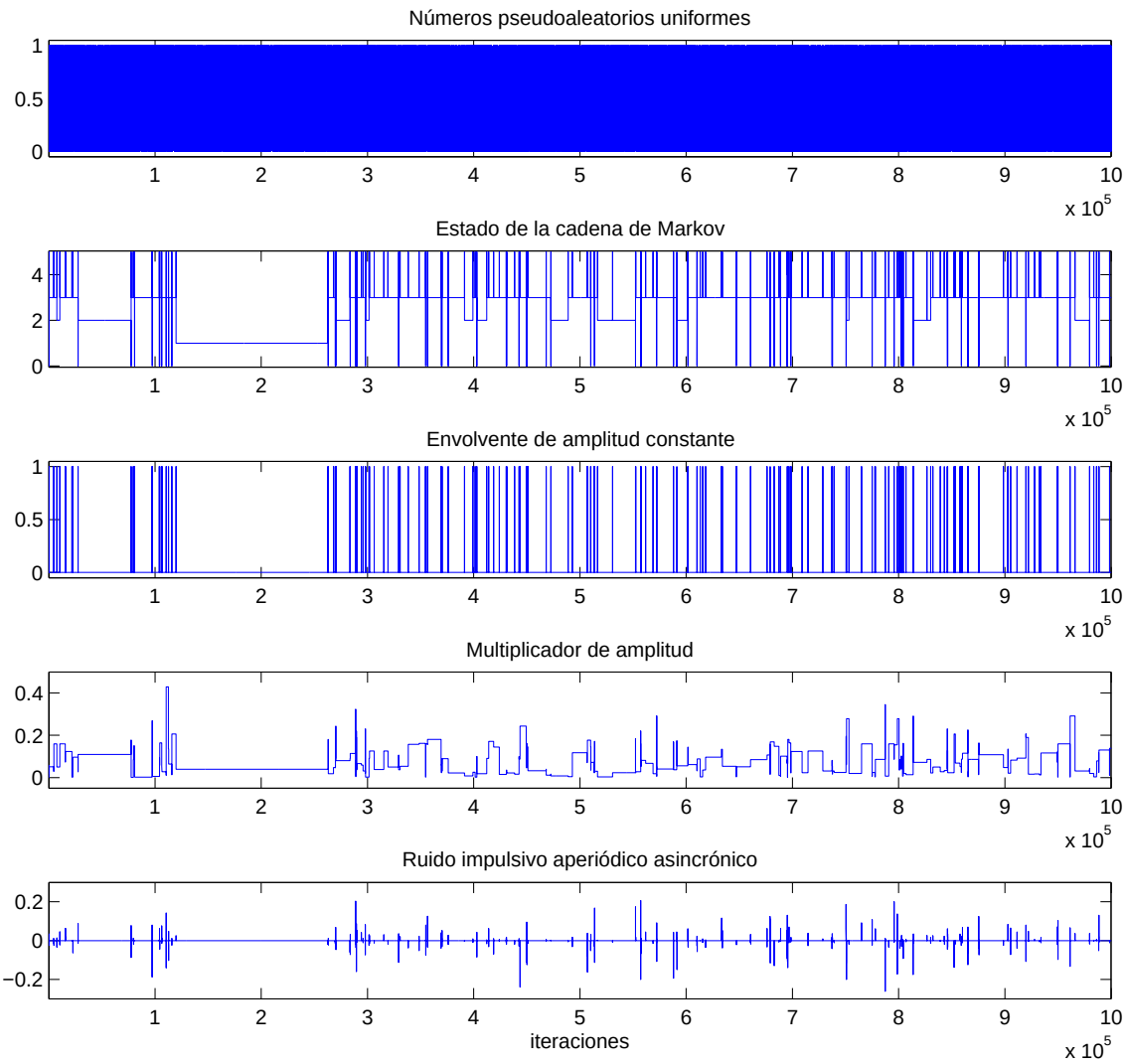


FIGURA 7.27: Emulación de ruido impulsivo aperiódico asincrónico.

En la figura 7.28 se observa con mayor detalle una porción de la misma simulación. En ésta justo transcurre el único evento del estado 4 que representa un impulso de ruido de duración considerable. A su vez, podemos ver como el factor de amplitud ocasiona impulsos de potencias muy distintas y así reproduce la variada diversidad de causas de los impulsos. Si se observa muy detalladamente se podrán notar desfasajes entre las señales, por ejemplo entre la señal de ruido y la envolvente. Esto se debe a que existen componentes como detectores de flanco, registros y multiplicadores que tienen latencia, algunos de manera forzada para cortar el camino combinacional lógico.

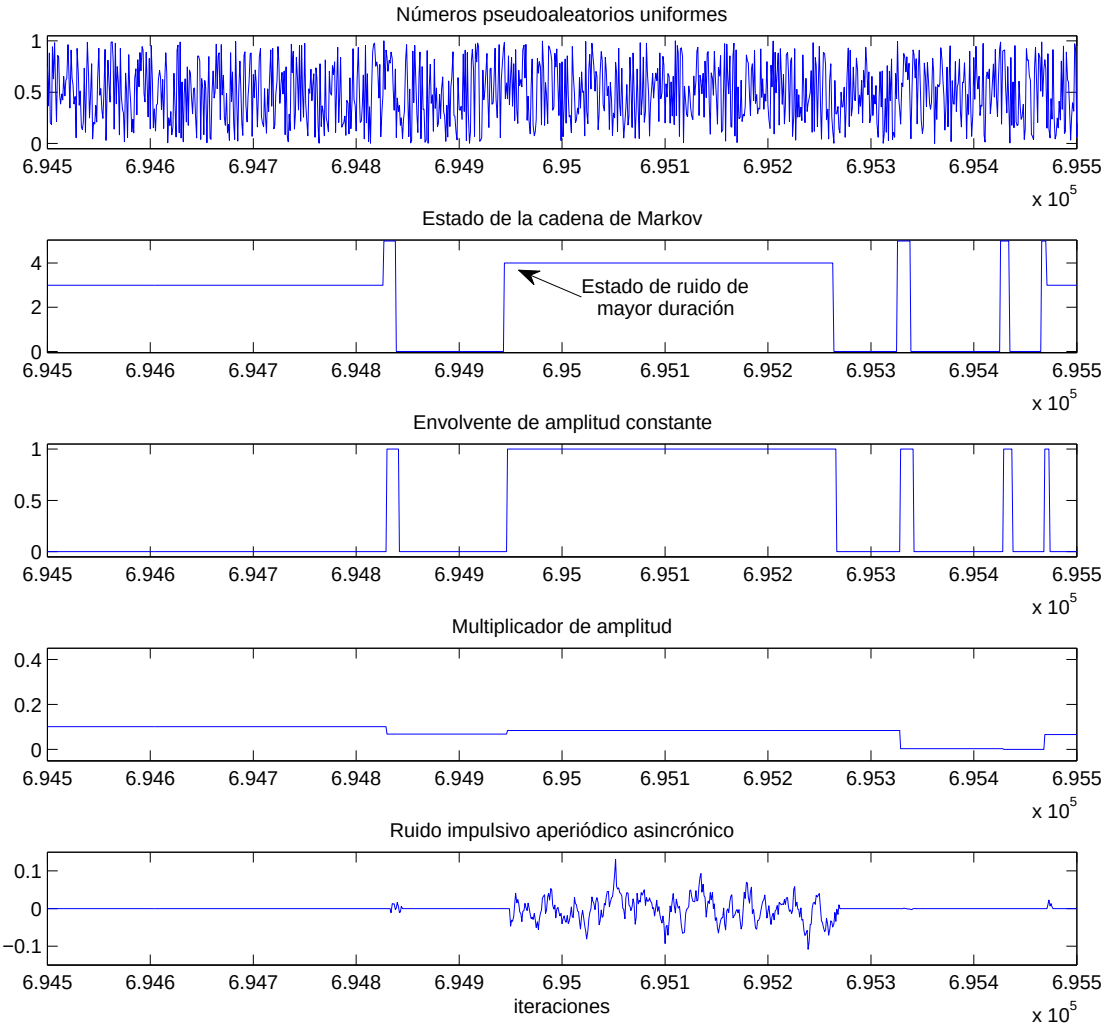


FIGURA 7.28: Detalle de la emulación de ruido impulsivo aperiódico asincrónico.

7.6.3. Comentarios

Acerca de las matrices de referencia y la tasa de impulsos En §3.11.1.4 se dijo que la tasa de impulsos suele ser 1 impulso por segundo para ambientes de baja perturbación y 100 impulsos por segundo para ambientes ruidosos, para un umbral de detección de 100 mV [1]. Sin embargo, la matriz clase *Moderate* sin nuestra cuantización tiene una tasa de 275,6 impulsos por cada millón de muestras. Si tomamos en cuenta la tasa de muestreo utilizada en [49] fue de 50 Msps, ese valor representa una tasa de 13780 impulsos por segundo, muy superior a los valores anteriormente mencionados. A su vez, en las mediciones de [49] se observaron 70000 impulsos en 333 minutos que da una tasa promedio de 4,5 impulsos por segundo. Sin embargo, la matriz propuesta por los autores de ese paper da una tasa de 2500 impulsos por segundo. Por otro lado, los impulsos contabilizados en dichas mediciones tuvieron un umbral

de detección de 100 mV, mientras que el modelo de generación de amplitudes puede generar impulsos de menor amplitud que no hubiesen sido detectados. Y por último, si se quisiesen generar dichas tasas de impulsos con el modelo particionado de Markov sería necesario utilizar precisiones superiores a los 24 bits. La justificación de dicha afirmación está basada en que la tasa promedio queda acotada[5] inferiormente por el estado de mayor $\bar{t_i}$ posible y esto lleva a que la cota es $f_s/2^N$ donde N es la cantidad de bits y f_s la frecuencia de muestreo. Si suponemos $f_s = 80$ Msps necesitamos 26 bits sólo para satisfacer la cota. Para obtener un valor promedio necesitaríamos varios bits más. En conclusión, nosotros nos enfocamos en poder representar los canales de referencia sugeridos por OPERA y de ser necesario extender la precisión esto no sería problema dada la simplicidad de nuestra arquitectura.

Comentario acerca de la cuantización de las matrices de referencia Anteriormente mencionamos que nosotros elegimos cuantizar las matrices para obtener un tipo de dato `UFix_17_16`. A continuación mostraremos en la tabla 7.3 las diferencias entre la versión cuantizada y la original. Es lógico que las diferencias más grandes ocurran en el caso de menor perturbación. Esto se debe a que el error de cuantización causado por la precisión numérica afecta en mayor proporción a valores chicos.

Clase	Tasa de impulsos por millón de muestras	Duración relativa de la perturbación
Moderate	275,6	0,270309 %
Moderate `UFix_17_16`	256,1	0,171073 %
Medium	541,3	0,480514 %
Medium `UFix_17_16`	547,2	0,482552 %
Strong	2741,9	3,021741 %
Strong `UFix_17_16`	2774,9	3,071887 %

Tabla 7.3: Comparación de matrices de probabilidades de transición de referencia vs sus versiones cuantizadas

7.7. Evaluación del ruido impulsivo en ráfagas

El ruido impulsivo en ráfagas es una característica particular del canal BPL y por lo tanto un componente importante del emulador. En esta sección podremos ver que se cumple con los requisitos **RQ_07** parte 6 y **RQ_12**.

[5] acá no estamos haciendo referencia a una cota formal sino suponiendo que el estado de mayor duración tiene un valor muy cercano a 1 de probabilidad estacionaria. Ver ecuación (3.49).

7.7.1. Comparación estadística

OPERA [3] no especifica matrices de referencia para este tipo de ruido según el modelo propuesto en [1] y planteado en §3.12. En cambio, dice utilizar como fuente para los impulsos individuales el mismo proceso de Markov para ruido impulsivo aperiódico asincrónico o el ruido impulsivo periódico asincrónico. En ambos casos resulta confuso interpretar que pasaría si habilito estos ruidos en simultáneo con el ruido en ráfagas, ya que al utilizar la misma fuente esto haría que los instantes de tiempo de los ruidos estén muy fuertemente correlacionados. Esta modalidad de implementación se puede ver en una captura de pantalla de la GUI que controla el emulador y ahí se ve que existen los tres casos: *Moderate*, *Medium* y *Strong* [3]. Además, es incongruente que la tasa de impulsos dentro de una ráfaga esté dada por cualquiera de las tres versiones de este proceso. La tasa de impulsos suele ser de varios órdenes de magnitud superior que el ruido impulsivo aperiódico asincrónico y es al utilizar el mismo proceso cuando surge la incongruencia. Por estos motivos hemos diseñado nuestras propias matrices para mostrar el funcionamiento de este bloque. Recapitulando, hay dos matrices de probabilidades de transición: la del nivel superior y la del nivel inferior. El nivel superior describe los intervalos de ráfagas y el nivel inferior los impulsos dentro del intervalo de ráfagas. Las matrices utilizadas son las siguientes

$$\mathbf{P}_{sup} = \begin{bmatrix} 0{,}999969482421875 & 0{,}000030517578125 \\ 0{,}005004882812500 & 0{,}994995117187500 \end{bmatrix} \tag{7.3}$$

$$\mathbf{P}_{inf} = \begin{bmatrix} 0{,}964996337890625 & 0{,}035003662109375 \\ 0{,}024993896484375 & 0{,}975006103515625 \end{bmatrix} \tag{7.4}$$

Analizando la matriz $\mathbf{P}_{sup}$ podemos sacar las siguientes conclusiones

- La siguiente distribución de probabilidad estacionaria

Nro de Estado	probabilidad estacionaria	$\bar{t_i}$ [muestras]	$\bar{t_i}$ [seg] a 80 Msps
0	0,993939393939394	32768,00	409,600000 μs
1	0,006060606060606	199,80	2,497561 μs

- Tasa de impulsos R_{imp}: 30,33 impulsos por cada millón de muestras, que a 80 Msps es 2426,6 impulsos por segundo.
- Duración relativa de los intervalos de ráfaga: 0,606061 %. Notar que esta métrica es distinta a la duración relativa de la perturbación. La causa de esto es que durante el intervalo de ráfaga hay intervalos sin ruido entre impulsos.

Se simularon 10^6 iteraciones y se obtuvieron los siguientes resultados que serán expuestos en las tablas 7.4 y 7.5. En la primer tabla podemos ver que tanto las probabilidades estacionarias como las duraciones promedio de los intervalos de ráfaga

estimados convergen a sus correspondientes valores teóricos. Esto implica que la calidad de números pseudoaleatorios generados es buena. En la segunda tabla vemos que la tasas de impulsos son muy parecidas aún cuando la cantidad de iteraciones es relativamente chica para una tasa teórica de 30,33 impulsos por millón.

Nro de Estado	Probabilidad estacionaria (teórica)	Probabilidad estacionaria (estimación)	Duración promedio en el estado i (teórica)	Duración promedio en el estado i (estimación) [N de $\frac{1}{N}\sum x_i$]
0	0,993939393939394	0,994375005624994	32768,00	35109,07 [29]
1	0,006060606060606	0,005624994375006	199,80	193,97 [29]

Tabla 7.4: Comparación 1 entre valores teóricos y simulados del proceso particionado de Markov del nivel superior

	Tasa de impulsos por millón de muestras	Duración relativa de los grupos de ráfagas
Teórico	30,33	0,606061 %
Simulado	29,00	0,562499 %

Tabla 7.5: Comparación 2 entre valores teóricos y simulados del proceso particionado de Markov del nivel superior

La matriz del proceso del nivel inferior ha sido diseñada teniendo en cuenta que durante una ráfaga la tasa de impulsos es muy alta. Analizando la matriz $\mathbf{P}_{inf}$ tenemos

- La siguiente distribución de probabilidad estacionaria

Nro de Estado	probabilidad estacionaria	$\bar{t_i}$ [muestras]	$\bar{t_i}$ [seg] a 80 Msps
0	0,416581892166836	28,57	0,357105 μs
1	0,583418107833164	40,01	0,500122 μs

- Tasa de impulsos dentro de la ráfaga $\mathsf{R}_{imp_{II}}$: 14581,89 impulsos por cada millón de muestras, que a 80 Msps es 1,17 millones de impulsos por segundo.
- Duración relativa de la perturbación dentro de una ráfaga: 58,341811 %.

Un punto a tener en cuenta en la simulación es que en realidad el proceso de nivel inferior es un proceso intervenido ya que se fuerzan las condiciones iniciales a que haya impulso (estado "1") en el comienzo de la ráfaga. Sin embargo, las simulaciones muestran que este hecho es de poca relevancia en tasas de impulsos muy elevadas ya que la cantidad de intervenciones es despreciables y además en este ejemplo existe un 58 % de probabilidad que ya se encuentre en este estado. Por estos motivos los

números de la simulación son como los de un proceso sin intervención. Nuevamente presentaremos la información en dos tablas

Nro de Estado	Probabilidad estacionaria (teórica)	Probabilidad estacionaria (estimación)	Duración promedio en el estado i (teórica)	Duración promedio en el estado i (estimación) [N de $\frac{1}{N}\sum x_i$]
0	0,416581892166836	0,418605581394419	28,57	28,93 [14468]
1	0,583418107833164	0,581394418605581	40,01	40,184 [14469]

Tabla 7.6: Comparación 1 entre valores teóricos y simulados del proceso particionado de Markov del nivel inferior

	Tasa de impulsos por millón de muestras	Duración relativa de la perturbación en una ráfagas
Teórico	14581,89	58,341811 %
Simulado	14467,98	58,139442 %

Tabla 7.7: Comparación 2 entre valores teóricos y simulados del proceso particionado de Markov del nivel inferior

7.7.2. Gráficos

A continuación presentaremos dos figuras con gráficas de las señales más características del generador de ruido impulsivo en ráfagas. Las señales graficadas son:

- *Números pseudoaleatorios uniformes*: estos son los que alimentan a la cadena de Markov del nivel superior.
- *Estado de la cadena de Markov del nivel superior*: son los estados, del 0 al 1, que va tomando la cadena de Markov, donde "1" indica la existencia de un intervalo de ráfaga.
- *Estado de la cadena de Markov del nivel inferior*: son los estados, del 0 al 1, que va tomando la cadena de Markov, donde "1" indica la existencia de un impulso. Notar que esta cadena corre continuamente pero sólo tiene relevancia dentro de un intervalo de ráfaga. Otro detalle de diseño importante es que se fuerza al estado "1" cuando se inicia el intervalo de ráfaga, de manera que se está forzando el estado inicial de la cadena en cada comienzo de ráfaga. Vale la pena mencionar que esta cadena de Markov tiene una fuente distinta de números pseudoaleatorios.
- *Envolvente efectiva de impulsos*: es la señal que determina si a la salida va a haber ruido o no. No contempla la potencia del impulso.
- *Multiplicador de amplitud*: esta señal es la salida de un proceso exponencial que mantiene fijo su valor durante cada intervalo de ráfaga. La motivación de mantener constante el valor es que se atribuye que los impulsos dentro de una ráfaga provienen de la misma causa. Se utiliza para multiplicar los impulsos y cambiar su potencia.

- *Ruido impulsivo en ráfagas*: es lo que queremos generar.

En la figura 7.29 se puede ver a través de la envolvente efectiva de impulsos que la tasa de impulsos es mucho menor que para el caso del ruido impulsivo aperiódico impulsivo (figura 7.27). También vemos que el estado de la cadena de Markov de nivel inferior alterna de estado entre "0" y "1" muy rápidamente, lo que implica una tasa de impulsos muy grande como es característico dentro de un intervalo de ráfaga.

En la figura 7.30 se observa con mayor detalle una porción de la misma simulación. En el gráfico se pueden ver claramente dos ráfagas y se pueden distinguir los impulsos dentro de cada ráfaga. También se observa que el multiplicador de amplitud, que es la salida de un proceso exponencial, sólo cambia de valor cuando comienza otra nueva ráfaga. De esta manera la potencia de los impulsos dentro de una ráfaga está correlacionada intentando emular a la realidad que en general se caracteriza por que dichos impulsos suelan provenir de una misma fuente.

7.8. Issues de simulación

Durante el proceso de verificación de nuestros diseños nos encontramos con un inconveniente para simular los ruidos impulsivos no periódicos. Tanto en el ruido impulsivo aperiódico asincrónico como en el ruido impulsivo en ráfagas, entran en juego eventos de muy baja probabilidad. Por este motivo es necesario trabajar con una cantidad de datos muy grande ($\sim 10^8$ o superior). Esto plantea un desafío adicional a la hora de desarrollar scripts en MATLAB. Sin embargo en *Simulink* es muy complicado trabajar con un número de iteraciones muy grande y existen dos problemas: el almacenamiento y la velocidad.

Para poder trabajar con grandes números de muestras deberíamos ser cautelosos y configurar adecuadamente los distintos componentes[6] capaces de almacenar señales en memoria para que almacenen sólo las muestras más recientes. Si quisiésemos volcar las señales al workspace nos veríamos limitados por la cantidad de memoria RAM[7]. Otra alternativa es direccionar las señales a un archivo. Pero si cargásemos el archivo en el workspace tendríamos la misma limitación de memoria. Sin embargo, existe la posibilidad de cargar porciones de la señales almacenadas. De esta manera, podríamos desarrollar scripts que procesen lotes de información y mitigar dicha limitación al costo de incrementar la dificultad de programación.

El problema de la velocidad es el más importante y no es mitigable. *Simulink* no fue diseñado para realizar simulaciones de grandes números sino que se enfocó en la funcionalidad y capacidad de simulación. Si bien *Simulink* posee métodos de aceleración ninguno de estos es aplicable cuando se utiliza junto con *System Generator*. Además, estas simulaciones no aprovechan de manera eficiente los nuevos procesadores *multi-core*. Para tener una idea de la velocidad de simulación, cuando nos

[6] por ejemplo, el *Scope* de *Simulink* y el *WaveScope* de *System Generator*.

[7] cada valor de cada señal es un *double* y por lo tanto son 8 Bytes por valor.

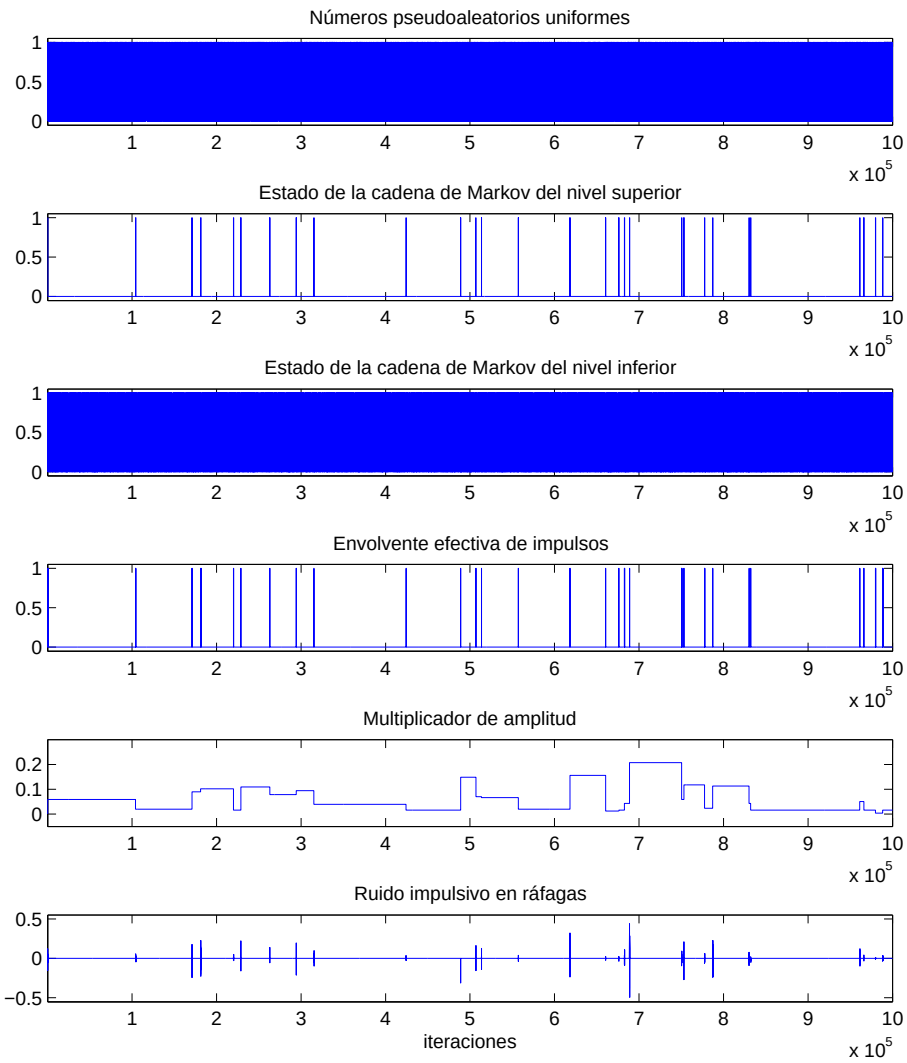


FIGURA 7.29: Emulación de ruido impulsivo en ráfagas

encontrábamos en las etapas finales del diseño simulaba a una tasa de 240 mil muestras por hora en un sistema *Intel Core i7 2600K @3.8 GHz*. A esa velocidad se necesitarían aproximadamente 17,4 días para simular 10^8 iteraciones, convirtiendo la tarea en impráctica.

Verificación por hardware En las últimas etapas de desarrollo decidimos incorporar métodos de verificación por hardware. Uno de ellos fue la incorporación de una propiedad intelectual de *Xilinx* llamada *Chipscope Pro* (ver apéndice G.2). Y el segundo método constó en agregar unos contadores para los ruidos impulsivos no periódicos. Estos contadores se leen desde el software de *MicroBlaze* mediante el uso de registros compartidos y luego se visualizan en el software GUI. Con ellos se puede estimar la tasa de impulsos por cada millón de muestras para el caso del ruido

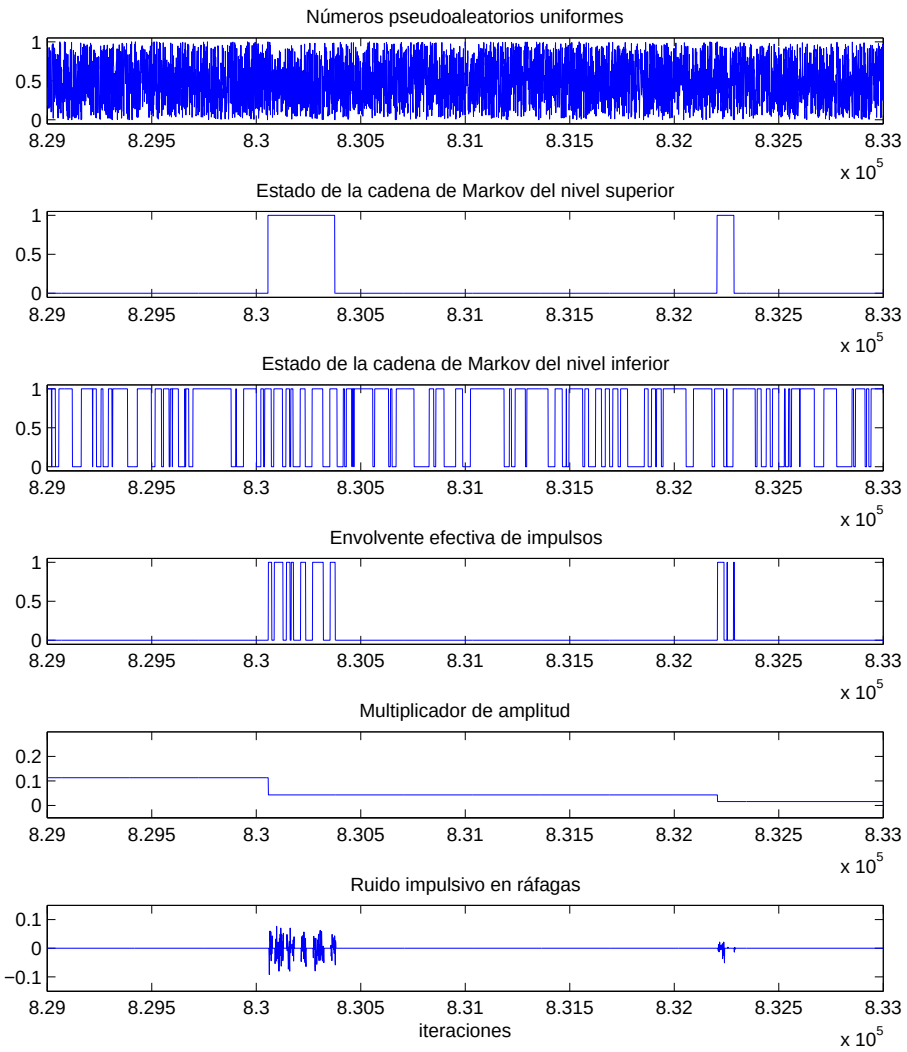


FIGURA 7.30: Detalle de la emulación de ruido impulsivo en ráfagas

impulsivo aperiódico asincrónico, o la tasa de ráfagas por cada millón de muestras para el caso del ruido impulsivo en ráfagas. La ventaja de este método es que el FPGA genera decenas de millones de muestras por segundo contra las 240 mil muestras por hora de la simulación en una PC. Por ejemplo, en la figura 7.31 se ve que la estimación de la tasa de impulsos del ruido impulsivo aperiódico asincrónico para el caso *Strong* se aproxima a la teórica (ver tabla 7.3). Para la estimación se ve que se utilizaron 732 millones de muestras, algo impracticable desde *Simulink* y *System Generator*.

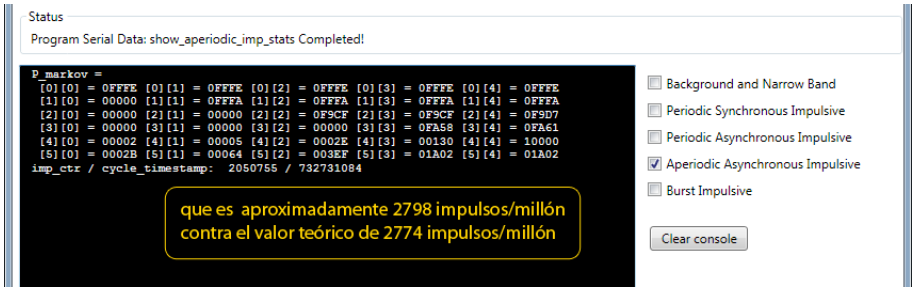


FIGURA 7.31: Captura de pantalla del software GUI. Contadores de estadísticas para el ruido impulsivo aperiódico asincrónico.

7.9. Evaluación del sistema completo

En esta sección mostraremos el efecto conjunto de todo el sistema. Para esto inyectaremos un pulso rectangular periódico de amplitud unitaria y veremos como es afectado por la transferencia del canal y los ruidos simulados. Hemos elegido el canal de referencia 9 (ver §A.1). De esta manera podremos apreciar cuan agresivo puede llegar a ser el canal BPL.

En primer lugar, en la figura 7.32 vemos el efecto del ruido impulsivo periódico sincrónico y del ruido impulsivo periódico asincrónico. Todavía no se observa el ruido de fondo y el de banda angosta porque a esta altura de la simulación la IFFT no ha empezado a producir salida. Aquí podemos observar que el ruido impulsivo periódico asincrónico tiene un efecto para nada despreciable contrario a lo que OPERA afirmaba. En este caso el ruido impulsivo asincrónico queda enmascarado dentro del otro ruido.

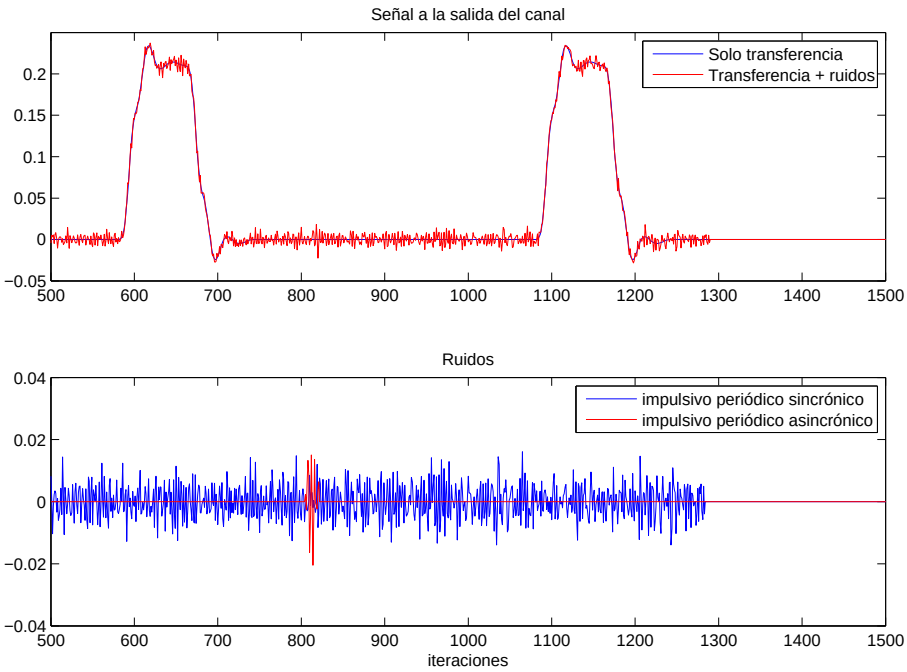


FIGURA 7.32: Detalle de la emulación. Efecto de los ruidos impulsivos periódicos

En segundo lugar, en la figura 7.33 vemos el efecto del ruido impulsivo aperiódico asincrónico característico de este canal. Notamos que su amplitud es comparable con la de la señal inyectada y por lo tanto su impacto será grave. Aquí sí podemos observar el impacto del ruido impulsivo periódico asincrónico y el del ruido de fondo junto con el ruido de banda angosta, que se aprecia en la oscilación del trazo verde.

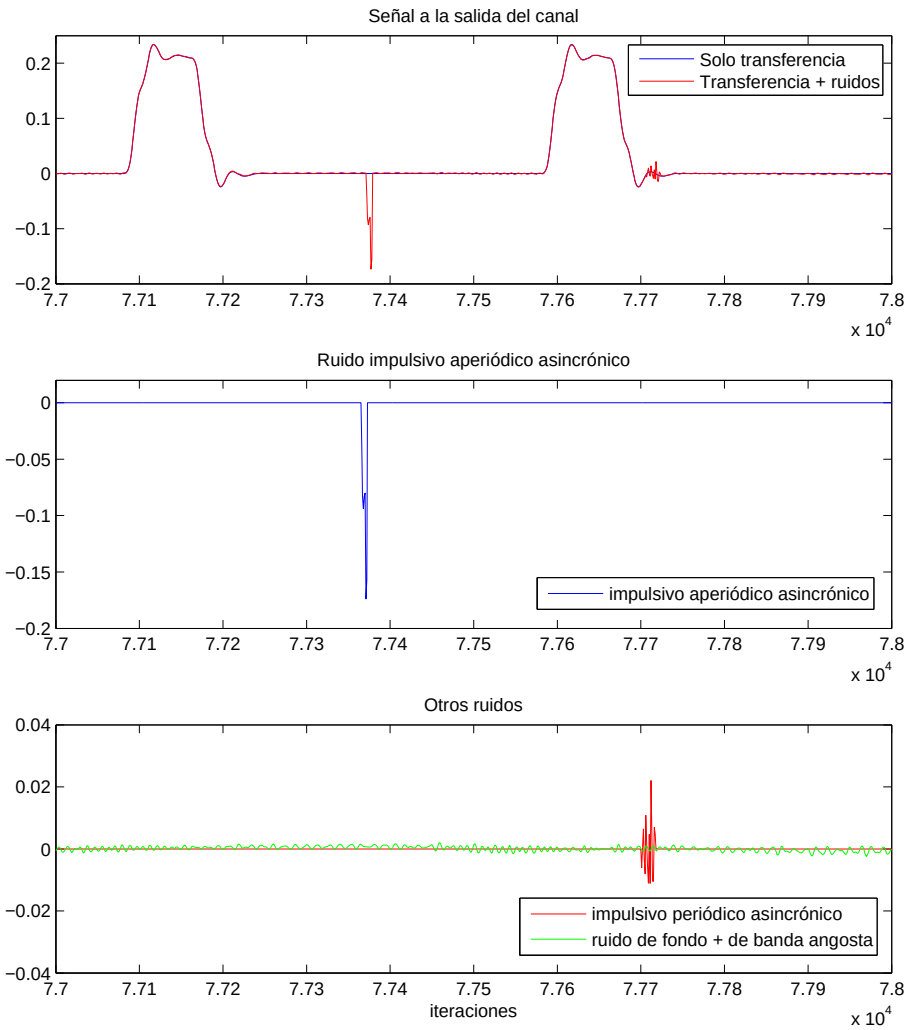


FIGURA 7.33: Detalle de la emulación. Efecto del ruido impulsivo aperiódico asincrónico y otros ruidos

En tercer lugar, en la figura 7.34 observamos una ráfaga de impulsos que justo se superpone con impulsos generados por el generador de ruido impulsivo aperiódico asincrónico. El impacto del ruido impulsivo en ráfagas es notablemente superior a todos los otros ruidos debido a su intensidad y a su extensa duración. Sin embargo en este caso veremos que la potencia de esta ráfaga ha sido moderada, a diferencia del próximo caso. A su vez se observan los efectos de los otros ruidos pero que sólo son apreciables fuera de la ráfaga.

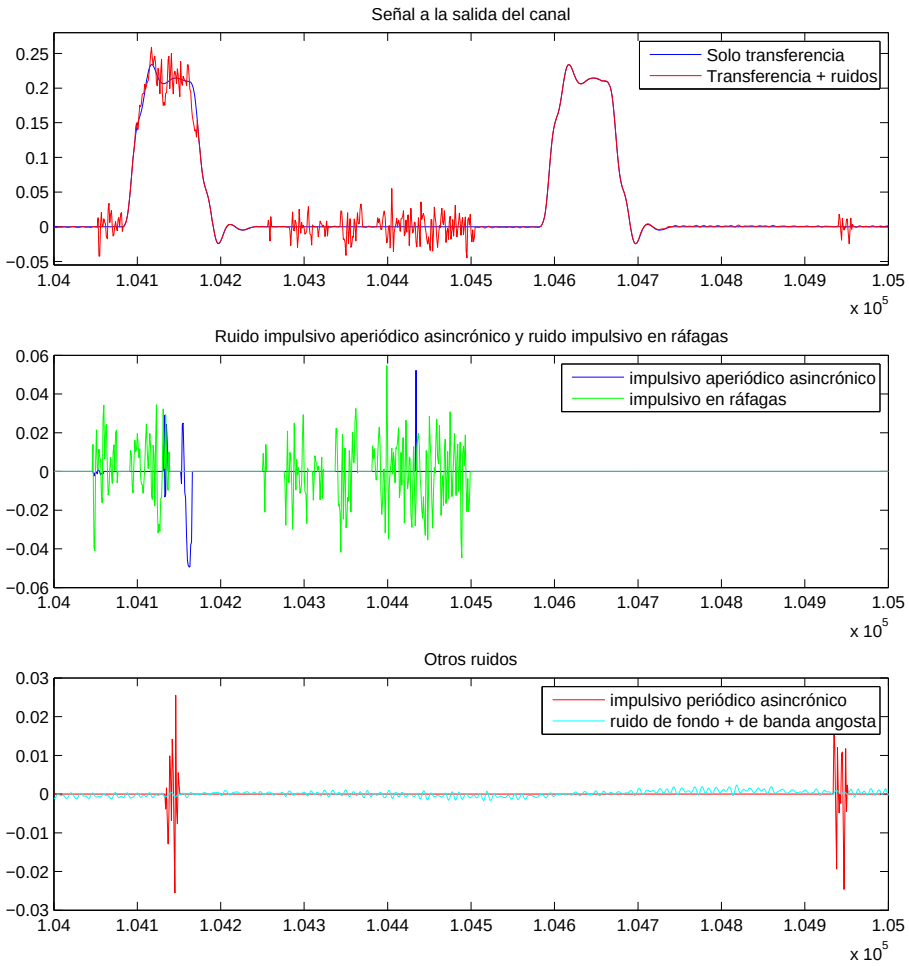


FIGURA 7.34: Detalle de la emulación. Efecto del ruido impulsivo en ráfagas y otros ruidos

En último lugar se puede observar el efecto de una ráfaga de gran magnitud en la figura 7.35. Probablemente provocará errores irrecuperables en la PPDU. La distorsión de la señal es del mismo orden llevando la SNR a aproximadamente 0 dB. También vemos la existencia de los otros ruidos pero es casi indistinguible debido a las características de esta ráfaga.

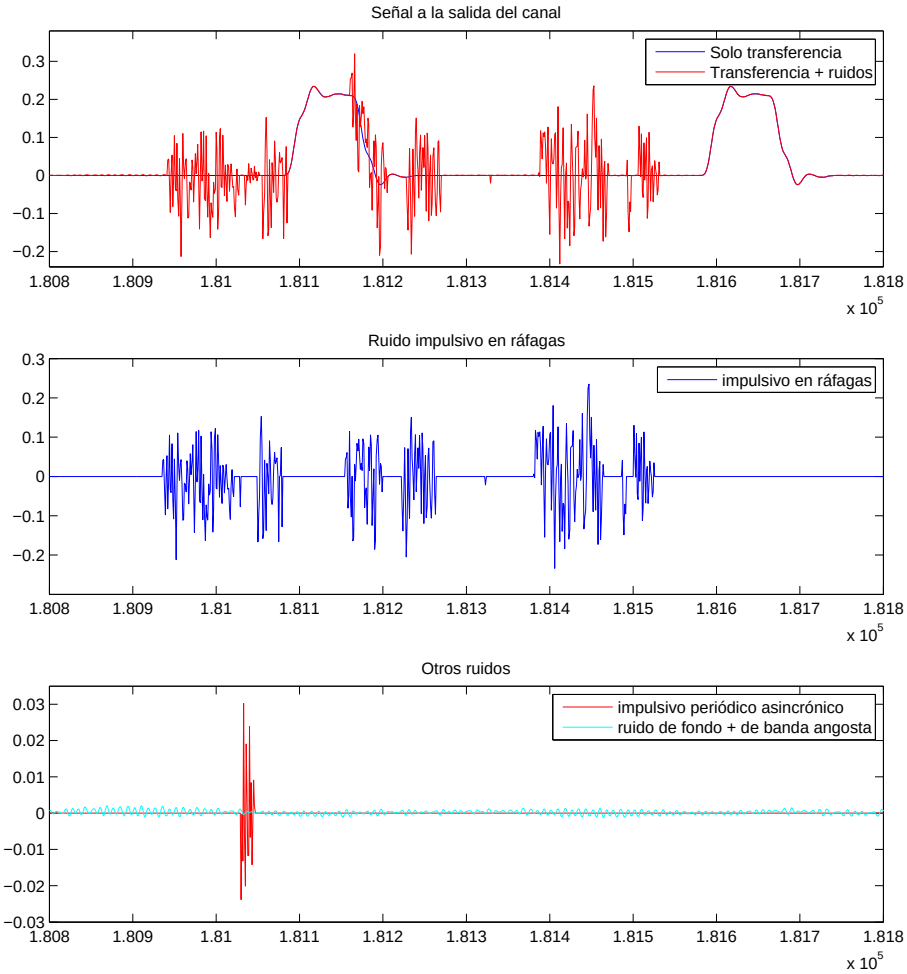


FIGURA 7.35: Detalle de la emulación. Efecto del ruido impulsivo en ráfagas. Intensidad fuerte

7.10. Resumen

En este capítulo se realizó la evaluación del diseño implementado en *System Generator*. La evaluación se basó en simulaciones que reproducen el comportamiento exacto dentro del FPGA. También se verificó el correcto funcionamiento de los distintos módulos que componen el emulador. A su vez se realizó un análisis estadístico sobre los ruidos impulsivos aperiódicos. Se mencionó la existencia de issues para la simulación de grandes cantidades de muestras para verificar estos últimos ruidos.

8

Conclusiones

8.1. Conclusiones

En este trabajo se ha realizado una investigación del canal de comunicaciones sobre líneas de tensión para comunicaciones de banda ancha. La información presentada abarca los temas de manera amplia, concisa y ordenada entrando en mayor detalle cuando es necesario e intentando proveer un contexto autocontenido de la situación. Esto permite a lectores novatos en este área introducirse fácilmente en el tema a diferencia de la mayoría de la bibliografía disponible[1] que suele ser fuerte en temas particulares e incompleta en otros.

En el desarrollo de sistemas de comunicaciones es fundamental verificar los diseños. En etapas tempranas esto se hace mediante simulaciones por computadora, pero a medida que el proyecto avanza y llega a materializarse son necesarias las pruebas de campo y de laboratorio. Para poder evaluar objetivamente a los prototipos es fundamental poder contar con un ambiente controlado. Las pruebas de campo son necesarias pero tienen baja controlabilidad y pueden ser costosas. Las pruebas de laboratorio con maquetas tienen un alto grado de control pero son aún más caras[2]. Entonces aparece la posibilidad de emular aquellos escenarios y es ahí es donde hace foco nuestro trabajo. La velocidad y complejidad de cálculos necesarios obligan a utilizar una estructura paralela. Los FPGAs tienen una gran capacidad de paralelización y de optimización ya que se restringe al hardware a realizar funciones específicas y no generales. Por esos motivos el diseño se ha realizado utilizando un FPGA.

El proceso de diseño se realizó en etapas que fueron desde lo más abstracto, los modelos matemáticos, hasta la etapa final de implementación en hardware, arquitec-

[1] hecho que es citado por Dostert en el prólogo de [13].

[2] al menos en este ámbito donde habría que construir un sistema de distribución eléctrica y tener absoluto control sobre el mismo.

turas para el FPGA. Este hecho se evidencia en la estructura de capítulos de esta tesis, donde en cada capítulo se va entrando cada vez en mayor detalle para cada uno de los componentes del sistema.

En base al estudio realizado se propone una arquitectura de implementación de un emulador de canal para ambientes BPL. La arquitectura expuesta no sólo cumple con los requisitos presentados por el consorcio OPERA [3] sino que extiende la capacidad de emulación dando como resultado una simulación más flexible y parecida a la realidad. Las mejoras son de evidente notoriedad en la generación de ruido coloreado, ruido de banda angosta y ruido impulsivo en ráfagas.

Unos de los objetivos complementarios de la tesis es facilitar la tarea de implementación para profesionales que tienen un perfil más orientado al procesamiento de señales y no al hardware. La tesis constituye entonces una guía que deja un *know-how* reutilizable para los integrantes del LPSC (Laboratorio de Procesamiento de Señales de las Comunicaciones) de la Facultad de Ingeniería de la UBA. Para ello se utilizaron herramientas de diseño de alto nivel provistas por Xilinx. El diseño de la lógica de procesamiento de señales fue realizado con el System Generator, mientras que el control de la misma se realizó utilizando el paquete Xilinx EDK[3] para sintetizar un *soft-processor* y programándolo. Estas herramientas aceleran el diseño y bajan la barrera inicial de conocimiento necesario para poder implementar en hardware. Sin embargo, nos encontramos con varios problemas causados por bugs de software, documentación incompleta o un mal soporte al kit XUPV5 comparado con los kits de Xilinx cuyo costo es alrededor del doble. Para solventar algunos inconvenientes hubo que modificar parámetros que requerían de un nivel de *expertise* considerable.

Esta tesis forma parte de un proyecto más grande donde se desea obtener un sistema completo de transmisión, recepción y testing en BPL y por lo tanto implica llegar a un prototipo de emulador de funcionalidad completa. Actualmente habría que extender este trabajo adicionando el módulo de adquisición y conversión de datos. Alineado con estos mismos intereses, se ha puesto gran énfasis en el desarrollo de una interfaz de control que permita que el emulador sea utilizado y configurado por personas no interiorizadas en este trabajo. Para ello se insertó un *soft-processor* en el FPGA que es el encargado de controlar cada uno de los módulos que componen el emulador. Un aporte importante de esta tesis es haber desarrollado una metodología de control y parametrización que no sólo es aplicable a este diseño sino a cualquier otro diseño realizado con System Generator. El *soft-processor* ejecuta un programa que se comunica via RS232 con un software GUI en una PC. Es este último el que presenta una interfaz amigable al usuario para poder controlar y configurar el emulador. El software del *soft-processor* se desarrolló en C y el de la GUI en C# y para ambos casos se cuenta con SDKs que permiten el desarrollo ágil y facilitan el debugging.

La experiencia de haber elegido implementar el emulador hizo surgir desafíos ocultos e interesantes donde realmente se aplica el ingenio del ingeniero. Como en

[3] Embedded Development Kit

todo diseño hemos debido experimentar con múltiples diseños para un misma tarea y reiteradas etapas de optimización que quizás no pueden apreciarse en este escrito.

Es importante evidenciar que el diseño final del emulador puede ser utilizado para simular otro tipo de canales debido a que el canal BPL tiene una mayor diversidad de fuentes de ruido. Entonces, en canales "más buenos" podemos anular ruidos no existentes en esos ambientes. Otro hecho notable es que debido a la metodología de diseño utilizada es fácil agregar o reemplazar nuevos bloques y controlar los mismos a través del *soft-processor*. Por ejemplo, fácilmente podría cambiarse el bloque de la transferencia por otro que sea capaz de simular canales *wireless*.

Por último, el diseño se validó con simulaciones en System Generator que son 100 % representativas del hardware sintetizado ya que son *bit-accurate* y *cycle-accurate* [46, p. 17].

8.2. Próximos Pasos

A continuación expondremos algunos posibles caminos para continuar este trabajo.

- Agregar los módulos de ADC/DAC y hacer pruebas con módems reales.
- Mejorar los módulos de ruido impulsivo aperiódicos asincrónico y ruido impulsivo en ráfagas para que puedan lograr tasas de impulsos más bajas.
- Realizar mediciones de canales (transferencia y ruidos) y estimar los parámetros que repliquen dicho canal para poder incorporarlos a la base de datos de configuraciones del emulador.
- Buscar optimizar el uso de recursos y la velocidad a través de la reingeniería de algunos bloques.

Apéndice A

Canales de referencia

En este apéndice describiremos los canales de referencia sugeridos por el consorcio OPERA en el documento *Theoretical postulation of PLC channel model* [1]. Se presentarán tanto los canales de redes de acceso como de redes hogareñas.

A.1. Redes de accesso

Para las redes de acceso se han definido tres clases de longitudes

- distancias cortas, alrededor de 150m
- distancias medias, alrededor de 250m
- distancias largas, alrededor de 350m

Debido al método de medición utilizado no se ha medido el tiempo absoluto de propagación. Ello explica por ejemplo porqué el canal de referencia 4 tiene su pico máximo antes que el canal de referencia 1, cuya longitud es menor que la del primero.

Para la evaluación de la transferencia se ha utilizado $v_p = 1{,}53 \cdot 10^8$ m/s como velocidad de propagación.

Observaciones Vale la pena repetir que estamos tratando con un modelo fenomenológico y por ello veremos valores de parámetros como ser $a_0 = 0$. Esto no implica que no haya atenuación debido sólo a la distancia (2.16) , sino que el factor $e^{-a_0 d}$ ha sido incluido en los coeficientes g_i del modelo (2.28).

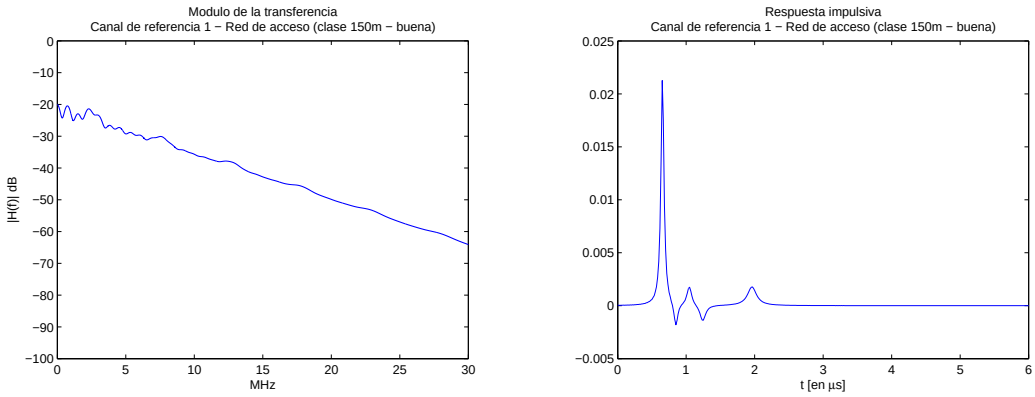
A.1.1. Clase de 150m

A.1.1.1. Canal de referencia 1 - Calidad Buena

Este canal representa un canal sin ramificaciones. Sólo se utilizan 5 caminos para su modelado. Exhibe baja atenuación y no contiene *notches* en frecuencias selectivas. La respuesta impulsiva está dominada por el eco principal y decae luego de 1,5μs. Por estos motivos, es casi un canal ideal.

Parámetros de atenuación		
$a_0 = 0$	$a_1 = 1{,}65 \cdot 10^{-9}[s/m]$	$k = 1$
Parámetros de camino		
i	$\mathbf{g}_i$	$\mathbf{d}_i$ $[m]$
1	0,09	100
2	−0,012	130
3	0,012	160
4	−0,012	190
5	0,022	300

Tabla A.1: Parámetros del canal de referencia (clase 150m, bueno)



(a) Módulo de la transferencia (b) Respuesta al impulso

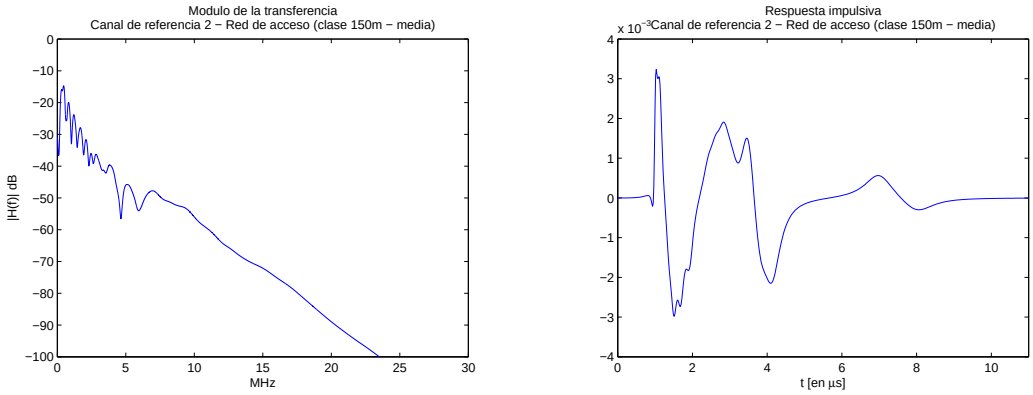
FIGURA A.1: Canal de referencia 1 - clase 150m - calidad buena

A.1.1.2. Canal de referencia 2 - Calidad Media

Este canal posee un comportamiento pasabajos más marcado. Si bien no presenta *notches* selectivos en frecuencia, la respuesta al impulso exhibe una forma compleja y por lo tanto se necesitan 17 caminos para su descripción en el modelo. La duración de la respuesta al impulso es bastante más larga y de alrededor de 8μs.

Parámetros de atenuación		
$a_0 = 0$	$a_1 = 2{,}8 \cdot 10^{-9}[s/m]$	$k = 1$
Parámetros de camino		
i	$\mathbf{g}_i$	$\mathbf{d}_i$ $[m]$
1	− 0,15	150,8
2	0,165	152,3
3	0,032	172
4	−0,014	210,4
5	−0,035	230
6	−0,035	258
7	−0,03	294
8	0,015	370
9	0,022	400
10	0,04	435
11	0,02	468
12	−0,015	494
13	0,0865	534
14	−0,062	581
15	−0,083	632
16	0,05	1070
17	−0,035	1224

Tabla A.2: Parámetros del canal de referencia (clase 150m, media)



(a) Módulo de la transferencia (b) Respuesta al impulso

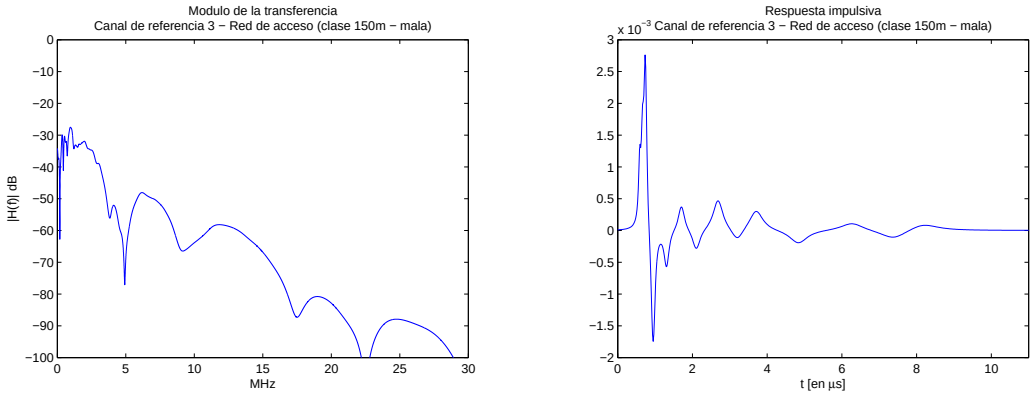
FIGURA A.2: Canal de referencia 2 - clase 150m - calidad media

A.1.1.3. Canal de referencia 3 - Calidad Mala

El tercer canal muestra una característica pasabajos similar al anterior pero ahora se observa *notches* selectivos en frecuencia. La medición fue realizada en una zona residencial donde las casas estaban dispuestas de manera regular. Esto se manifiesta en el gran número de ecos que se observan en la respuesta al impulso.

Parámetros de atenuación		
$a_0 = 0$	$a_1 = 2{,}8 \cdot 10^{-9}[s/m]$	$k = 1$
Parámetros de camino		
i	$\mathbf{g}_i$	$\mathbf{d}_i$ $[m]$
1	0,01832	113,2
2	0,00516	90,1
3	0,00765	101,8
4	−0,01031	143
5	−0,008	148
6	−0,00711	200
7	0,00676	261
8	−0,00676	322
9	0,01263	411
10	−0,00622	490
11	0,01156	567
12	−0,00978	740
13	0,00747	960
14	−0,01049	1130
15	0,00871	1250

Tabla A.3: Parámetros del canal de referencia (clase 150m, mala)



(a) Módulo de la transferencia (b) Respuesta al impulso

FIGURA A.3: Canal de referencia 3 - clase 150m - calidad mala

A.1.2. Clase de 250m

Los canales de esta clase tiene un comportamiento pasabajos más fuerte que la clase de 150 m.

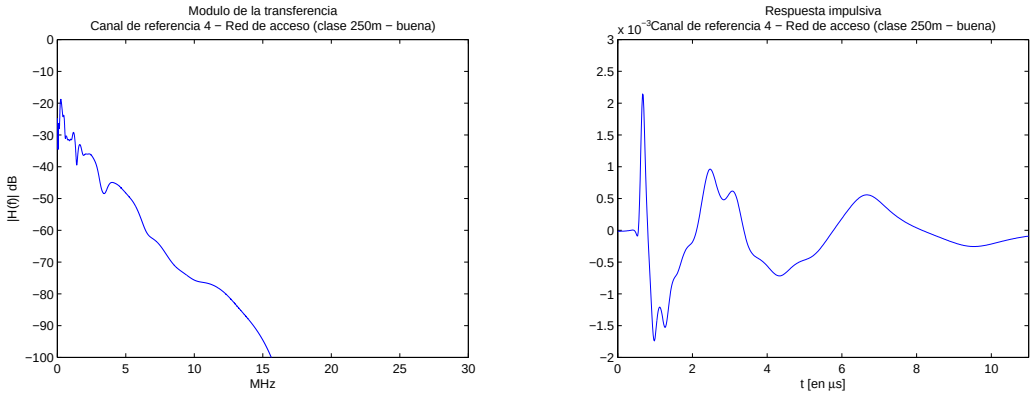
A.1.2.1. Canal de referencia 4 - Calidad Buena

Este canal no presenta *notches* en frecuencia pero sin embargo la respuesta al impulso es larga y con una forma compleja por lo que deben utilizarse 14 caminos

para modelarla. En este escenario, en el mejor de los casos se podría aprovechar un ancho de banda de aproximadamente 14 MHz. Este cálculo se fundamenta en que un valor típico de emisión máxima de señal es -50 dBm que junto con la atenuación de 100 dB tenemos un nivel de señal en el receptor de -150 dBm que es igual al ruido de fondo. Además se supuso que se utiliza el espectro desde 1 MHz hacia las frecuencias superiores.

Parámetros de atenuación		
$a_0 = 0$	$a_1 = 5 \cdot 10^{-9}[s/m]$	$k = 1$
Parámetros de camino		
i	$\mathbf{g}_i$	$\mathbf{d}_i$ $[m]$
1	− 0,007	85
2	0,034	103
3	−0,03	148
4	−0,031	195
5	−0,013	245
6	−0,015	315
7	0,053	376
8	−0,022	438
9	0,07	476
10	−0,04	530
11	−0,052	660
12	−0,04	800
13	0,088	1015
14	−0,053	1450

Tabla A.4: Parámetros del canal de referencia (clase 250m, buena)



(a) Módulo de la transferencia (b) Respuesta al impulso

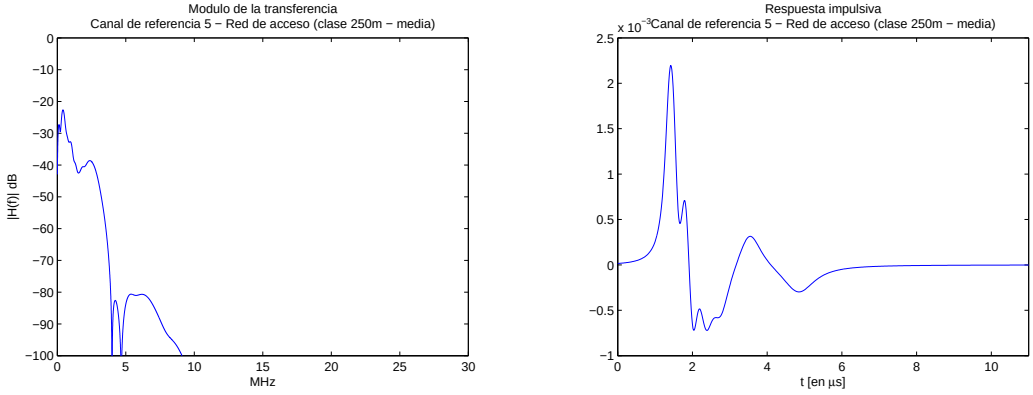
FIGURA A.4: Canal de referencia 4 - clase 250m - calidad buena

A.1.2.2. Canal de referencia 5 - Calidad Media

Este segundo canal de esta clase tiene una característica pasabajos todavía más marcada que a su vez se ve superpuesta con *notches* en frecuencia en el rango de los 4 MHz. Sin embargo, la respuesta al impulso es más corta que en el anterior canal y por ello sólo 12 caminos son suficientes para su modelado.

Parámetros de atenuación		
$a_0 = 0$	$a_1 = 4{,}5 \cdot 10^{-9}[s/m]$	$k = 1$
Parámetros de camino		
i	$\mathbf{g}_i$	$\mathbf{d}_i$ $[m]$
1	0,0302	211,5
2	0,0445	228
3	−0,0178	243
4	−0,0213	254
5	0,0587	278
6	−0,0658	306
7	0,032	330
8	−0,0356	360
9	0,0089	390
10	−0,0267	420
11	0,0267	540
12	−0,0267	740

Tabla A.5: Parámetros del canal de referencia (clase 250m, media)



(a) Módulo de la transferencia (b) Respuesta al impulso

FIGURA A.5: Canal de referencia 5 - clase 250m - calidad media

A.1.3. Clase de 350m

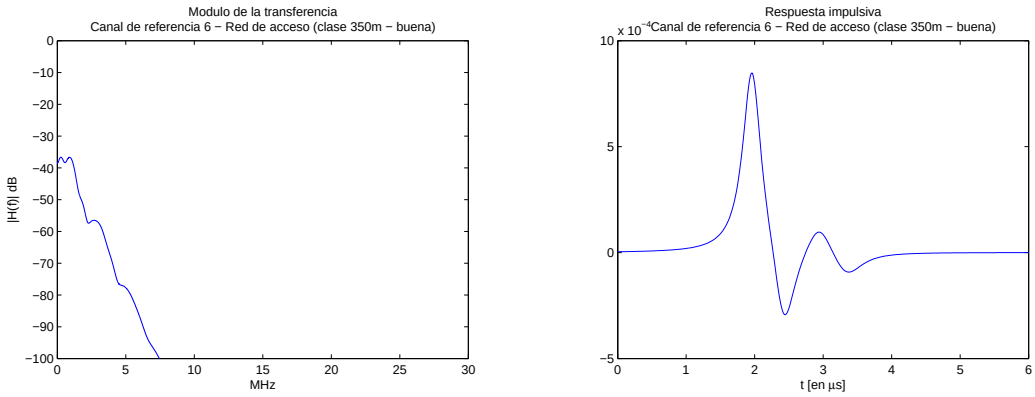
Esta clase de canales tiene una mayor atenuación con la característica que en las bajas frecuencias ya posee una atenuación superior a 40 dB.

A.1.3.1. Canal de referencia 6 - Calidad Buena

Este canal representa un canal con pocas ramificaciones y por lo tanto no presenta *notches* en frecuencia. La respuesta al impulso decae luego de poco tiempo por ello 5 caminos son suficientes para su modelado. Se observa que los pulsos que la componen son más anchos y de allí su efecto pasabajos más pronunciado.

Parámetros de atenuación		
$a_0 = 8 \cdot 10^{-3}[1/m]$	$a_1 = 3{,}5 \cdot 10^{-9}[s/m]$	$k = 1$
Parámetros de camino		
i	$\mathbf{g}_i$	$\mathbf{d}_i$ $[m]$
1	0,26	300
2	0,05	350
3	−0,3	370
4	0,25	450
5	−0,35	510

Tabla A.6: Parámetros del canal de referencia (clase 350m, buena)



(a) Módulo de la transferencia (b) Respuesta al impulso

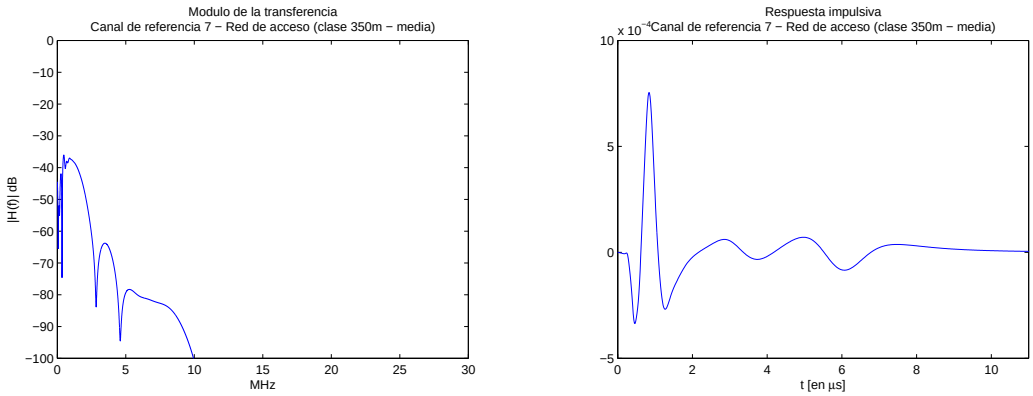
FIGURA A.6: Canal de referencia 6 - clase 350m - calidad buena

A.1.3.2. Canal de referencia 7 - Calidad Media

Este canal muestra algunos *notches* en frecuencia. En particular se observan *notches* a frecuencias irregulares con una acumulación en las bajas frecuencias y por ello se necesitan 13 caminos.

Parámetros de atenuación		
$a_0 = 0$	$a_1 = 9 \cdot 10^{-9}[s/m]$	$k = 1$
Parámetros de camino		
i	$\mathbf{g}_i$	$\mathbf{d}_i$ $[m]$
1	0,00039	40
2	−0,0045	68
3	−0,0062	86
4	0,0281	129
5	−0,0169	185
6	−0,0028	237
7	−0,0056	235
8	0,0051	230
9	0,0112	450
10	−0,0141	560
11	0,1125	830
12	−0,1687	895
13	0,0675	1000

Tabla A.7: Parámetros del canal de referencia (clase 350m, media)



(a) Módulo de la transferencia (b) Respuesta al impulso

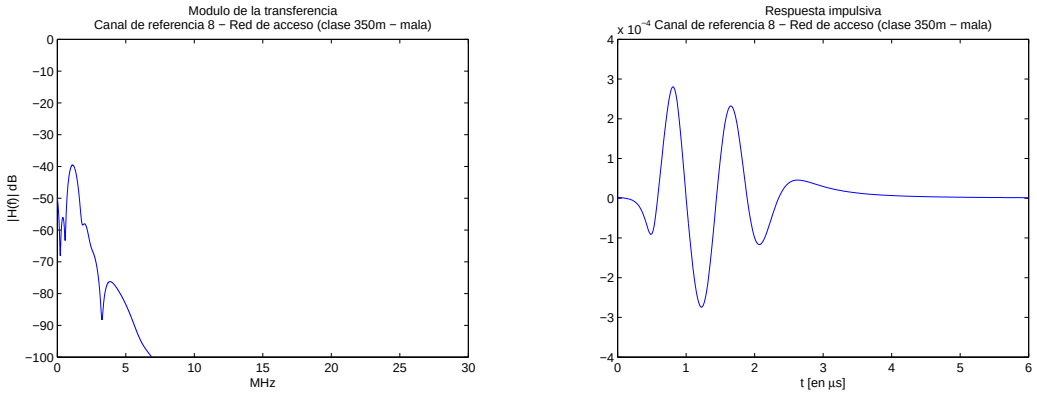
FIGURA A.7: Canal de referencia 7 - clase 350m - calidad media

A.1.3.3. Canal de referencia 8 - Calidad Mala

Este último canal de referencia de esta clase manifiesta el comportamiento pasa-bajos más intenso. Se observa que posee más de 100 dB de atenuación a los 7 MHz. Su respuesta al impulso tiene una forma simple y por ello se utilizan 6 caminos.

Parámetros de atenuación		
$a_0 = 0$	$a_1 = 1{,}1 \cdot 10^{-8}[s/m]$	$k = 1$
Parámetros de camino		
i	$\mathbf{g}_i$	$\mathbf{d}_i$ $[m]$
1	− 0,0039	78
2	0,0156	126
3	−0,034	191
4	0,0715	256
5	−0,122	306
6	0,076	330

Tabla A.8: Parámetros del canal de referencia (clase 350m, mala)



(a) Módulo de la transferencia (b) Respuesta al impulso

FIGURA A.8: Canal de referencia 8 - clase 350m - calidad mala

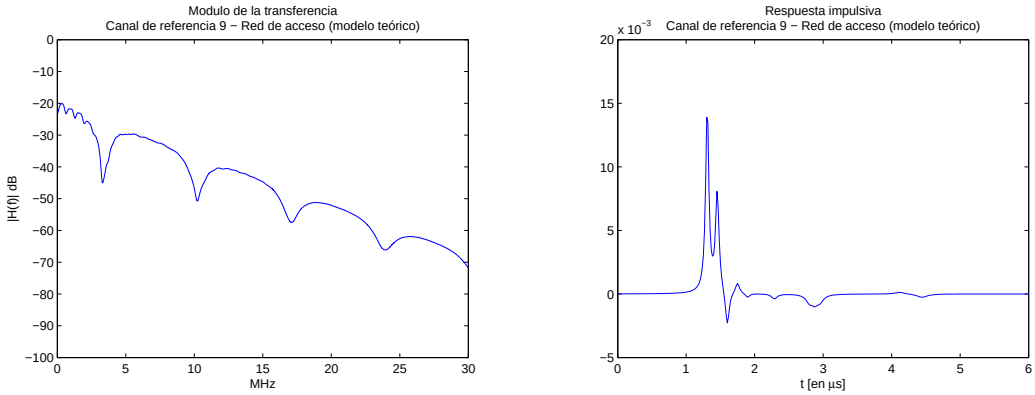
A.1.4. Canal de referencia 9 - Modelo Teórico

Este último canal presenta *notches* selectivos en frecuencia a intervalos regulares. Este comportamiento corresponde a un canal con una sola ramificación. Es decir que con 2 ó 3 caminos tendríamos una buena aproximación general. Los caminos adicionales amplifican los *notches* y causan un leve *ripple* en bajas frecuencias.

Parámetros de atenuación		
$a_0 = 0$	$a_1 = 8{,}8 \cdot 10^{-10}[s/m]$	$k = 1$

Parámetros de camino		
i	$\mathbf{g}_i$	$\mathbf{d}_i$ $[m]$
1	0,064	200
2	0,038	222,4
3	−0,015	244,8
4	0,005	267,5
5	−0,002	290
6	−0,003	350
7	−0,005	428
8	−0,006	440
9	−0,005	452
10	0,002	630
11	−0,004	680

Tabla A.9: Parámetros del canal de referencia (clase 350m, mala)



(a) Módulo de la transferencia (b) Respuesta al impulso

FIGURA A.9: Canal de referencia 9 - modelo teórico

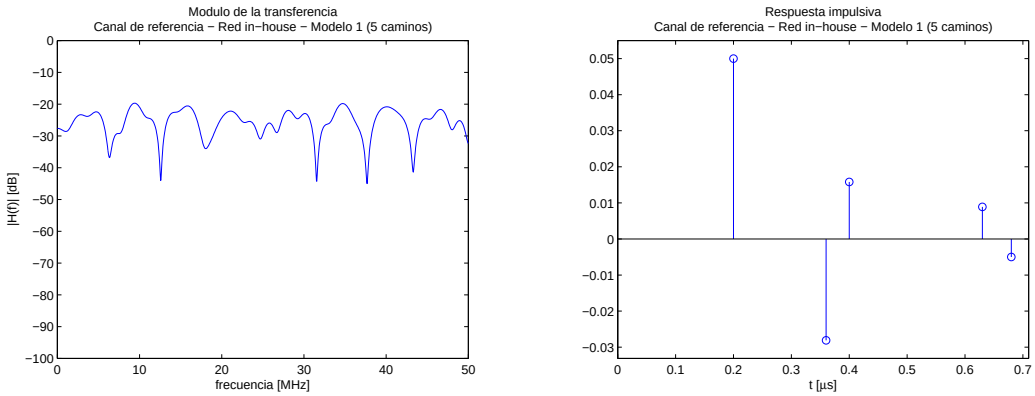
A.2. Redes in-house

En esta sección presentaremos los 4 canales de referencia para redes in-house Los parámetros característicos de estos canales se corresponden con los prefijados por la tabla 2.3.

A.2.1. Modelo 1 (5 caminos)

i	$\mathbf{h}(t_i)$	$\mathbf{t}_i$ $[\mu s]$
1	0,05	0,20
2	−0,0281	0,36
3	0,0158	0,40
4	0,0089	0,63
5	−0,005	0,68

Tabla A.10: Parámetros del canal de referencia in-house (modelo 1)



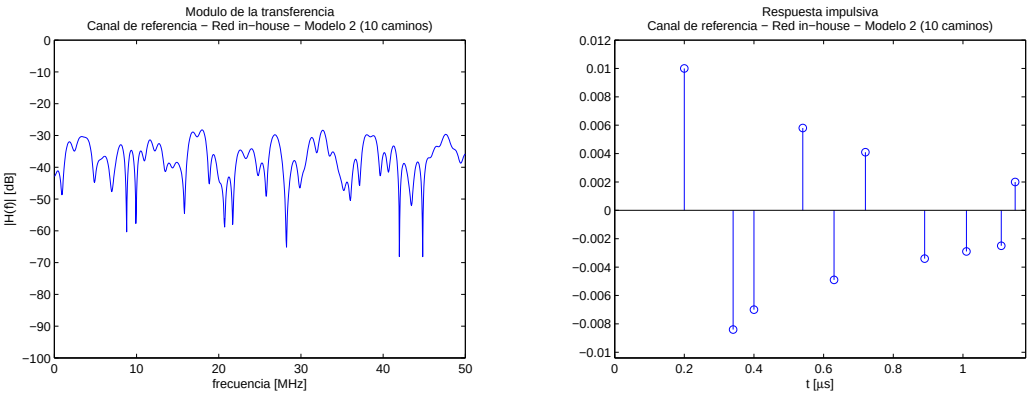
(a) Módulo de la transferencia (b) Respuesta al impulso

FIGURA A.10: Canal de referencia - Red in-house - Modelo 1

A.2.2. Modelo 2 (10 caminos)

i	$\mathbf{h}(t_i)$	$\mathbf{t}_i$ $[\mu s]$
1	0,01	0,20
2	−0,0084	0,34
3	−0,007	0,40
4	0,0058	0,54
5	−0,0049	0,63
6	0,0041	0,72
7	−0,0034	0,89
8	−0,0029	1,01
9	−0,0025	1,11
10	0,0020	1,15

Tabla A.11: Parámetros del canal de referencia in-house (modelo 2)



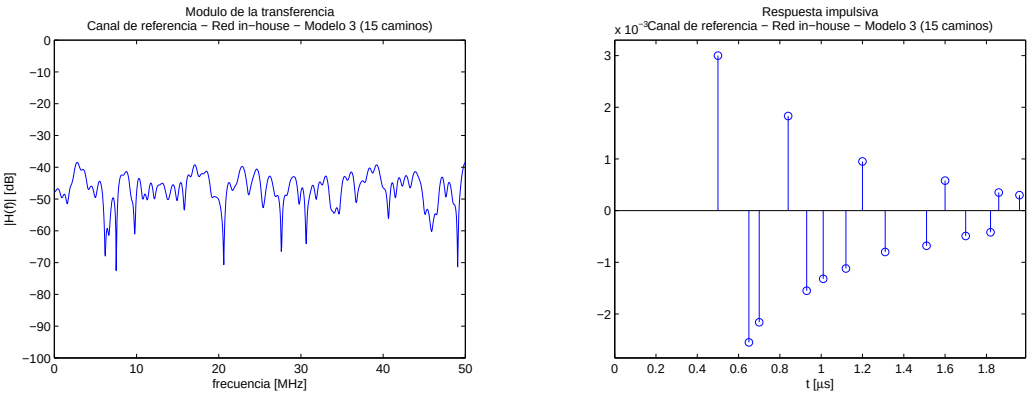
(a) Módulo de la transferencia (b) Respuesta al impulso

FIGURA A.11: Canal de referencia - Red in-house - Modelo 2

A.2.3. Modelo 3 (15 caminos)

i	$\mathbf{h}(t_i)$	$\mathbf{t}_i$ $[\mu s]$
1	$0{,}3 \cdot 10^{-2}$	0,50
2	$-0{,}255 \cdot 10^{-2}$	0,65
3	$-0{,}216 \cdot 10^{-2}$	0,70
4	$0{,}183 \cdot 10^{-2}$	0,84
5	$-0{,}155 \cdot 10^{-2}$	0,93
6	$-0{,}132 \cdot 10^{-2}$	1,01
7	$-0{,}112 \cdot 10^{-2}$	1,12
8	$0{,}095 \cdot 10^{-2}$	1,20
9	$-0{,}08 \cdot 10^{-2}$	1,31
10	$-0{,}068 \cdot 10^{-2}$	1,51
11	$0{,}058 \cdot 10^{-2}$	1,60
12	$-0{,}049 \cdot 10^{-2}$	1,70
13	$-0{,}042 \cdot 10^{-2}$	1,82
14	$0{,}035 \cdot 10^{-2}$	1,86
15	$0{,}03 \cdot 10^{-2}$	1,96

Tabla A.12: Parámetros del canal de referencia in-house (modelo 3)



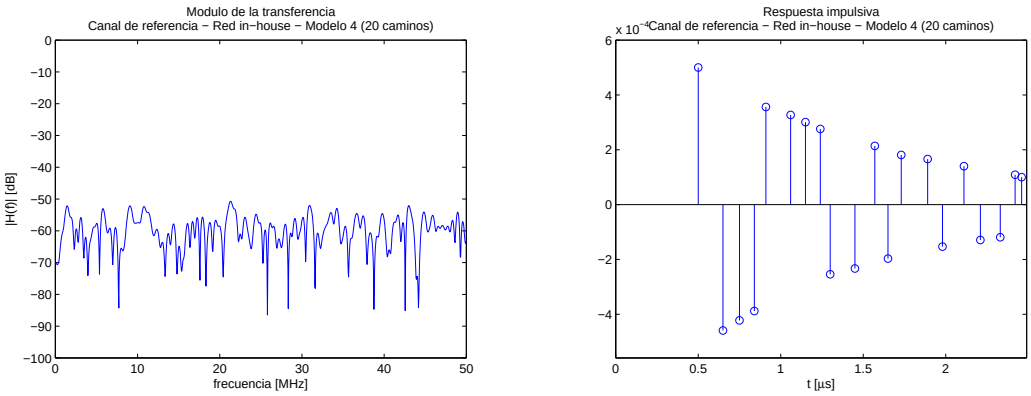
(a) Módulo de la transferencia (b) Respuesta al impulso

FIGURA A.12: Canal de referencia - Red in-house - Modelo 3

A.2.4. Modelo 4 (20 caminos)

i	$\mathbf{h}(t_i)$	$\mathbf{t}_i$ $[\mu s]$
1	$0{,}5 \cdot 10^{-3}$	0,50
2	$-0{,}459 \cdot 10^{-3}$	0,65
3	$-0{,}422 \cdot 10^{-3}$	0,75
4	$-0{,}388 \cdot 10^{-3}$	0,84
5	$0{,}356 \cdot 10^{-3}$	0,91
6	$0{,}327 \cdot 10^{-3}$	1,06
7	$0{,}301 \cdot 10^{-3}$	1,15
8	$0{,}276 \cdot 10^{-3}$	1,24
9	$-0{,}254 \cdot 10^{-3}$	1,30
10	$-0{,}233 \cdot 10^{-3}$	1,45
11	$0{,}214 \cdot 10^{-3}$	1,57
12	$-0{,}197 \cdot 10^{-3}$	1,65
13	$0{,}181 \cdot 10^{-3}$	1,73
14	$0{,}166 \cdot 10^{-3}$	1,89
15	$-0{,}153 \cdot 10^{-3}$	1,98
16	$0{,}14 \cdot 10^{-3}$	2,11
17	$-0{,}129 \cdot 10^{-3}$	2,21
18	$-0{,}119 \cdot 10^{-3}$	2,33
19	$0{,}109 \cdot 10^{-3}$	2,42
20	$0{,}1 \cdot 10^{-3}$	2,46

Tabla A.13: Parámetros del canal de referencia in-house (modelo 4)



(a) Módulo de la transferencia (b) Respuesta al impulso

FIGURA A.13: Canal de referencia - Red in-house - Modelo 4

Apéndice B

Modelo eléctrico

B.1. Approximación de γ para bajas péridas

$$\gamma = \sqrt{(R' + j\omega L')\,(G' + j\omega C')} \tag{B.1}$$

$$\gamma = \sqrt{j\omega L' \left(\frac{R'}{j\omega L'} + 1\right) j\omega C' \left(\frac{G'}{j\omega C'} + 1\right)} \tag{B.2}$$

$$\gamma = j\omega\sqrt{L'C'}\sqrt{\left(1 - j\frac{R'}{\omega L'}\right)}\sqrt{\left(1 - j\frac{G'}{\omega C'}\right)} \tag{B.3}$$

pero si $R' \ll \omega L'$ y $G' \ll \omega C'$

y utilizamos la siguiente aproximación $\sqrt{1-x} \approx 1 - \frac{x}{2}$

$$\gamma \approx j\omega\sqrt{L'C'}\left(1 - j\frac{R'}{2\omega L'}\right)\left(1 - j\frac{G'}{2\omega C'}\right) \tag{B.4}$$

$$\gamma \approx j\omega\sqrt{L'C'}\left(1 - j\frac{R'}{2\omega L'} - j\frac{G'}{2\omega C'} \quad \underbrace{-\frac{R'G'}{4\omega^2 L'C'}}_{\text{infinitésimo de orden 2}}\right) \tag{B.5}$$

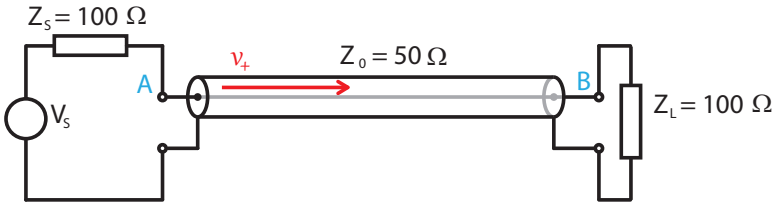
$$\gamma \approx j\omega\sqrt{L'C'}\left(1 - j\frac{R'}{2\omega L'} - j\frac{G'}{2\omega C'}\right) \tag{B.6}$$

y como aproximamos $Z_0 \approx \sqrt{\frac{L'}{C'}}$

$$\gamma \approx \frac{1}{2}\frac{R'}{Z_0} + \frac{1}{2}G'Z_0 + j\omega\sqrt{L'C'} \tag{B.7}$$

B.2. Ejemplo 1 de cálculo de caminos

Dada la siguiente configuración



tenemos $r_B = 1/3 \quad t_B = 4/3 \quad r_A = 1/3$
A continuación vemos la tabla de caminos

N° camino	Camino de la señal	peso g_i	valor g_i	Longitud del camino d_i
1	$A \to B$	t_B	$4/3$	l
2	$A \to B \to A \to B$	$t_B r_B^2$	$4/3 \cdot 1/3^2$	$3l$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
N	$A \to (B \to A)^{N-1} \to B$	$t_B r_B^{2(n-1)}$	$4/3 \cdot 1/3^{2(n-1)}$	$2(n-1)l$

Si queremos calcular la tensión en estado estacionario en el nodo B deberemos sumar la contribución de cada camino

$$V_B = \frac{4}{3}\left(1 + \frac{1}{3^2} + \frac{1}{3^4} + \ldots + \frac{1}{3^{n-1}} + \ldots\right) v_+ \tag{B.8}$$

Usando $\sum_{n=0}^{\infty} a^{2n} = \frac{1}{1-a^2} \qquad |a| < 1$ tenemos

$$V_B = \frac{4}{3} \cdot \frac{9}{8} = 1,5 v_+ \tag{B.9}$$

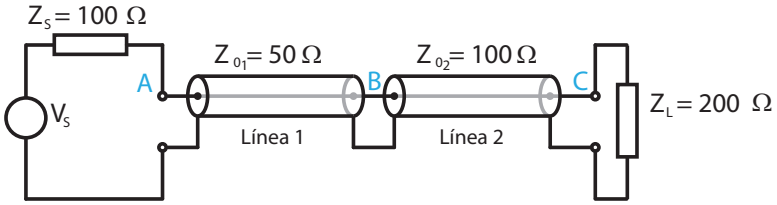
Sin embargo, si calculamos cual debió ser el valor de la fuente de tensión para que haya v_+ en la línea de transmisión

$$V_S \frac{50}{100 + 50} = v_+ \Rightarrow V_S = 3v_+ \tag{B.10}$$

obtenemos que $V_S = 3v_+$ por lo que no hubo ganancia de tensión alguna.

B.3. Ejemplo 2 de cálculo de caminos

Dada la siguiente configuración



$$\begin{array}{ll} & r_{1A} = 1/3 \\ \text{Entonces} & r_{1B} = 1/3 \quad t_{1B} = 4/3 \quad t_{2B} = 2/3 \\ & r_{2C} = 1/3 \quad t_{2C} = 4/3 \end{array}$$

A continuación vemos la tabla de caminos

N° camino	Camino de la señal	peso g_i	Longitud del camino d_i
1	$A \to B \to C$	$t_{1B}t_{2C}$	$l_1 + l_2$
2	$A \to B \to A \to B \to C$	$r_{1B}r_{1A}t_{1B}t_{2C}$	$3l_1 + l_2$
3	$A \to B \to C \to B \to C$	$t_{1B}r_{2C}r_{2B}t_{2C}$	$l_1 + 3l_2$
4	$A \to B \to C \to B \to A \to B \to C$	$t_{1B}r_{2C}\underbrace{t_{2B}r_{1A}t_{1B}}_{\frac{8}{9}r_{1A}}t_{2C}$	$3l_1 + 3_2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Vemos que pasar de la línea 2 a la 1 y luego de la 1 a la 2 tiene un costo para la amplitud de la señal, sin haber tenido en cuenta el impacto de la reflexión que tuvo que haber para que la propagación de la señal cambie de sentido

$$t_{2B} \cdot t_{1B} = 2/3 \cdot 4/3 = 8/9 < 1$$

Entonces concluimos con que a mayor distancia de camino, más chico tenderá a ser el valor de g_i sin importar que algunos coeficientes de transmisión sean mayores que 1. Es por ello que para el modelo alcanza con que tomemos los caminos de mayor importancia y obviemos aquellos cuyo g_i ya es demasiado chico como para aportar significativamente pero que siempre aumentan la complejidad del cálculo.

Apéndice C

Fórmulas

C.1. Similitud de métodos de conversión de probabilidades a otros tiempo de muestreo

Hay dos posibles métodos para convertir las probabilidades de transición correspondientes a un tiempo de muestreo t_s a otro tiempo de muestreo t'_s.

- $p'_{i,i} = e^{-\lambda_i t'_s} = p_{i,i}^{\frac{t'_s}{t_s}}$
- $p'_{i,i} = 1 - \frac{t'_s}{t_s}(1 - p_{i,i})$

El resto de las probabilidades se ajustan para que cierren a 1 con la fórmula

$$p'_{i,j} = p_{i,j}\frac{1 - p'_{i,i}}{1 - p_{i,i}} \quad i \neq j$$

El primer método conserva la mediana $t_{50\,\%,i}$, ecuación (3.43). El segundo método conserva $\bar{t_i}$ y la distribución de probabilidad estacionaria.

Si bien son dos métodos distintos, se comportan muy parecido para los valores con solemos trabajar y ésto lo probaremos a continuación.

En nuestro contexto las matrices de probabilidades de transición tiene valores $p_{i,i}$ cercanos a uno. Por lo tanto definimos

$$p_{i,i} \triangleq 1 - \Delta_{i,i} \qquad \Delta_{i,i} \approx 0 \tag{C.1}$$

Entonces desarrollaremos la serie de Taylor del primer método entorno a $\Delta_{i,i} = 0$. Primero expresamos $p'_{i,i}$ como una función de $\Delta_{i,i}$.

$$p'_{i,i} = p_{i,i}^{\frac{t'_s}{t_s}} = (1 - \Delta_{i,i})^{\frac{t'_s}{t_s}} \quad \Rightarrow \quad p'_{i,i}(\Delta_{i,i}) = (1 - \Delta_{i,i})^{\frac{t'_s}{t_s}} \tag{C.2}$$

Desarrollamos la serie de Taylor hasta el primer orden y tenemos

$$p'_{i,i}(\Delta_{i,i}) \approx p'_{i,i}(\Delta_{i,i})\big|_{\Delta_{i,i}=0} + \frac{1}{1!}\frac{d}{d\Delta_{i,i}}p'_{i,i}(\Delta_{i,i})\bigg|_{\Delta_{i,i}=0} \cdot (\Delta_{i,i} - 0) \tag{C.3}$$

si evaluamos en $\Delta_{i,i} = 0$ llegamos a

$$p'_{i,i}(\Delta_{i,i}) \approx 1 + \frac{t'_s}{t_s} \cdot (\Delta_{i,i} - 0) \tag{C.4}$$

pero usando $p_{i,i} \triangleq 1 - \Delta_{i,i}$ tenemos

$$\boxed{p'_{i,i}(\Delta_{i,i}) \approx 1 + \frac{t'_s}{t_s} \cdot (1 - p_{i,i})} \tag{C.5}$$

Es decir que llegamos a la misma expresión que para el segundo método. La aproximación es mejor si $\Delta_{i,i}$ se aproxima a 0, que es nuestro caso.

Apéndice D

Procesos Aleatorios

D.1. Proceso Autoregresivo de primer orden

La siguiente ecuación describe el modelo del proceso autoregresivo de primer orden.

$$x[n] = \varphi \cdot x[n-1] + e[n] \tag{D.1}$$

donde $e[n]$ es ruido blanco Gaussiano de media nula y varianza σ_N^2.

Si deseamos que el proceso sea estacionario, podemos calcular la media como

$$\mu_X = E\{x[n]\} = E\{\varphi \cdot x[n-1] + e[n]\} = \varphi \cdot \mu_X + 0$$

Por lo tanto, si $\varphi \neq 1$ entonces

$$\mu_X = 0 \tag{D.2}$$

La varianza de $x[n]$ la podemos calcular de la siguiente manera

$$\begin{aligned} \sigma_X^2 &= E\{x[n]^2\} \\ &= E\{\varphi^2 \cdot x[n-1]^2 + e[n]^2 + 2\varphi \cdot x[n-1]e[n]\} \end{aligned}$$

como $x[n-1]$ es una combinación lineal de $e[n-1], e[n-2], \ldots$ entonces $e[n]$ y $x[n-1]$ son independientes. Entonces

$$\sigma_X^2 = \varphi^2 \sigma_X^2 + \sigma_N^2 \quad \Rightarrow \quad \sigma_X^2 = \frac{\sigma_N^2}{1-\varphi^2} \tag{D.3}$$

con la salvedad de que para que la varianza sea finita se debe cumplir la condición $-1 < \varphi < 1$.

Densidad espectral de potencia La transferencia del proceso la podemos hallar aplicando la ecuación

$$S_{XX}(f) = |H(f)|^2 S_{ee}(f) \tag{D.4}$$

donde $S_{ee}(f)$ es la densidad espectral de potencial del ruido blanco Gaussiano y es

$$S_{ee}(f) = \sigma_N^2 \quad 0 \leqslant f < 1 \tag{D.5}$$

y $H(f)$ es la transferencia y se puede describir por

$$H(f) = \frac{1}{1 - \varphi \cdot e^{-j2\pi f}} \quad 0 \leqslant f < 1 \tag{D.6}$$

y por lo tanto

$$|H(f)|^2 = \frac{1}{1 - 2\varphi \cdot \cos(2\pi f) + \varphi^2} \quad 0 \leqslant f < 1 \tag{D.7}$$

Entonces

$$S_{XX}(f) = \frac{\sigma_N^2}{1 - 2\varphi \cdot \cos(2\pi f) + \varphi^2} \quad 0 \leqslant f < 1 \tag{D.8}$$

D.2. Breve referencia sobre Cadenas de Markov

La cadena de Markov describe aquellos procesos aleatorios discretos cuyo valor futuro sólo depende de su valor presente o de una cantidad limitada de valores pasados. El comportamiento del proceso es descripto por estados. El sistema está compuesto por n estados z_i $(i = 1, \ldots, n)$. Las transiciones ocurren sólamente a momentos discretos en el tiempo $0, 1, \ldots, k$ que son múltiplos de intervarlo de muestreo t_s, si es que se discretizó el tiempo continuo. Además, la función de salida $\Phi(k)$ en el instante k sólo depende del estado actual.

$$\Phi(k) = \Phi(z(k) = z_i) \tag{D.9}$$

Para representarlos ilustrativamente se suele utilizar un grafo donde los nodos representan los estados y las flechas con sus pesos expresan la probabilidad de transición $p_{i,j}$ del estado i al estado j $(i, j = 1, \ldots, n)$.

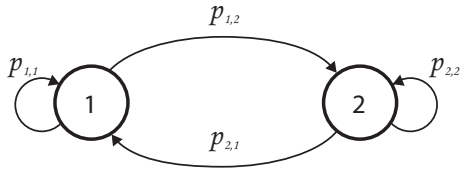


FIGURA D.1: Cadena de Markov de 2 estados

Todas las propiedades estadísticas de la cadena de Markov quedan descriptas por la matriz de probabilidades de transición $\mathbf{P}$.

$$\mathbf{P} = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & p_{n-1,n} \\ p_{n,1} & \cdots & p_{n,n-1} & p_{n,n} \end{bmatrix} \tag{D.10}$$

donde por ser probabilidades

$$p_{i,j} \leqslant 1 \qquad i,j = 1,2,\ldots,n \tag{D.11}$$

$$\sum_{j=1}^{n} p_{i,j} = 1 \qquad i = 1,2,\ldots,n \tag{D.12}$$

D.3. Deducción de la tasa de transición

Tenemos una cadena de Markov particionada [18] en dos grupos A y B. El grupo A tiene ν estados y el grupo B w estados. En el modelo de Fritchman sólo se puede transicionar de un estado a otro estado del grupo opuesto o permanecer en el mismo estado inicial. Suponiendo que estoy en el estado $i \in A$ analizaremos la probabilidad de transicionar a otro estado del otro grupo, la cual se expresa como

$$P\left(\text{transicionar de } i \in A \text{ a otro grupo}\right) = P\left(i \to B\right) \tag{D.13}$$

El hecho de pasar de i a cualquier estado de B puede expandirse a pasar de i a cualquiera de los estados de B, lo cual se expresa matemáticamente como una unión de sucesos.

$$P\left(i \to B\right) = P\left(\bigcup_{j \in B} i \to j\right) \tag{D.14}$$

como dichos sucesos que componen la unión son disjuntos entonces las probabilidades de los sucesos de la unión se pueden sumar aritméticamente

$$P\left(i \to B\right) = \sum_{j \in B} P\left(i \to j\right) = \sum_{j \in B} p_{i,j}\mathbf{p}_i \tag{D.15}$$

debido a la característica particionada de nuestra cadena

$$\sum_{j \in B} p_{i,j}\mathbf{p}_i = (1 - p_{i,i})\mathbf{p}_i \tag{D.16}$$

Ahora que tenemos la probabildad de que haya una transición del estado i a cualquier estado del grupo B nos interesa conocer la cantidad de iteraciones que esperaremos

ver en promedio para que dicha transición ocurra. Para ello utilizamos como herramienta a la variable aleatoria Bernoulli(que llamaremos Z), donde definimos como éxito del experimento a la transición $i \to B$ y como fracaso a la NO transición. Por lo tanto

$$p_{Bernoulli} = P\left(i \to B\right) \tag{D.17}$$

y

$$E[Z] = \frac{1}{p_{Bernoulli}} \tag{D.18}$$

A su vez, como cada iteración dura t_s segundos, podemos traducir dicho resultado a un valor temporal.

$$\bar{t}_{i\to B} = \frac{t_s}{p_{Bernoulli}} \tag{D.19}$$

$\bar{t}_{i\to B}$ es el tiempo promedio entre transiciones del estado $i \in A$ a un estado cualquiera en B. La tasa a la que dichas transiciones ocurren en $\frac{1}{\bar{t}_{i\to B}}$. Si interpretamos que A es el grupo de estados no perturbados y B el grupo de estados perturbados, la transición considerada, $i \to B$, es el comienzo de un impulso. Por lo tanto la tasa de impulsos debido a esta transición $i \to B$ es

$$\mathsf{R}_{imp_i} = \frac{1}{t_{i\to B}} = \frac{p_{Bernoulli}}{t_s} = \frac{P\left(i \to B\right)}{t_s} \tag{D.20}$$

usando las ecuaciones (D.15), (D.16) para desarrollar $P\left(i \to B\right)$ y (3.40) tenemos

$$\mathsf{R}_{imp_i} = \frac{(1 - p_{i,i})\mathbf{p}_i}{t_s} = \frac{\mathbf{p}_i}{\frac{t_s}{(1-p_{i,i})}} = \frac{\mathbf{p}_i}{\bar{t}_i} \tag{D.21}$$

Entonces, para obtener la tasa de impulsos promedio del sistema es

$$\boxed{\mathsf{R}_{imp} = \sum_{i\in A} \mathsf{R}_{imp_i} = \sum_{i\in A} \frac{\mathbf{p}_i}{\bar{t}_i}} \tag{D.22}$$

Vale la pena recalcar que nosotros obtuvimos la tasa de comienzos de impulsos. Existe la otra alternativa de planteo que considera los fines de impulsos. Si bien se llegará a otra expresión en la cual la sumatoria será sobre los elementos del otro grupo, se puede probar que la tasa será la misma.

D.4. Cálculo de probabilidades estacionarias

Dada la cadena de Markov particionada descrita por las ecuaciones (3.28), (3.29), (3.30), (3.31), (3.32), (3.33) y (3.34), que tiene la siguiente matríz de transición

$$\mathbf{P} = \left[\begin{array}{ccccccc} u_{1,1} & & 0 & & u_{1,\nu+1}\cdot g_{w+1,1} & \cdots & u_{1,\nu+1}\cdot g_{w+1,w} \\ & \ddots & & & \vdots & & \vdots \\ 0 & & u_{\nu,\nu} & & u_{\nu,\nu+1}\cdot g_{w+1,1} & \cdots & u_{\nu,\nu+1}\cdot g_{w+1,w} \\ \hdashline g_{1,w+1}\cdot u_{\nu+1,1} & \cdots & g_{1,w+1}\cdot u_{\nu+1,\nu} & & g_{1,1} & & 0 \\ \vdots & & \vdots & & & \ddots & \\ g_{w,w+1}\cdot u_{\nu+1,1} & \cdots & g_{w,w+1}\cdot u_{\nu+1,\nu} & & 0 & & g_{w,w} \end{array}\right]$$

se pueden calcular las probabilidades estacionarias ($\mathbf{p} = \mathbf{p}\cdot\mathbf{P}$ s. t. $\sum_j \mathbf{p}_j = 1$)

$$\mathbf{p} = [\mathbf{p}_1 \cdots \mathbf{p}_\nu \ \mathbf{p}_{\nu+1} \cdots \mathbf{p}_n]$$

utilizando las siguiente variables auxiliares

$$K_{U,i} = \frac{u_{\nu+1,i}}{1-u_{i,i}}, \quad i = 1, \cdots, \nu, \tag{D.23}$$

$$K_{G,i} = \frac{g_{w+1,i}}{1-g_{i,i}}, \quad i = 1, \cdots, w \tag{D.24}$$

$$K = \sum_{i=1}^{\nu} K_{U,i} + \sum_{i=1}^{w} K_{G,i} \tag{D.25}$$

de manera que

$$\mathbf{p}_i = \begin{cases} \frac{K_{U,i}}{K} & i = 1, \cdots, \nu \\ \frac{K_{G,i-\nu}}{K} & i = \nu+1, \cdots, n \end{cases} \tag{D.26}$$

recordar que $n = \nu + w$

Apéndice E

Detalles de implementación

E.1. Implementación del proceso AR 1

En los filtros retroalimentados implementados en punto fijo es vital estudiar la posibilidad de overflow en el loop además de su estabilidad numérica y fidelidad en la representación de la respuesta al impulso.

E.1.1. Estudio del overflow

El peor caso para tener en cuenta el overflow se da con el φ más cercano a uno posible. Dada la representación `Fix_16_15` utilizada este valor es $\varphi = 1 - 2^{-15} = 0{,}999969482421875$. Si suponemos la entrada de un escalón unitario resolviendo la ecuación en diferencias (D.1) para un estado estacionario tenemos que

$$\lim_{n\to\infty} y[n] = \left.\frac{1}{1-\varphi}\right|_{\varphi=1-2^{-15}} = 2^{15} \tag{E.1}$$

Por lo tanto, necesitaríamos contar con al menos 15 bits en la parte entera si no queremos que ocurra overflow en adición a los bits fraccionarios. Aún peor, si la entrada no la restringimos a amplitud unitaria sino que ahora cuenta con 4 bits de amplitud debemos sumar dicha cantidad de bits a los ya mencionados. Sin embargo, como ingenieros, siempre hay que tener en cuenta el entorno en el que funciona el filtro y apartarnos *adecuadamente* del mundo ideal. En nuestro caso tenemos una entrada que es la simulación de un ruido blanco Gaussiano de varianza unitaria cuyo tipo de dato es `Fix_12_7`. Este tipo de entrada tiene una probabilidad extremadamente baja[1] de tener como salida una secuencia de números del mismo signo y de una

[1] o nula ya que no tenemos un ruido Gaussiano ideal sino que tenemos un pseudoruido generado cuya secuencia es finita y no contempla una salida patológica del tipo considerado.

amplitud significactiva como para causar overflows. Entonces, bajo estas condiciones operativas podemos empezar a analizar cuál sería una cantidad razonables de bits de asignar a la parte entera para que la probabilidad de overflow sea muy baja. Por lo tanto, se realizaron simulaciones cuyos resultados serán expresados a continuación.

Las simulaciones utilizaron un valor de $\varphi = 1 - 2^{-15}$ lo cual implica una desvío estándar teórico del proceso AR 1 $\sigma_{AR1} \approx 128$ según la ecuación (D.3). Desde el punto de vista numérico, estas simulaciones fueron hechas con punto flotante de precisión `double` ya que se quiere hacer un análisis estadístico del proceso AR 1 en sí y no de su versión cuantizada de punto fijo. La simulación final en la cual basamos nuestra conclusión contó de 2^{33} o $8,59 \cdot 10^9$ iteraciones y el valor absoluto máximo fue aproximadamente $712,95$ que equivale a $5,57$ desvíos estándar. La cantidad de iteraciones se eligió de manera de tener más de 1 minuto de funcionamiento simulado de nuestro emulador. Por ejemplo, a 100 Msps representa 86,9 segundos o a 80 Msps equivale a 107,3 segundos.

En conclusión, en una primer instancia podríamos decir que con 6 desvíos estándares estamos cubiertos y en caso de que haya overflow será un evento raro que afectará una señal cuya intención original es la de ser un ruido. Por lo que será un glitch en un ruido y que podría llegar a interpretarse como un ruido de otra naturaleza, por ejemplo impulsivo. Entonces tenemos que $6 \cdot \sigma_{AR1} \approx 768$ y $\log_2(768) \approx 9,58$ y por lo tanto necesitaremos de al menos 10 bits para la parte entera del loop. Eso se traduce en un tipo de datos `Fix_(11+XX)_XX`. Si queremos un rendimiento eficiente de nuestro diseño debemos utilizar un solo multiplicador ya que como éste está en el loop y su delay no puede ser mayor a 1, la cascada de varios multiplicadores para obtener mayor precisión impactaría significativamente en la longitud del camino combinacional. Entonces, debemos tener en cuenta las restricciones de hardware impuestas por nuestro FPGA y esto se traduce en que los multiplicadores son de 25 x 18 y por lo tanto la precisión máxima según este criterio será `Fix_25_14`.

Respuesta al escalón Si utilizamos `Fix_25_14` significa que el máximo valor positivo en el loop será $2^{10} - 1 + (1 - 2^{-14}) \approx 1024$ y entonces según la ecuación (E.1) la respuesta al escalón no tendrá overflow hasta un valor $\varphi = 0{,}999023437441792$. Pero como expusimos anteriormente esto no es de mayor importancia ante una entrada aleatoria blanca y un fin ulterior de ser un ruido.

E.1.2. Estudio de la precisión numérica fraccionaria

Si bien determinamos el tipo de dato a utilizar en el loop para satisfacer un criterio en la parte entera, ahora estudiaremos el efecto que esto tiene en la parte fraccionaria. Para ello compararemos la respuesta al impulso para distintos valores de φ. Además plantearemos una tipo de dato alternativo que sacrifica un bit de la parte entera, `Fix_25_15`. Vale la pena notar que las simulaciones que generaron las figuras que mostraremos a continuación debieron ser generadas con el System Generator ya

que el toolbox de punto fijo de MATLAB no emula idénticamente al sumador y el multiplicador de Xilinx.

En las figuras E.1, E.2, E.3, E.4, E.5 y E.6 se pueden ver las respuestas impulsivas para distintos valores de φ en precisión `double`, y `Fix_25_15`. En general, se observa que el error, diferencia respecto a `double`, es la mitad para `Fix_25_15` comparándola contra `Fix_25_14`. La diferencia se hace más notoria cuanto más chicos son los valores de amplitud de la respuesta impulsiva. Esto ocurre cuando φ tiende a uno. A su vez el error relativo es cada vez más grande viendo que las amplitudes del error se asemejan a las amplitudes de la verdadera respuesta al impulso. A medida que φ aumenta se puede notar que la respuesta impulsiva de precisión de punto fijo pasa a ser una función lineal a tramos, figura E.4. Para valores muy cercanos a 1, se observa que la precisión de punto fijo influye de manera tal que la multiplicación de números muy chicos decae en forma recta y no exponencial.

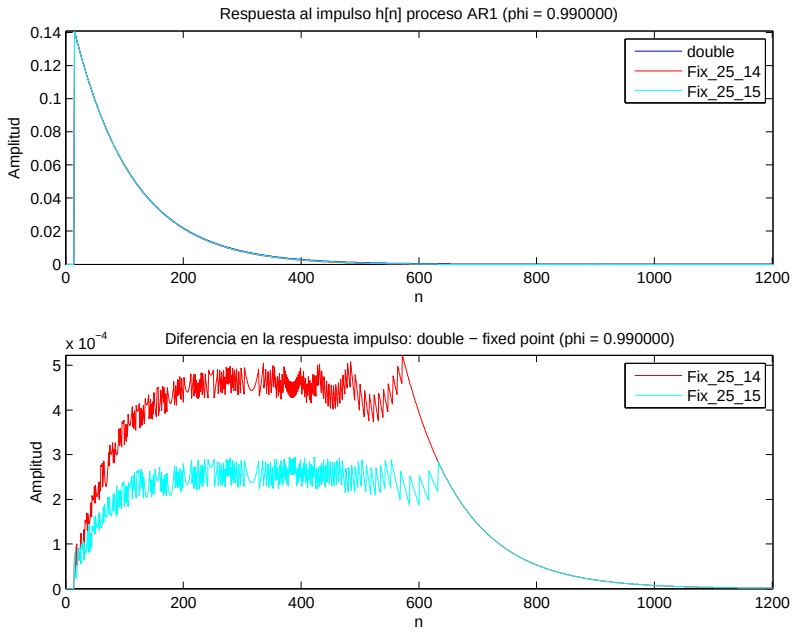


FIGURA E.1: Efectos de precisión numérica en la respuesta al impulso para $\varphi = 0{,}99$

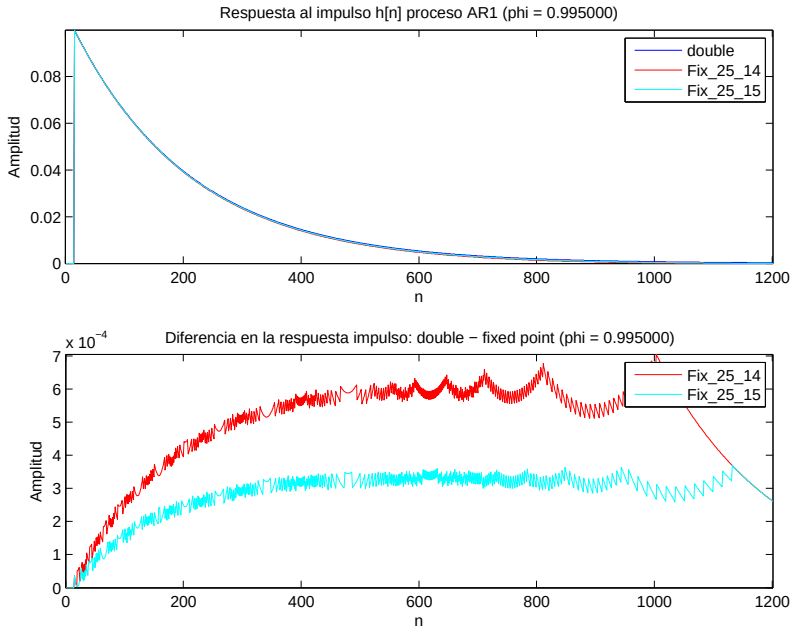


FIGURA E.2: Efectos de precisión numérica en la respuesta al impulso para $\varphi = 0{,}995$

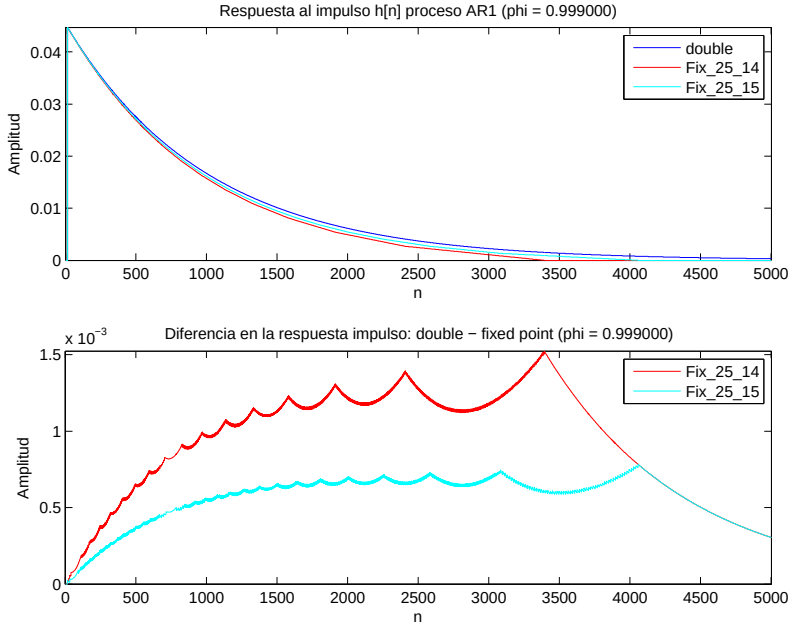


FIGURA E.3: Efectos de precisión numérica en la respuesta al impulso para $\varphi = 0{,}999$

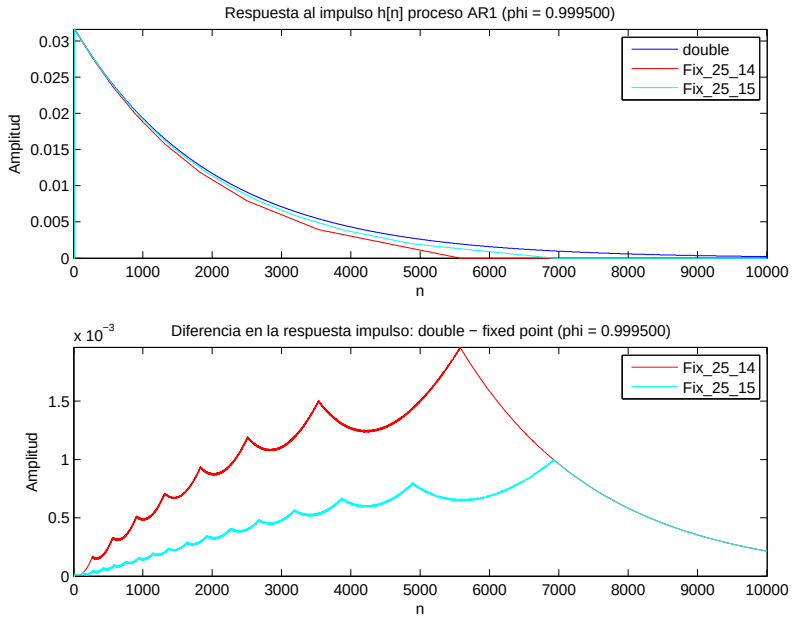


FIGURA E.4: Efectos de precisión numérica en la respuesta al impulso para $\varphi = 0{,}9995$

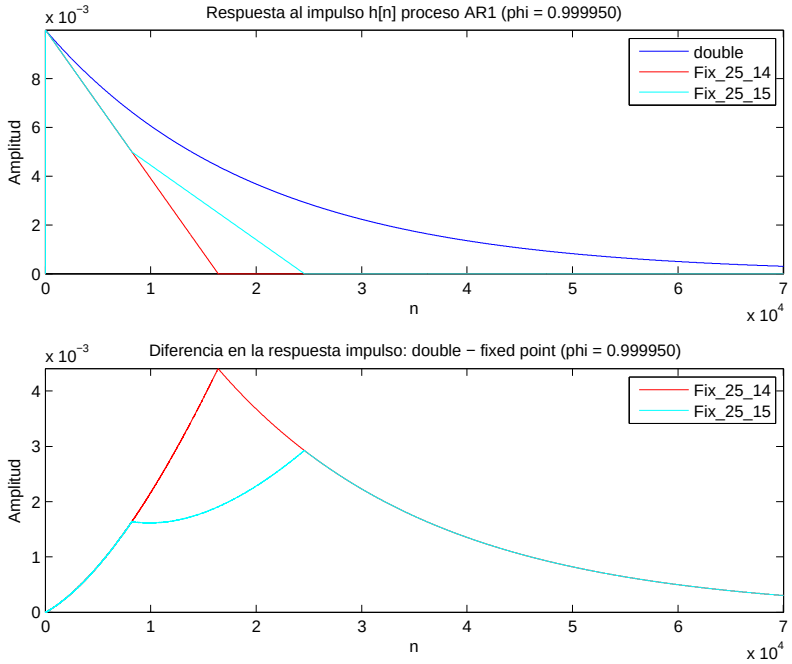


FIGURA E.5: Efectos de precisión numérica en la respuesta al impulso para $\varphi = 0{,}99995$

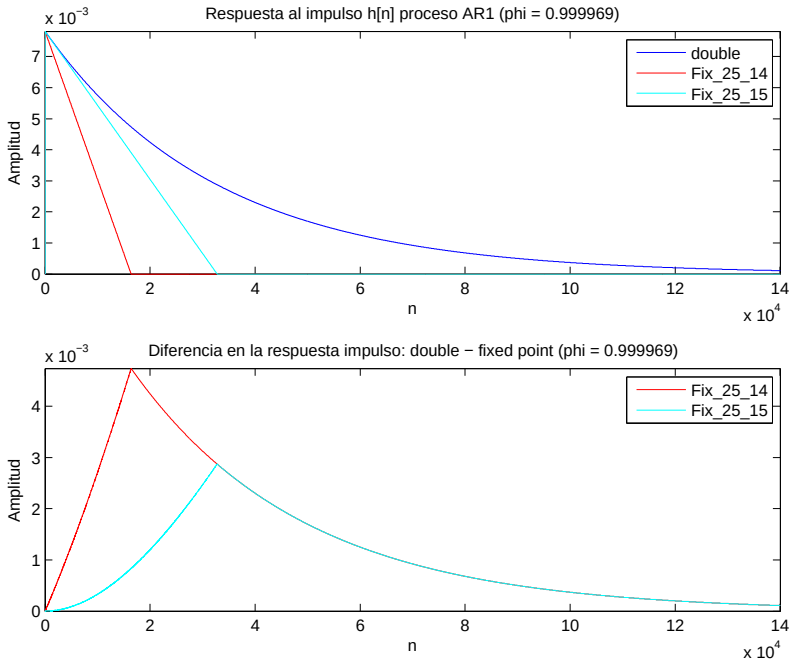


FIGURA E.6: Efectos de precisión numérica en la respuesta al impulso para $\varphi = 1 - 2^{-15}$

Impacto del bit extra en el overflow Decidimos realizar una simulación para ver cual era la tasa de overflow que se podía producir si ahora teníamos 9 bits para la parte entera para $\varphi = 1 - 2^{-15}$. Entonces hicimos ensayos del proceso AR 1 donde se interrumpía la ejecución cuando se encontraba un overflow. Se almacenaba la cantidad de iteraciones que transcurrieron hasta el overflow y repetimos este proceso 400 veces. Luego con ese grupo de 400 valores realizamos las siguientes estadísticas. En la figura E.7 se observa el histograma de los tiempos de ocurrencia y se ve que sigue aproximadamente una distribución exponencial. La media de iteraciones fue $6{,}14 \cdot 10^7$ y la mediana tuvo un valor de $4{,}47 \cdot 10^7$ iteraciones. Por lo tanto, teniendo en cuenta dichos valores podemos desestimar el efecto de los overflows por tener una baja tasa de ocurrencia y podemos contar un bit más cuyo aporte para φ muy cercanos a 1 es considerable. Por último, en la figura E.8 se puede ver el efecto del bit adicional en el error. Si bien la respuesta impulsiva para $\varphi = 1 - 2^{-15}$ es una recta para los casos `Fix_25_14` y `Fix_25_15`, la salida del proceso AR 1 en ambos casos es muy similar a la de precisión `double` y a su vez `Fix_25_15` tiene la mitad del error que `Fix_25_14`.

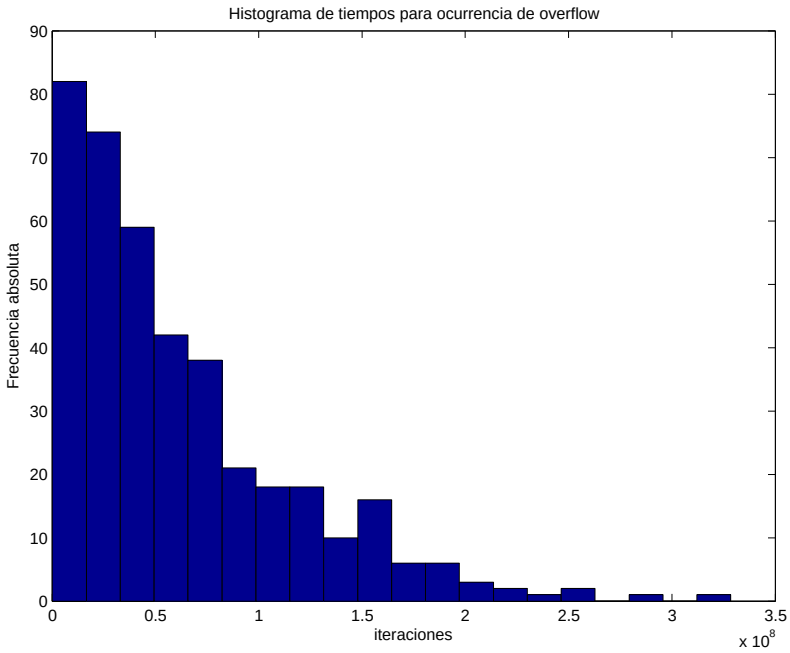


FIGURA E.7: Histograma de tiempos de overflow para $\varphi = 1 - 2^{-15}$

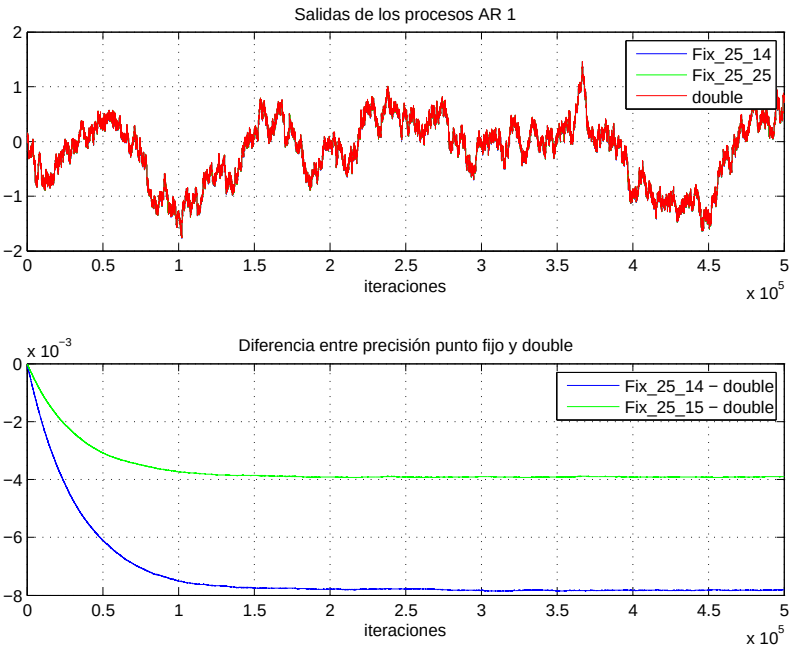


FIGURA E.8: Comparación de distintas precisiones de punto fijo en procesos AR 1 para $\varphi = 1 - 2^{-15}$

E.1.3. Elección del tipo de datos

En conclusión de lo expuesto anteriormente desde el punto de vista del overflow y la precisión fraccionarias hemos decidido elegir como tipo de datos del loop a `Fix_25_15`.

E.2. Parametrización del *core* de la FFT

Cuando decidimos implementar el ruido de banda angosta utilizando la IFFT tuvimos que tomar varias decisiones para configurar adecuadamente el bloque de *Xilinx "Fast Fourier Transform 7.1"*. Los parámetros a definir eran:

- **Longitud**: se determinó en función del ancho de banda de cada portadora. Se deseaba tener una resolución en frecuencia de alrededor de 4,3 kHz trabajando a 80 Msps. Por lo tanto, el valor potencia de 2 más cercano era 16384.
- **Tipo de arquitectura**: existen cuatro tipos de arquitecturas
  - *Pipelined, Streaming I/O*: permite el procesamiento continuo de datos.
  - *Radix-4, Burst I/O*: carga y procesa los datos de manera separada usando un enfoque iterativo. Utiliza menos recursos que la opción *pipelined* pero tiene un mayor tiempo de transformada.
  - *Radix-2, Burst I/O*: es similar a la *Radix-4* pero con una *Butterfly* más chica. Esto significa que ocupa menos recursos pero a su vez tarda más que la *Radix-4*.
  - *Radix-2 Lite, Burst I/O*: basada en la *Radix-2* pero utiliza una variante de multiplexación en tiempo para consumir menos recursos. Como consecuencia es más lenta.

  En nuestro caso la elección fue fácil debido a los requerimientos de velocidad de procesamiento y la conveniencia del procesamiento continuo de datos.
- ***Phase Factor Width***: fue determinado en función de simulaciones. Nosotros antitransformamos una señal real par. Por lo tanto, deberíamos obtener parte imaginaria nula. Sin embargo, aparecían picos de muy baja amplitud por errores numéricos. Entonces aumentamos la cantidad de bits hasta que la ocurrencia de los mismos sea muy baja. El valor en que ocurría ello era a los 21 bits para una entrada de 16 bits. Luego, cuando nos encontramos limitados por la cantidad de recursos que ocupaba el diseño completo redujimos dicho número a 20 bits y observamos que la disminución de los mismo era casi despreciable con el tamaño total del *core* y por lo tantos decidimos mantener 21 bits.
- **Escalaje**: elegimos *Unscaled* porque es la opción recomendada si nuestra entrada es aleatoria y puede tener grandes fluctuaciones de amplitud según la hoja de datos [45].
- **Modo de redondeo**: elegimos *Convergent Rounding* porque nos da menor error cuadrático y es importante cuando trabajamos con amplitudes chicas, por ejemplo para representar ruido de fondo coloreado.

- **Orden de la salida**: elegimos *Natural Order* ya que necesitamos que la señal en el tiempo represente las frecuencias que deseamos.
- **Número de etapas usando Block RAM**: usamos 5 que es lo mínimo para nuestro tamaño de la IFFT. No usamos más porque este es un recurso limitado en nuestro diseño.
- **Multiplicadores Complejos**: elegimos *3-multiplier structure* para poder tener buena velocidad y recursos optimizados.
- ***Butterfly arithmetic***: elegimos usar *CLB Logic* en lugar de *slices XtremeDSP* ya que son un recurso escaso.

Apéndice F

Tutoriales de herramientas Xilinx

F.1. Síntesis de un *soft-processor MicroBlaze*

Una vez que hemos decidido incluir un procesador *MicroBlaze* primero deberemos configurarlo y exportar su diseño para que pueda ser utilizado por otras herramientas como el *System Generator*, el *Xilinx SDK* y el *ISE*. Para realizar la configuración y exportación se utiliza la herramienta de software *Xilinx Platform Studio* que viene en la Embedded Edition.

A continuación describiremos los pasos realizados para poder configurar nuestro *soft-processor*. El nivel de detalle de los pasos descriptos será tal como para que otra persona pueda realizarlo sin tener demasiado conocimiento del tema.

1. Abrimos la herramienta *Xilinx Platform Studio*. Para ello vamos a:
   Start Menu→All Programs→Xilinx ISE Design Suite 13.4→EDK→Xilinx Platform Studio

2. Luego crearemos una nueva plataforma:
   File→New BSB Project...

3. Aparecerá una pantalla como la que se ve en la figura F.1. Ahí deberemos elegir donde guardaremos nuestro proyecto. En nuestro caso hemos creado directorio y el path es *F:\Tesis\XPS_tutorial*. El nombre del archivo por default es *system.xmp*. Luego debemos elegir un tipo de sistema interconexión AXI o PLB. Si bien AXI es más nuevo y mejor, elegiremos PLB debido a que es una tecnología consolidada y probada y además no tenemos requerimientos elevados de performance. Por último, y es muy importante, en nuestro caso deberemos incluir un directorio para que busque bibliotecas adicionales. El path del directorio no es lo importante sino lo que contiene. Como mencionamos anteriormente nosotros utilizamos el kit Digilent XUPV5. Este **no** es soportado out-of-the-box como

los kits oficiales de Xilinx[1] aunque Xilinx ofrezca soporte en la página e indique que es muy similar al kit ML505. Peor aún, las bibliotecas que se encuentran rápidamente[2] no funcionan y no se indica el porqué. Tras investigar se encuentra otra página[3] donde indica que a partir de la versión 12.2 la biblioteca no funciona y se deben utilizar otros archivos. Una vez resuelto esto se clickea *OK*.

4. En la próxima pantalla elegiremos *"I would like to create a new design"* y luego presionaremos *Next*.

5. Es en este paso donde se refleja el resultado de haber incluido la nueva biblioteca que nos habilitó el soporte para el kit XUPV5. Para ello elegiremos la parametrización tal cual se muestra en la figura F.3, allí vemos la opción *XUPV5-LX110T Evaluation Platform*.

6. Luego elegiremes *Single-Processor System* y continuamos con *Next*. Figura F.4.

7. En este paso elegiremos las características básicas del procesador como son la frecuencia de operación y la cantidad de memoria local que poseen. Nosotros elegiremos 125 MHz y 64 KB de RAM respectivamente como se muestra en la figura F.5

8. Ahora deberemos elegir cuáles periféricos tendrá conectados nuestro procesador. En la figura F.6 se puede observar una gran cantidad de periféricos como memoria DDR2_SDRAM, SRAM, Dip switches, LEDs, etc. Sin embargo, nuestros requisitos son simples y por lo tanto elegiremos sólo tres periféricos: una UART RS232, un bus de instrucciones y otro bus para los datos. La UART la configuraremos en 115200 bps, 8 bits de datos, sin paridad y sin usar interrupciones. Debido a que no elegimos SRAM no tendremos la posiblidad de elegir caché, figura F.7.

9. Por último se nos muestra la página de resumen de la configuración. Para continuar clickeamos *Finish*. Figura F.8.

10. A continuación se verá la interfaz del *Xilinx Platform Studio* que está compuesta por varios paneles, que no describiremos en este tutorial para ser sucintos y porque no lo necesitaremos para mostrar la metodología de diseño. Figura F.9. Por ejemplo, allí se puede ver como es la interconexión de los componentes con los distintos buses.

Los pasos recién descriptos nos permitirán tener una plataforma de *MicroBlaze* para importarla en *System Generator*. Más adelante nos explayaremos en los pasos

[1] cuyo precio es superior.

[2] `http://www.xilinx.com/univ/xupv5-lx110t-bsb.htm` y `http://www.xilinx.com/univ/xupv5-lx110t/design_files/EDK-XUPV5-LX110T-Pack.zip`

[3] `http://www.xilinx.com/products/boards/xupv5/base-system-builder.htm`

a realizar para tener una plataforma completa que interactúe con nuestra lógica de emulación.

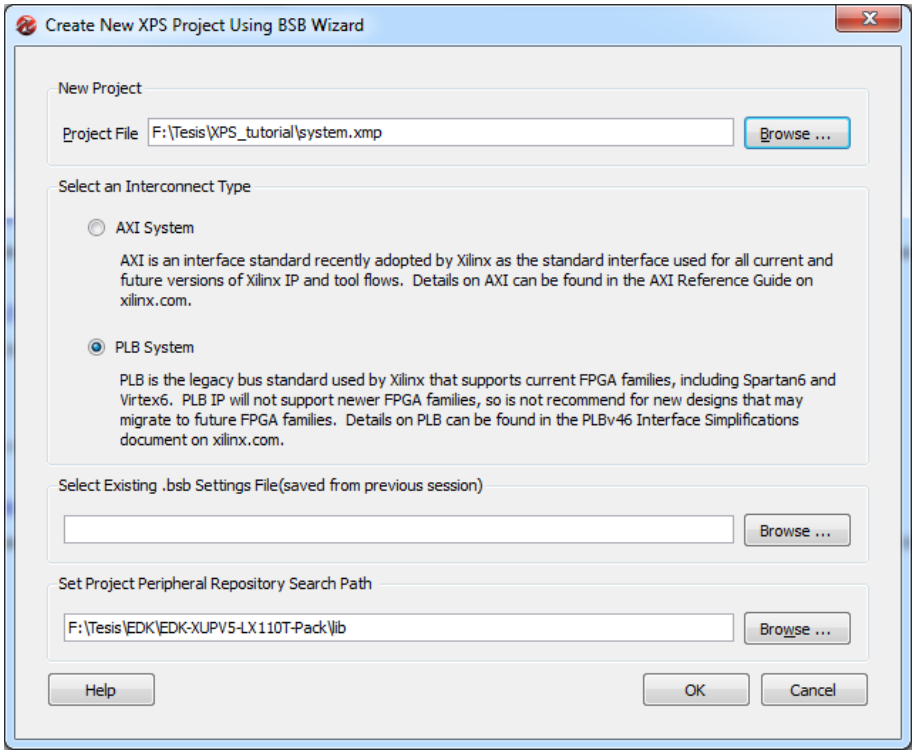


FIGURA F.1: *Xilinx Platform Studio*. Ventana de creación de nuevo proyecto.

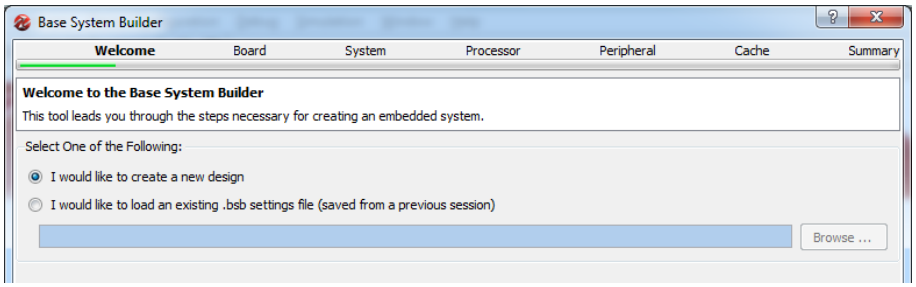


FIGURA F.2: *Xilinx Platform Studio*. Creación de nuevo diseño.

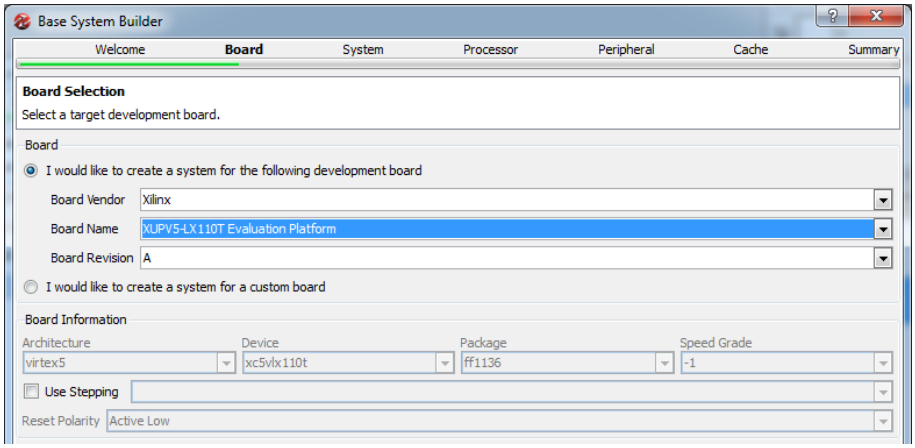


FIGURA F.3: *Xilinx Platform Studio*. Elección del kit de desarrollo.

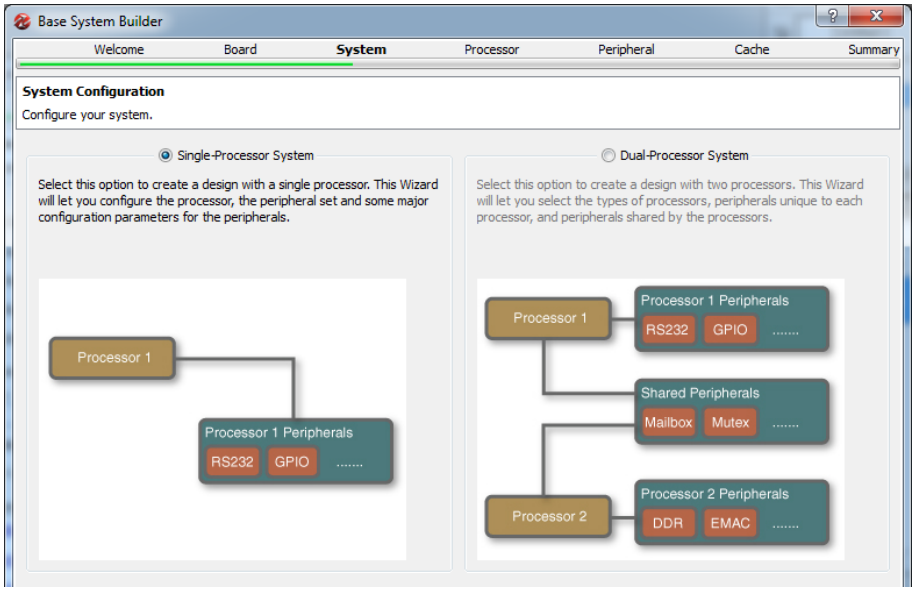


FIGURA F.4: *Xilinx Platform Studio*. Configuración single-processor.

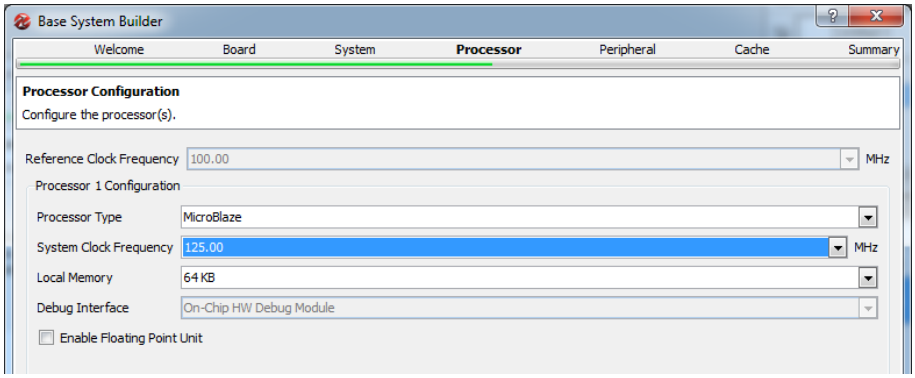


FIGURA F.5: *Xilinx Platform Studio*. Parámetros de frecuencia y memoria.

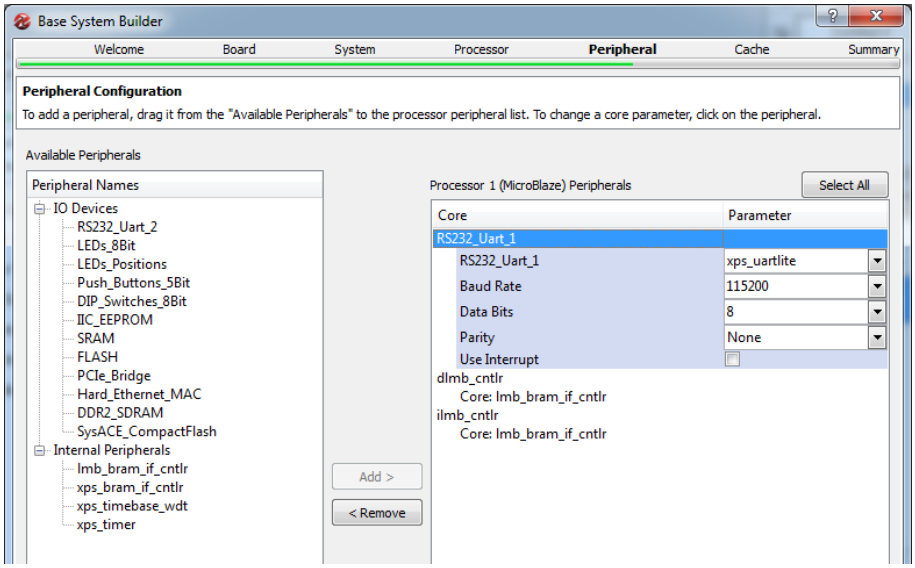


FIGURA F.6: *Xilinx Platform Studio*. Elección de periféricos.

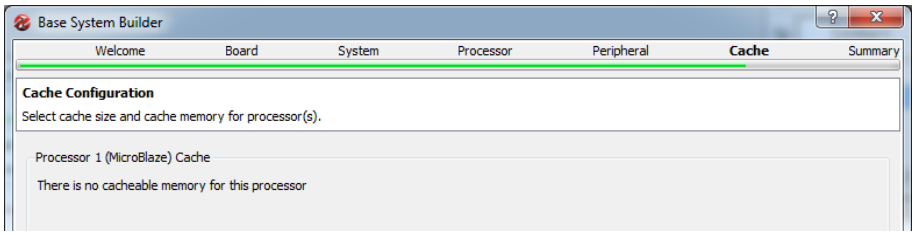


FIGURA F.7: *Xilinx Platform Studio*. Configuración de caché.

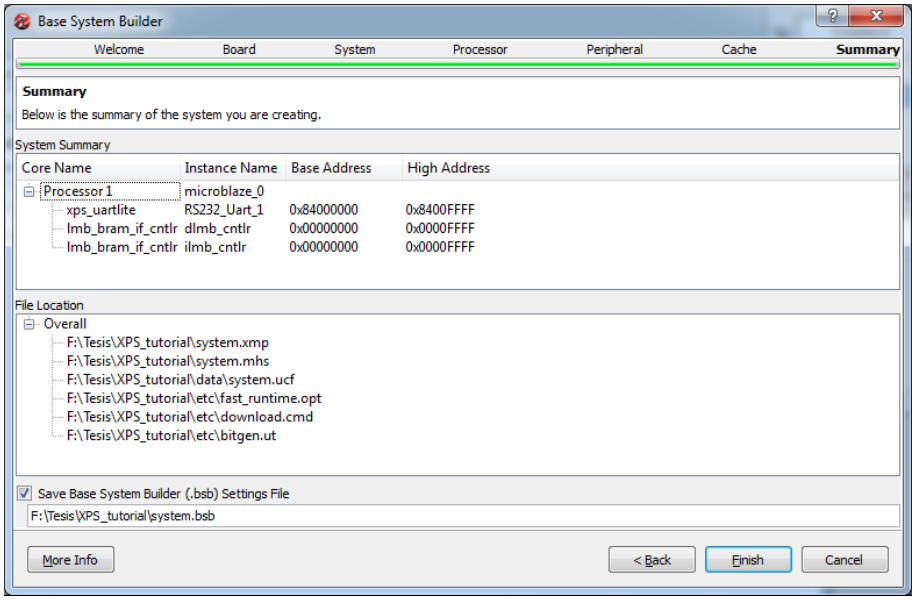


FIGURA F.8: *Xilinx Platform Studio*. Resumen del diseño.

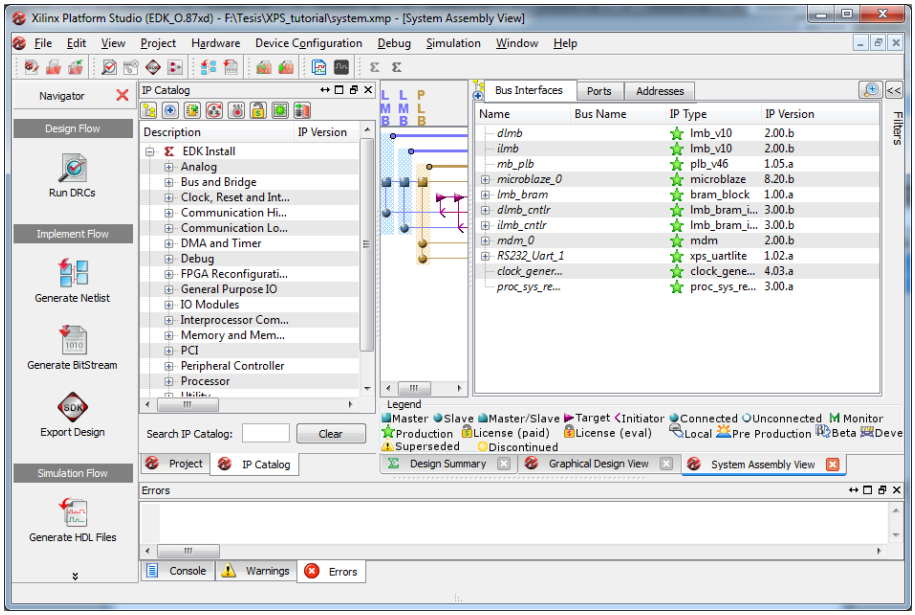


FIGURA F.9: Interfaz del *Xilinx Platform Studio*.

F.2. Asociación de los recursos en *MicroBlaze*

En §F.1 generamos una plataforma de *MicroBlaze* y ahora la importaremos en *System Generator* para asociar los recursos compartidos que queremos controlar. En esta etapa se configurará un periférico que permite acceder a dichos recursos

y a su vez se configurará este periférico para ese conjunto de recursos compartidos. Básicamente se realiza un *memory map* donde se establece una dirección de memoria a cada recurso.

A continuación describiremos los pasos realizados para poder importar la plataforma de *MicroBlaze* en *System Generator*. El nivel de detalle de los pasos descriptos será tal como para que otra persona pueda realizarlo sin tener demasiado conocimiento del tema.

1. Abrir el modelo de Simulink/*System Generator* al cual le queremos importar nuestro *soft-processor*. Desde el *Simulink Library Browser* ir a *Libraries*→*Xilinx Blockset*→*Control Logic*→*EDK Processor* y debemos arrastrar dicho bloque a nuestro modelo. Una vez que lo arrastramos se verá tal como muestra la figura F.10.

FIGURA F.10: *System Generator*. Token EDK Processor.

2. Luego vamos al modelo de Simulink/*System Generator* y hacemos doble-click sobre el bloque *EDK Processor* y nos aparecerá una ventana (figura F.11) que nos indica que está buscando las memorias compartidas que tiene nuestro modelo.

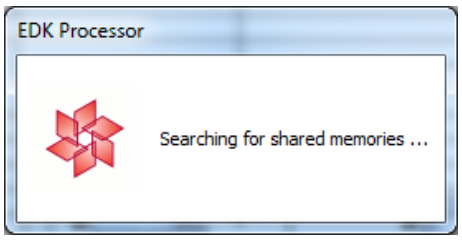


FIGURA F.11: *System Generator*. Buscando recursos recompartidos.

3. Una vez abierta la ventana del *EDK Processor*, en el tab *Basic* debemos elegir qué hacer con el procesador eligiendo una opción en el combo box *"Configure Processor for"*. Acá existen dos opciones: *"EDK pcore generation"* y *"HDL netlisting"* como muestra la figura F.12. La primera se usa para que nuestro diseño de *System Generator* se sintetize en un periférico, por eso la **p** en *pcore*. La que nos interesa es la segunda, *"HDL netlisting"*, que inserta el *MicroBlaze* en nuestro diseño de *System Generator*. Una vez elegida esa opción, nos aparecerá una ventana para abrir un archivo `.xmp` (Xilinx Microprocessor Project).

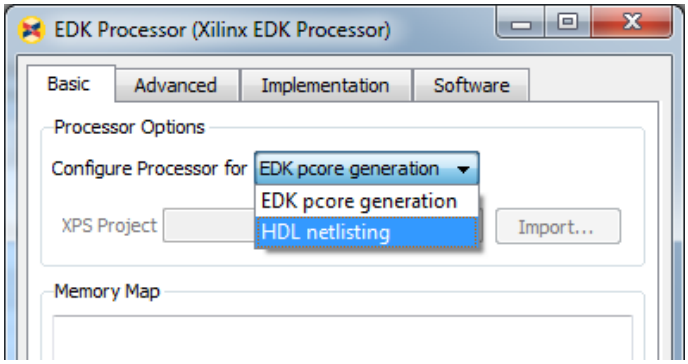


FIGURA F.12: *EDK Processor Block*. Importación de plataforma XPS.

4. Entonces elegiremos el archivo que generamos en §F.1 llamado *F:\Tesis\XPS_tutorial\system.xmp*. Desde este momento, *System Generator alterará*[4] nuestra plataforma de *MicroBlaze*, por ejemplo insertando el periférico que se conecta con los recursos compartidos. Ese proceso se nos indica con un cartel como el de la figura F.13. Terminado este proceso veremos los siguientes valores de parámetros, figura F.14.

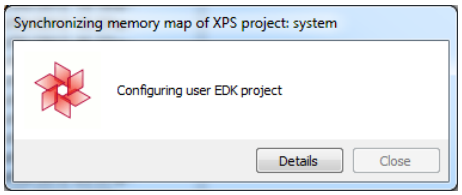


FIGURA F.13: *System Generator*. Configurando/Modificando el proyecto XPS.

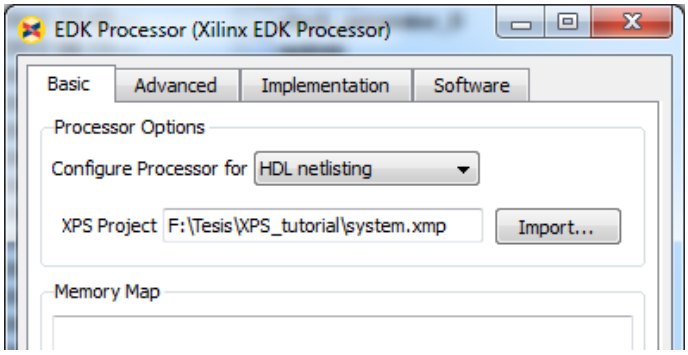


FIGURA F.14: *EDK Processor Block*. Parámetros de la plataforma importada.

5. Una vez importada la *netlist* de nuestra plataforma debemos elegir cuáles recursos se *mapearán* para que sean accesibles. En la figura F.15 observamos que

[4] Por eso es recomendable hacer un backup del directorio donde se encuentra el proyecto de XPS.

el panel *Memory map* está vacío, y que abajo hay un combo box llamado *"Available Memories"* en el cual observamos los nombres de las memorias y registros compartidos de nuestro diseño. Entonces, seleccionaremos *<all>* y presionaremos *Add*. Finalmente veremos que el panel *Memory map* se pobló con los nombres de nuestros recursos compartidos y que en el combo box ahora dice *<empty>* (figura F.16).

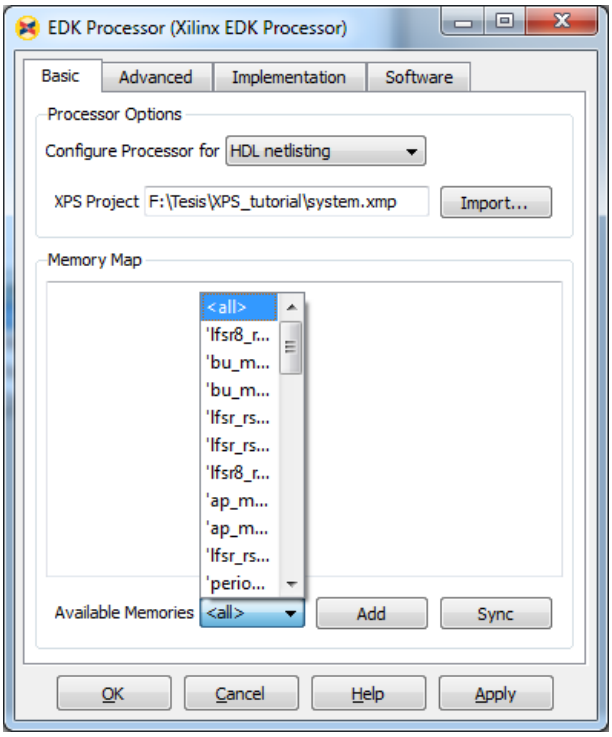


FIGURA F.15: *EDK Processor Block*. Recursos compartidos disponibles.

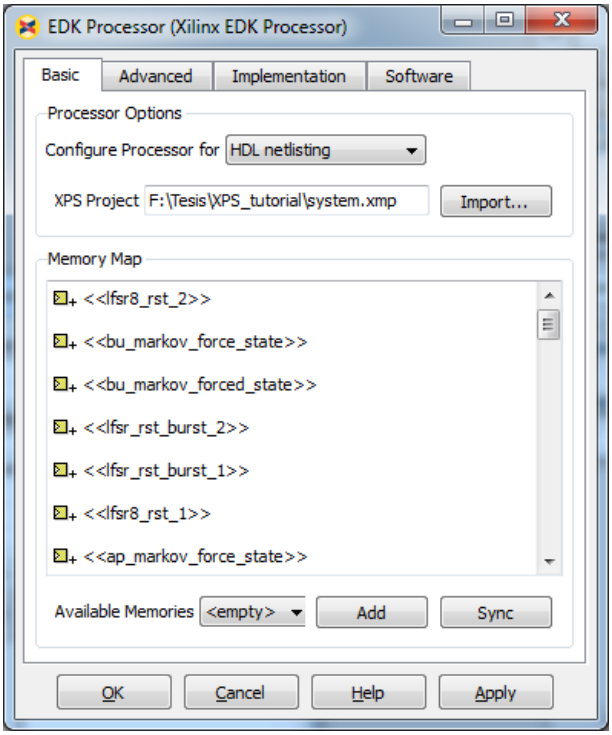


FIGURA F.16: *EDK Processor Block*. Recursos compartidos importados.

6. Por último, en la figura F.17 mostramos el tab *Implementation* en el cuál se puede ver que se utilizarán dos clocks separados, uno para el diseño de *System Generator* y otro para el *soft-processor*, y también vemos el archivo de restricciones `.ucf` que nos vimos obligados a modificarlo.

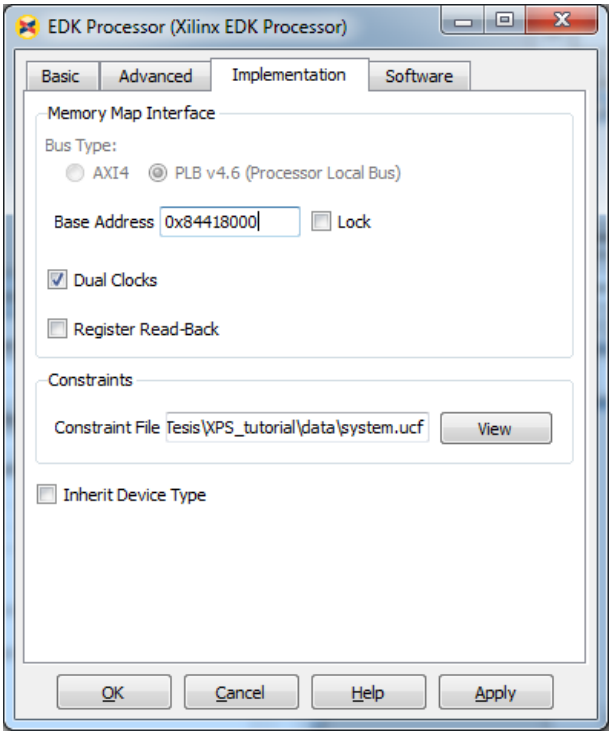


FIGURA F.17: *EDK Processor Block*. Tab *Implementation*.

7. Debido a que nuestra plataforma de *MicroBlaze* cuenta con el periférico UART, si intentamos sintetizar el diseño de *System Generator* obtendremos un error de norma eléctrica del tipo `Conflicting IO Standards`. Esto se debe a que hay pines en el mismo banco[5] que intentan imponer normas eléctricas distintas, por ejemplo `LVCMOS25` y `LVCMOS33`. Entonces modificamos el archivo *F:\Tesis\XPS_tutorial\data\system.ucf* de la siguiente manera:

   - donde decía

     ```
     Net fpga_0_RS232_Uart_1_RX_pin LOC = AG15  |  IOSTANDARD=LVCMOS33;
     Net fpga_0_RS232_Uart_1_TX_pin LOC = AG20  |  IOSTANDARD=LVCMOS33;
     ```

     pusimos

     ```
     Net fpga_0_RS232_Uart_1_RX_pin LOC = AG15;
     Net fpga_0_RS232_Uart_1_TX_pin LOC = AG20;
     ```

   - donde decía

     ```
     Net fpga_0_clk_1_sys_clk_pin LOC = AH15  |  IOSTANDARD=LVCMOS33;
     ```

     pusimos

     ```
     Net fpga_0_clk_1_sys_clk_pin LOC = AH15  |  IOSTANDARD=LVCMOS25;
     ```

[5] las salidas están agrupadas por bancos.

F.3. Usando *Hardware Co-Simulation*

Hardware Co-Simulation es la funcionalidad que nos permite que parte del diseño se sintetice en el FPGA mientras que otra parte quede en *Simulink System Generator*. Debido a que nuestro kit XUPV5 no es soportado de igual manera que los kits oficiales de *Xilinx* debemos hacer un proceso adicional. El fin de dicho proceso es agregar a nuestro kit como posible *Compilation Target*.

Para crear un nuevo *Compilation Target* para nuestro kit y utilizar *Hardware Co-Simulation* debemos realizar los siguientes pasos:

1. Hacer doble-click sobre el token *System Generator* y se abrirá una ventana.

2. En "Compilation" ir a *Hardware Co-Simulation→New Compilation Target...* (ver figura F.18), nos aparecerá una ventana y llenaremos la información de la siguiente manera (ver figura F.19)

   - **Board Name**: Digilent XUPV5LX110T
   - **System Clock**: 100 MHz **Pin Location**: AH15
   - **JTAG Options**: Hago Detect (con el cable USB JTAG conectado a la placa encendida) y ello da como resultado

     ```
     Boundary Scan Position: 5
     IR Lengths: 16, 16, 8, 8, 10
     ```

   - **Targeatable Device**: *Virtex5→xc5vlx110t-1 ff1136*
   - **Non-Memory-Mapped Ports**: Para llenar esta información debo presionar *Add...* en la sección *Non-Memory-Mapped Ports* y se abrirá la ventana de la figura F.20. Si quiero utilizar LEDS como indicadores lo completo de manera que en el archivo `.ucf` [6] me termine quedando lo siguiente.

     ```
     NET LED8bit(0) LOC = "H18"; # led8bit contingent
     NET LED8bit(0) PULLUP; # led8bit contingent
     NET LED8bit(1) LOC = "L18"; # led8bit contingent
     NET LED8bit(1) PULLUP; # led8bit contingent
     NET LED8bit(2) LOC = "G15"; # led8bit contingent
     NET LED8bit(2) PULLUP; # led8bit contingent
     NET LED8bit(3) LOC = "AD26"; # led8bit contingent
     NET LED8bit(3) PULLUP; # led8bit contingent
     NET LED8bit(4) LOC = "G16"; # led8bit contingent
     NET LED8bit(4) PULLUP; # led8bit contingent
     NET LED8bit(5) LOC = "AD25"; # led8bit contingent
     NET LED8bit(5) PULLUP; # led8bit contingent
     NET LED8bit(6) LOC = "AD24"; # led8bit contingent
     NET LED8bit(6) PULLUP; # led8bit contingent
     NET LED8bit(7) LOC = "AE24"; # led8bit contingent
     NET LED8bit(7) PULLUP; # led8bit contingent
     NET LED_C LOC = "E8"; # led_c contingent
     NET LED_C PULLUP; # led_c contingent
     NET LED_E LOC = "AG23"; # led_e contingent
     NET LED_E PULLUP; # led_e contingent
     NET LED_N LOC = "AF13"; # led_n contingent
     ```

[6] en nuestro caso el archivo se llamaba c:\Xilinx\13.4\ISE_DS\ISE\sysgen\plugins\compilation\Hardware Co-Simulation\Digilent XUPV5LX110T\digilent_xupv5lx110t.ucf.

```
NET LED_N PULLUP; # led_n contingent
NET LED_S LOC = "AG12"; # led_s contingent
NET LED_S PULLUP; # led_s contingent
NET LED_W LOC = "AF23"; # led_w contingent
NET LED_W PULLUP; # led_w contingent
```

3. Presiono Install. Esto me también me genera la libreria de non-memory-mapped ports que se reflajará como nuevos bloques en *System Generator*.

4. Ahora que creamos el *Compilation Target*, para sintetizar una parte de nuestro diseño hay que crear un subsistema que contenga lo que queremos ejecutar en el FPGA. Este subsistema debe contener un token *System Generator*. Luego haremos doble-click sobre éste y elegiremos en "Compilation" nuestro kit "Digilent XUPV5LX110T" tal como se ve en la figura F.21. Finalmente, presionaremos *Generate*. Este proceso puede tardar varios minutos dependiendo de la complejidad del diseño y la velocidad de la PC.

5. Una vez terminado el proceso aparecerá un nuevo bloque. Por ejemplo, nosotros lo aplicamos sólo al *soft-processor MicroBlaze* y por ello no tiene puertos (ver figura F.22.

6. Cuando vayamos a ejecutar una simulación primero deberíamos fijarnos en qué modo queremos correr el FPGA (ver figura F.23). Existen dos opciones: *free running clock* y *single stepped clock*. En el primero no hay ningún sincronismo entre el modelo simulado por software y lo que está funcionando en el FPGA. Mientras que en el segundo, la simulación será igual que cuando la lógica no estaba sintetizada dentro del FPGA. Para más información sobre estos modos referirse a [46].

7. Si nuestro subsistema contiene un bloque *EDK Processor* al clickear sobre el nuevo bloqué tendremos una pestaña *Software* desde donde pondremos ejecutar el SDK para crear nuestros programas tal como se ve en la figura F.24.

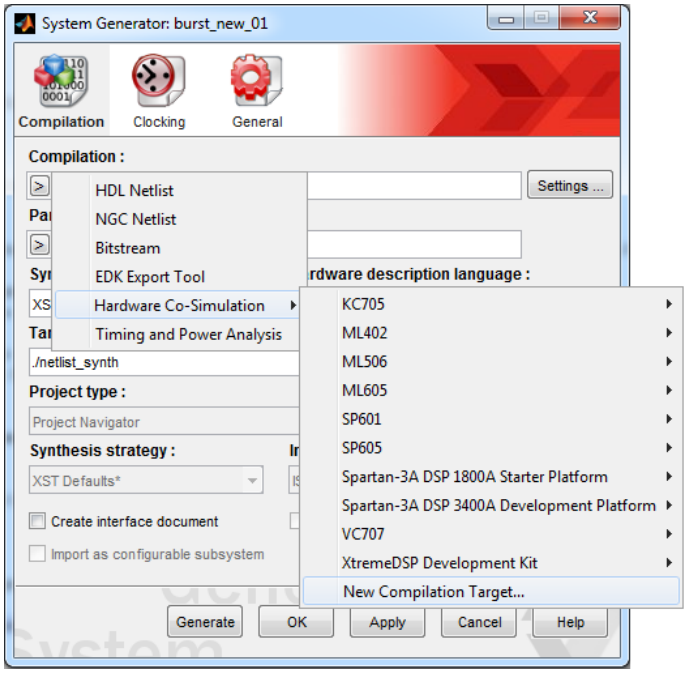


FIGURA F.18: Menú para crear de un nuevo *Compilation Target*

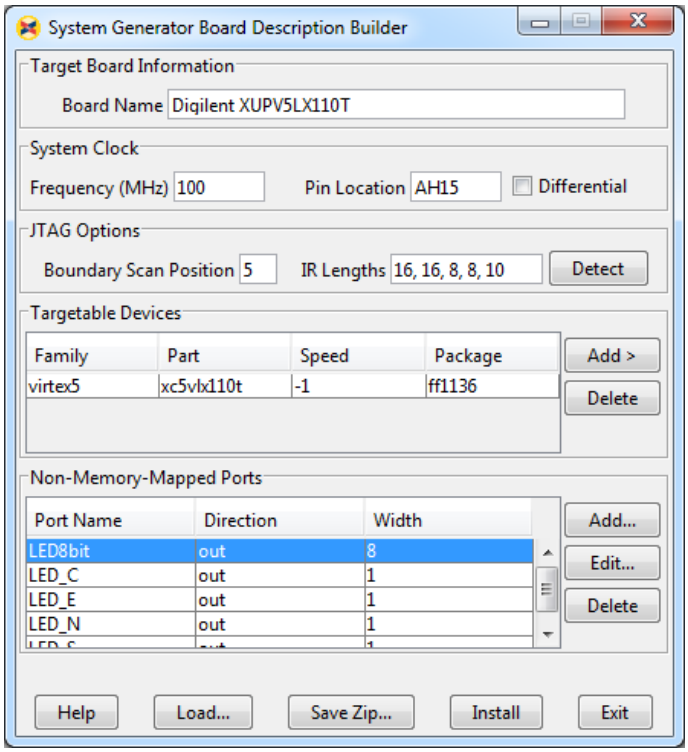


FIGURA F.19: Ventana de configuración del nuevo *Compilation Target*

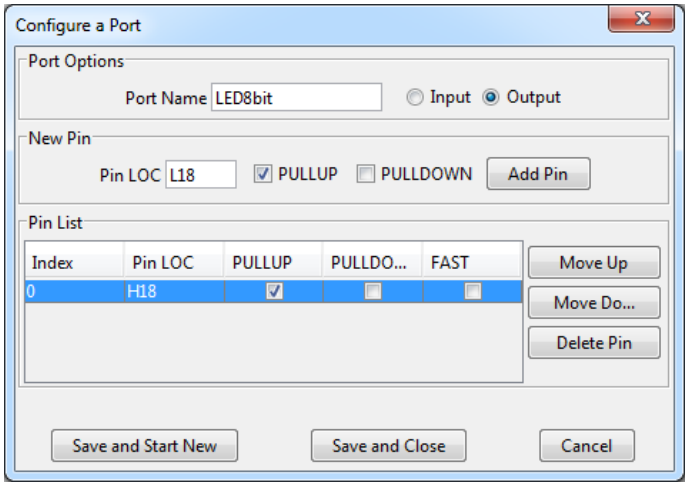


FIGURA F.20: Configurando un *non-memory-mapped port*

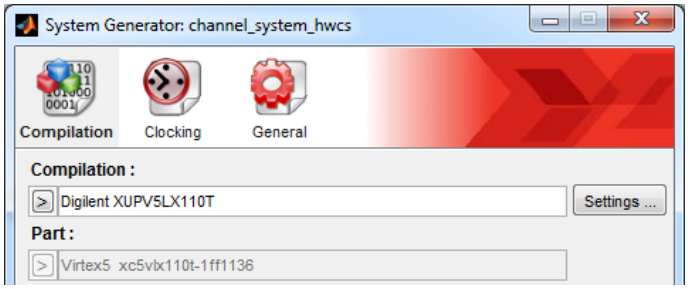


FIGURA F.21: Eligiendo nuestro *Compilation Target* para generar el bloque de Co-Simulación

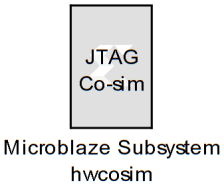


FIGURA F.22: Bloque de Co-Simulación

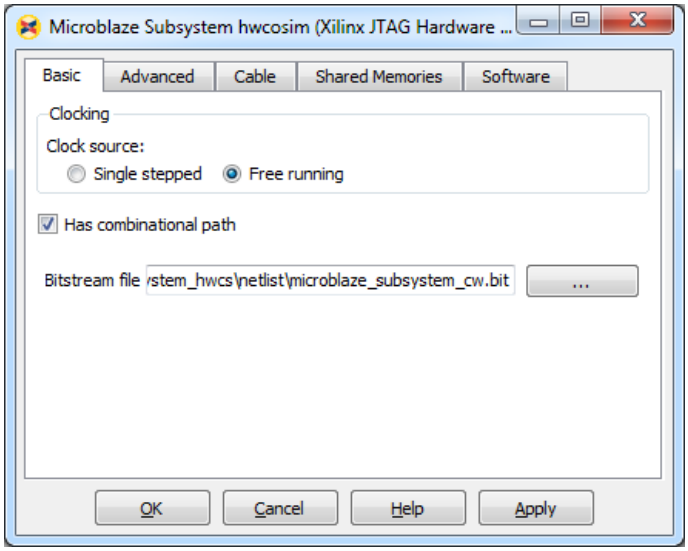


FIGURA F.23: Opciones de clock del bloque de Co-Simulación

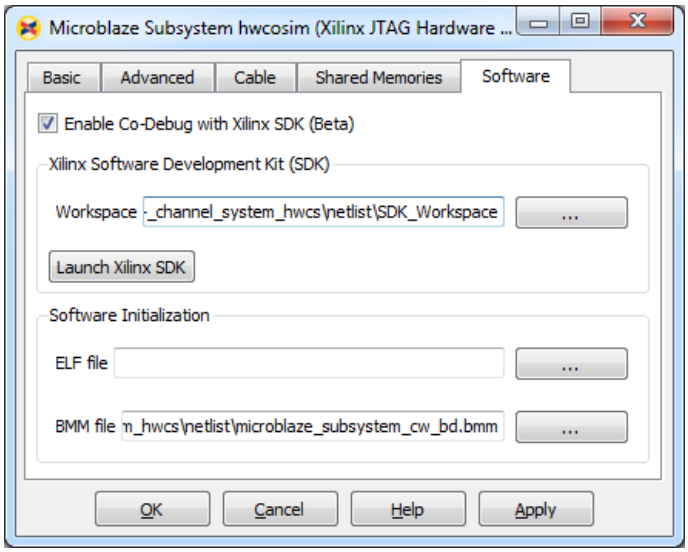


FIGURA F.24: Pestaña *Software* para diseños *MicroBlaze*

F.4. Generando el *Bitstream* y configurando *Xilinx SDK*

En este apéndice veremos cómo configurar el SDK para poder crear proyectos de software para nuestro diseño de *System Generator* que incluye un bloque *EDK Processor*. En este caso trabajaremos sobre el diseño compilado en *Bitstream*, es decir la imagen con que se programa el FPGA. Por lo tanto, primero veremos cómo

sintetizar nuestro diseño en un Bitstream que dará como resultado un archivo de extensión `.bit`.

1. Una vez que tenemos el diseño vamos a la ventana de *Simulink* de mayor jerarquía donde se encuentra el token *System Generator* (figura F.25). Hacemos doble click y bajo la sección *Compilation* configuraremos varios parámetros y dejaremos el resto en su valor predeterminado.

   - **Compilation**: aquí elegiremos *Bitstream*.
   - **Part**: aquí debemos elegir el modelo de nuestro FPGA, que para el kit XUPV5 es *Virtex5 xc5vlx110t-1ff1136* que se elige en 4 pasos: *Virtex5*→*xc5vlx110t*→*-1*→ff1136.
   - **Target directory**: aquí elegimos el directorio donde irán todos los archivos producto de la síntesis. El valor predeterminado es un path relativo y está especificado en formato UNIX y es *"./netlist"*, pero también es posible poner un path absoluto *"C:\Proyectos\emulador_synth"*. Es **recomendable** que el directorio se encuentre en la misma unidad de disco que el software de *Xilinx* sino existe un bug[7] con el utilitario `data2mem` y sino deberemos este utilitario ejecutarlo a mano[8].

FIGURA F.25: Token *System Generator*

2. Una vez realizada la configuración presionamos *Generate* (figura F.26). Este proceso puede tardar bastante tiempo dependiendo del tamaño del diseño y la velocidad de la máquina en la cual se ejecuta. Podría llegar a tardar horas con lo cual hay que estar seguros de qué vamos a sintetizar.

[7] `http://www.xilinx.com/support/answers/46140.htm`

[8] Este fue nuestro caso, y se soluciona abriendo una consola de Xilinx y ejecutando el comando que falló (el cual podemos obtener de la consola del SDK). Pero primero hay que estar situado en la misma unidad de disco de donde se encuentran los parámetros.

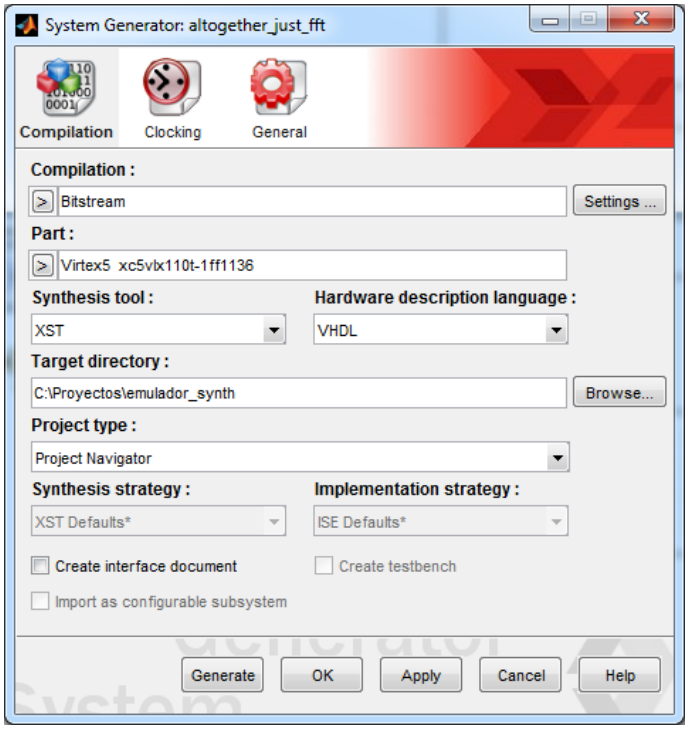


FIGURA F.26: Configuración de *System Generator* para generar un *Bitstream*

3. Una vez terminado el proceso, si todo salió bien, aparecerá una ventana que dice *Compilation Successfully*. Allí se recomienda presionar el botón *Show Reports* que nos muestra los reportes a través del *Xilinx ISE Reports* de una manera más accesible. Si no lo hacemos tendremos que leer los reportes manualmente y estos son archivos que pueden ocupar varios MegaBytes. Uno de los procesos que se realizan en la compilación es la exportación de la plataforma de *MicroBlaze* al SDK.

4. A diferencia de cuando hacemos *Hardware Co-Simulation* el proceso de compilación no nos configura automáticamente el SDK. Por lo tanto, debemos hacerlo manualmente. Primero crearemos un directorio que será el Workspace del SDK. El directorio será *"C:\Proyectos\emulador_synth\SDK_Workspace"*. Entonces procederemos a abrir el programa *Xilinx SDK* y nos pedirá que le indiquemos el Workspace (figura F.27). Luego iremos a *File→New→Xilinx Hardware Platform Specification* (figura F.28). Luego bajo la sección "Target Hardware Specification" presionaremos el botón *Browse...* Allí debemos ir al directorio *SDK_Export\hw* relativo a nuestro *Target directory* que ingresamos previamente. Allí habrá un archivo llamado *system.xml*. Al abrir ese archivo se nos completarán todos los campos de la pantalla anterior y le daremos click al botón *Finish* (figura F.29). Finalmente veremos en el panel izquierdo *Project Explorer* la plataforma de hardware (figura F.30).

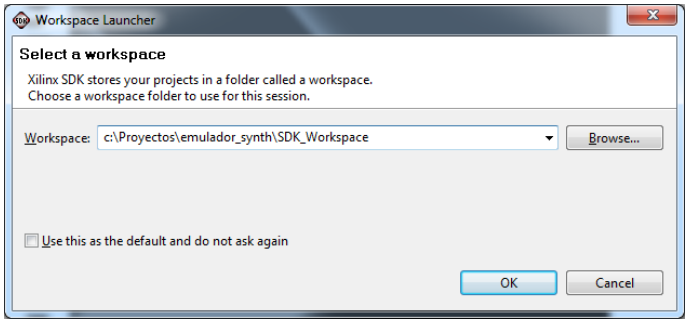


FIGURA F.27: *Xilinx SDK*. Elección del workspace

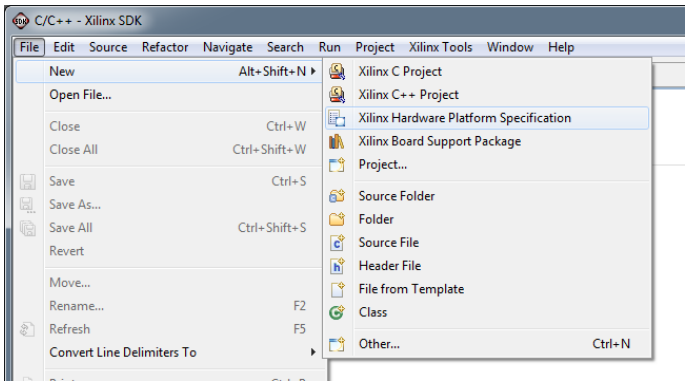


FIGURA F.28: *Xilinx SDK*. Nueva plataforma de hardware

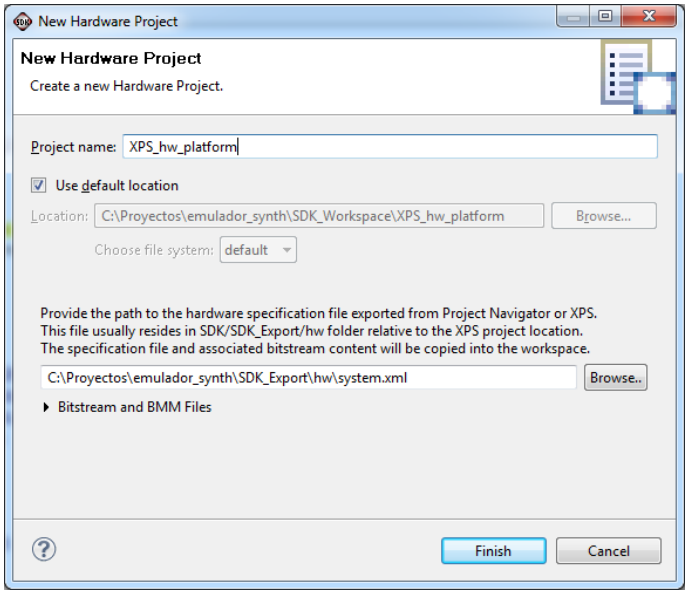


FIGURA F.29: *Xilinx SDK*. Parámetros de la nueva plataforma de hardware

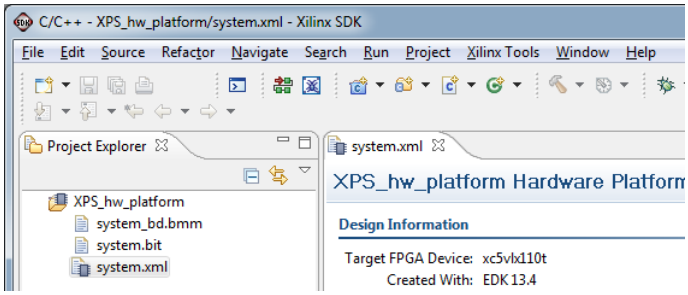


FIGURA F.30: *Xilinx SDK*. Panel *Project Explorer*

5. Luego debemos crear el Board Support Package que es el software base que contiene los drivers que necesitamos para desarrollar nuestros programas. Pero aquí es donde debemos hacer un paso adicional[9] para que encuentre el código del driver de la interfaz que intercomunica el procesador *MicroBlaze* con el diseño lógico. Para ello debemos ir a *Window→Preferences→Xilinx SDK→Repositories*. Luego en *Local Repositories* clickeamos *New...*. Allí deberemos poner el directorio *SDK_Export\sysgen_repos* que se encuentra relativo a nuestro *Target directory*. Pero especificaremos el path de manera absoluta (figura F.31).

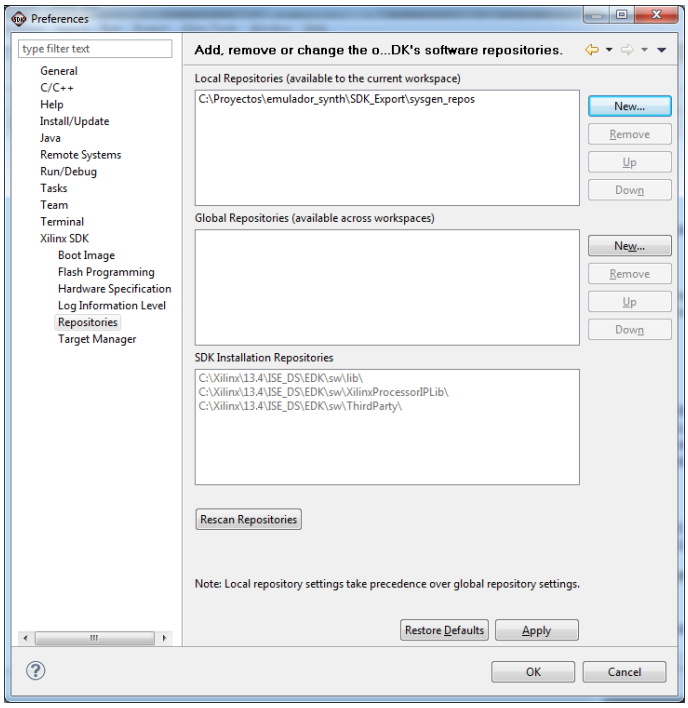


FIGURA F.31: *Xilinx SDK*.Configurando repositorio de drivers

[9] en *Hardware Co-Simulation* se configura automáticamente

6. Para evitar un bug[10] debemos crear el Board Support Package desde el wizard de *Xilinx C Project*. Entonces vamos a *File→New→Xilinx C Project* (figura F.32). Entonces elegiremos un *Project name*, por ejemplo "first_program" y bajo *Select Project Template* seleccionaremos *System Generator example* (figura F.33). Luego clickeamos *Next* y en la próxima pantalla se encuentra elegido *Create a new Board Support Package project*. Allí elegimos un *Project name*, por ejemplo "microblaze_bsp_0" y presionamos *Finish* (figura F.34). Es recomendable guardar ese proyecto como base, ya que si intentamos crear otro nuevo proyecto con el template *System Generator example* eligiendo como *Board Support Package* a "microblaze_bsp_0" obtendremos el error del bug **invalid command name get_stdout**. En la figura F.35 observamos el *Project Explorer* del SDK que contiene nuestra plataforma de hardware, el Board Support Package y nuestro primer programa, donde mostramo el archivo *sg_hello_world.c*.

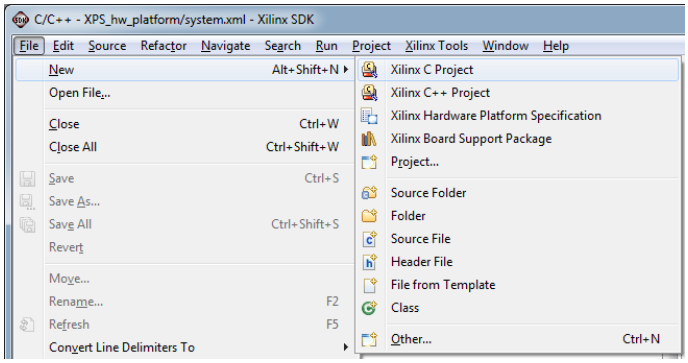


FIGURA F.32: *Xilinx SDK*. Nuevo proyecto de C de *Xilinx*

[10] este se manifiesta como "invalid command name get_stdout" en el wizard de creación de *Xilinx C Project* cuando se elije el template *System Generator example* para un BSP existente

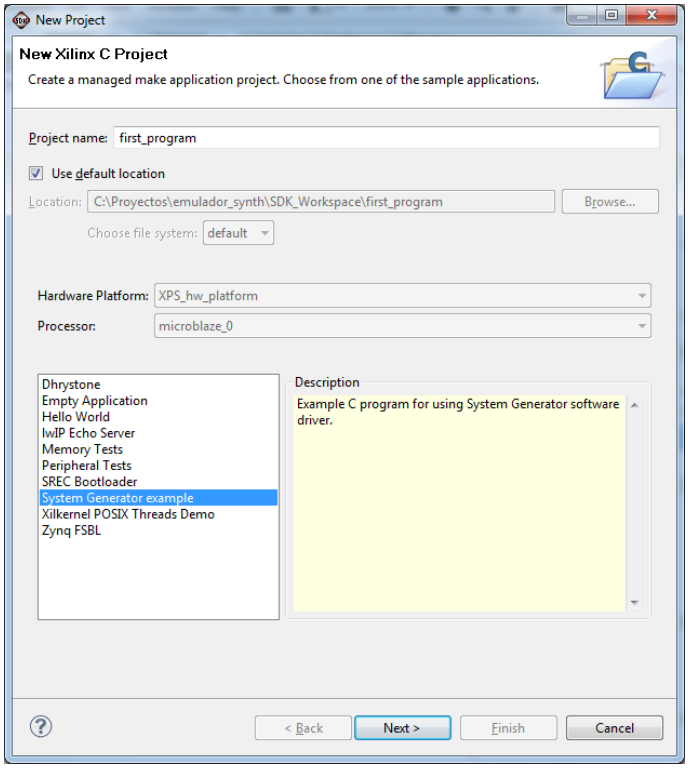


FIGURA F.33: *Xilinx SDK*. Configuración de proyecto de C de *Xilinx*

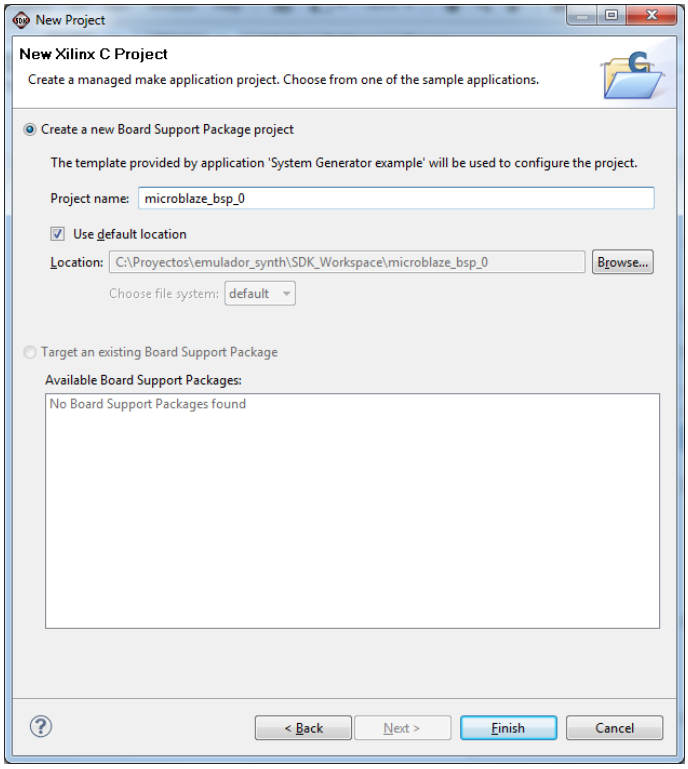


FIGURA F.34: *Xilinx SDK*. Creación de Board Support Package

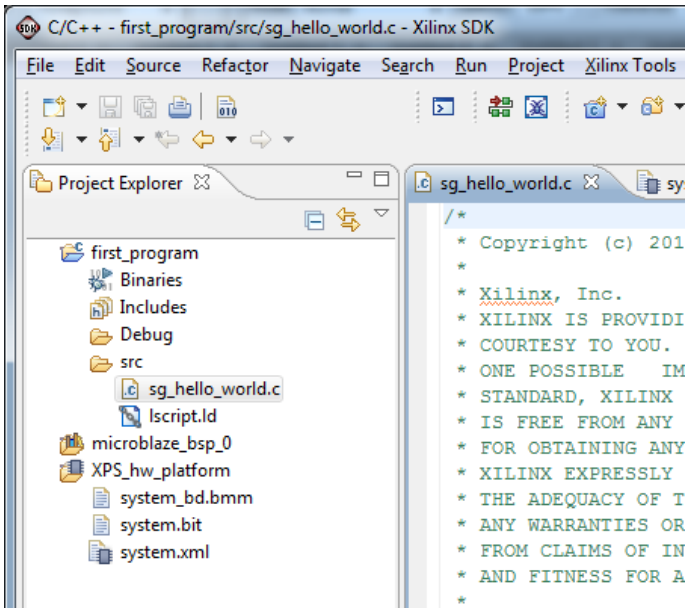


FIGURA F.35: *Xilinx SDK*. Panel *Project Explorer* mostrando todo lo realizado.

7. Un tip a tener en cuenta a la hora de compilar el software de *MicroBlaze* en modo *Release* es que pueden surgir problemas. En programación es común

esperar que la versión *Release* ocupe menos que la versión *Debug*. Sin embargo, en nuestro programa el SDK decía que nuestro software no entraba en los 64 KBytes de memoria local. A su vez, la compilación arrojaba nuevos *warnings*.

```
warning: dereferencing type-punned pointer will break strict-aliasing rules
```

El problema estaba en que en la *Building Configuration* para la versión *Release* estaba el flag de optimización `-O3`. Esto puede arreglarse alterando dicho parámetro a otro modo de optimización que no de estos *warnings*, como por ejemplo `-O0` que es de la versión *Debug*.

F.5. API de interfaz con System Generator

Cuando importamos la plataforma de *MicroBlaze* en el *System Generator* este último la modificó y le agregó un periférico que se comunica con los recursos compartidos. Este periférico cuenta con un driver que implementa una API y a través de las funciones de dicha API podemos leer o escribir aquellos recursos. La documentación de esta API siempre es adjuntada en el directorio de compilación elegido en el *System Generator Token* en el path relativo *SDK_Export/sysgen_repos/drivers/sg_plbiface_v1_00_a/doc/htn*
A continuación mostraremos breves líneas de código que ejemplifican como usar la API.

- Lo primero que hay que hacer es inicializar la interfaz de comunicación con *System Generator*

```
xc_iface_t *iface; /* System Generator Interface */
/*  Software driver initialization assuming Pcore ID  0 (zero) */
XC_CfgInitialize(&iface, &SG_PLBIFACE_ConfigTable[0]);
if (iface == NULL) print("ERROR: unable to access config table\r\n");
```

- Después hay que asociar el recurso compartido con su tipo de datos que lo puede manejar, en este caso es un "From Register"

```
xc_status_t status;
xc_to_reg_t *toreg_periodic_sync_enable; /* "To Register" struct, van apareados */
status = XC_GetShMem(iface, "periodic_sync_enable", (void **) &toreg_periodic_sync_enable);
if (status != XC_SUCCESS )
  print("ERROR: unable to associate shared register periodic_sync_enable\r\n");
```

- Luego podemos escribir a un recurso compartido, en este caso es un "From Register"

```
xc_status_t status;
u32 dato = 1;
status = XC_Write(iface, toreg_per_imp_sync_pulse_load->din, dato); /* Possible values 0 1 */
if (status != XC_SUCCESS ) print("ERROR: could not write register\r\n");
```

- En el caso de la memoria compartida, la asociación es similar pero el tipo de datos distinto. Si bien es posible accederla usando `XC_Read`, `XC_Write` y `XC_GetAddr` es conveniente usarla a través del puntero porque es más eficiente y más util para trabajar con *arrays* con el operador `++`.

```
u32 delays[NUM_STAGE] = {20,19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1};
int *plb_addr;
xc_status_t status;
xc_shram_t *shram_delay;

/* Associate Shared Memory */
status = XC_GetShMem(iface, "delay_forward", (void **) &shram_delay);
if (status != XC_SUCCESS )
  print("ERROR: unable to associate shared register periodic_sync_enable\r\n");

/* Getting base addresses */
plb_addr = (int*) shram_delay->addr;
/* Load initial configuration of delays for the forward channel */
for (i = 0; i<NUM_STAGE; i++)
  *(plb_addr++) = delays[i];
```

- y finalmente en el caso de una FIFO hay que chequear que no esté llena antes de escribirla

```
xc_to_fifo_t *tofifo_coef_din;
u32 fifo_full;
u32 lpfilter[LPFILTER_SIZE] = {0x07ff,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

for (i = 0; i < LPFILTER_SIZE; i++) {
    /* Check the "full" port of shared memory "tofifo_coef_din" */
    do {
        XC_Read(iface, tofifo_coef_din->full, &fifo_full);
    } while (fifo_full == 1);
    status = XC_Write(iface, tofifo_coef_din->din, lpfilter[i]);
    if (status != XC_SUCCESS ) print("ERROR: could not write FIFO tofifo_coef_din\r\n");
}
```

Apéndice G

Datos técnicos del emulador

G.1. Sintaxis de comandos

La sintaxis de comandos se resume en la próxima página. Para interpretar hay que conocer la semántica siguiente

- Los corchetes `[·]` delinean las partes de un comando
- El número inmediato que le sigue al comando indica la cantidad de dígitos hexadecimales de esa parte `[#digitos·]`. Luego se utiliza como delimitador al símbolo "`:`".
- El número entre paréntesis indica un valor o un rango de valores posibles `[#digitos:(valor) ·]` o `[#digitos:(desde..hasta) ·]`
- El texto a continuación provee una descripción de dicha parte del comando `[#digitos:(valor/rango) descripción]` o `[#digitos:(desde..hasta) descripción]`
- Finalemente los comandos están compuestos por varias partes.

Sintaxis de Comandos

```
[1: Destino][Index parameter/s]..[Value]

[1:(1) Input Delay Forward][2:(0x01..0x14) Delay tap number][4:(0x0000..0x003f) Delay value]
[1:(2) Input Coeff Forward][2:(0x01..0x14) Coeff tap number][4:(0x0000..0xffff) Coeff value]
[1:(3) Input Delay Backward][2:(0x01..0x14) Delay tap number][4:(0x0000..0x003f) Delay value]
[1:(4) Input Coeff Backward][2:(0x01..0x14) Coeff tap number][4:(0x0000..0xffff) Coeff value]
[1:(5) Input Aperiodic Markov Matrix][1:(0..5) matrix row][1:(0..4) matrix column][8:(0x00000000..0x00010000) probability UFix_17_16]
[1:(6) Input Burst Markov Matrix][1:(0..1) matrix row][1:(0) matrix column][8:(0x00000000..0x0000ffff) probability UFix_16_16]
[1:(7) Input Periodic Noise][1:(1) Sync][1:(1..2) Pulse width/Pulse period][8:(0x00000000..0x00ffffff) cycles]
                           [1:(2) Async][1:(1..2) Pulse width/Pulse period][8:(0x00000000..0x0000ffff) cycles]
[1:(8) Input LP Filter Forward][2:(0x00..0x1c) coeff tap number][4:(0x0000..0xffff) Coeff value]
[1:(9) Input Narrow Band][1:(1..2) Power Envelope/AR1 coeff][4:(0x0000..0x2000) index][4:(0x0000..0xffff) value]
[1:(a) Input 8 but width Amplitude LFSR seed][1:(1..2) Lower 32/Upper 8][8:(0x00000000.0xffffffff) Partial Seed Value]
[1:(b) Input Coloring Filter][2:(0x00..0x14) coeff tap number][4:(0x0000..0xffff) Coeff value]
[1:(c) Show][2:(0x01) Markov Matrix]
            [2:(0x02) LFSR User Input Array]
            [2:(0x03) Aperiodic Impulse Statistics]
            [2:(0x04) Burst Markov Matrix 1st Level]
            [2:(0x05) Burst Markov Matrix 2nd Level]
            [2:(0x06) Burst Group Statistics]
            [2:(0x07) LFSR Seed Aperiodic]
            [2:(0x07) LFSR Seed Burst 1st Level]
            [2:(0x08) LFSR Seed Burst 2nd Level]
[1:(d) Input 16 bit width LFSR Seed][1:(0x0..0x0f) LFSR index][1:(1..2) Lower 32/Upper 8 bits][8:(0x00000000.0xffffffff) Partial Seed Value]
[1:(e) Set registers][2:(0x01) Aperiodic Markov Force State][4:(0x0000..0x0001) Value]
                     [2:(0x02) Aperiodic Markov Forced State][4:(0x0000..0x0005) Value]
                     [2:(0x03) Aperiodic LFSR Reset][4:(0x0000..0x0001) Value]
                     [2:(0x04) Periodic Impulsive Sync Load][4:(0x0000..0x0001) Value]
                     [2:(0x05) Periodic Impulsive ASync Load][4:(0x0000..0x0001) Value]
                     [2:(0x06) Aperiodic Impulse Statistics Counters Reset][4:(0x0000..0x0001) Value]
                     [2:(0x07) Burst LFSR 1st Level Reset][4:(0x0000..0x0001) Value]
                     [2:(0x08) Burst LFSR 1st Level Reset][4:(0x0000..0x0001) Value]
                     [2:(0x09) Burst Markov Force State 1st Level][4:(0x0000..0x0001) Value]
                     [2:(0x0a) Burst Markov Forced State 1st Level][4:(0x0000..0x0001) Value]
                     [2:(0x0b) Burst Group Statistics Counters Reset][4:(0x0000..0x0001) Value]
[1:(f) Load][2:(0x01) Aperiodic Markov Matrix Forward]
            [2:(0x02) Aperiodic Markov Matrix Backward]
            [2:(0x03) Aperiod LFSR Load]
            [2:(0x04) LP Filter Forward]
            [2:(0x05) Burst Markov 1st Level Matrix Forward]
            [2:(0x06) Burst Markov 2nd Level Matrix Forward]
            [2:(0x07) Burst LFSR Load 1st Level Forward]
            [2:(0x08) Burst LFSR Load 2nd Level Forward]
            [2:(0x09) Aperiodic Amplitude 8 bit LFSR Load]
            [2:(0x0a) Burst Amplitude 8 bit LFSR Load]
            [2:(0x0b) Coloring Filter Periodic Synchronous]
            [2:(0x0c) Coloring Filter Periodic Asynchronous]
            [2:(0x0d) Coloring Filter Aperiodic Asynchronous]
            [2:(0x0e) Coloring Filter Burst]
```

G.2. Funcionamiento del sistema

En esta apéndice describiremos el funcionamiento del emulador y cómo interactúan los dos softwares: el que corre en el *MicroBlaze* y el software GUI. Si bien los diseños se verificaron en la etapa de diseño nosotros decidimos incorporar una herramienta de *Xilinx* llamada *Chipscope Pro*. Ésta permite extraer señales del FPGA cuando este se encuentra en condiciones operativas. *Chipscope Pro* es una propiedad intelectual compuesta por varios *cores* de los cuales los más usados son:

- *Chipscope Pro ICON*: Integrated CONtroller, que es el encargado de proveer una interfaz entre el FPGA y el JTAG Boundary Scan. Se encarga de controlar a los otros *cores*.
- *Chipscope Pro ILA*: Integrated Logic Analyzer, que es un analizador lógico configurable que provee diversas capacidades de trigger y almacenamiento de señales. Este *core* se conecta con el *core ICON*.

Las señales se extraen a través del JTAG que es controlado por el software *Chipscope Pro Analyzer*. Desde este software se configuran las condiciones de trigger e incluso se pueden exportar las señales capturadas para analizarlas con MATLAB.

Antes de poner en funcionamiento el sistema debemos conectar todos los cables. Debemos conectar el cable USB del JTAG a la PC, y el cable del puerto serie a la PC[1] a través de un cable *null modem*. Luego debemos arrancar la aplicación *Chipscope Pro Analyzer* y el software GUI que desarrollamos cuya interfaz se observa en la figura G.1. Nosotros hemos creado un proyecto de *Chipscope* que tiene guardada la configuración de los triggers, la configuración de los dispositivos de la JTAG *chain* y de los gráficos de señales entre otras cosas. Entonces cargamos dicho proyecto. Ahora podemos proceder a encender el kit XUPV5. En el software GUI configuramos los parámetros de la UART y clickeamos *Open Port*.

[1] en nuestro caso contábamos con un puerto serie por USB.

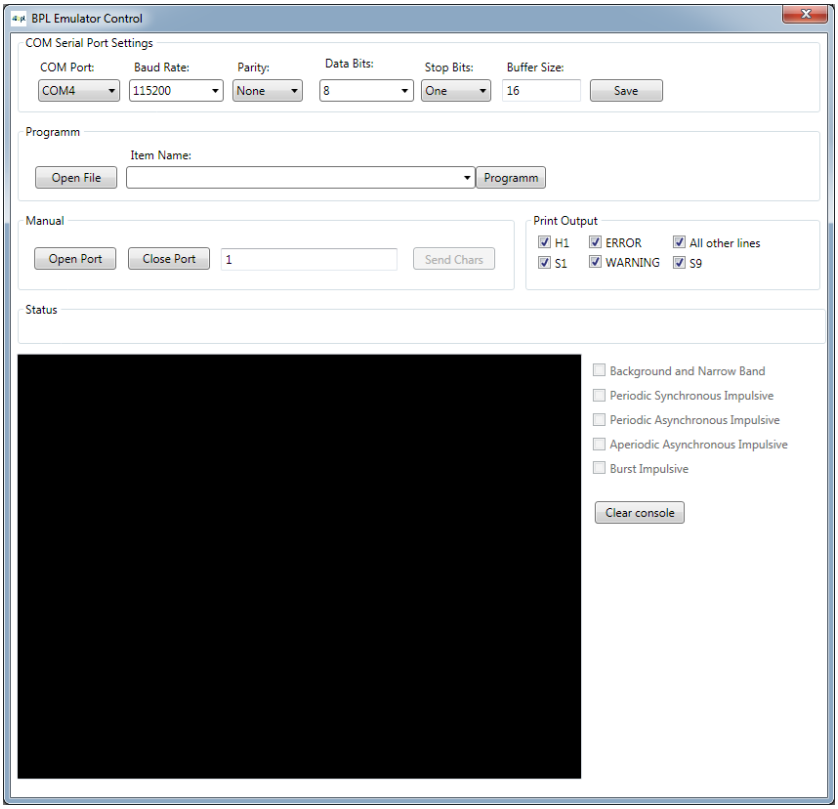


FIGURA G.1: Interfase del software GUI

Luego configuramos el *Bitstream* en el FPGA tal como se ve en la figura G.2.

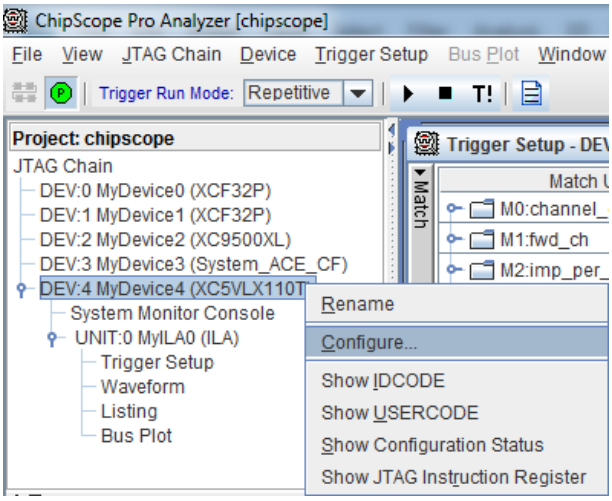


FIGURA G.2: Menú de configuración del FPGA en *Chipscope Pro Analyzer*

Una vez que carga el software vemos que en el cuadro de texto se imprime el menú principal (figura G.3). En esa misma figura podemos ver que los ruidos se

encuentran todos desactivados. Aquí se ve claramente la interacción entre el software GUI y el software de *MicroBlaze* cuya salida se observa en esa consola. También vemos a la derecha una sección *Print Output* donde se encuentran *checkboxes* que nos permiten filtrar qué tipo de información se dibuja en la consola. La descripción de cada tipo de mensaje es la siguiente:

- **H1** son mensajes de nivel 1 cuyo destino es una persona humana. Cuanto mayor es el nivel mayor es el nivel de detalle interno del software que describen los mensajes.
- **S1** son mensajes de nivel 1 cuyo destino es un software.
- **S9** son mensajes de nivel 9 cuyo destino es un software.
- **ERROR** son mensajes de error.
- **WARNING** son mensajes de advertencia.
- **All other lines** todo lo que no se corresponde con las categorías anteriores.

Vale la pena mencionar que por más que se filtren los mensajes el software de *MicroBlaze* los sigue transmitiendo por el puerto serie. Y durante el tiempo que transmite no está atendiendo a los datos que ingresan a su puerto serie, y por lo tanto puede desbordar la FIFO de 16 carácteres como antes mencionamos.

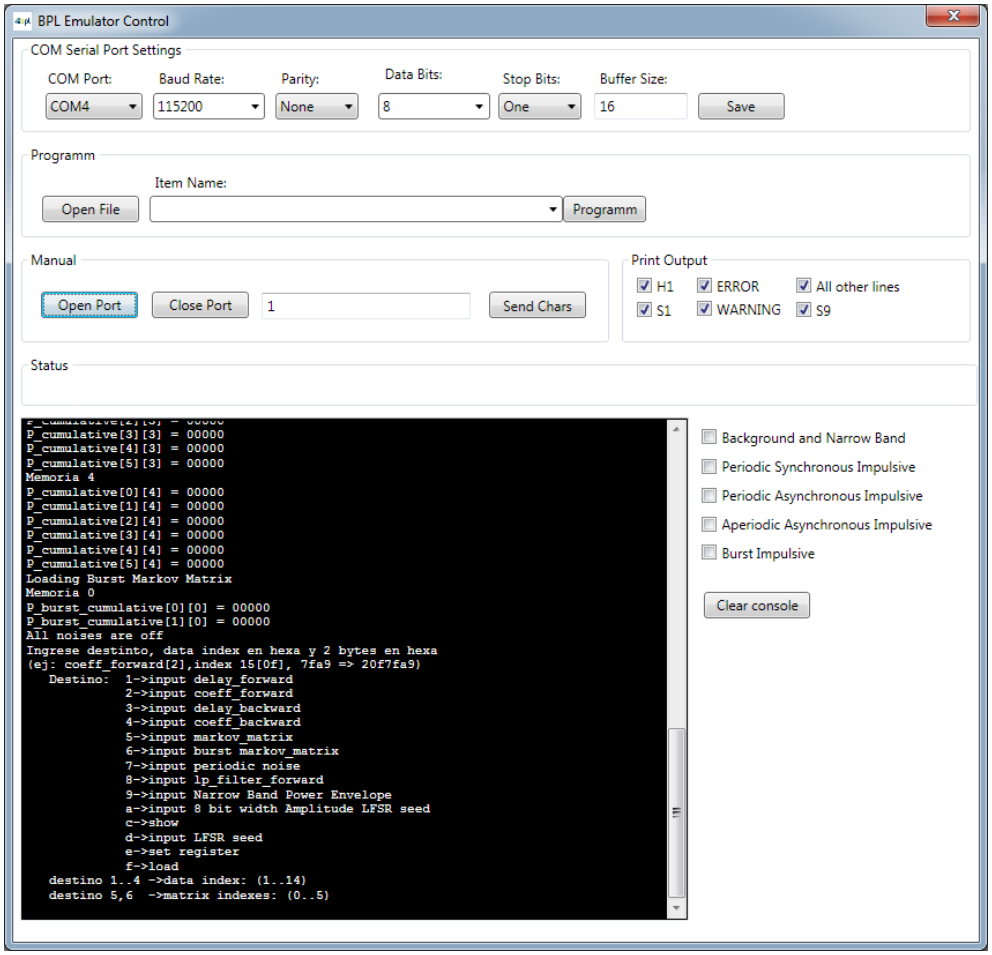


FIGURA G.3: Menú principal del emulador (en consola)

Para poder cargar configuraciones primero debemos abrir el archivos de configuraciones usando el botón *Open File*. En este caso abrimos el archivo generado por el script de MATLAB y cargamos la matriz *Moderate* correspondiente al ruido impulsivo aperiódico asincrónico (figura G.4). Allí se observa el contenido de las memorias compartidas que son las columnas de la matriz de probabilidades acumuladas de transición.

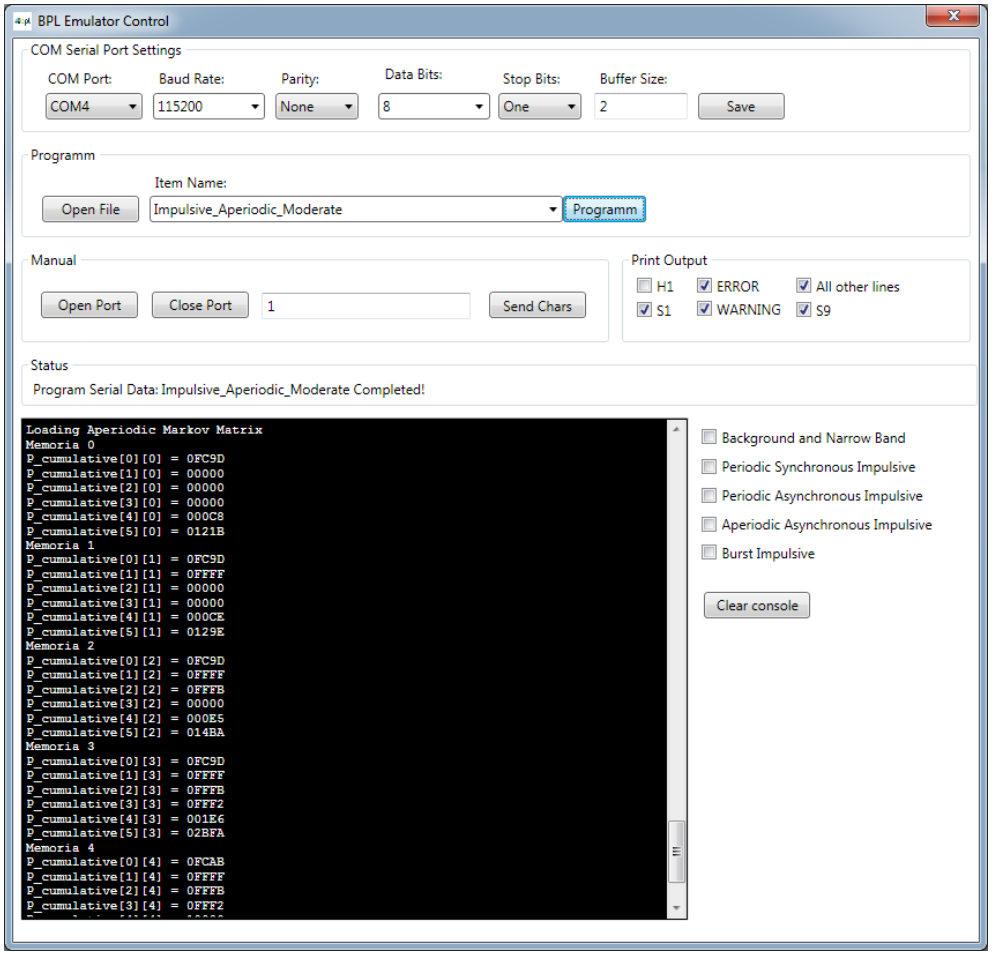


FIGURA G.4: Carga de configuración de matriz para ruido impulsivo aperiódico asincrónico

Luego en el *Chipscope Pro Analyzer* configuramos el trigger para que se dispare cuando vea un impulso y le damos la orden capturar repetitivamente o modo *REPETITIVE RUN* (figura G.5). Notar en la figura que elegimos el trigger cuyo *Trigger Condition Name* es "Aperiodic".

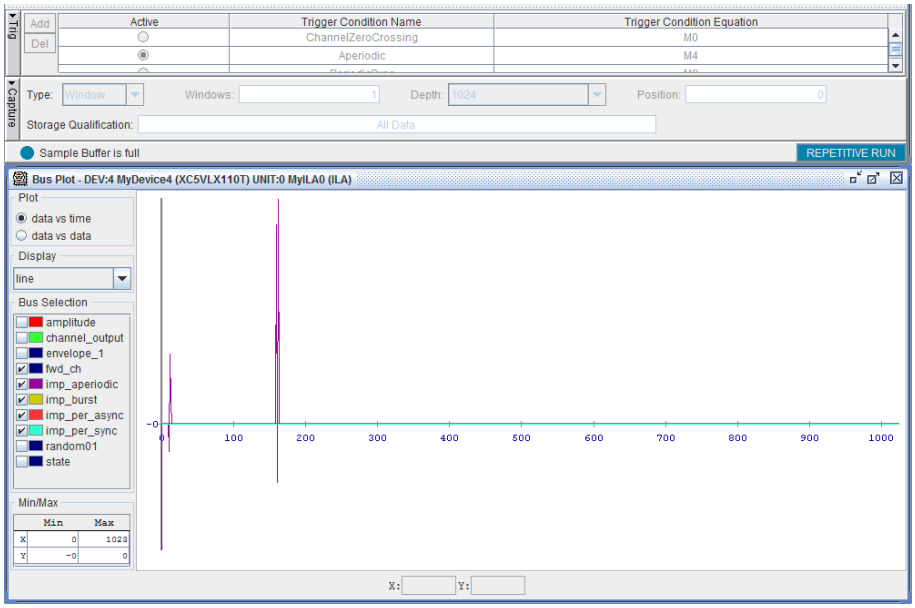


FIGURA G.5: *Chipsocpe Pro Analyzer*. Captura de señal de ruido impulsivo aperiódico asincrónico

Después cargamos la configuración correspondiente al ruido impulsivo en ráfagas y configuramos el trigger para que se dispare al ver una ráfaga (figura G.6). Notar en la figura que elegimos el trigger cuyo *Trigger Condition Name* es "Burst".

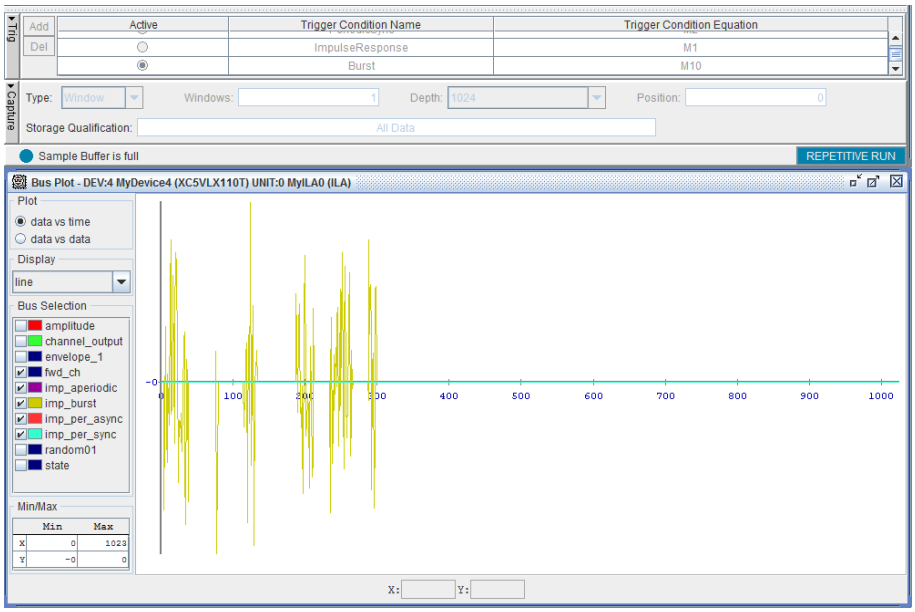


FIGURA G.6: *Chipsocpe Pro Analyzer*. Captura de señal de ruido impulsivo en ráfagas

Luego cargamos un escenario de ruido periódico asincrónico de 400 KHz con un ancho de impulso de 0,2 μs, calculado para 80 Msps. En este caso pusimos un trigger forzado de disparo repetitivo, ya que siempre se van a observar impulsos en un intervalo de 1024 muestras.

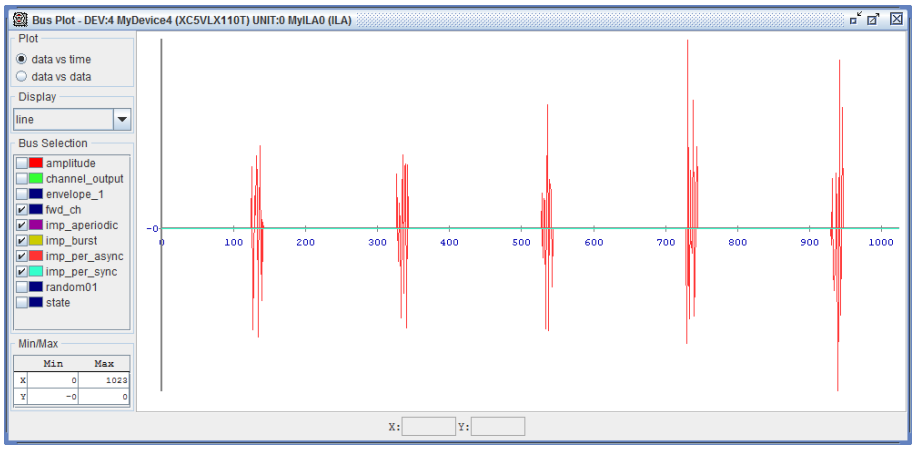


FIGURA G.7: *Chipsocpe Pro Analyzer*. Captura de señal de ruido impulsivo periódico asincrónico

Después configuramos un escenario de ruido periódico sincrónico de 120 Hz con un ancho de impulso de 10 μs. En este caso es necesario establecer un trigger porque ocurre un impulso cada 666 mil muestras. Este ruido se observa en la figura G.8. Notar en la figura que elegimos el trigger cuyo *Trigger Condition Name* es "PeriodicSync".

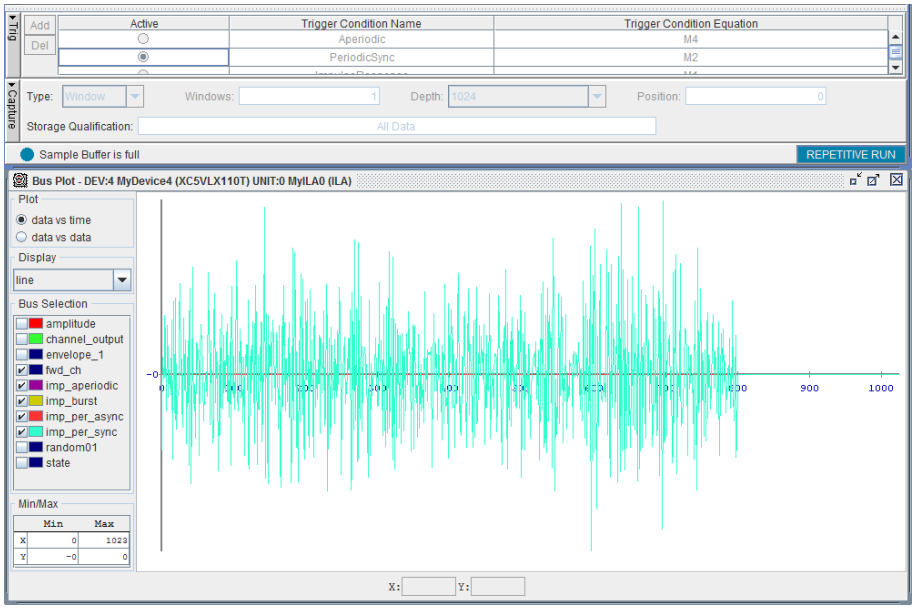


FIGURA G.8: *Chipsocpe Pro Analyzer*. Captura de señal de ruido impulsivo periódico sincrónico

Finalmente se cargó un escenario de ruido de banda angosta que se observa en la figura G.9. Aquí también utilizamos un trigger forzado de disparo repetitivo.

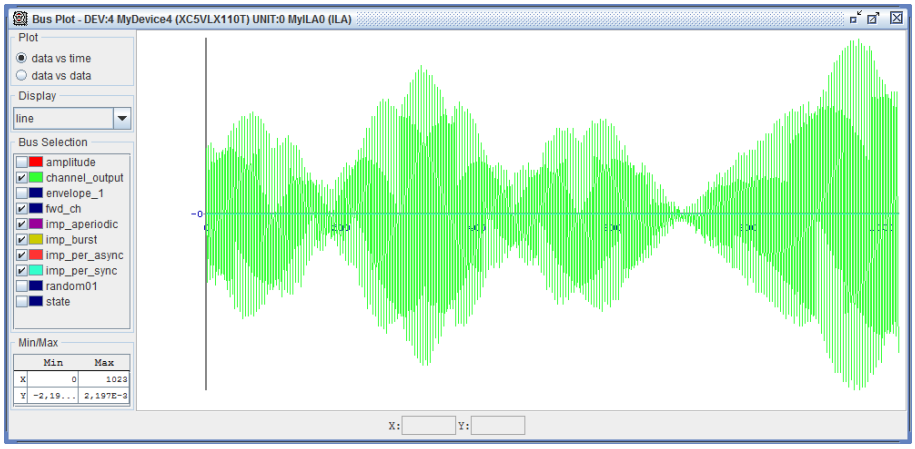


FIGURA G.9: *Chipsocpe Pro Analyzer*. Captura de señal de ruido de fondo coloreado y ruido de banda angosta

En esta última figura, la G.10, se observa la funcionalidad que tiene el software de mostrar los valores de los parámetros cargados. Allí vemos la matriz de probabilidades acumuladas de transición para el ruido impulsivo aperiódico asincrónico para el escenario *Strong*. Más abajo vemos estadísticas de impulsos tras ejecutar el comando *show_aperiodic_imp_stats*, el primer valor es la cantidad de impulsos y el segundo es la cantidad de ciclos de clock o muestras (1 muestra/clock)

```
imp_ctr / cycle_timestamp:  2050755 / 732731084
```

Su correspondencia con el valor teórico[2] es muy buena. Hay que tener en cuenta que se utilizaron 732 millones de muestras, cifra que parece muy grande para una simulación pero que sólo representa algunos segundos en el FPGA. La motivación principal de la introducción de estos contadores fue la de tener un método de verificación del correcto funcionamiento de los ruidos impulsivos además de las simulaciones. Si bien las simulaciones son un instrumento suficiente[3] para la verificación, la cantidad de tiempo simulado para hacer estimaciones de eventos de muy baja probabilidad tiene que ser grande y eso las hace impracticables en herramientas como *System Generator* en *Simulink*.

[2] Ver tabla 7.3.

[3] Xilinx nos especifica que las simulaciones son *bit-accurate* y *cycle-accurate* [46, p. 17].

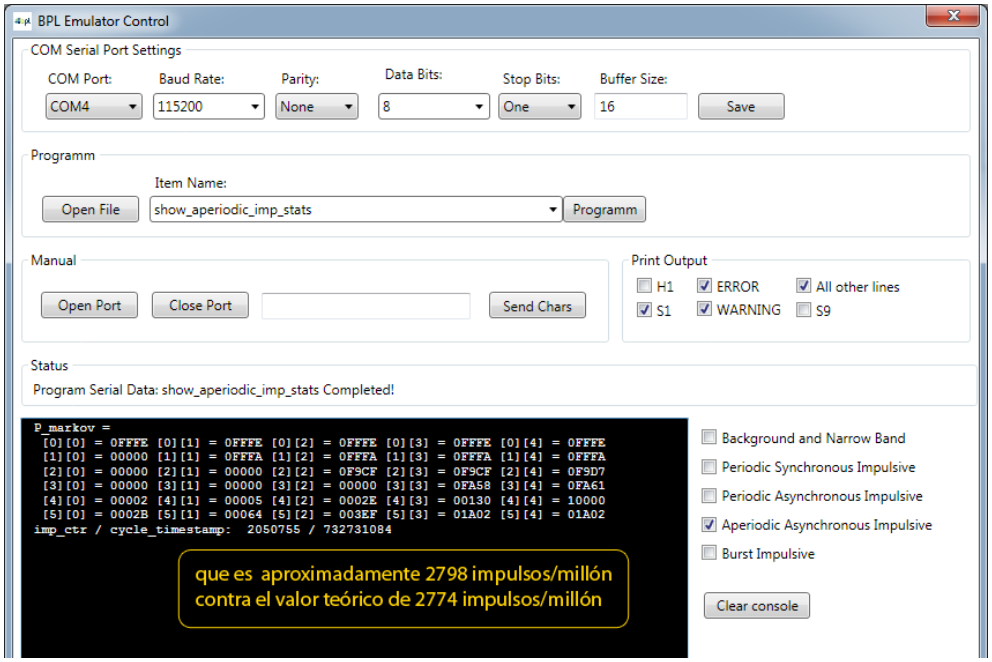


FIGURA G.10: Verificación a través de contadores de estadísticas para el ruido impulsivo aperiódico asincrónico

Bibliografía

[1] Open PLC Research Alliance. Theoretical postulation of PLC channel model. Archivo: OP_WP1_DeliverableD4.doc, March 2005.

[2] Open PLC Research Alliance. Pathloss as a function of frequency, distance and network topology for various LV and MV European powerline networks. Archivo: OP_WP1_D5_v0.9.doc, April 2005.

[3] Open PLC Research Alliance. Design and implementation of the channel emulator. Archivo: OP2_WP1_D7.2 Channel Emulator Design and Implementation 1.2.1.2-3_v0.3.doc, June 2008.

[4] Altera. *Stratix Device Handbook: Implementing High Performance DSP functions in Stratix & Stratix GX devices*, volume 2. Altera, September 2004.

[5] D. Benyoucef. A New Statistical Model of the Noise Power Density Spectrum for Powerline Communication. *Proceedings of the 7th International Symposium on Power-Line Communications and its Applications (ISPLC)*, pages 136–141, 2003.

[6] E. Boutillon, J.L. Danger, and A. Ghazel. Design of high speed AWGN communication channel emulator. *Analog Integrated Circuits. Signal Proceedings*, 34 (2):133–142, 2003.

[7] G.E.P. Box and M.E. Muller. A note on the generation of random normal deviates. *The Annals of Mathematical Statistics*, 29(2):610–611, 1958.

[8] R. Broadridge. Power-line modems and networks. *IEE 4th International Conference on Metering Applications and Tariffs for Electricity Supply IEE Conf. Publ.*, pages 294–296, 1984.

[9] F. J. Cañete, L. Díez, J. A. Cortés, and J. T. Entrambasaguas. Broadband Modelling of Indoor Power-Line Channels. *IEEE Transactions on Consumer Electronics*, 48(1):175–183, February 2002.

[10] F. J. Cañete, J. A. Cortés, L. Díez, and J. T. Entrambasaguas. Modeling and Evaluation of the Indoor Power-Line Transmission Medium. *IEEE Transactions on Consumer Electronics*, 41(4):41–47, April 2003.

[11] M.H.L. Chan and R.W. Donaldson. Amplitude, width, and interarrival distributions for noise impulses on intrabuilding power line communications. *IEEE Transactions on Electromagnetic Compatibility*, 31(3):320–323, 1989.

[12] H. Chaouche, F. Gauthier, A. Zeddam, M. Tlich, and M. Machmoum. Time Domain Modeling of Powerline Impulsive Noise at Its Source. *Journal of Electromagnetic Analysis and Applications*, 3:359–367, September 2011.

[13] K. Dostert. *Powerline Communications*. Prentice Hall, 2001.

[14] K. Dostert. Telecommunications over the power distribution grid - possibilities and limitations. *Proceedings of the International Symposium on Power-Line Communications and Its Applications (ISPLC)*, pages 1–9, 2nd-4th April 1997.

[15] Donald E. Knuth. *Seminumerical algorithms*, volume 2. Addison-Wesley Publishing Company, 1969.

[16] EICHHOFF. Blocking filters to enable multimaster deployment. OPERA Deliverable D1.2, April 2008.

[17] B. Ezio and P. Torino. Coding and Modulation for a Horrible Channel. *IEEE Communications Magazine*, pages 92–98, 2003.

[18] Bruce D. Fritchman. A Binary Channel Characterization Using Partitioned Markov Chains. *IEEE Transactions on Information Theory*, 13(2):221–227, April 1967.

[19] S. Galli. PLC Standardization Progress and Some PHY Considerations. `http://cms.comsoc.org/SiteGen/Uploads/Public/Docs_ISPLC_2009_/keynotes/ISPLC09_Keynote_Web_New.pdf`, 2009.

[20] A. Ghazel, E. Boutillon, J.L. Danger, G. Gulak, and H. Laamari. Design and performance analysis of a high speed awgn communication channel emulator. *Communications, Computers and signal Processing. IEEE Pacific Rim Conference on*, 2:374–377, 2001.

[21] O. G. Hooijen. On the channel capacity of the residential power circuit used as a digital communications medium. *IEEE Communications Letters*, 2(10), October 1998.

[22] H. Hrasnica and R. Lehnert. Powerline communications in telecommunication access area (Powerline communications im tk-zugangsbereich). *VDE World Microtechnologies Congress (MICRO.tec2000)*, September 2000.

[23] ITU-T. Applications of ITU-T G.9960, ITU-T G.9961 transceivers for Smart Grid applications: Advanced metering infrastructure, energy management in the home and electric vehicles, June 2010.

[24] D.U. Lee, W. Luk, J.D. Villasenor, and P.Y.K. Cheung. A gaussian noise generator for hardware-based simulations. *IEEE Transactions on Computers*, 53 (12):1523–1534, October 2004.

[25] G. Marsaglia. The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness. `http://www.stat.fsu.edu/pub/diehard/`, January 2012.

[26] P. Martin. An Analysis of Random Number Generators for a Hardware Implementation of Genetic Programming using FPGAs and Handel-C. Technical Report CSM-358, January 2002.

[27] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[28] A. V. Oppenheim, R. W. Schafer, and J. R. Buck. *Discrete-time Signal Processing*. Prentice Hall, New Jersey, 2nd edition, 1998.

[29] K. K. Parhi. Algorithm Transformation Techniques for Concurrent Processors. *Proceedings of the IEEE*, 77(12):1879–1895, December 1989.

[30] H. Philipps. Modelling of powerline communication channels. *International Symposium on Power Line Communications and Its Applications (ISPLC)*, pages 14–21, March 1999.

[31] H. Philipps. *Hausinterne Stromversorgungsnetze als Übertragungswege für hochratige digitale Signale*. Shaker Verlag, Aachen, March 2002.

[32] M. C. Pike. Algorithm 267: random normal deviate [G5]. *Commun. ACM*, 8:606, October 1965. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/365628.365649. URL `http://doi.acm.org/10.1145/365628.365649`.

[33] J. G. Proakis. *Digital Communications*. Mc Graw Hill, 4th edition, 2000.

[34] S. Ramo, J. R. Whinnery, and T. Van Duzer. *Fields and Waves in Communication Electronics*. John Wiley & Sons, 3rd edition, 1994.

[35] K. Ravinder, R. Ashish, S. Hardev, and M. Jagjit. Design and Implementation of High Speed IIR and FIR Filter using Pipelining. *International Journal of Computer Theory and Engineering*, 3(2):292–295, April 2011.

[36] S. Sancha, F. J. Cañéte, L. Díez, and J. T. Entrambasaguas. A Channel Simulator for Indoor Power-line Communications. *Proceedings of the IEEE International Symposium on Power Line Communications and Its Applications (ISPLC)*, pages 104–109, 2007.

[37] A. Schwager. *Powerline Communications: Significant Technologies to become Ready for Integration*. PhD thesis, Duisburg-Essen, 2010.

[38] A. Skrzypczak, P. Siohan, and J.P. Javaudin. Application of the OFDM/OQAM Modulation to Power Line Communications. *International Symposium on Power Line Communications and Its Applications (ISPLC)*, pages 71–76, Jun 2007.

[39] D. Thomas, W. Luk, P. H. W. Leong, and J. D. Villasenor. Gaussian Random Number Generators. *ACM Computing Surveys*, 4(39), 2007.

[40] D. Umehara, H. Nishiyori, and Y. Morihiro. Performance Evaluation of CMFB Transmultiplexer for Broadband Power Line Communications under Narrowband Interference. *International Symposium on Power Line Communications and Its Applications (ISPLC)*, pages 50–55, October 2006.

[41] S. A. White. Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review. *IEEE ASSP Magazine*, 1989.

[42] Wikipedia. High-voltage direct current. `http://en.wikipedia.org/wiki/High-voltage_direct_current`, December 2011.

[43] Xilinx. Efficient Shift Registers, LFSR Counters, and Long Pseudorandom Sequence Generator. Application Note. `http://www.xilinx.com/support/documentation/application_notes/xapp052.pdf`, July 1996.

[44] Xilinx. Additive White Gaussian Noise(AWGN) Core v1.0. `http://www.xilinx.com/support/documentation/ip_documentation/awgn.pdf`, October 2002.

[45] Xilinx. LogiCORE IP Fast Fourier Transform 7.1 Product Specification. `http://www.xilinx.com/support/documentation/ip_documentation/xfft_ds260.pdf`, 2011.

[46] Xilinx. System Generator for DSP User Guide v13.2. `http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_2/sysgen_user.pdf`, July 2011.

[47] M. Zimmermann. *Energieverteilnetze als Zugangsmedium für Telekommunikationsdienste*. Shaker Verlag, Aachen, 2000.

[48] M. Zimmermann and K. Dostert. A Multi-Path Signal Propagation Model for the Power Line Channel in the High Frequency Range. *International Symposium on Power Line Communications and Its Applications (ISPLC)*, 1999.

[49] M. Zimmermann and K. Dostert. An Analysis of the Broadband Noise Scenario in Powerline Networks. *International Symposium on Power Line Communications and Its Applications (ISPLC)*, 2000.

[50] M. Zimmermann and K. Dostert. A Multipath Model for the Power Line Channel. *IEEE Transaction On Communications*, 50(4):553–559, April 2002.