UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

IMPLEMENTACIÓN EN FPGA DE UN MODULADOR DIGITAL GMSK

T E S I S

Que para obtener el título de

INGENIERO EN TELECOMUNICACIONES

P R E S E N T A:

MIGUEL ÁNGEL ALVARADO ZARAGOZA

DIRECTOR DE TESIS:
Dr. Esaú Vicente Vivas

Ciudad Universitaria, 2013

Jurado asignado:

Presidente:	Dr. Carlos Rivera Rivera
Secretario:	M.I. Ricardo Mota Marzano
Vocal:	Dr. Esaú Vicente Vivas
1er. Suplente:	Dra. Fátima Moumtadi
2do. Suplente:	Dr. José María Matías Maruri

Esta tesis se realizó en:
Instituto de Ingeniería, U.N.A.M.

Director de tesis:
Dr. Esaú Vicente Vivas

_____

Firma

A mis padres y hermanos.

Agradecimientos

En esta sección quiero agradecer:

- En primer lugar a Dios, a quien le debemos nuestra existencia.
- A mis padres: Verónica e Isidoro, por brindarme todo su apoyo pese a muchos sacrificios.
- A mis hermanos: Mayra Sarahí y Sergio, por apoyarme y confiar en mí pese a muchas peleas.
- A todos los demás miembros de mi familia: abuelos, tíos, primos, cuñados, sobrinos sin excepción; quienes me han acompañado en todo momento.
- A la Universidad Nacional Autónoma de México, mi alma máter.
- A la Facultad de Ingeniería.
- Al Instituto de Ingeniería.
- Al Programa Nacional de Becas (PRONABES), por el apoyo económico que me otorgó a lo largo de la carrera.
- A mis amigos, por brindarme su valiosa amistad, aunque en verdad son muchos para mencionarlos en estas líneas, pero hago mención especial a Mario Alberto Corona, Enrique Munguía, Rodolfo Alarcón, Ángel N. Rodriguez, Albert Zavala, Oscar Vázquez, Juan Andrés Pérez Celis.
- A mis compañeros y amigos de carrera, por los buenos y malos momentos que pasamos juntos; con mención especial a Liliana Reyes, Marcos J. Sánchez y José Gustavo Alonso.
- A mis compañeros de laboratorio, a quienes debo gran parte de mi formación profesional: C.Dr. Mario Alberto Mendoza, C.Dra. Paloma Pedrajas, M.I. Emilio Jiménez, M.I. Rodrigo Cordova, Ing. Alejandro Castilla, Ing. Eduardo Vizcaíno, Alberto Muñiz, Gustavo, Bernardo, Jorge y Pedro.
- A Dra. Fátima Moumtadi, Dr. Carlos Rivera, M.I. Ricardo Mota y Dr. José María Matías Maruri; por aceptar ser parte de mi jurado, por su amabilidad y comentarios, los cuales complementaron este trabajo.
- A mi asesor de tesis Dr. Esaú Vicente Vivas, por darme la oportunidad de pertenecer a su grupo de trabajo y sobre todo por la confianza puesta en mí.

Resumen

Este trabajo de tesis presenta la propuesta, el diseño y la implementación en FPGA de un modulador digital GMSK-IQ, el cual pretende ser parte de un sistema de radio definido por software para aplicaciones en satélites. Este desarrollo tendrá la capacidad de actualizarse mediante reconfiguración en vuelo y dar solución a problemas de masa, volumen y costo, factores importantes en los sistemas aeroespaciales.

La modulación GMSK es una modulación en frecuencia de fase continua y de alta eficiencia espectral, por lo que es muy popular en las comunicaciones móviles como GSM y en las comunicaciones aeroespaciales.

Por otra parte la computación reconfigurable va ganando terreno en soluciones que requieren más capacidad y velocidad de procesamiento, así como flexibilidad; ya que las plataformas reconfigurables permiten diseñar sistemas digitales utilizando el concepto de *"paralelismo"*, disminuyendo así el tiempo de ejecución; además dichas plataformas pueden ser reconfiguradas total o parcialmente cuando se requiera sin comprometer su desempeño.

Abstract

This thesis presents the proposal, design and FPGA implementation of a digital modulator GMSK-IQ, which aims to be part of a software defined radio (SDR) system for satellite applications. This development will have the capability to upgrade by in fligh reconfiguration and to solve problems of mass, volume and cost, major factors in aerospace systems.

GMSK modulation (Gaussian Minimum Shift Keying) is a phase continuos frecuency modulation (PCFM) and high spectral efficiency, so it is very popular in mobile communications such as GSM and aerospace communications.

Moreover reconfigurable computing is gaining ground in solutions that require more processing performance: capability and speed, as well as flexibility. Reconfigurable platforms allow you to design digital systems using the concept of *"paralell"*, reducing execution time; furthermore, such platforms may be totally or partially reconfigured as required without compromising its performance.

Índice general

Lista de acrónimos

ABEL	Advanced Boolean Expression Language
AHDL	Altera Hardware Description Language
AHPL	A Hardware Programming Language
ASIC	Application Specific Integrated Circuit
ASK	Amplitude Shift Keying
ASP	Analog Signal Processing
ADC	Analog-to-Digital Converter
BiCMOS	Bipolar CMOS
CDL	Computer Design Language
CLB	Configurable Logic Block
CMOS	Complementary Metal Oxide Semiconductor
CPLD	Complex PLD
CPM	Continuous Phase Modulation
DAC	Digital-to-Analog Converter
DDL	Data Description Language
DSP	Digital Signal Processing
EECMOS	Electrically Erasable CMOS
EPLD	Enhanced PLD
EPROM	Electrical Programmable PROM
ET-Rx	Estación Terrena Transmisora
ET-Tx	Estación Terrena Receptora
FM	Frequency Modulation
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
FSK	Frequency Shift Keying
GAL	Generic Array Logic
GMSK	Gaussian MSK
GSM	Global System for Mobile Communications
GPP	General Purpose Processor
HDL	Hardware Description Language
IC	Integrated Circuit
IQ	In phase & Quadrature
ISI	Inter Symbol Interference
IBM	International Business Machines
IEEE	Institute of Electrical and Electronics Engineers

IDL	Interactive Design Language
IOB	Input Output Block
INTELSAT	International Telecommunications Satellite Organization
ISPS	Instruction Set Processor Specification
LSB	Least Significant Bit
LSI	Large Scale Integration
LUT	Look-Up Table
MOS	Metal Oxide Semiconductor
MSB	Most Significant Bit
MSI	Medium Scale Integration
MSK	Minimum Shift Keying
NCO	Numeric Controlled Oscillator
NRZ	Non Return-to-Zero
OLMC	Output Logic Macro Cell
PCB	Printed Circuit Board
PDSP	Programmable Digital Signal Processors
PLA	Programmable Logic Array
PLD	Programmable Logic Device
PROM	Programmable ROM
PSK	Phase Shift Keying
PSM	Programmable Switch Matrix
RAM	Random Access Memory
RCA	Radio Corporation of America
RF	Radio Frequency
ROM	Read Only Memory
RTL	Register Transfer Level
SDR	Software Defined Radio
SOC	System On Chip
SR	Software Radio
SSI	Small Scale Integration
TI	Texas Instruments
TTL	Transistor Transistor Logic
TV	Televisión
VCO	Voltage Controlled Oscillator
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VIUF	VHDL International User's Forum
VLSI	Very Large Scale Integration
XNOR	Exclusive NOR

Lista de símbolos

α'_{k-1}	bit inferior al bit actual α'_k
α'_k	bit actual
α'_{k+1}	bit superior al bit actual α'_k
λ	Longitud de onda
ϕ	Ángulo
σ^2	Varianza de la función de densidad Gaussiana
θ	Fase
ω	Frecuencia angular
A	Amplitud
B	Ancho de banda
B_N	Ancho de banda del ruido
e	Función exponencial
f	Frecuencia
f_1	Frecuencia del símbolo 1
f_2	Frecuencia del símbolo 2
f_c	Frecuencia de la señal portadora
$g(t)$	Respuesta a un pulso rectangular del filtro Gaussiano
$g_{k_2}(t)$	Trayectoria Gaussiana correspondiente a $\alpha'_{k-1} = 0, \alpha'_k = 1, \alpha'_{k+1} = 0$
$g_{k_3}(t)$	Trayectoria Gaussiana correspondiente a $\alpha'_{k-1} = 0, \alpha'_k = 1, \alpha'_{k+1} = 1$
$g_{k_7}(t)$	Trayectoria Gaussiana correspondiente a $\alpha'_{k-1} = 1, \alpha'_k = 1, \alpha'_{k+1} = 1$
h	Índice de modulación
$h(t)$	Respuesta al impulso
$\|H(f)\|$	Respuesta en frecuencia
Hz	Hertz
$Q(t)$	Función Q
$\mathrm{erf}(t)$	Función de error
R	Tasa de bits
s	segundos
$s(t)$	Señal modulada
$s_1(t)$	Símbolo 1 modulado
$s_2(t)$	Símbolo 2 modulado
t	Tiempo
T_b	Periodo del bit

Lista de figuras

Lista de tablas

Capítulo 1

Introducción

Este trabajo de tesis fue realizado en el Instituto de Ingeniería de la UNAM (IIUNAM), en el grupo de desarrollo de tecnología aeroespacial, el cual posee una línea de investigación orientada al desarrollo de un modulador-demodulador digital GMSK con arquitectura reconfigurable, basado en tecnología FPGA para aplicaciones en satélites, tal desarrollo tendrá la capacidad de actualizarse y adaptarse a las condiciones exigidas tanto por su aplicación como por el ambiente espacial.

1.1. Motivación

Actualmente los sistemas satelitales son una opción necesaria para las comunicaciones a distancia, ya que permiten ofrecer servicios de telecomunicaciones a regiones y localidades geográficamente aisladas o de difícil acceso, en las cuales los sistemas de comunicaciones terrestres no ofrecen cobertura o su despliegue resulta económicamente inviable.

Por otra parte los satélites de comunicaciones han prestado servicio durante décadas, evolucionando constantemente en tecnología y aplicaciones. En tal periodo, estos sistemas satelitales han demostrado su versatilidad, multiplicando los servicios que proporcionan. Sin embargo, sus desarrollos están llegando a restricciones y dificultades de índole técnica que deben considerarse con gran cuidado en su planeación y operación, como son la saturación de la órbita geoestacionaria, la aparente insuficiencia del espectro radioeléctrico y la interferencia entre ellos. Por lo tanto en esta tesis se pretende dar solución a algunos de estos problemas con la implementación en FPGA de un modulador GMSK[1], el cual será capaz de actualizarse mediante reconfiguración en vuelo y dar solución a problemas de masa, volumen y costo, factores importantes en los sistemas aeroespaciales.

1.2. Objetivo

Proponer, diseñar, implementar y validar en FPGA un modulador digital GMSK funcional y eficiente.

[1]Esquema de modulación muy popular en las comunicaciones inalámbricas debido a su eficiencia espectral y a la robustez de su señal en ambientes hostiles.

1.3. Generalidades

1.3.1. Sistema de comunicaciones

El objetivo esencial de los sistemas de comunicaciones es la transmisión de información de un punto de origen o *fuente* hacia un punto *destino* por medio de un canal físico. Un sistema de comunicaciones se define como aquel que mediante el empleo de técnicas y dispositivos adecuados realizan el transporte de información entre dos o varios puntos a través de un medio de transmisión.

Los sistemas de comunicaciones, figura 1.1, están compuestos por tres partes principales: *transmisor, canal o medio de transmisión y receptor:*

a) Transmisor: Es el encargado de modificar la señal de información, de tal manera que pueda ser adecuada para su transmisión. Esta adaptación implica procesos como filtrado, codificación, modulación, amplificación, etc.

b) Medio de transmisión: Es aquél a través del cual viaja la información del transmisor al receptor.

c) Receptor: Es el encargado de recibir la señal modulada y de realizar las operaciones inversas del transmisor como son: amplificación, filtrado, demodulación, decodificación, etc.; con la finalidad de obtener la información.

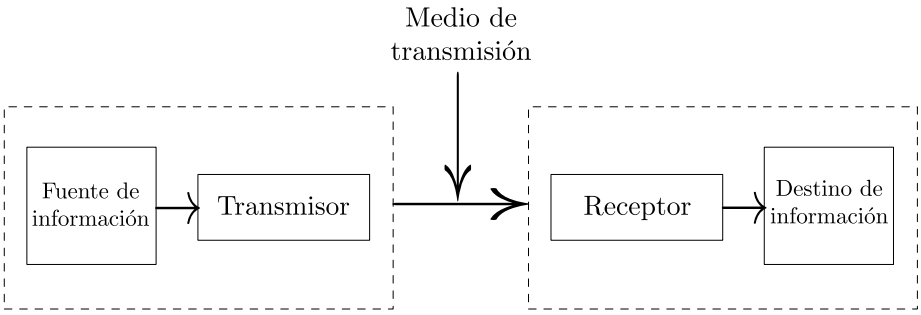


Figura 1.1: Diagrama a bloques simplificado de un sistema de comunicaciones.

1.3.2. Comunicaciones vía satélite

Un sistema de comunicaciones vía satélite está conformado por un satélite artificial que orbita la Tierra, el cual recibe señales de una estación terrena transmisora (ET-Tx), las procesa y las envía de regreso a Tierra para su recepción en una o más estaciones terrenas receptoras (ET-Rx), figura 1.2.

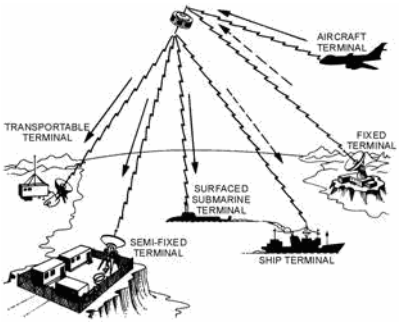


Figura 1.2: Sistema de comunicaciones vía satélite.[2]

1.3.2.1. Elementos de un sistema de comunicaciones por satélite

Un sistema de comunicaciones vía satélite esta compuesto por un *segmento espacial* y un *segmento terrestre*:

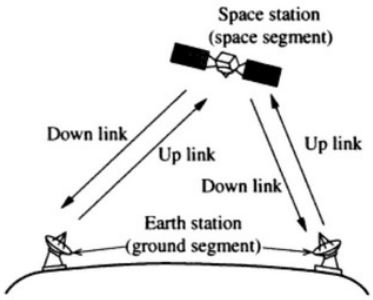


Figura 1.3: Diagrama conceptual de un sistema de comunicaciones vía satélite.

Segmento espacial

El segmento espacial esta constituido por el satélite o constelaciones de satélites que orbitan en el espacio exterior.

El *satélite* se define como un aparato fabricado por el hombre y lanzado al espacio para *girar de forma útil* alrededor de la Tierra o algún otro cuerpo celeste. El satélite está conformado por diversos subsistemas, los cuales se nombran y se describen a continuación:

- Estructura: Sus principales funciones son mantener a los demás subsistemas en una posición fija y proveer resistencia mecánica al sistema durante el lanzamiento.
- Potencia: Sus principales funciones son la de almacenar, regular y proveer energía eléctrica a todos los subsistemas. Está integrado por baterías, paneles solares, reguladores de voltaje, cargadores de baterías, etc.
- Control de posición y estabilización: Su función es la de mantener al satélite en alguna posición específica y darle estabilidad. Este subsistema puede ser activo (por ruedas inerciales, por propulsión química, etc.) o pasivo (por giro, por gravedad, etc) y está compuesto por sensores de navegación inercial y actuadores activos o pasivos.

[2] http://www.rfcafe.com/references/electrical/NEETS%20Modules/NEETS-Module-17-4-1-4-10.htm

- Control térmico; Su función principal es mantener a los subsistemas del satélite en un rango de temperatura operacional. Este subsistema puede ser pasivo (películas reflejantes) o activos (resistencias y refrigerantes).
- Telemetría y comando: Su función principal es conocer el estado del satélite y comandarlo. Está compuesto por sensores y sistema de comunicaciones.
- Computadora de vuelo: Su función principal es coordinar las diversas acciones entre los subsistemas. Está compuesto por microprocesadores, bus y memoria principalmente.
- Carga útil (Payload): Es el subsistema que define la aplicación o misión del satélite.

Segmento terrestre

El segmento terrestre está integrado por las distintas estaciones terrenas destinadas en la recepción y transmisión de señales a través del satélite o satélites en órbita. Existen tres tipos de estaciones terrenas:

- Fijas: Estas estaciones están diseñadas para mantenerse fijas en un lugar, es decir, permanecen inmóviles durante la comunicación con el satélite.
- Transportables: Estas estaciones están diseñadas para ser movibles, pero una vez ubicada en un lugar se deben mantener fijas para realizar la comunicación vía satélite.
- Móviles: Estas estaciones están diseñadas para comunicarse con el satélite en movimiento. Estas se definen de acuerdo a su localización en la superficie terrestre, tales como *móvil terrestre, móvil aeronáutico o móvil marítimo.*

1.3.3. Ventajas de los sistemas de comunicaciones por satélite

- Área de cobertura: Un satélite o constelación de satélites puede tener una cobertura regional, nacional, internacional, continental o global, la cual depende de la órbita en el que se encuentre. De esta manera, se pueden brindar diferentes servicios a zonas de difícil acceso.
- Alta capacidad: Los satélites de comunicaciones utilizan altas frecuencias, lo que representa el uso de grandes anchos de banda.

1.3.4. Radio definido por software (SDR)

Años atrás se habían llevado a cabo el diseño de sistemas de radiocomunicaciones de manera analógica; elementos de los sistemas de comunicaciones, tales como filtros, osciladores, moduladores, amplificadores, mezcladores y demás elementos del sistema fueron diseñados y construidos empleando un gran número de componentes electrónicos analógicos, definiendo estrictamente su comportamiento en hardware y limitándolos a una sola función específica. En 1992 Joseph Mitola introduce el término "radio definido por software" (SDR), para definir una tecnología en los sistemas de radio cuya funcionalidad se encuentra descrita y definida

por medio de software, minimizando la necesidad de realizar modificaciones en hardware durante actualizaciones tecnológicas, ofreciendo así una gran flexibilidad al sistema.

En [3] se define SDR como *"un radio que es substancialmente definido en software y cuyo comportamiento en capa física puede ser alterado de manera significativa haciendo cambios en su software"*. En términos generales, un SDR es un dispositivo de radio, el cual permite la manipulación de sus principales características (banda de frecuencia, ancho de banda, esquemas de modulación y codificación, entre otras.) a partir de software, obteniendo así sistemas multi-servicio, multi-estándar y multi-banda.

Figura 1.4: SDR: receptor CR-1 de la compañía CommRadio.

1.3.5. Arquitecturas de cómputo reconfigurables

Hoy en día, la tecnología no deja de crecer y evolucionar debido al ámbito de la modernidad y la automatización, día a día se observan dispositivos de almacenamiento de grandes capacidades, plataformas de computo más sofisticadas, sistemas de comunicaciones más rápidos, etc. Las tendencias actuales vienen produciendo incansables cambios progresivos en todas las metodologías de diseño, ya que estas deben contemplar cierta flexibilidad para aceptar modificaciones o actualizaciones en plazos cada vez más cortos. Por tal razón, hoy en día la "computación reconfigurable" va ganando terreno en las opciones de implementación, ya que es un nuevo paradigma de diseño enfocado a crear sistemas digitales, los cuales pueden ser reconfigurados total o parcialmente cuando se requiera, debido a que utilizan hardware cuya funcionalidad puede modificarse en tiempo de ejecución, otorgando al sistema una enorme flexibilidad sin comprometer su desempeño.

El propósito fundamental de las arquitecturas reconfigurables es adaptar libremente el hardware a las aplicaciones que se requieran.

1.3.6. Procesamiento digital de señales con dispositivos de lógica programable

En la actualidad, el procesamiento digital de señales (DSP) sobre dispositivos de lógica programable está tomando mayor relevancia en los sistemas de comunicaciones, esto se deriva del hecho de que el procesamiento no se limita únicamente en procesos en banda base, sino que se esta llevando hasta etapas de transmisión, donde las velocidades de muestreo y de procesamiento no pueden ser alcanzadas por PDSPs (procesador digital de señal programable) convencionales.

1.3.6.1. Procesamiento digital de señales

El procesamiento de señales ha sido usado para transformar o manipular señales analógicas o digitales a lo largo del tiempo. El DSP ha encontrado muchas aplicaciones que van desde las comunicaciones, voz, audio o procesamiento de señales biomédicas hasta en instrumentación y robótica. La tabla 1.1 muestra algunas aplicaciones del DSP.

Tabla 1.1: Aplicaciones con procesamiento digital de señales (DSP)

Uso	Algoritmo DSP
Propósito General	Filtrado, convolución, detección, correlación, estimación espectral y transformada de Fourier.
Procesamiento de voz	Codificación y decodificación, encriptación y desencriptación, reconocimiento y síntesis de voz, identificación de voz, cancelación de eco, etc.
Procesamiento de audio	Codificación y decodificación en alta definición (hi-fi), cancelación de ruido, ecualización de audio, mezclado de audio, etc.
Procesamiento de imágenes	Compresión y descompresión, reconocimiento de imágenes, mejoramiento de imágenes, etc.
Sistemas informáticos	Buzón de voz, fax, modéms, telefonía celular, moduladores y demoduladores, encriptación y desencriptación de datos, comunicaciones digitales, radio, televisión, comunicación inalámbrica, etc.
Control	Ingeniería de control, control de vibración, monitores de sistemas de potencia, robótica, etc.
Instrumentación	Generación de formas de onda, instrumentación científica, radar, sonar, etc.

1.3.6.2. Ventajas del procesamiento digital de señales

El DSP se ha convertido en una técnica madura y ha reemplazado a los sistemas que utilizan procesamiento analógico de señales (ASP) en muchas aplicaciones debido a diversas ventajas, las cuales se mencionan a continuación.

- Los sistemas de DSP permiten realizar, de forma económica, tareas que serían difíciles o incluso imposibles de realizar empleando sistemas electrónicos analógicos, como la Transformada de Fourier Rápida (FFT).
- Insensibilidad ante variación de condiciones ambientales como la temperatura. La eficacia en el funcionamiento de los componentes electrónicos depende fuertemente de su temperatura.

- Reprogramabilidad o reconfigurabilidad. Debido a que los sistemas de DSP están basados en procesadores programables o dispositivos reconfigurables, estos poseen la capacidad de programarse o reconfigurarse para realizar diferentes tareas a lo largo de su vida útil, en contraste, un sistema analógico requiere reconstruirse total o parcialmente si desea realizar una tarea diferente a la que se construyo.
- Tamaño: Un sistema DSP se puede albergar en un solo chip (procesadores o dispositivos lógicos programables), sin embargo, un sistema analógico está constituido por varias piezas de hardware como capacitores, inductores, resistores, etc., los cuales hacen que el sistema crezca en tamaño dependiendo de su complejidad.

1.3.6.3. Limitaciones del procesamiento digital de señales

A pesar de las grandes ventajas que presenta el DSP con respecto al ASP, el procesamiento digital presenta limitaciones que deben ser consideras en el diseño de sistemas, tales como:

- Rango dinámico limitado. La amplitud del rango dinámico disponible será fijado por el número de bits empleados para representar la muestra, dando origen a fenómenos de saturación o de truncado. Por lo que mientras más bits tenga la muestra, mayor será la precisión en los cálculos posteriores y serán menos los errores generados.
- Ancho de banda limitado por la frecuencia de muestreo. De acuerdo al teorema de Nyquist o del muestreo, el cual establece que la frecuencia mínima de muestreo debe ser mínimo 2 veces la frecuencia de la señal, evitando así pérdidas de información y logrando resultados aceptables.
- Error debido al proceso de cuantización.

  El proceso de cuantización consiste en representar una muestra analógica por el entero más próximo de acuerdo a una escala definida por un número finito de niveles. Este proceso genera:

  - ⋆ Un error de cuantización debido a la diferencia entre el valor real y el valor muestreado de la señal. Cuanto mayor sea el número de bits utilizado para representar la muestra, menor será este error.
  - ⋆ Una degradación de la señal como consecuencia de la pérdida de la información inherente a la representación de la señal analógica mediante muestras digitales.

Capítulo 2

Dispositivos lógicos programables

2.1. Introducción

Hoy en día, el nivel de integración alcanzado en el desarrollo de la microelectrónica ha hecho posible la implantación de sistemas electrónicos complejos en un circuito integrado SOC (Sistemas en Chip), mejorando día a día en velocidad, confiabilidad, consumo de potencia y sobre todo en área de diseño.

El proceso de miniaturización de los sistemas electrónicos comenzó con la interconexión de componentes (resistores, capacitores, bobinas, etc.) colocados en un chasis reducido y separados a una distancia mínima. Posteriormente se diseñaron y construyeron los primeras placas de circuitos impresos (PCB), un medio para sostener mecánicamente y conectar eléctricamente componentes electrónicos, a través de rutas o pistas de material conductor, grabadas en hojas de cobre laminadas sobre un sustrato no conductor.

Más tarde, en 1946 y 1947 se construyó el transistor de difusión planar, permitiendo en 1960 la fabricación del primer circuito integrado monolítico, integrando ciento de transistores, diodos, resistores, capacitores y más elementos sobre una oblea de silicio.

En la actualidad, la integración de sistemas sobre circuitos integrados (ICs) se supera día a día con el surgimiento de nuevas tecnologías de fabricación, consiguiendo desarrollar sistemas cada vez más complejos. Sin embargo, el tiempo de desarrollo de nuevos productos es muy largo, por lo que solamente se aplica exclusivamente para componentes con alto volumen de producción.

En la década de los sesenta, Hon & Sequin y Conway & Mead introdujeron el concepto de "lógica programable" como una forma más rápida y directa de construir aplicaciones sobre un circuito integrado, independizando el proceso de fabricación del proceso de diseño.

2.2. Dispositivos lógicos programables (PLD)

Un dispositivo lógico programable (PLD) es un componente electrónico usado para construir un "circuito digital reconfigurable", el cual a diferencia de una compuerta lógica (función única), tiene una infinidad de funciones a lo largo de su vida útil. Un PLD debe ser reconfigurado en su hardware de acuerdo a la función que se desea realizar.

Los PLDs son una atractiva opción para la integración de sistemas electrónicos en un solo IC, ya que además de reducir espacio, estos dispositivos pueden ser personalizados desde el exterior mediante diversas técnicas de programación. El diseño de aplicaciones se basa en bibliotecas y mecanismos de mapeado de funciones, mientras que su implementación solo requiere una fase de programación del dispositivo, la cual se realiza en unos segundos.

En la actualidad los PLDs son usados para reemplazar circuitos de pequeña escala de integración (SSI), de mediana escala de integración (MSI), e incluso a circuitos de muy alta escala de integración (VLSI), ya que reducen de manera significativa los costos en el diseño de aplicaciones. Los PLDs se clasifican por su arquitectura interna, es decir la distribución física de sus componentes internos, los cuales definen las características del dispositivo, Tabla 2.1.

Tabla 2.1: Dispositivos lógicos programables

Dispositivo	Descripción
PROM	Memoria programable de solo lectura
PLA	Arreglo lógico programable
PAL	Lógica de arreglos programables
GAL	Arreglo lógico genérico
CPLD	Dispositivo lógico programable complejo
FPGA	Arreglos de compuertas programables en campo

Estructura interna de un PLD

Algunos dispositivos de lógica programable están compuestos por arreglos o matrices que pueden ser fijos o programables como PROM, PLA, PAL y GAL, mientras que los CPLD y FPGA están estructurados mediante bloques reconfigurables y celdas lógicas de alta densidad, respectivamente.

La arquitectura básica de un PLD está formada por un arreglo de compuertas AND y OR conectadas en la entrada y salida del dispositivo.

a) Arreglo AND. Está compuesto por varias compuertas lógicas AND interconectadas a un arreglo programable, el cual contiene fusibles en cada punto de intersección, como se muestra en la figura 2.1. En esencia, la programación del arreglo consiste en fundir o apagar los fusibles para descartar la combinaciones lógicas que no serán utilizadas. Las variables que serán usadas se conectan a las entradas de la compuerta lógica mediante el fusible intacto.

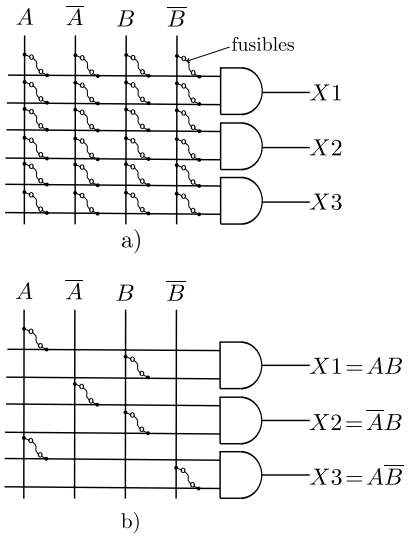


Figura 2.1: Arreglo de compuertas lógicas AND.

b) Arreglo OR. Está formado por un conjunto de compuertas lógicas OR interconectadas a un arreglo programable, el cual contiene fusibles en cada punto de intersección, como se muestra en la figura 2.2. Este tipo de arreglo es similar al arreglo AND, por lo que la forma de programación es la misma.

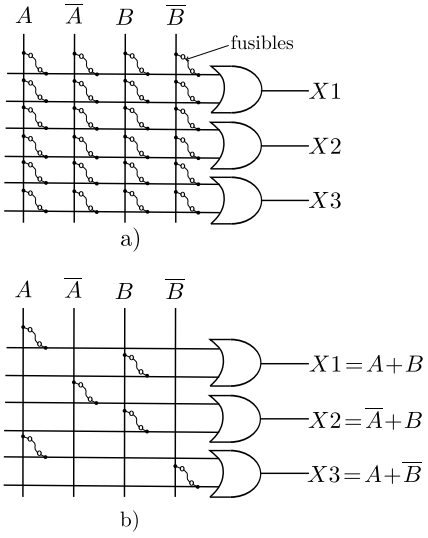


Figura 2.2: Arreglo de compuertas lógicas OR.

A continuación se describen las características de los diferentes PLD a partir de su estructura básica.

2.2.1. PROM

La memoria programable de solo lectura (PROM) está formada básicamente por un arreglo no programable de compuertas AND conectado a un arreglo programable OR, como se muestra en la figura 2.3. Debido a las limitaciones ocasionadas por el arreglo fijo de compuertas AND, la PROM no se utiliza como un dispositivo lógico sino como una memoria direccionable.

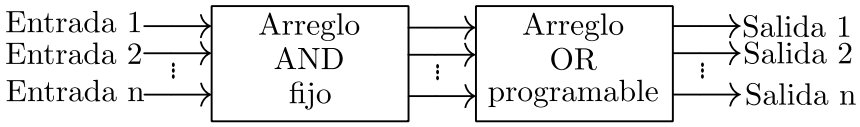


Figura 2.3: Estructura interna de la PROM.

2.2.2. PLA

La PLA se desarrolló para superar las limitaciones de la PROM. En esencia, el arreglo lógico programable está compuesto por un arreglo programable de compuertas AND conectado a un arreglo programable de compuertas OR, tal como se muestra en la figura 2.4. Este dispositivo también es llamado FPLA (Arreglo Lógico Programable en Campo), ya que es el usuario y no el fabricante quien lo programa.

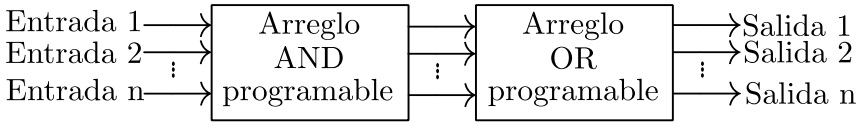


Figura 2.4: Estructura interna del PLA.

2.2.3. PAL

El PAL fue desarrollado para superar algunas limitaciones del PLA como lo son los retardos provocados por la utilización de dos arreglos programables y la complejidad del circuito. Este arreglo lógico programable se encuentra definido por un arreglo AND programable y un arreglo OR fijo, como se ilustra en la figura 2.5.

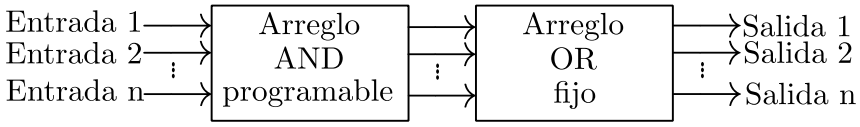


Figura 2.5: Estructura interna del PAL.

La arquitectura interna de un PAL está compuesta básicamente por circuitos combinacionales (compuertas lógicas AND,OR y NOT y buffers tri-estados) como muestra la figura 2.6, la cual corresponde a un PAL16L8. De forma general, cada término producto se programa en el arreglo AND, mientras que en el arreglo OR se realiza la suma lógica de los términos requeridos.

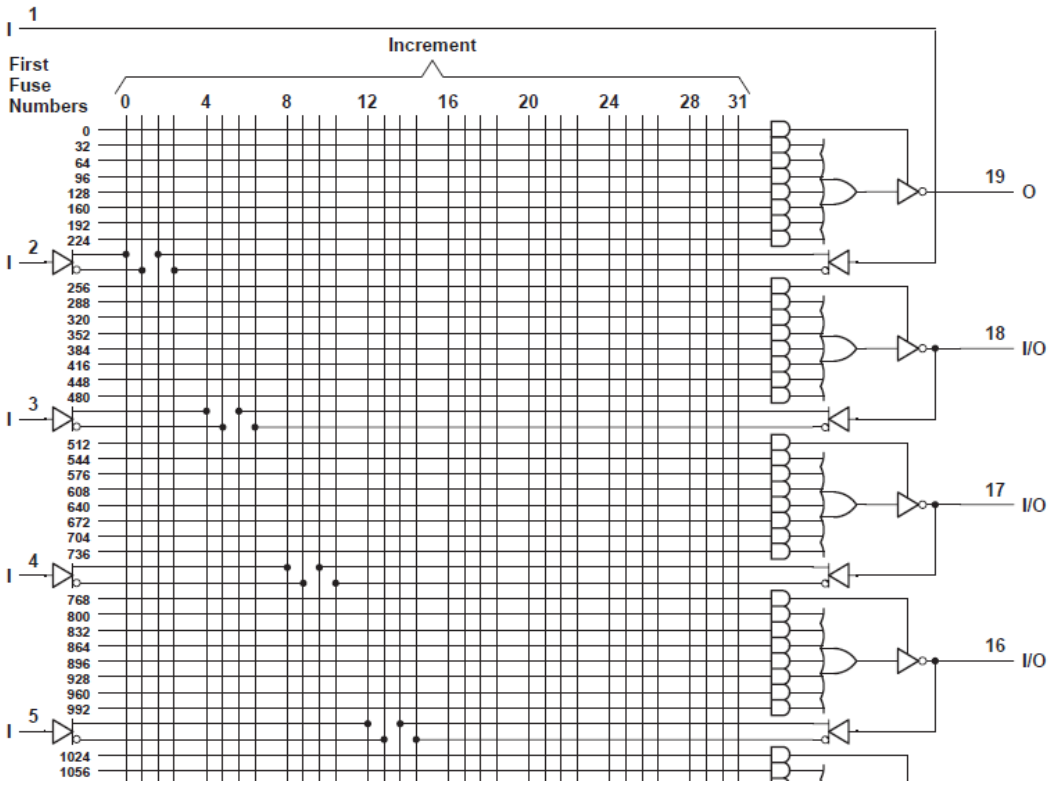


Figura 2.6: Estructura interna del PAL16L8.

2.2.4. GAL

El arreglo lógico genérico (GAL) es similar al PAL, ya que su estructura interna básica es similar (un arreglo AND programable y un arreglo OR fijo). Su principal diferencia radica en que el GAL es reprogramable y permite la configuración de sus salidas mediante programación.

Estos dispositivos usan una tecnología E^2CMOS (Electrically Erasable CMOS, CMOS borrable eléctricamente), en lugar de tecnología bipolar y fusibles, la cual permite programarlo infinidad de veces. Es decir, el GAL estás formado por celdas programables, las cuales están conectadas en los puntos de intersección de los arreglos programables (AND o OR) tal como los fusibles en los PLD anteriores, sin embargo en esta configuración solo se activan o desactivan las celdas programables de acuerdo a la función lógica requerida, otorgándole al dispositivo la característica de ser reprogramable.

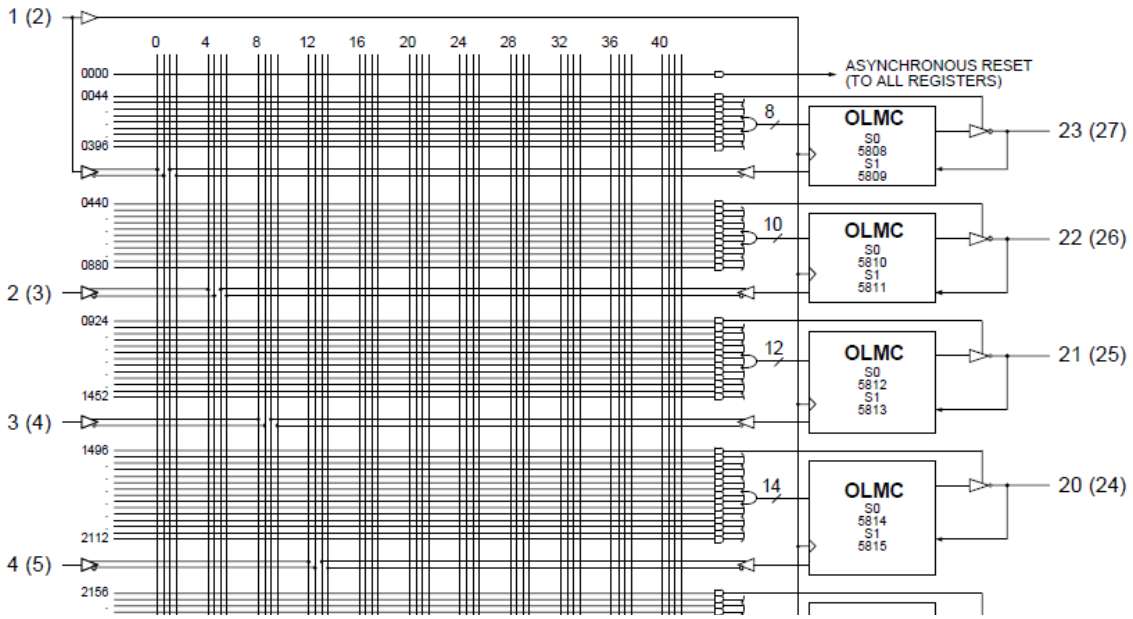


Figura 2.7: Estructura interna del GAL.

La arquitectura de un GAL se ve de manera de ejemplo en la figura 2.7, la cual corresponde a una GAL22V10. En la imagen se pueden observar los arreglos AND y OR colocados similarmente como en el PAL, sin embargo en la arquitectura del GAL sobresale el uso de *macroceldas lógicas de salida* (OLCM, por output logic macrocells). La macrocelda controla internamente el comportamiento que tendrán las entradas y salidas de las señales entre el arreglo lógico, y las terminales I/0 del dispositivo, figura 2.8. Bajo el control de una OLCM, cada salida puede designarse en forma general en:

a) Modo secuencial, la función lógica de salida pasa a través de un flip-flop.

b) Modo combinacional, la función lógica de salida es el resultado de una suma de productos Booleanos en donde no interviene ningún flip-flop.

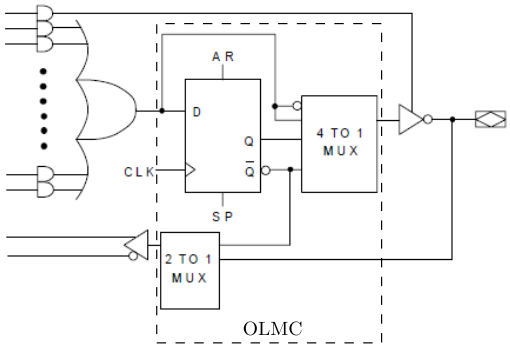


Figura 2.8: Estructura interna de una OLCM.

2.3. Dispositivos lógicos programables de alto nivel de integración

Los PLD de alto nivel de integración surgieron a partir de la necesidad de integrar sistemas electrónicos en un solo circuito integrado (SOC). Una característica adicional que poseen estos PLD además de la reducción de espacio y costo, son las grandes velocidades y frecuencias de operación que soportan, parámetro importante en sistemas informáticos y de comunicaciones de hoy en día. Estos dispositivos se han vuelto una opción muy atractiva para los desarrolladores de soluciones electrónicas, ya que les permite lanzar al mercado sus productos con mayor rapidez, dado al paradigma de diseño usado en estas plataformas reconfigurables, el cual les permite realizar modificaciones al sistema en un período de tiempo corto.

2.3.1. CPLD

Un dispositivo lógico programable complejo (CPLD) consiste en varios PLD agrupados en bloques en un solo CI. Se clasifican como de alto nivel de integración, ya que tienen una capacidad aproximada de 50 PLD sencillos, por tal razón, en ocasiones es llamado como EPLD (PLD mejorado).

La arquitectura del CPLD se basa en un arreglo de múltiples bloques lógicos (similares al GAL) conectados por medio de señales canalizadas desde la interconexión programable (PI). La unidad PI es la encargada de conectar los bloques lógicos y los bloques de entrada/salida del dispositivo sobre las redes apropiadas, figura 2.9.

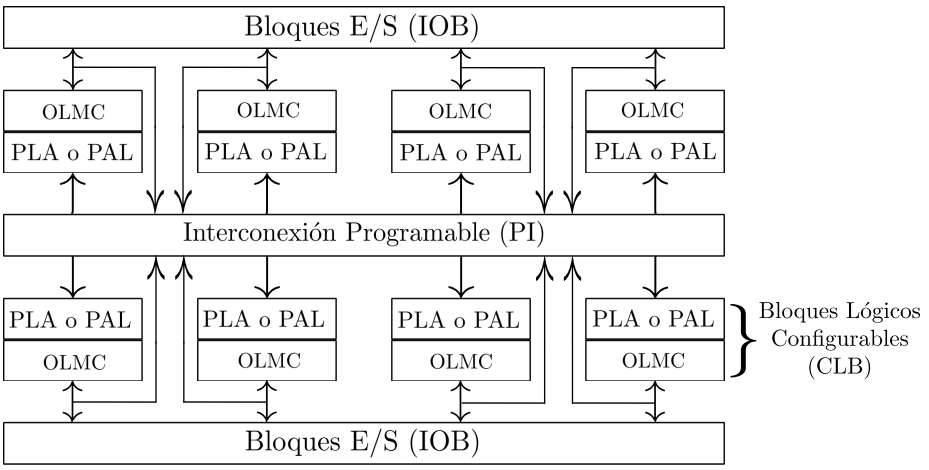


Figura 2.9: Estructura interna de un CPLD.

Los bloques lógicos son celdas generadoras de funciones y están constituidos por arreglos programables de términos producto (arreglo AND), esquemas de distribución/suma de términos productos provenientes del arreglo AND y por OMLC similares a las del GAL, figura 2.8. El tamaño de los bloques lógicos es un parámetro importante a considerarse, ya que determina cuánta lógica se puede implementar en el CPLD. Por otro lado, las celdas de entrada/salida en general no forman parte del bloque lógico, sino un módulo independiente.

2.3.2. FPGA

La arquitectura de los FPGA está formada por tres elementos configurables: bloques lógicos configurables (CLB), bloques de entrada/salida (IOB) y canales de comunicación, figura 2.10. De forma general, los CLB se comunican entre sí y con los IOB por medio de los canales de comunicación. A diferencia de los CPLD, la capacidad de los FPGA se establece por la cantidad de compuertas que posee.

Los bloques lógicos o celdas generadoras de funciones permiten la implementación de cualquier función booleana representada en la forma de suma de productos. Los CLB implementan el diseño lógico a través de bloques generadores de funciones o LUTs (Tablas de búsqueda), los cuales permiten almacenar la lógica requerida sobre una memoria interna.

En un dispositivo FPGA, los CLB están ordenados en arreglos de matrices programables (PSM), los cuales direccionan las salidas de un bloque a otro. Las terminales de entrada/salida del FPGA pueden estar conectadas directamente al PSM o CLB, o se pueden conectar a través de los canales de comunicación.

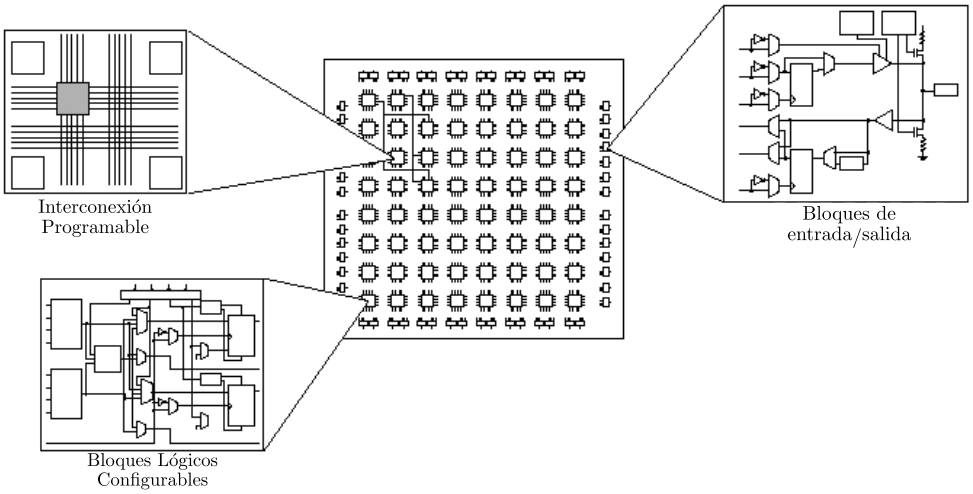


Figura 2.10: Estructura interna de un FPGA.

A pesar de que el FPGA y el CPLD poseen en su arquitectura bloques lógicos, la diferencia entre ambos dispositivos radica en el número de flip-flops contenidos en el componente, mientras que la arquitectura del FPGA posee gran cantidad de registros, la del CPLD mantiene una baja densidad. En la siguiente tabla se presentan otras diferencias entre ambas arquitecturas. [4]

Tabla 2.2: Diferencias entre FPGA y CPLD

Características	CPLD	FPGA
Densidad	Baja a media	Media a alta
Arquitectura	Arreglo de múltiples PLD	Arreglos de compuertas Registros + RAM
Frecuencia de operación	Superiores a los 200 MHz	Superiores a los 135 MHz
Aplicaciones	Contadores rápidos Lógica combinacional Máquinas de estado	Procesamiento digital de señales Arquitectura de computadoras Diseño con registros

Capítulo 3

Lógica programable y lenguaje descriptor de hardware (HDL)

Debido a la creciente necesidad de integrar un mayor número de aplicaciones en un solo circuito integrado, se desarrollaron nuevas herramientas de diseño que ayudan en la integración de sistemas complejos. En la década de los cincuenta aparecieron los lenguajes de descripción de hardware (HDL) como una opción de diseño y desarrollo de sistemas electrónicos. En los años setenta aparecen lenguajes como IDL de IBM, TI-HDL de Texas Instruments, ZEUS de General Electric, entre otros, los cuales estaban orientados al área industrial, por lo que no estaban disponibles fuera de la empresa que los manejaba, y los lenguajes orientados al ámbito universitario como el AHPL, DDL, CDL, ISPS, etc., los cuales carecían de soporte y mantenimiento. El desarrollo continuó y en la década de los ochenta surgieron lenguajes como VHDL, Verilog, ABEL 5.0 y AHDL, considerados ya como lenguajes descriptores de hardware, ya que permitieron describir un problema lógico a nivel funcional.

Estos lenguajes tienen la capacidad de describir sistemas lógicos en distintos niveles de abstracción (funcional, transferencia de registros (RTL) y lógico o a nivel de compuertas). A continuación se definen los niveles de abstracción desde el punto de vista de simulación y síntesis del circuito.

- Algorítmico: Se refiere a la relación funcional entre las entradas y salidas del circuito o sistema, sin hacer referencia a la realización final.
- Transferencia de registros (RTL): Consiste en la partición del sistema en bloques funcionales sin considerar a detalle la realización final de cada bloque.
- Lógico o a nivel de compuertas: El circuito o sistema se expresa en términos de ecuaciones lógicas.

3.1. VHDL, lenguaje de descripción de hardware

En la década de los ochenta, VHDL surge como un un lenguaje estándar, capaz de soportar el proceso de diseño de sistemas electrónicos complejos, con propiedades para reducir el tiempo de diseño y los recursos requeridos. Este lenguaje fue creado por el departamento de defensa de los Estados Unidos como parte del programa "Very High Speed Integrated Circuits" (VHSIC), a partir del cual se detectó la necesidad de contar con un medio estándar para facilitar el diseño y desarrollo de sistemas complejos en un solo chip. El programa VHSIC tenía como principal objetivo reducir el tiempo de diseño de circuitos integrados.

Después de varias versiones revisadas por el gobierno de los Estados Unidos, industrias y universidades, el Instituto de Ingenieros Eléctricos Electrónicos (IEEE, por Institute of Electrical and Electronics Engineers) publicó en diciembre de 1987 el estándar IEEE Std 1076-1987[1]. Tiempo después surge la necesidad de describir en VHDL todos los ASIC creados por el departamento de defensa de los Estados Unidos, por lo que en 1993 se adoptó el estándar adicional de VHDL IEEE 1164.

En la actualidad VHDL se considera como un estándar para la descripción, modelado de sistemas digitales. Este lenguaje presenta diversas características que lo hacen uno de los HDL más utilizados a nivel industrial.

3.1.1. Ventajas al diseñar sistemas digitales con VHDL

A continuación se enumeran algunas ventajas que representa desarrollar circuitos integrados con VHDL:

- Disponibilidad pública. VHDL es un estándar no sometido a patente o a alguna marca registrada, por lo que cualquier persona, empresa o institución puede utilizarlo sin restricciones. Además, el IEEE lo mantiene y documenta, por lo que se tiene la garantía de estabilidad y soporte.
- Independencia tecnológica de diseño. VHDL se diseñó para soportar diversas plataformas tecnológicas, como son los PLD, FPGA y los ASIC.
- Independencia en tecnología. VHDL se creó para que funcione de igual manera en circuitos diseñados con tecnología bipolar, MOS, CMOS, BICMOS, etc.
- Capacidad descriptiva en distintos niveles de abstracción. VHDL permite diseñar en cualquiera de los diferentes niveles de abstracción (algoritmico, RTL y lógico) y combinarlos entre sí para conseguir el diseño final.
- Independencia de proveedores. Debido a que VHDL es un lenguaje estándar, permite que las descripciones o modelos generados para un dispositivo de cierta compañía sean compatibles para cualquier otro dispositivo de compañía diferente.
- Reutilización del código. VHDL permite reutilizar código en diversos diseños, sin importar que hayan sido generados para una plataforma (FPGA, CPLD, ASIC, etc.) y una tecnología (MOS, CMOS, bipolar, etc.) en particular.

[1] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=26487&contentType=Standards

3.1.2. Desventajas al diseñar sistemas digitales con VHDL

A pesar de las grandes ventajas que posee VHDL, a continuación se mencionan algunas desventajas importantes que deben considerase al momento de diseñar un sistema digital.

- Posible pérdida de la libertad de diseño. Con la finalidad de facilitar el esfuerzo que requiere el diseño de aplicaciones, algunas compañías de dispositivos programables proveen herramientas de diseño con características adicionales al lenguaje (características no estándar), por lo que la libertad de diseño se ve limitada.
- Lenguaje difícil de aprender. Debido que VHDL fue desarrollado por un comité, en el cual se debió satisfacer las diversas opiniones de sus miembros, VHDL presenta alta complejidad, convirtiéndolo en un lenguaje difícil de aprender. [4]

3.1.3. VHDL en la actualidad

El proceso de estandarización de VHDL no se detuvo con la primera versión (VHDL'87), sino que ha continuado con la versión (VHDL'93) y se realizan periódicamente actualizaciones, mejoras y metodologías de uso. Entre estas actualizaciones esta la extensión analógica (1076.), la cual permite la utilización de un lenguaje único en todas las tareas de síntesis y simulación de sistemas electrónicos digitales, analógicos o mixtos.

La actividad en torno a VHDL es muy grande. Muchos países han creado grupos de trabajo cuya base de desarrollo es VHDL, además se realizan periódicamente conferencias, congresos y demás reuniones para presentar trabajos. [2]

[2] http://www.vhdl.org/

Capítulo 4

Modulación GMSK y principios teóricos

En este capítulo se definen los conceptos fundamentales de modulación, siguiendo la descripción de los esquemas de modulación FSK (modulación por desplazamiento de frecuencia) y MSK (modulación por desplazamiento mínimo de frecuencia), los cuales son clave para comprender la modulación GMSK (modulación por desplazamiento mínimo de frecuencia con prefiltrado Gaussiano), la cual se explica en la última sección del capítulo.

4.1. Modulación

El proceso de modulación consiste en la modificación de ciertas características de una señal denominada *portadora*, en función de las características de otra señal llamada *moduladora o de información*, con el objetivo de obtener una nueva señal más adecuada para la transmisión, a la cual se le denomina señal *modulada*.

Existen fundamentalmente dos tipos de modulación: analógica y digital. La modulación es analógica cuando se emplea como portadora una señal continua, la modulación es digital si la portadora es una señal discreta, en forma más precisa, la modulación digital implica una transformación digital por medio de la cual la señal en banda base se cambia de un lenguaje simbólico a otro. Si la señal en banda base es originalmente una función continua en el tiempo, esta previamente debe muestrearse y cuantificarse para ser digitalizada. Cualquiera que sea el tipo de modulación, este proceso es *reversible*, esto es, el mensaje se puede recuperar en el receptor mediante el proceso inverso o *demodulación*.

Algunas causas por las que una señal debe modularse son: la radiación electromagnética eficiente, realizar transmisión múltiple, reducir el ruido y las interferencias, empleo eficiente del espectro radioeléctrico, etc.

a) *Radiación electromagnética eficiente*. La radiación eficiente de una señal se consigue cuando la antena radiadora tiene una longitud de por lo menos 1/10 de la *longitud de onda* (λ) de la señal que se desea radiar. Bajo estas condiciones, la radiación directa de una señal de audio de 1000 Hz requeriría una antena cuyas dimensiones físicas estarían en el orden de los 30 km. Mediante la modulación, la señal de 1000 Hz se puede mover a una señal cuya frecuencia es más alta, reduciendo así la longitud física de la antena.

b) *Transmisión múltiple*. Mediante la modulación, el espectro de una señal de información se puede trasladar a diferentes posiciones en el dominio de la frecuencia. Por lo que los espectros de diferentes señales previamente moduladas pueden mezclarse y transmitirse por un mismo canal sin que se interfieran.

c) *Reducción del ruido y las interferencias*. Mediante determinados tipos de modulación se puede reducir considerablemente el ruido y las interferencias, sin embargo, para conseguir esto, el sistema requiere un ancho de banda de transmisión mucho mayor que el ancho de banda de la señal en banda base. La combinación apropiada entre el ancho de banda de transmisión y la reducción aceptable del ruido constituye uno de los aspectos más interesantes en el diseño de los sistemas de telecomunicaciones.

d) *Empleo eficiente del espectro radioeléctrico*. Mediante la modulación se puede hacer un uso racional y eficiente del espectro, es decir, este proceso permite convenientemente acomodar diferentes señales en las diferentes bandas de frecuencia del espectro, permitiendo controlar y administrar este recurso finito e invaluable. [6]

4.1.1. Modulación digital

La modulación digital es el proceso por el cual los símbolos digitales (bits), se transforman en formas de ondas compatibles con el medio de transmisión o a las características del canal. En el caso de la modulación en *banda base* esas formas de onda son pulsos, sin embargo cuando se necesite realizar la transmisión sobre un canal, la señal de información modula una señal sinusoidal denominada portadora, por lo que a este tipo de modulación es llamada pasa-banda.

Además, a la hora de elegir un esquema de modulación digital se debe contemplar alcanzar los siguientes objetivos: máxima velocidad de transmisión, mínima probabilidad de error, mínima potencia transmitida, mínimo ancho de banda del canal, máxima inmunidad a interferencias, mínima distorsión frente a la no linealidad de los amplificadores y por supuesto mínima complejidad en su construcción. Sin embargo, en la realidad no es posible obtener estas características juntas, por lo que cada sistema en particular debe evaluar los compromisos entre los diferentes parámetros para determinar cuáles son los que imponen las condiciones más restrictivas en el diseño.

En cuanto a los tipos de modulaciones digitales, existen varios que son la contraparte de las técnicas de modulación analógica, ya que modulan amplitud, fase o frecuencia de una portadora: ASK (modulación por desplazamiento de amplitud), FSK (modulación por desplazamiento de frecuencia), PSK (modulación por desplazamiento de fase), además se han inventado otras variantes con la finalidad de optimizar y mejorar dichas modulaciones digitales.

Por otra parte las modulaciones digitales presentan ventajas respecto a las modulaciones analógicas, tales como:

1. Mayor inmunidad frente a ruido e interferencias

2. Posibilidad de utilizar técnicas de procesamiento digital de señales como filtros adaptativos, ecualizadores de canal, etc., mediante dispositivos PDSP, FPGA y ASIC, los cuáles son más estables y de menor costo que los circuitos analógicos

3. Posibilidad de emplear técnicas de compresión de información

4. Posibilidad de emplear técnicas de encriptado [7]

4.1.2. Modulación pasa-banda

Como se mencionó antes, la modulación pasa-banda es el proceso por el cual una señal de información es convertida a una forma de onda sinusoidal en el caso de la modulación digital, y partiendo de que una señal sinusoidal tiene tres características fundamentales, las cuáles son: amplitud, frecuencia y fase. La modulación pasa-banda puede ser definida como el proceso mediante el cual la amplitud, frecuencia o fase de una señal portadora, o una combinación de estas características son variados conforme a la señal de información a transmitir. La forma general de una señal portadora es:

$$s(t) = A(t)\cos\phi(t) \tag{4.1}$$

donde $A(t)$ es la amplitud variable en el tiempo y $\phi(t)$ es el ángulo variable en el tiempo. Por lo que es conveniente escribir:

$$\phi(t) = \omega_0 t + \theta(t) \tag{4.2}$$

por lo tanto:

$$s(t) = A(t)\cos[\omega_0 t + \theta(t)] \tag{4.3}$$

donde ω_0 es la frecuencia en radianes de la portadora y $\phi(t)$ es su fase.

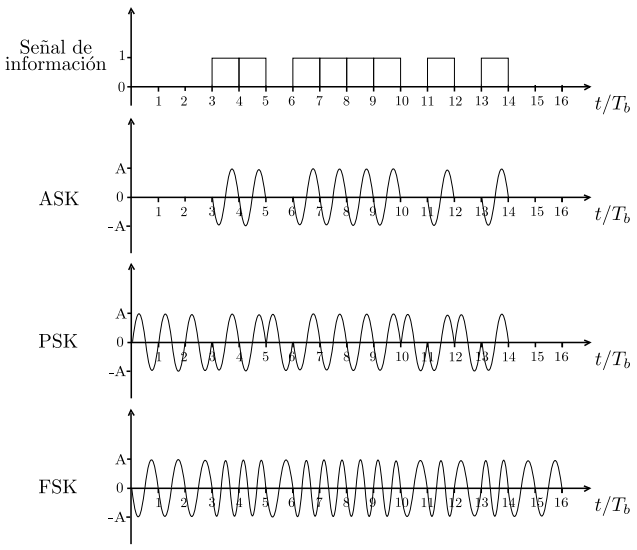


Figura 4.1: Esquemas básicos de modulación pasa-banda

4.2. Modulación FSK

La modulación FSK es similar a la modulación en frecuencia (FM)[1], por lo que FSK es un esquema de modulación en donde la señal de información (señal en banda base) modula a la señal portadora mediante cambios discretos en frecuencia.

En la modulación binaria FSK (BFSK o 2FSK), un bit de dato es representado por un tono con frecuencia f_1 y el otro bit de dato es representado por un tono con frecuencia f_2. Las frecuencias f_1 y f_2 son escogidas con tal que los símbolos $s_1(t)$ y $s_2(t)$ sean ortogonales entre sí, en el intervalo de señalización $T = T_b$, lo cual significa que f_1 y f_2 deben satisfacer la siguiente ecuación:

$$f_1 = \frac{2n+m}{4T}, \quad f_2 = \frac{2n-m}{4T} \quad \text{y} \quad f1 - f2 = \frac{m}{2T} \tag{4.4}$$

En donde n y m son números enteros.

La forma de onda de un símbolo en una modulación FSK está dada por:

$$s_i(t) = \sqrt{\frac{2E_b}{T_b}} \cos(2\pi f_i t), \quad \begin{cases} 0 \leq t < T_b \\ i = 1, 2 \end{cases} \tag{4.5}$$

Donde $E_b = E$ es la energía promedio del bit, f_1 y f_2 son las frecuencias portadoras que representan cada bit, y T_b es la duración del símbolo.

La frecuencia nominal de la portadora f_c se define como la media aritmética de las frecuencias de las señales portadoras de FSK, es decir:

$$f_c = \frac{f_1 + f_2}{2} = \frac{n}{2T} \tag{4.6}$$

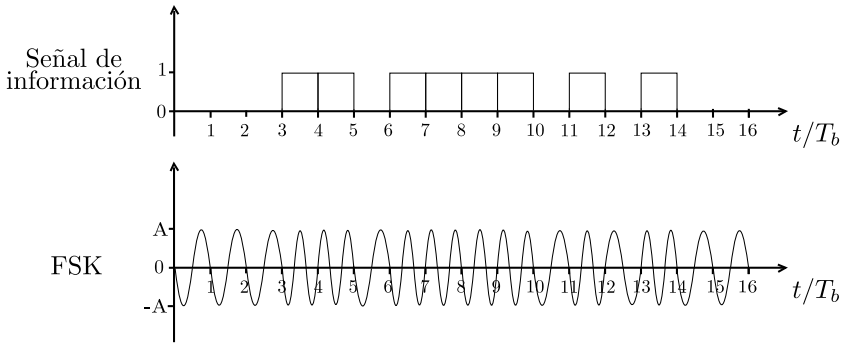


Figura 4.2: Modulación FSK

[1]FM- modulación analógica en frecuencia

4.3. Modulación MSK

La modulación MSK, También conocida como "Fast FSK", fue desarrollada a finales de 1950's. Sus características como: envolvente constante, espectro compacto y un buen rendimiento respecto al error son deseables en muchos sistemas de comunicaciones digitales. Su utilización va desde los sistemas de comunicación vía satélites, sistemas de posicionamiento y navegación, sistemas de comunicaciones inalámbricas, etc.

El esquema de modulación MSK puede ser visto como un caso especial de CPFSK (modulación por desplazamiento de frecuencia de fase continua), por lo que MSK es una modulación en frecuencia de fase continua, la cual puede ser expresada como:

$$s_i(t) = \sqrt{\frac{2E_b}{T_b}} \cos[(2\pi f_c t + \theta(t))], \tag{4.7}$$

Donde E_b es la energía promedio del bit, T_b es la duración del bit y $\theta(t)$ es la fase variable en el tiempo de la señal modulada partiendo de la fase de la portadora $2\pi f_c t$. La derivada respecto al tiempo de la fase $\theta(t)$ da lugar al cambio instantáneo de la frecuencia angular de CPFSK. Entonces, en un intervalo de tiempo dado, $\theta(t)$ se incrementa o decrementa linealmente dependiendo de la señal a transmitir, por lo que:

$$\theta(t) = \theta(0) \pm \frac{\pi h}{T_b} t, \quad 0 \leq t < T_b, \tag{4.8}$$

Donde $\theta(0)$ es el valor acumulado de la fase hasta $t = 0$ y $h = \Delta f T_b$ es el índice de modulación, el cual mide la desviación de frecuencia. En MSK $h = \frac{1}{2}$, se obtiene la mínima separación entre frecuencias ortogonales, razón por el cual es llamada MSK.

Extendiendo el resultado en (4.7) para cualquier instante de tiempo, la fase de la señal modulada puede ser determinada por:

$$\theta(t) = \theta(0) + \frac{\pi h}{T_b} \int_0^t b(u)\, du; \tag{4.9}$$

Donde $b(t) \in \pm 1$ es la información codificada en NRZ.[2]

[2]NRZ-(Non-Return-to-Zero). Es un código de línea utilizado en Telecomunicaciones para representar la información, en el cual el "-1" representa un "0" lógico y el "+1" representa un "1" lógico.

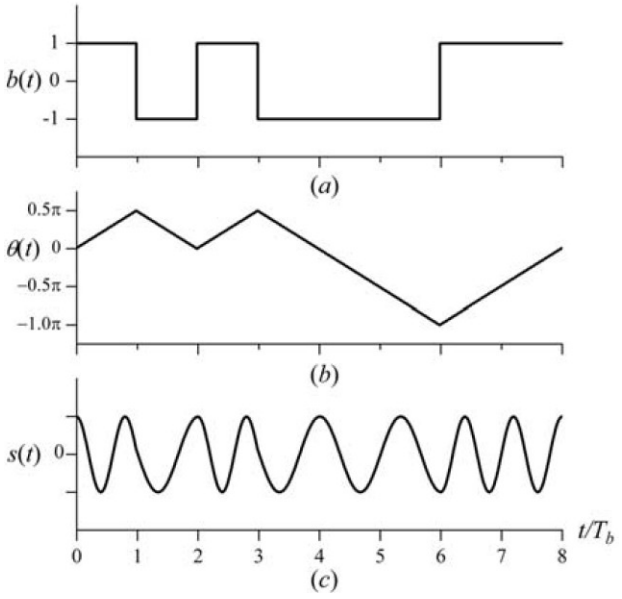


Figura 4.3: a) señal de información, b) señal de fase c) señal modulada en MSK.

4.4. Modulación GMSK

La modulación GMSK fue introducida por primera vez por Murota, Hirada y Kinoshita en 1981 como un esquema de modulación de envolvente constante con un alta eficiencia espectral para comunicaciones en los 900 MHz. En términos simples, GMSK es un esquema de modulación de fase continua (CPM) de respuesta parcial con $h = 0.5$, el cual se obtiene al filtrar la señal de información (pulsos rectangulares) utilizando un filtro paso bajos Gaussiano antes de ser modulada mediante MSK.

Como observamos anteriormente, una señal de CPFSK puede ser descrita mediante la ecuación (4.7), donde la fase $\theta(t)$ esta dada por (4.9). Combinando estas expresiones, podemos escribir una señal modulada en fase continua como:

$$\begin{aligned} s(t) &= \sqrt{\frac{2E_b}{T_b}}\cos[2\pi f_c t + \theta(t)] \\ &= \sqrt{\frac{2E_b}{T_b}}\cos[2\pi f_c t + \theta(0) + \frac{\pi h}{T_b}\int_0^t b(u)\,du] \\ &= \sqrt{\frac{2E_b}{T_b}}\cos[2\pi f_c t + \theta(0) + \frac{\pi h}{T_b}\int_{-\infty}^t b(u)\,du], \end{aligned} \tag{4.10}$$

Donde $\theta(t)$ es la variación de fase con respecto a la fase de la portadora. La forma de onda $b(t)$ es definida como:

$$b(t) = \sum_{k=-\infty}^{\infty} a_k g(t - kT_b), \tag{4.11}$$

Donde $a_k \in \pm 1$, de tal forma que $a_k = +1$ para un bit de dato "1" y $a_k = -1$ para un bit de dato "0". La función $g(t)$ corresponde a un pulso rectangular de amplitud $1/T_b$ y de duración T_b para MSK. Para GMSK, $g(t)$ es un pulso gaussiano dado por:

$$g(t) = \frac{1}{2T_b} \mathrm{Q}\left(\frac{2\pi BT_b}{\sqrt{\ln 2}}\left(\frac{t}{T_b} - 1\right)\right) - \frac{1}{2T_b} \mathrm{Q}\left(\frac{2\pi BT_b}{\sqrt{\ln 2}} \frac{t}{T_b}\right), \quad \text{para } 0 \leq BT_b < \infty \tag{4.12}$$

Donde $\mathrm{Q}(t)$ está dada por:

$$\mathrm{Q}(x) = \int_x^\infty \frac{1}{\sqrt{2\pi}} \exp(-\frac{y^2}{2})\, dy \tag{4.13}$$

Y T_b es la duración del bit y B es el ancho de banda a 3 dB del filtro paso bajo Gaussiano, el cual está relacionado con el ancho de banda del ruido B_N por:

$$\frac{B}{B_N} = 2\sqrt{\frac{\ln 2}{\pi}} = 0.93944 \tag{4.14}$$

Las expresiones (4.10), (4.11) y (4.12) pueden ser vistas como la representación matemática en síntesis directa [3] de la señal de GMSK.

El producto BT_b en (4.12) es el parámetro que gobierna el comportamiento del filtro Gaussiano. La magnitud de la respuesta en frecuencia de este filtro es también Gaussiano y está dado por:

$$|H(f)| = \exp\left[-\left(\frac{f}{B}\right)^2 \frac{\ln 2}{2}\right] \tag{4.15}$$

El pulso gaussiano en (4.12) es el resultado de la convolución entre un pulso rectangular de duración T_b y la respuesta al impulso de un filtro paso bajo Gaussiano, dado por:

$$h(t) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{t^2}{2\sigma^2}\right), \qquad \sigma^2 = \frac{\ln 2}{(2\pi B)^2} \tag{4.16}$$

La cual es una función de densidad Gaussiana de media cero y varianza σ^2. La figura 4.4 muestra el pulso gaussiano $g(t)$ para diferentes valores de BT_b. Note que cuando BT_b se incrementa, el pulso gaussiano tiende a la función de un pulso rectangular. Para $BT_b = \infty$ no existe filtrado, por lo que la modulación GMSK se convierte en una modulación MSK convencional. Note también que el pulso $g(t)$ en (4.12) es definido con un tiempo de duración infinito y es el resultado de un proceso de filtrado no-causal. Por lo tanto, para resolver este problema, en la practica el pulso es truncado y la respuesta al impulso es desplazada de manera arbitraria una cantidad suficiente para convertirse en un sistema causal.

[3] Síntesis Directa. Es un procedimiento mediante el cual se generan formas de onda partiendo de la expresión matemática que definen la expresión en el tiempo y de valores numéricos que definen los parámetros representativos de una señal.

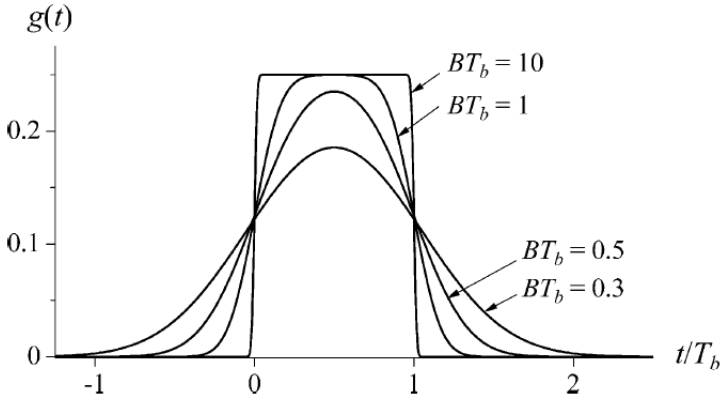


Figura 4.4: Respuesta a un pulso rectangular del filtro Gaussiano

El filtro Gaussiano da origen en GMSK a que las transiciones de fase sean suaves en comparación con las transiciones de fase que se tienen en MSK. Aunque este suavizado de fase resulta en una densidad espectral de potencia compacta, el modulador GMSK sufre de Interferencia intersimbólica (ISI) causada debido al hecho de que los pulsos $g(t)$ no están confinados dentro del intervalo del bit, es decir, GMSK es una señalización de respuesta parcial.Esta interferencia provoca una disminución en el rendimiento.

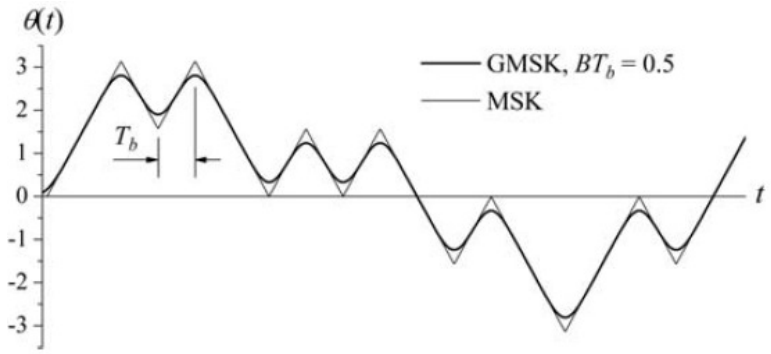


Figura 4.5: Comparación entre la evolución de fase de MSK y GMSK. Unidades en radianes

4.4.1. Densidad espectral de potencia (PSD) de GMSK

La figura 4.6 muestra la PSD en banda base estimada mediante simulaciones numérica para algunos valores de BT_b, en la cual se observa que a menor valor de BT_b, menor es el ancho de la señal; además se pueden observar las características espectrales de la modulación GMSK adoptada en GSM ($BT_b = 0.3$) [4].

[4] Este valor de BT_b compensa y equilibra la eficiencia espectral con la disminución del rendimiento debido al ISI

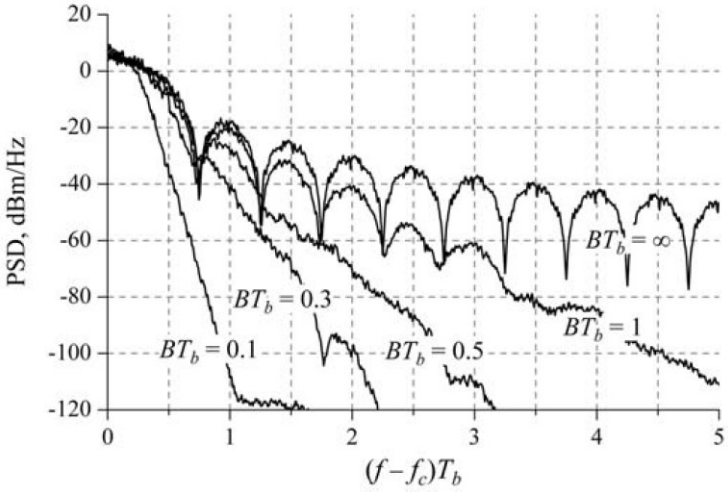


Figura 4.6: Eficiencia espectral de potencia (PSD) de GMSK estimada mediante simulaciones numéricas para diferentes BT_b.

4.4.2. Ventajas de la modulación GMSK

La modulación GMSK ofrece diversas ventajas en los sistemas de radiocomunicaciones. Algunas ventajas son:

* Mayor eficiencia espectral en comparación con otros tipos de modulación.
* Permite la utilización de amplificadores no lineales en su señal sin afectar la información que transporta, ya que esta no depende de variaciones en amplitud.
* Mayor inmunidad ante los efectos producidos por el ruido en el medio de transmisión, ya que la modulación no depende de variaciones en amplitud.
* Reducción de interferencias en canales vecinos, gracias al pre-filtrado Gaussiano.

4.4.3. Desventajas de la modulación GMSK

A pesar de que la modulación GMSK ha sido utilizado por largo tiempo debido a su eficiencia espectral, existen algunas desventajas consecuencia del ahorro de espectro, tales como:

* Se necesita mayor potencia en la transmisión, para lograr superar las señales ruidosas en el receptor dada la interferencia entre símbolo producida por el filtro Gaussiano.
* Surge la necesidad de la implementación de un receptor de mayor complejidad, ya que no existen cambios definidos de fase en la señal modulada, lo que dificultad la sincronización de datos.

4.4.4. Implementación de GMSK

Existen dos métodos para generar GMSK, uno es mediante modulación en frecuencia (FM) y el otro mediante modulación IQ (en fase-cuadratura). A continuación se describen ambos métodos, destacando sus ventajas y desventajas.

4.4.4.1. Implementación de GMSK mediante modulación en frecuencia

La implementación de GMSK mediante modulación en frecuencia, figura 4.7, está constituido básicamente por el filtro Gaussiano y un modulador en frecuencia basado en VCO (oscilador controlado por voltaje), cuyo índice de modulación debe ser exactamente 0.5. Aunque es fácil su implementación, esta arquitectura del modulador GMSK no es adecuada para una demodulación del tipo coherente, ya que el índice de modulación de un transmisor basado en VCO suele variar a lo largo del tiempo debido a la temperatura.

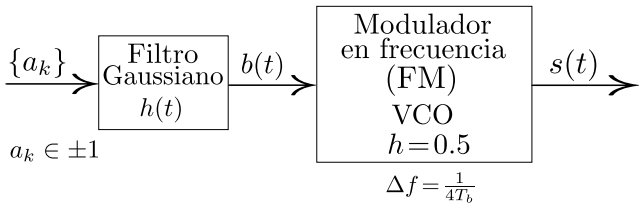


Figura 4.7: Diagrama a bloques del modulador GMSK implementado mediante un modulador en frecuencia (FM).

4.4.4.2. Implementación de GMSK mediante modulación IQ

Para expresar GMSK como una modulación IQ, se aplica la identidad trigonométrica: $\cos(a+b) = \cos(a)\cos(b) - \operatorname{sen}(a)\operatorname{sen}(b)$ a la ecuación (4.10), obteniendo:

$$s(t) = \sqrt{\frac{2E_b}{T_b}}\left\{ \cos(2\pi f_c t)\cos\left[\frac{\pi h}{T_b}\int_{-\infty}^{t} b(u)du\right] - \operatorname{sen}(2\pi f_c t)\operatorname{sen}\left[\frac{\pi h}{T_b}\int_{-\infty}^{t} b(u)du\right]\right\} \quad (4.17)$$

El diagrama a bloques del modulador GMSK IQ se muestra en la figura 4.8, con este tipo de implementación el índice de modulación se puede mantener exactamente en 0.5.

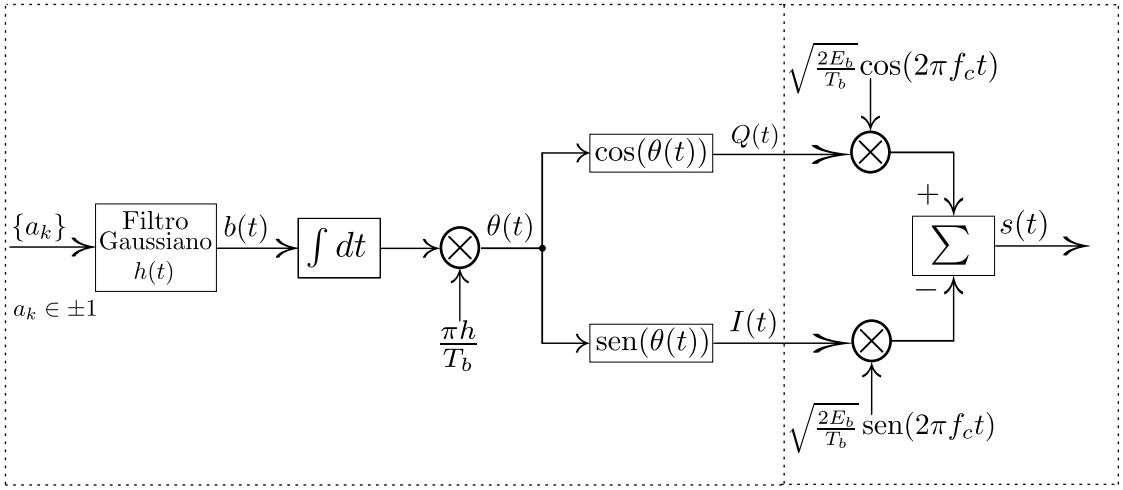


Figura 4.8: Diagrama de bloques del modulador GMSK implementado mediante un modulador en cuadratura (IQ).

A continuación se describe cada uno de los bloques que conforman el modulador.

a) Filtro Gaussiano

Dada una señal digital a_k codificada en NRZ (figura 4.9), que pasa a través de un filtro Gaussiano, cuya respuesta a un pulso rectangular está dada como:

$$g(t) = \frac{1}{2T_b}\,\mathrm{Q}\left(\frac{2\pi BT_b}{\sqrt{\ln 2}}\left(\frac{t}{T_b}-1\right)\right) - \frac{1}{2T_b}\,\mathrm{Q}\left(\frac{2\pi BT_b}{\sqrt{\ln 2}}\frac{t}{T_b}\right), \quad \text{para } 0 \le BT_b < \infty$$

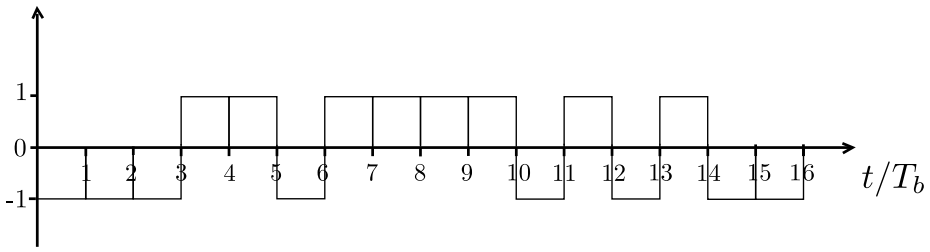


Figura 4.9: Señal digital a_k codificada en NRZ.

El efecto individual que produce el filtro sobre los pulsos se muestra en la figura 4.10, en la cual se observa la interferencia entre símbolo (ISI) ocasionada por la característica Gaussiana.

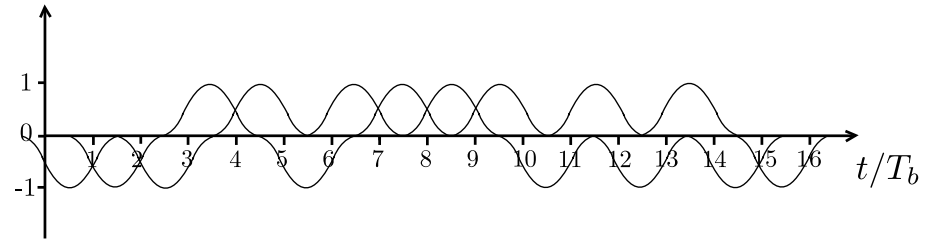


Figura 4.10: Filtrado Gaussiano, efecto individual.

Sumando la energía total de los pulsos filtrados, resulta la señal de la figura 4.11.

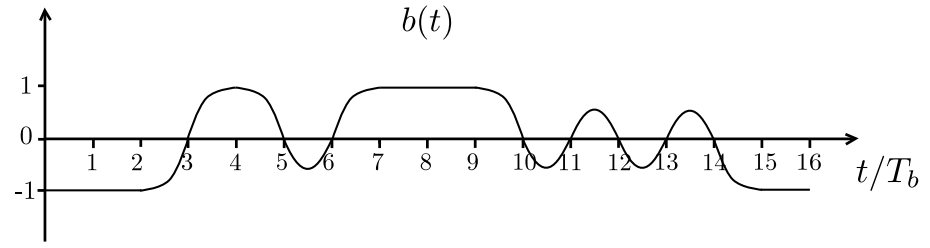


Figura 4.11: Filtrado Gaussiano, efecto sumado.

Para implementar fisicamente el filtro Gaussiano, su respuesta es escalada y truncada en el tiempo; condicionando la respuesta del k bit, en el intervalo $kT_b \le |t| \le (k+1)T_b$ a solo depender del bit de interés, α_k, y sus bits vecinos más cercanos, α_{k-1} (bit anterior al bit α_k) y α_{k+1} (bit superior al bit α_{m+1} y tomando en cuenta que:

$$\begin{aligned} \mathrm{Q}\left(\frac{2\pi BT_b}{\sqrt{\ln 2}}\right) &\cong 0 \\ \mathrm{Q}\left(-\frac{2\pi BT_b}{\sqrt{\ln 2}}\right) &\cong 1 \end{aligned} \tag{4.18}$$

Y sabiendo que la señal digital codificada en NRZ, la cual varía de -1 a 1 es equivalente a una señal compuesta por un tren de pulsos que varía de 0 a 2 menos una constante igual a 1, por lo tanto la respuesta del filtro gaussiano $g(t)$ se puede reescribir como:

$$\begin{aligned} g_k(t) &\equiv \sum_{i=k-1,k,k+1} (\alpha_i + 1)\left[\mathrm{Q}\left(\frac{2\pi BT_b}{\ln 2}\left(\frac{t}{T_b} - (i+1)\right)\right) - \mathrm{Q}\left(\frac{2\pi BT_b}{\ln 2}\left(\frac{t}{T_b} - i\right)\right)\right] - 1 \\ &\cong (\alpha_{k-1} + 1)\,\mathrm{Q}\left(\frac{2\pi BT_b}{\ln 2}\left(\frac{t}{T_b} - (k)\right)\right) \\ &+ (\alpha_k + 1)\left[\mathrm{Q}\left(\frac{2\pi BT_b}{\ln 2}\left(\frac{t}{T_b} - (k+1)\right)\right) - \mathrm{Q}\left(\frac{2\pi BT_b}{\ln 2}\left(\frac{t}{T_b} - k\right)\right)\right] \\ &+ (\alpha_{k+1} + 1)\left[1 - \mathrm{Q}\left(\frac{2\pi BT_b}{\ln 2}\left(\frac{t}{T_b} - (k+1)\right)\right)\right] - 1 \end{aligned} \tag{4.19}$$

Puesto que la función $\mathrm{Q}(x)$ se puede ser expresada en términos de la función de error "$\mathrm{erf}(x)$" como: $\mathrm{Q}(x) = (1/2)[1 + \mathrm{erf}(x/\sqrt{2})]$ y teniendo que, $\alpha_i' = (1/2)(\alpha_i + 1)$ y $\beta = \pi B\sqrt{2/\ln 2}$, la ecuación 4.19 puede ser reescrita como:

$$\begin{aligned} g_k(t) &\cong \alpha_{k-1}'[1 - \mathrm{erf}(\beta(t - kT_b))] \\ &\quad + \alpha_k'[\mathrm{erf}(\beta(t - kT_b)) - \mathrm{erf}(\beta(t - (k+1)T_b))] \\ &\quad + \alpha_{k+1}'[1 + \mathrm{erf}(\beta(t - (k+1)T_b))] - 1 \end{aligned} \tag{4.20}$$

Donde $\alpha_i' \in 0, 1$, ya que, $\alpha_i \in \pm 1$.

Por lo tanto existen 8 posibles formas de onda o trayectorias "g_k" correspondientes a la combinación de los bits $\alpha_{k-1}'\ \alpha_k'\ \alpha_{k+1}'$, los cuales caracterizan la respuesta del filtro en el intervalo del k bit. La tabla 4.1 muestra las posibles respuestas en el intervalo $0 \leq t \leq T_b$, asumiendo $k = 0$ por simplicidad, [13].

Tabla 4.1: Posibles respuestas o trayectorias del filtro Gaussiano en el intervalo $0 \leq t \leq T_b$

$\alpha'_{k-1}\alpha'_k\alpha'_{k+1}$	i	$g_{k_i}(t)$
000	0	-1
001	1	$\mathrm{erf}(\beta(t-T_b))$
010	2	$\mathrm{erf}(\beta t) - \mathrm{erf}(\beta(t-T_b)) - 1$
011	3	$\mathrm{erf}(\beta t)$
100	4	$-\mathrm{erf}(\beta t)$
101	5	$-\mathrm{erf}(\beta t) + \mathrm{erf}(\beta(t-T_b)) + 1$
110	6	$-\mathrm{erf}(\beta(t-T_b))$
111	7	1

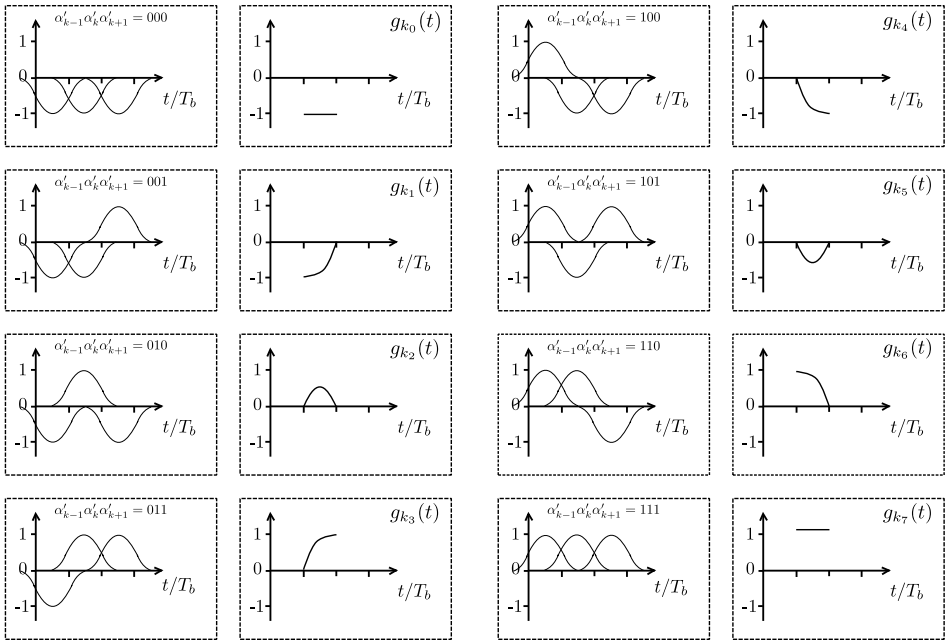


Figura 4.12: Respuestas o trayectorias del filtro Gaussiano de acuerdo a la excitación: $\alpha'_{k-1}\alpha'_k\alpha'_{k+1}$.

Se observa que existen tres trayectorias independientes: $g_{k_2}(t)$, $g_{k_3}(t)$ y $g_{k_7}(t)$, de las cuales se pueden generar las cinco trayectorias restantes mediante simples operaciones.

$$
\begin{aligned}
g_{k_0}(t) &= -g_{k_1}(t) \\
g_{k_1}(t) &= g_{k_3}(t) - g_{k_2}(t) - g_{k_7}(t) = g_{k_3}(t - T_b) \\
g_{k_4}(t) &= -g_{k_3}(t) \\
g_{k_5}(t) &= -g_{k_2}(t) \\
g_{k_6}(t) &= -g_{k_1}(t)
\end{aligned}
\tag{4.21}
$$

En la implementación física, las trayectorias $g_{k_2}(t)$, $g_{k_3}(t)$ y $g_{k_7}(t)$ son almacenadas en LUTs y mediante operaciones aritméticas se generan las cinco trayectorias restantes.

b) Integrador

Para asegurar que la respuesta del filtro Gaussiano produzca en un pulso un cambio de fase igual a $\pi/2$, se debe elegir una constante K que satisfaga la siguiente ecuación:

$$\int_0^{T_b} Kg(t)\,dt = \frac{\pi}{2} \tag{4.22}$$

Por lo tanto, la señal $b(t)$ se integra con respecto al tiempo de t a ∞ para obtener una señal $\theta(t)$ o señal de fase, figura 4.13.

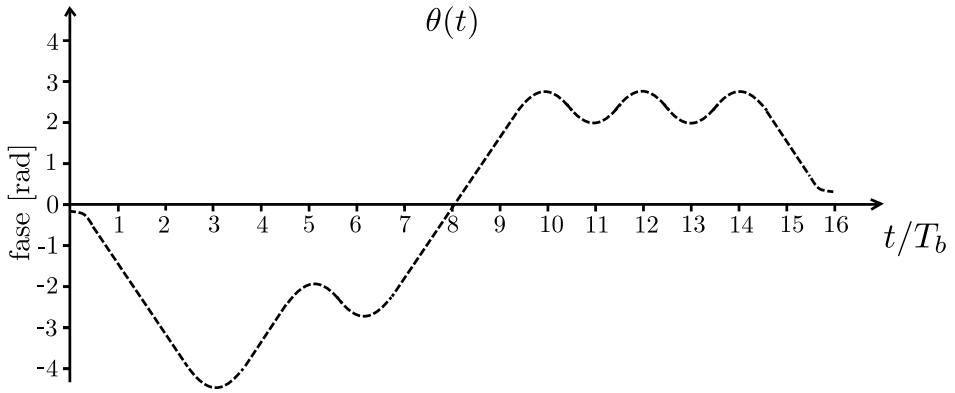


Figura 4.13: Señal de fase $\theta(t)$ resultante.

c) Procesador IQ en banda base

Para obtener las señales I (in phase) y Q (in quadrature) en banda base, se debe obtener el *seno* y *coseno* de la señal $\theta(t)$.

La señal $I(t)$ se obtiene con el $\cos(\theta(t))$ y la señal $Q(t)$ con el $\text{sen}(\theta(t))$, dichas señales están representadas gráficamente en la figura 4.14.

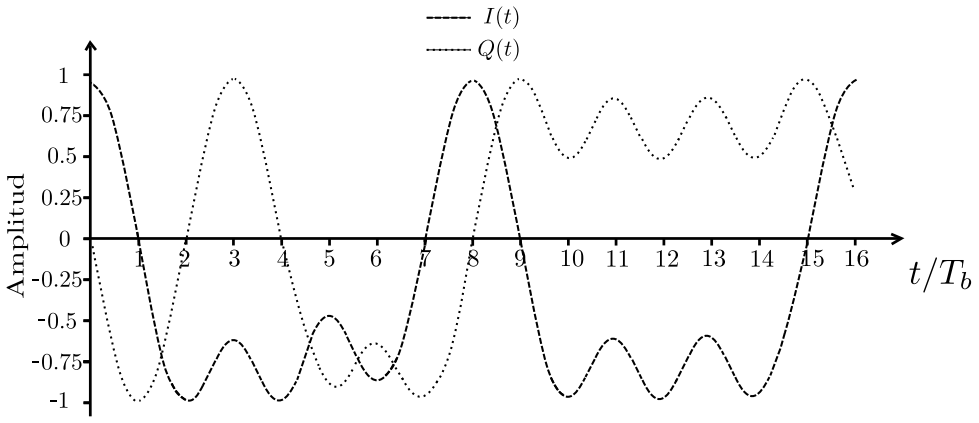


Figura 4.14: Señales $I(t)$ y $Q(t)$ resultantes.

c) Modulador IQ

Después de obtener las señales $I(t)$ y $Q(t)$ en banda base, estas deben ser moduladas de forma IQ, tal como lo describe la siguiente expresión:

$$s(t) = \cos(2\pi f_c t) \cdot Q(t) - \text{sen}(2\pi f_c t) \cdot I(t) \tag{4.23}$$

Donde f_c es la frecuencia de la portadora.

La señal $s(t)$ corresponde a la señal GMSK y está representada gráficamente en la figura 4.15, [15].

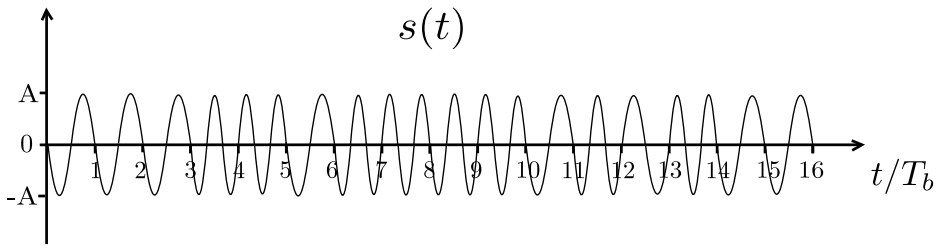


Figura 4.15: Señal modulada $s(t)$.

Capítulo 5

Implementación en FPGA del modulador digital GMSK

En este capítulo se describe la implementación en FPGA del modulador GMSK en su versión IQ, debido a las ventajas que ofrece comparado con el modulador GMSK-FM, las cuales se mencionaron en la subsubsección 4.4.4.2.

El flujo de diseño que hay que seguir para poder implementar sistemas digitales en FPGA se muestra en la figura 5.1

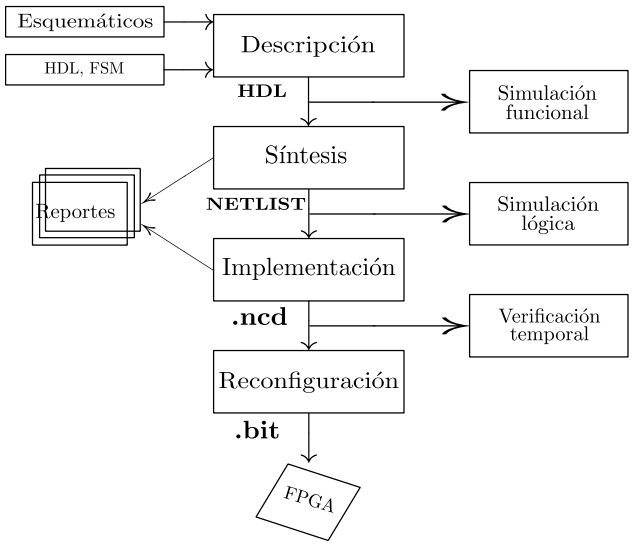


Figura 5.1: Flujo de diseño con FPGAs

Como se observa en la figura 5.1, el primer paso en el flujo de diseño es la descripción en hardware del sistema mediante un HDL. En esta tesis se utilizó VHDL debido a las ventajas descritas en la subsección 3.1.1.

5.1. Descripción en hardware del modulador GMSK

La descripción en hardware del modulador GMSK se realizó con la metodología de diseño "top-down"[1]. Un primer nivel de abstracción para el modulador GMSK se muestra en la figura 5.2, ilustrando el proceso de modulación mediante un sistema.

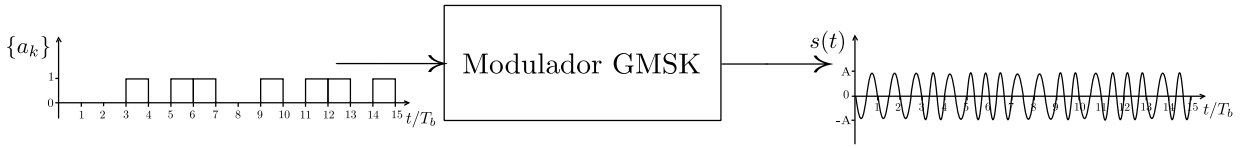


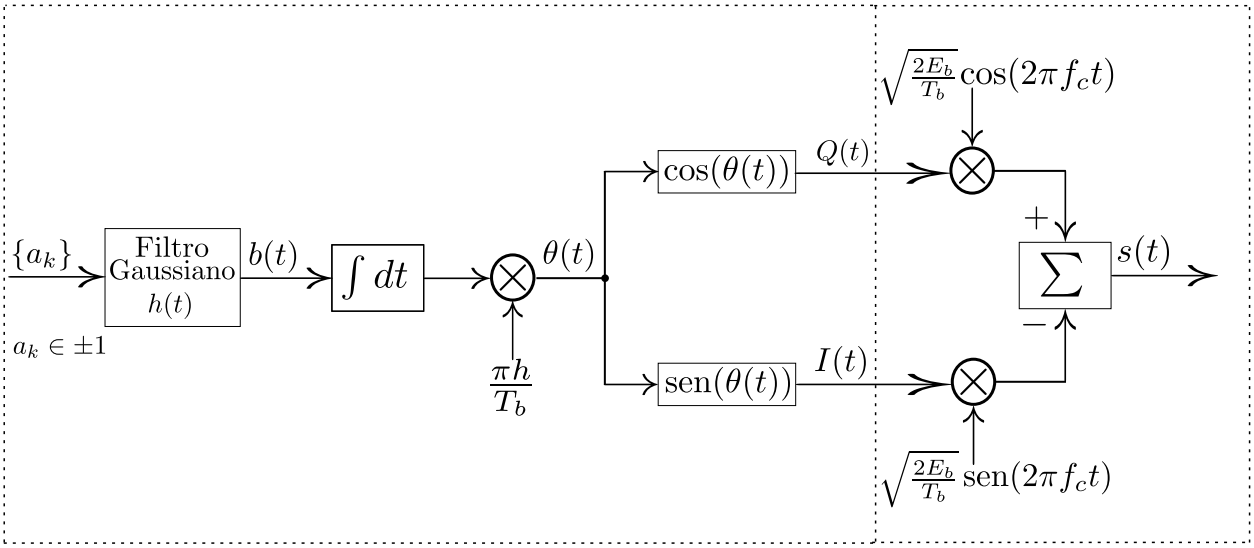
Figura 5.2: Primer nivel de abstracción del Modulador GMSK.

La entrada del sistema corresponde a la señal de información (digital) y la salida del sistema es la señal modulada.

Un segundo nivel de abstracción se muestra en figura 5.3, en donde el modulador GMSK es visto como un sistema digital.

Figura 5.3: Segundo nivel de abstracción del modulador GMSK.

Para proponer la arquitectura que define el funcionamiento del modulador se partió de su diagrama a bloques (figura 4.8)



[1]La metodología de diseño "top-down" es un método de diseño en donde primero son definidas funciones de alto nivel, posteriormente se va bajando su nivel de abstracción

Por lo que la arquitectura propuesta es:

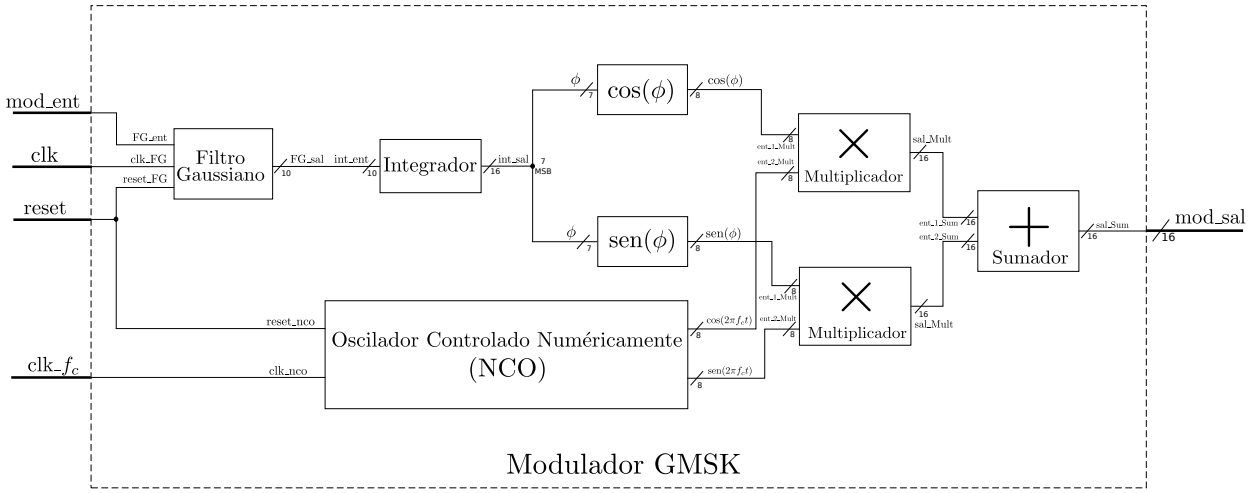


Figura 5.4: Arquitectura propuesta para el modulador GMSK.

Como se observa en la figura 5.4, la arquitectura del modulador se divide en sub-arquitecturas, las cuales se describen a continuación.

5.1.1. Filtro Gaussiano

Siguiendo la metodología de diseño top-down, el primer nivel de abstracción del filtro Gaussiano se muestra en la figura 5.5.

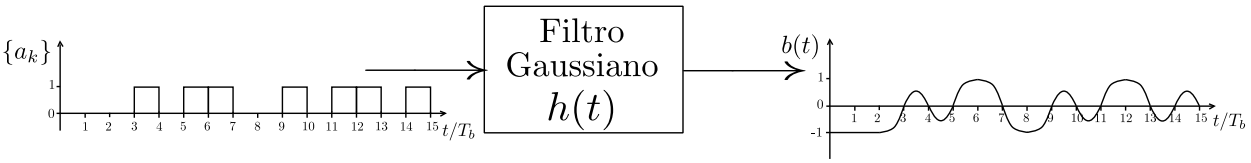


Figura 5.5: Primer nivel de abstracción del filtro Gaussiano.

La entrada del sistema corresponde a un tren de pulsos (señal digital) y la salida del sistema corresponde a la señal filtrada y codificada en NRZ.

Un segundo nivel de abstracción se muestra en la figura 5.6, en donde el filtro Gaussiano es visto como un sistema digital.

Figura 5.6: Segundo nivel de abstracción del filtro Gaussiano.

La arquitectura que define el funcionamiento del filtro Gaussiano se muestra en la figura 5.7, la cual está basada en la arquitectura propuesta en [16], la cual almacena en LUTs las trayectorias $g_{k_2}(t)$, $g_{k_3}(t)$ y $g_{k_7}(t)$ y mediante operaciones lógicas y aritméticas genera las trayectorias restantes.

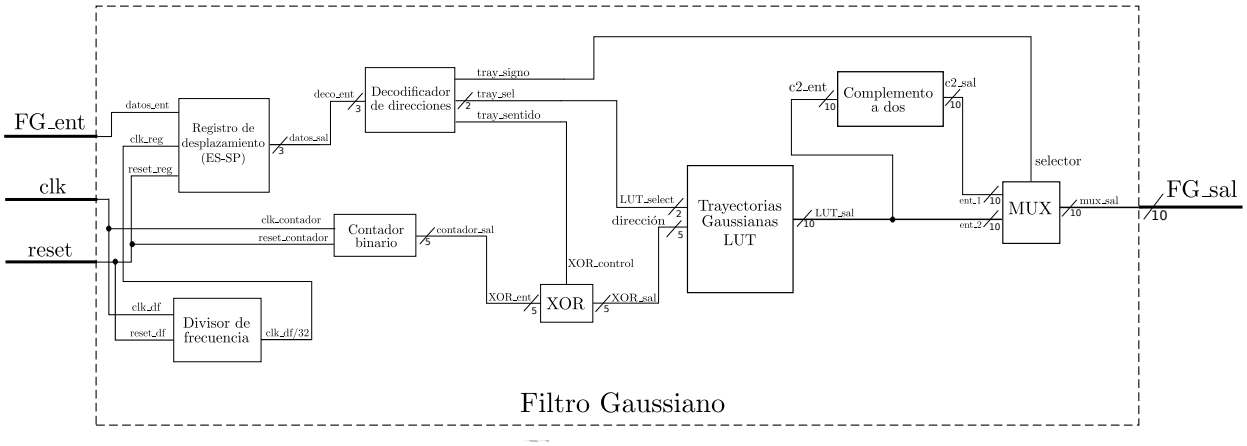


Figura 5.7: Arquitectura propuesta para el filtro Gaussiano.

La arquitectura del filtro Gaussiano (figura 5.7) también se subdivide en sub-arquitecturas, las cuales en conjunto realizan el proceso de filtrado.

En la figura 5.10 se ilustra el funcionamiento del filtro de acuerdo a la excitación. $\{a_k\} = 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, ..$

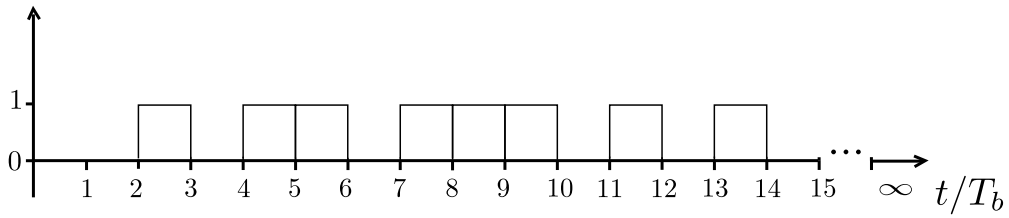


Figura 5.8: Señal de entrada supuesta.

Suponiendo el análisis en $t/T_b = 1$, es decir, cuando $\alpha'_k = 0$ (bit actual), $\alpha'_{k-1} = 1$ (bit anterior) y $\alpha'_{k+1} = 0$ (bit superior), donde la señal de salida es:

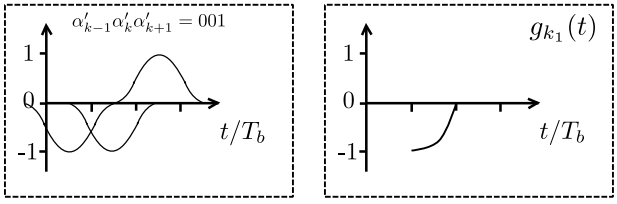


Figura 5.9: Señal de salida del filtro Gaussiano de acuerdo a la combinación: $\alpha'_k = 0$, $\alpha'_{k-1} = 0$ y $\alpha'_{k+1} = 1$.

Figura 5.10: Proceso para el filtrado Gaussiano.

Como se observa en la figura 5.10, el filtro posee un retardo característico de $2T_b$, ya que el "registro de desplazamiento ES-SP" debe guardar los valores del bit superior y el bit inferior.

El "registro de desplazamiento" es alimentado por una señal de reloj con una frecuencia igual a: $f_{clk\_datos} = f_{clk}/32$, correspondiendo a: $f_{clk\_datos} = 1/T_b$.

En $t/T_b = 1$ la salida del registro es: $a_k = 0$ (bit actual), $a_k = 1$ (bit posterior) y $a_{k-1} = 1$ (bit anterior).

En consiguiente, los bits: $\alpha'_{k-1} = 0$, $\alpha'_k = 0$y $\alpha'_{k+1} = 1$ son alimentados al "decodificador de direcciones", el cual genera la salida: "$1, [00], 1$", los cuales corresponden al *signo de la trayectoria, trayectoria, sentido de la trayectoria*, respectivamente. En este ejemplo, el signo de la trayectoria es negativo, la trayectoria seleccionada es la g_{k_4} y el sentido de la trayectoria es descendente.

En el intervalo de tiempo: $t/T_b = 1 \leq t \leq t/T_b = 2$, el "contador digital" accede a 32 direcciones de la "LUT de trayectorias gaussianas"(las 32 direcciones corresponden a las muestras de la trayectoria seleccionada de acuerdo a los bits de *trayectoria*) de forma ascendente o descendente según el bit de *sentido de la trayectoria* y obtiene su "complemento a dos". Al al final, la trayectoria correspondiente a la combinación de los tres bits (a_k, a_k, a_{k-1}) es seleccionada mediante el "mux" de acuerdo al bit *signo de la trayectoria*.

A continuación se describen los módulos que integran la arquitectura del Filtro Gaussiano.

5.1.1.1. Registro de desplazamiento serie-paralelo

Un registro de desplazamiento son circuitos secuenciales construidos con flip-flops cuya función es memorizar temporalmente, así como desplazar datos hacia la izquierda o hacia la derecha.

Un registro de desplazamiento serie-paralelo es aquel que convierte la información recibida de forma serial a un formato paralelo.

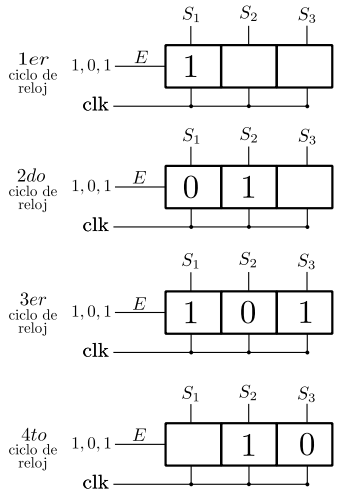


Figura 5.11: Registro de desplazamiento ES-SP (entrada serie- salida paralela).

El registro de desplazamiento ES-SP que se utilizó en la arquitectura del filtro Gaussiano posee los puertos de entrada y salida que se muestran en la figura 5.12.

Figura 5.12: Registro de desplazamiento utilizado en la arquitectura del filtro Gaussiano.

La descripción en hardware en VHDL del registro de desplazamiento serie-paralelo se encuentra en el **Apéndice C**.

5.1.1.2. Divisor de frecuencia

Un divisor de frecuencia es un dispositivo que produce a su salida una frecuencia menor que la entrada. Básicamente un divisor de frecuencia es implementado mediante un contador cuyo módulo corresponde al número por el que se quiere dividir la frecuencia.

$$f_{out} = \frac{f_{in}}{M} \tag{5.1}$$

Donde M es el módulo del contador

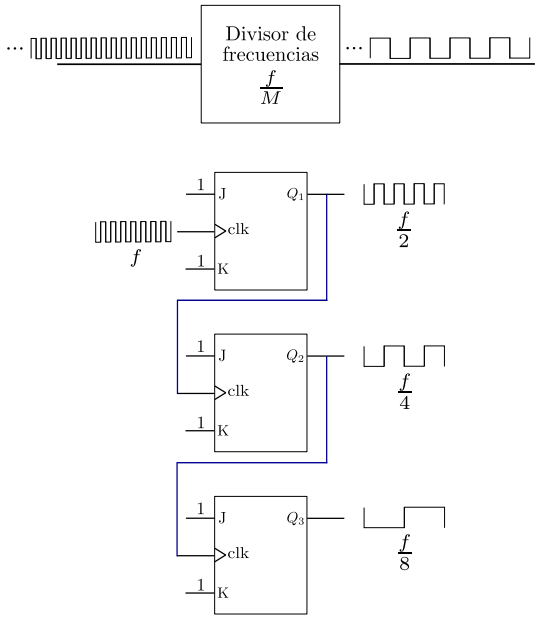


Figura 5.13: Divisor de frecuencia.

El divisor de frecuencia que se utilizó en la arquitectura del filtro Gaussiano posee los puertos de entrada y salida que se muestran en la figura 5.14.

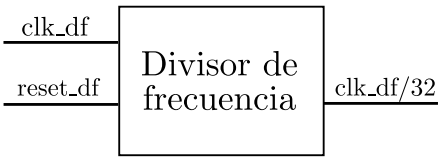


Figura 5.14: Divisor de frecuencia utilizado en la arquitectura del filtro Gaussiano.

La descripción en hardware en VHDL del divisor de frecuencia se encuentra en el **Apéndice C**.

5.1.1.3. Contador digital

Los contadores digitales son dispositivos formados por biestables[2] que se emplean para contar los pulsos de una señal de reloj. Los contadores poseen una entrada por la que llega la señal de reloj, cuyos pulsos son contados y varias salidas en las que se expresa, en un determinado código binario, el número de pulsos que se han contado hasta el momento.

Los contadores se definen mediante diversas características, entre las más importantes se encuentran:

- El tipo de contador: asíncrono o síncrono
- El módulo del contador
- La forma del contador: ascendente o descendente
- El código binario empleado por las salidas para expresar el número de pulsos contados

Los contadores asíncronos son aquellos en el que la señal de reloj solo se aplica al primer biestable de la cadena de biestables que lo forman. En los contadores síncronos, la señal de reloj se aplica a todos los biestables que lo forman.

El módulo del contador hace referencia al número de estados distintos que pueden tomar sus salidas, en otras palabras es el número de pulsos que es capaz de contar.

Un contador es ascendente sí cuenta de abajo a hacia arriba, ejemplo : 0, 1, 2, ... n, ó es descendente sí cuenta de arriba hacia abajo, ejemplo: n, n-1, ... , 2, 1, 0.

Un contador puede utilizar diferentes códigos para representar su salida, ejemplos de estos son el binario,gray, bcd, exceso de 3, etc.

[2]Los biestables son circuitos electrónicos (flip-flop o LATCH) capaces de permanecer en uno de dos estados posibles durante un tiempo indefinido en ausencia de perturbaciones. Esta característica es utilizada en electrónica digital para memorizar información

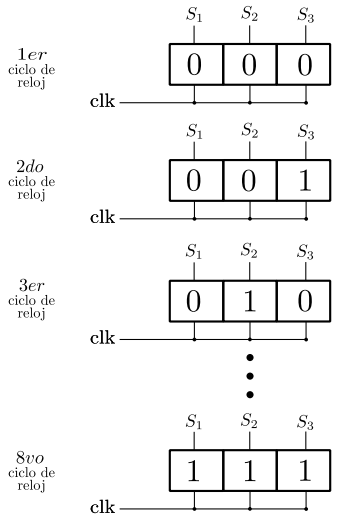


Figura 5.15: Contador binario.

El contador usado en la arquitectura del filtro Gaussiano es del tipo asíncrono, binario, de módulo 32 y puede ser descendente o ascendente. Éste contador binario posee los puertos de entrada y salida que se muestran en la figura 5.16.

Figura 5.16: Contador binario utilizado en la arquitectura del filtro Gaussiano..

La descripción en hardware en VHDL del contador binario se encuentra en el **Apéndice C**.

5.1.1.4. Decodificador de direcciones

El decodificador de direcciones se modeló como un sistema combinacional.

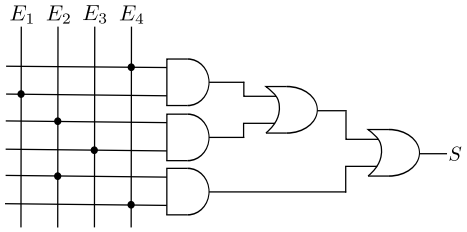


Figura 5.17: Sistema digital combinacional.

La tabla de verdad del decodificador de direcciones se muestra en la tabla 5.1

Tabla 5.1: Tabla de verdad para el decodificador de direcciones

decodificador_ent{2,1,0}	trayectoria_signo	trayectoria_sel{1,0}	trayectoria_sentido
0 0 0	1	1 0	0
0 0 1	1	0 0	1
0 1 0	0	0 1	0
0 1 1	0	0 0	0
1 0 0	1	0 0	0
1 0 1	1	0 1	0
1 1 0	0	0 0	1
1 1 1	0	1 0	0

El decodificador de direcciones que se utilizó en la arquitectura del filtro Gaussiano posee los puertos de entrada y salida que se muestran en la figura 5.18.

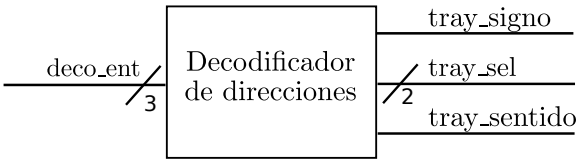


Figura 5.18: Decodificador de direcciones utilizado en la arquitectura del filtro Gaussiano.

La descripción en hardware en VHDL del decodificador de direcciones se encuentra en el **Apéndice C**.

5.1.1.5. XOR

La finalidad de la compuerta lógica XOR es la de configurar al contador de forma ascendente o descendente. La tabla de verdad de la compuerta lógica es la siguiente:

Tabla 5.2: Tabla de verdad de la compuerta lógica XOR del filtro Gaussiano

XOR_entrada	XOR_control	XOR_salida
00000	0	00000
00001	0	00001
00010	0	00010
00011	0	00011
00100	0	00100
00101	0	00101
.....	0	.....
01000	0	01000
01001	0	01001
.....	0	.....
11110	0	11110
11111	0	11111
00000	1	11111
00001	1	11110
00010	1	11101
00011	1	11100
00100	1	11011
00101	1	11010
.....	1	.....
01000	1	10111
01001	1	10110
.....	1	.....
11110	1	00001
11111	1	00000

El módulo XOR que se utilizó en la arquitectura del filtro Gaussiano posee los puertos de entrada y salida que se muestran en la figura 5.19.

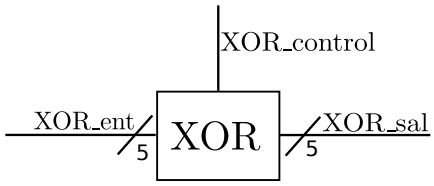


Figura 5.19: Módulo XOR utilizado en el filtro Gaussiano.

La descripción en hardware en VHDL del módulo XOR se encuentra en el **Apéndice C**.

5.1.1.6. Tabla de búsqueda

La tabla de búsqueda se implementó sobre una ROM, segmentada en cuatro secciones.

- La primera sección corresponde a las 32 muestras de la trayectoria $g_{k_4}(011)$
- La segunda sección corresponde a las 32 muestras de la trayectoria $g_{k_3}(010)$
- La tercera sección corresponde a las 32 muestras de la trayectoria $g_{k_8}(111)$
- La cuarta sección corresponde a las 32 muestras de la trayectoria de error $S_E(---)$.

De las trayectorias g_{k_4}, g_{k_3} y g_{k_8} se generan trayectorias restantes mediante operaciones aritméticas y lógicas, mientras que la trayectoria de error $S_E(---)$ es un indicador de errores en la arquitectura del modulador.

La LUT que se utilizó en la arquitectura del filtro Gaussiano posee los puertos de entrada y salida que se muestran en la tabla 5.20.

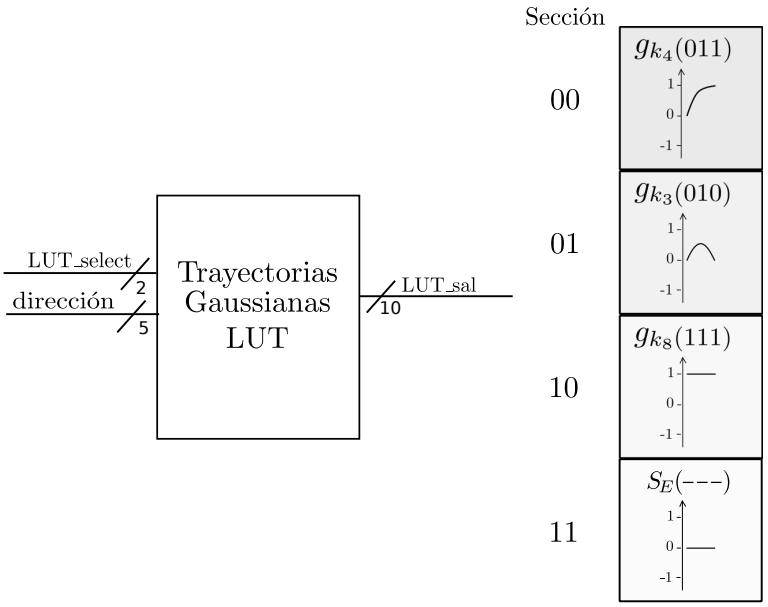


Figura 5.20: Tabla de búsqueda utilizada en la arquitectura del filtro Gaussiano.

Las trayectorias gaussianas utilizadas en la arquitectura del filtro Gaussiano están definidas con $BT_b = 0.3$, mismo valor de parámetro utilizado en GSM.

Los valores de las trayectorias g_{k_4}, g_{k_3} y g_{k_8} que fueron almacenados en memoria se encuentran en la tabla 5.3.

Tabla 5.3: Valores de las trayectorias g_{k_4}, g_{k_3} y g_{k_8} que fueron almacenados en memoria

Trayectoria	Valores {1...32}							
$g_{k_4}(011)$	0	0.059	0.117	0.176	0.235	0.292	0.348	0.401
	0.454	0.503	0.552	0.597	0.640	0.679	0.716	0.751
	0.785	0.814	0.840	0.865	0.886	0.906	0.924	0.939
	0.953	0.965	0.975	0.982	0.990	0.994	0.998	1
$g_{k_3}(010)$	0	0.051	0.100	0.151	0.200	0.241	0.290	0.331
	0.360	0.391	0.421	0.450	0.472	0.481	0.491	0.500
	0.500	0.491	0.481	0.472	0.450	0.421	0.391	0.360
	0.331	0.290	0.241	0.200	0.151	0.100	0.050	0
$g_{k_8}(111)$	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1
$S_E(---)$	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	0

Los valores almacenados en memoria están representadas en punto fijo con un formato signado $0Q9$ en complemento a dos.

La descripción en hardware en VHDL de la tabla de búsqueda se encuentra en el **Apéndice C**.

5.1.1.7. Complemento a dos

En la implementación del filtro Gaussiano se utilizó el complemento a dos para representar los números signados debido a que permite operar de manera directa operaciones aritméticas y no posee una doble representación del cero como en el caso del complemento a uno.

La arquitectura propuesta para el complemento a dos es la mostrada en la figura 5.21. Como se observa en la figura 5.21, el complemento a dos se implementó mediante dos sub-arquitecturas: "complemento a uno" y un "sumador", tal como se obtiene de manera práctica.

El módulo del complemento a dos que se utilizó en la arquitectura del filtro Gaussiano posee los puertos de entrada y salida que se muestran en la figura 5.21

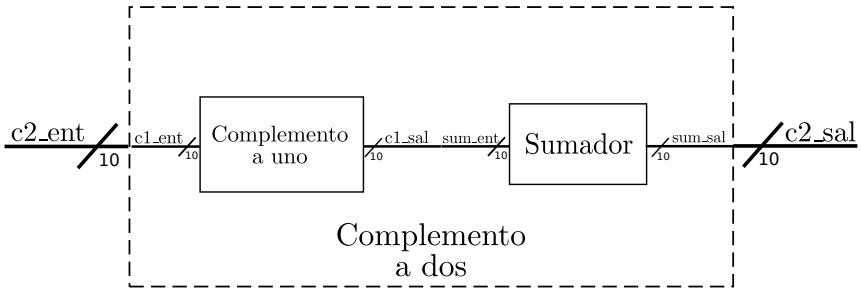


Figura 5.21: Módulo complemento a dos utilizado en la arquitectura del filtro Gaussiano.

La descripción en hardware en VHDL del módulo del complemento a dos se encuentra en el **Apéndice C**.

5.1.1.8. Multiplexor

Un multiplexor digital es un circuito combinacional con varias entradas (I_i) y una salida (Z) que toma el valor de una de las entradas seleccionadas mediante una entrada de control (S_j). La entrada seleccionada se determina en función de la combinación de las entradas de control, por lo que si se tiene N entradas de datos, se necesitan al menos n entradas de control para seleccionarlas, de tal forma que $N \leq 2^n$.

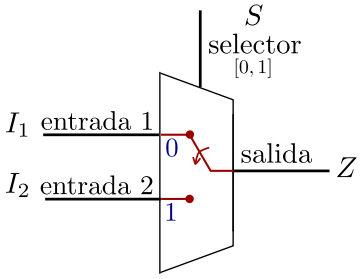


Figura 5.22: Multiplexor 2×1.

El multiplexor que se utilizó en la arquitectura del filtro Gaussiano posee los puertos de entrada y salida que se muestran en la figura 5.23.

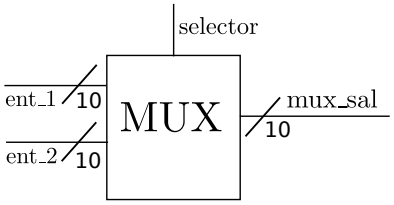


Figura 5.23: Multiplexor utilizado en la arquitectura del filtro Gaussiano.

La descripción en hardware en VHDL del multiplexor se encuentra en el **Apéndice C**.

5.1.2. Integrador

La arquitectura propuesta para el integrador se muestra en la figura 5.24, la cual la conforma un "acumulador", con el cual se asegura que la fase $\theta(t)$ de la señal a transmitir incremente

o decremente linealmente $\pi/2$.

$$\theta(t) = \theta(0) \pm \frac{\pi}{2T_b}t, \quad 0 \leq t < T_b$$

Donde $\theta(0)$ es el valor acumulado de la fase en $t = 0$.

Como se observa en la expresión matemática, un pulso previamente filtrado incrementa o decrementa su fase $\pi/2$ en un periodo de bit T_b.

Entonces, para determinar la longitud de la palabra de salida en bits del acumulador, se supone el caso en donde la suma de fase sea 2π, el cual corresponde a un periodo completo de las señales seno y coseno, resolviendo así el problema de desbordamiento, ya que al desbordarse el acumulador en 2π, éste automáticamente regresa a 0

Por lo tanto, para obtener una fase acumulada de 2π son necesarios 4 pulsos. Recordar que cada pulso tiene 32 muestras, las cuales están cuantificadas a 10 bits y representadas en complemento a dos.

$$32\,\text{muestras} \times \frac{512(\text{máximo valor})}{\text{muestras}} \times 4T_b = 65536\,(\text{máximo valor en un sistema signado}) \tag{5.2}$$

Por lo que la longitud de la palabra de salida en bits corresponde a:

$$log_2(65536) = 16 \text{ bits} \tag{5.3}$$

El módulo integrador que se utilizó en la arquitectura del modulador GMSK posee los puertos de entrada y salida que se muestran en la figura 5.24.

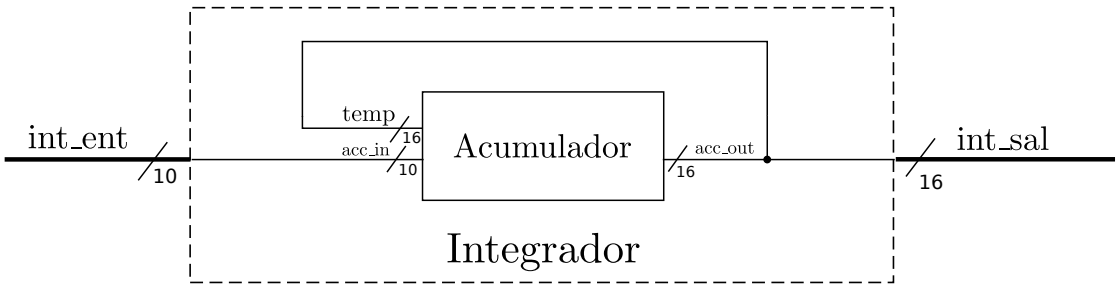


Figura 5.24: Integrador utilizado en la arquitectura del modulador GMSK.

La descripción en hardware en VHDL del Integrador se encuentra en el **Apéndice C**.

5.1.3. LUT seno y LUT coseno

El módulo para el cálculo de seno y coseno se implementó mediante "Tablas de búsqueda" (LUTs), debido a su fácil construcción en comparación con el algoritmo CORDIC (Computadora Digital para Rotación de Coordenadas). Otra razón por la que se decidió implementar LUTs fue debido a las pocas muestras por símbolo que se utilizaron (32 muestras), significando así pocas muestras de la función seno y coseno a utilizar, resultando en el consumo de menos recursos en el FPGA, en comparación del algoritmo CORDIC.

Las LUTs seno y coseno pueden tener n muestras o valores, los cuales se obtienen con las siguientes expresiones:

$$\text{LUT seno} = \sum_{i=0}^{n-1} \text{sen}\left(\frac{2\pi i}{n}\right) \tag{5.4}$$

$$\text{LUT coseno} = \sum_{i=0}^{n-1} \cos\left(\frac{2\pi i}{n}\right) \tag{5.5}$$

Las LUTs que se implementaron para el modulador GMSK poseen 128 muestras, las cuales están representadas en punto fijo con un formato signado $0Q7$ en complemento a dos.

En las salidas de dichas tablas se obtienen las señales $Q(t)$ e $I(t)$.

Los módulos seno y coseno que se utilizaron en la arquitectura del modulador GMSK poseen los puertos de entrada y salida que se muestran en la figura 5.25.

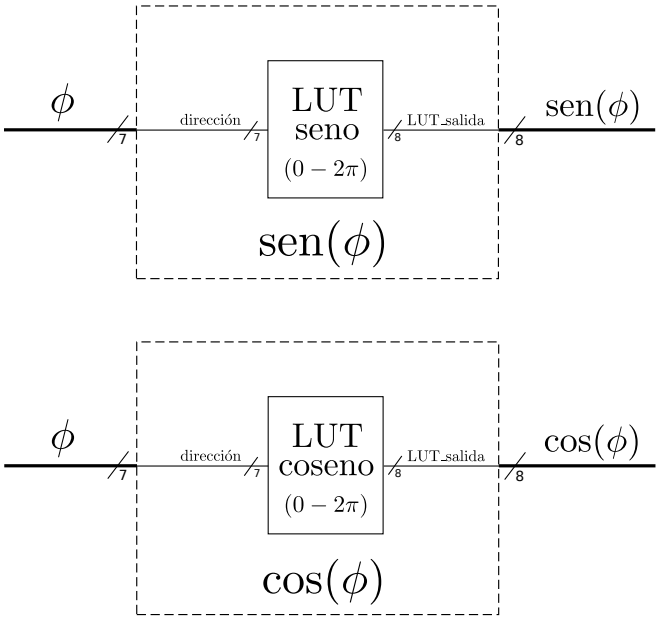


Figura 5.25: Módulos seno y coseno utilizados en la arquitectura del modulador GMSK.

La descripción en hardware en VHDL de las LUTs se encuentra en el **Apéndice C** y la tabla con sus valores se encuentran en el **Apéndice B**.

5.1.4. Oscilador controlado numéricamente (NCO)

El NCO es un generador de señales, cuya salida es la representación discreta en el tiempo de una forma de onda. Los NCOs ofrecen varias ventajas sobre otros tipos de osciladores en términos de agilidad, exactitud, estabilidad y confiabilidad.

La arquitectura del NCO está formada por un contador y dos tablas de búsqueda: seno y coseno (figura 5.26). El contador recorre las direcciones de las memorias en las cuales están guardados los valores del seno y coseno, ecuaciones (5.4) y (5.5).

En la salida del NCO se obtienen dos señales sinusoidales:

$$\text{sen}(2\pi f_c t) \text{ y } \cos(2\pi f_c t)$$

Las frecuencia f_c del NCO se obtiene con la siguiente expresión:

$$f_c = \frac{f_{clk}}{n} \tag{5.6}$$

Donde n es el módulo del contador o el número de muestras de la señal sinusoidal en un periodo; y f_{clk} es la frecuencia del reloj.

Las LUTs contienen $n = 25$ muestras, las cuales están representadas en punto fijo con un formato signado $0Q7$ en complemento a dos. $n = 25$ genera una $f_c = 40\,\text{kHz}$ con $f_{clk} = 1\,\text{MHz}$, dichas frecuencias se utilizaron para validar el modulador GMSK.[3]

El NCO que se utilizó en la arquitectura del modulador GMSK posee los puertos de entrada y salida que se muestran en la figura 5.26.

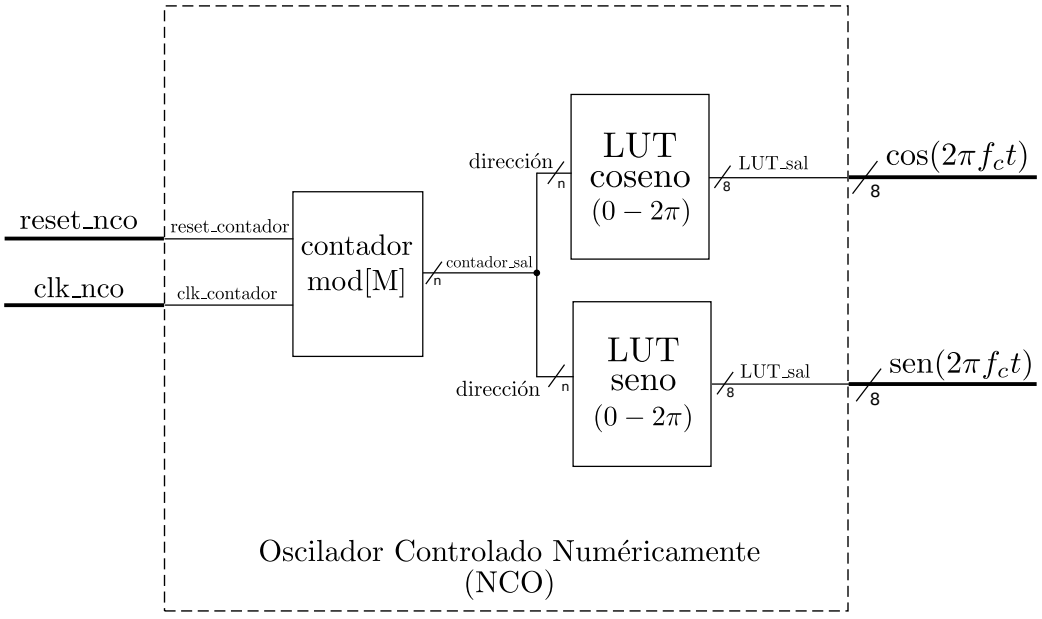


Figura 5.26: NCO utilizado en la arquitectura del modulador GMSK.

La descripción en hardware en VHDL del NCO se encuentra en el **Apéndice C**.

La tabla con los valores de LUTs del seno y coseno implementadas en NCO se encuentran en el **Apéndice B**.

[3]Generalmente en el diseño de NCOs el valor de f_{clk} es fijo, por lo que el valor de n es el que se varía para conseguir la f_c requerida

5.1.5. Multiplicador

El multiplicador digital funciona como modulador, ya que multiplica la señal portadora con la señal moduladora (I o Q)

$$\text{sen}(2\pi f_c t) \cdot I(t) \;\text{ y }\; \cos(2\pi f_c t) \cdot Q(t) \tag{5.7}$$

Esta arquitectura multiplica dos palabras de x y y bits para obtener un resultado de $x + y$ bits.

El módulo multiplicador que se utilizó en la arquitectura del modulador GMSK posee los puertos de entrada y salida que se muestran en la figura 5.27.

Figura 5.27: Módulo multiplicador utilizado en la arquitectura del modulador GMSK.

La descripción en hardware del multiplicador en VHDL se encuentra en el **Apéndice C**.

5.1.6. Sumador

El sumador digital suma las señales moduladas $\cos(2\pi f_c t) \cdot Q(t)$ y $-\text{sen}(2\pi f_c t) \cdot I(t)$ para obtener la señal modulada GMSK.

$$s(t) = \cos(2\pi f_c t) \cdot Q(t) + -\,\text{sen}(2\pi f_c t) \cdot I(t) \tag{5.8}$$

Esta arquitectura suma o resta dos palabras de x bits, cuyo resultado también es de x bits.

El módulo sumador que se utilizó en la arquitectura del filtro Gaussiano posee los puertos de entrada y salida que se muestran en la figura 5.28.

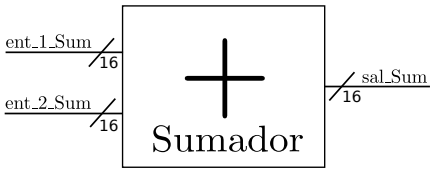


Figura 5.28: Módulo sumador utilizado en la arquitectura del filtro Gaussiano.

La descripción en hardware del sumador en VHDL se encuentra en el **Apéndice C**.

5.2. Síntesis, implementación y reconfiguración

Los procesos de : síntesis, implementación y reconfiguración, se realizaron en el ambiente de desarrollo ISE (Integrated Software Environment) para un FPGA Spartan 6 (XC6SLX16-CSG324), tanto el software como el dispositivo pertenecen a la compañía Xilinx. A la interfaz de usuario gráfica (GUI) de ISE se le denomina *Project Navigator*, la cual se encarga de gestionar las etapas del flujo de diseño. Esta interfaz se monta sobre un conjunto de programas, los cuales participan en cada etapa de diseño (síntesis, implementación y reconfiguración).

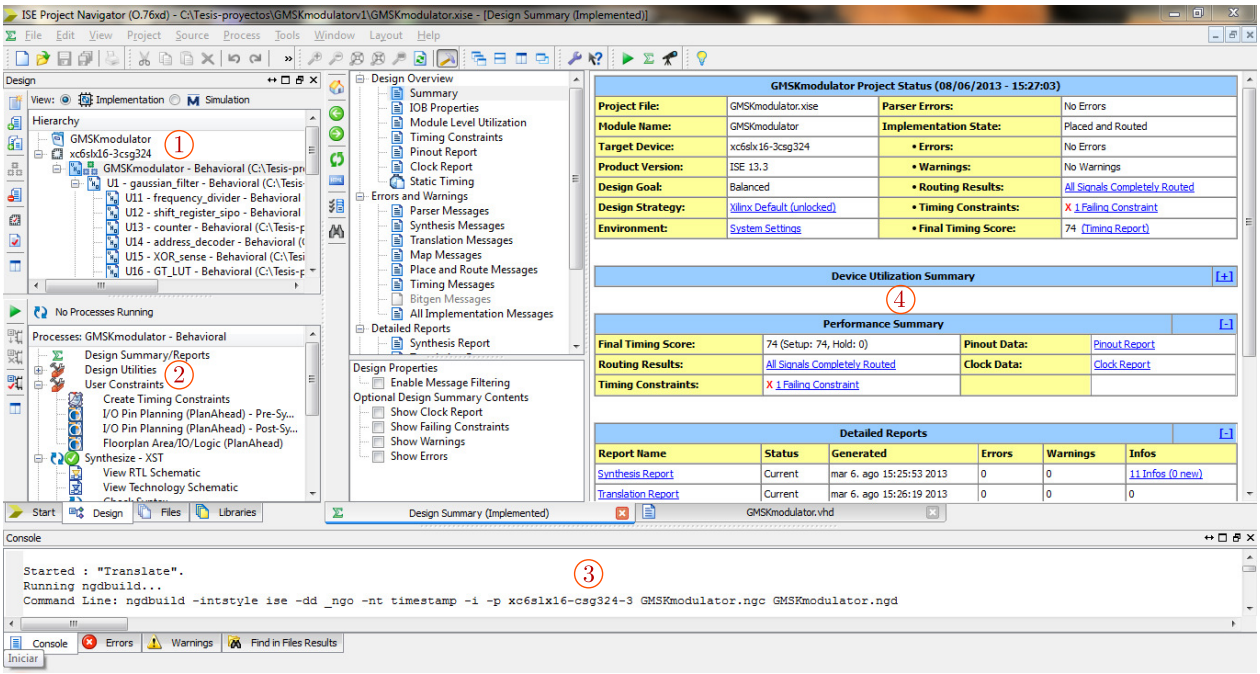


Figura 5.29: Ambiente de desarrollo ISE-Project Navigator

En la figura 5.29 se distinguen 4 áreas principales:

1.- Área de archivos. En esta área se muestran los diferentes archivos fuentes que integran el proyecto, a manera de árbol. Si un proyecto está formado por varios módulos, estos se ubicarán de manera jerárquica en el árbol.

2.- Área de mensajes. En esta área se muestran los mensajes e informes de las tareas que se realizan en cada proceso. Además indican errores y advertencias surgidas en el proyecto.

3.- Área de procesos. En esta área aparecen los procesos que se pueden aplicar a los diferentes archivos del proyecto. Para cada proceso se muestra el estado del mismo, si ha sido ejecutado, y en caso afirmativo si el resultado ha sido correcto.

4.- Área de trabajo. Esta área está destinada para visualizar y/o editar cualquier archivo ligado al proyecto. Además, esta área puede ser un navegador de internet para la consulta de información acerca de errores o advertencias surgidas en el diseño.

5.2.1. Síntesis

El proceso de síntesis consiste en reducir una descripción de alto nivel de abstracción a un nivel de compuertas lógicas que pueda ser implementado en el circuito, dicho de otra manera, la síntesis es el proceso mediante el cual una descripción es convertida en un listado de conexiones (netlist) entre las compuertas, registros, multiplexores, etc. de un CPLD o FPGA. Este proceso genera archivos NGC (.ngc).

La síntesis e implementación de la arquitectura del modulador GMSK se realizó en un FPGA **Spartan 6 XC6SLX16** con empaquetado CSG324.

A continuación se muestran algunos esquemas RTL del modulador GMSK y el reporte estimado de recursos utilizados en el FPGA generados por la síntesis.

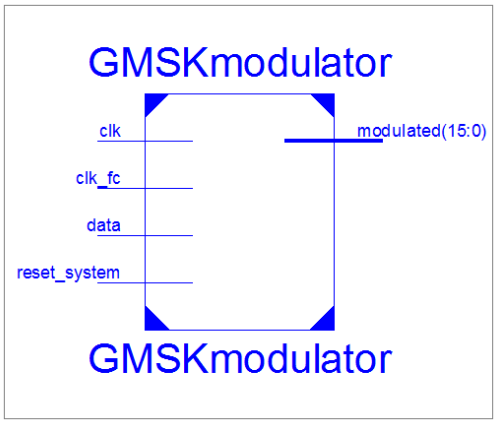


Figura 5.30: Esquema RTL del modulador GMSK (primer nivel de jerarquía) generado por *ISE*

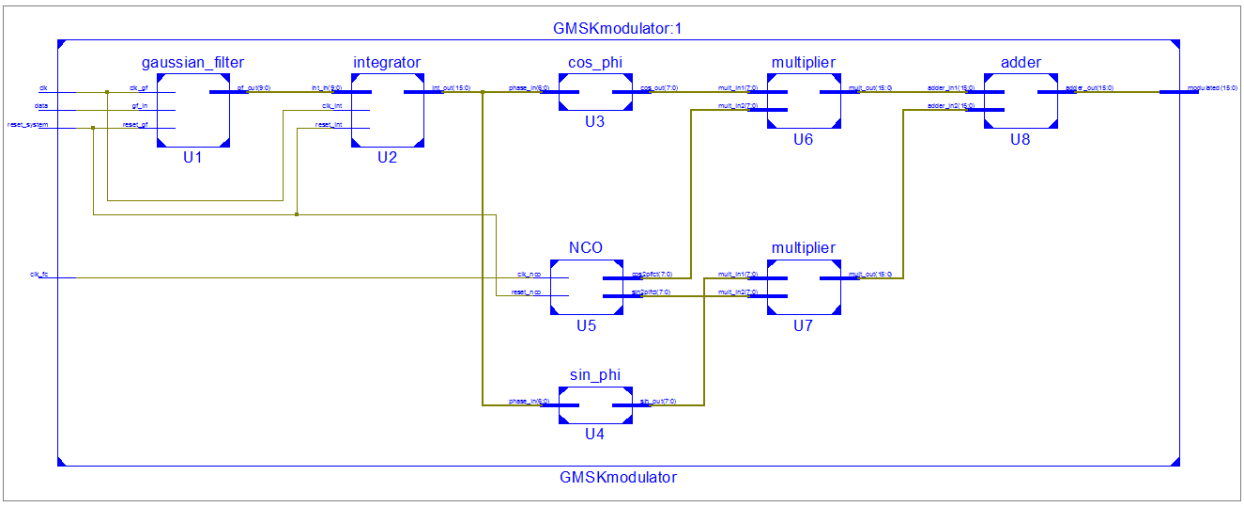


Figura 5.31: Esquema RTL del modulador GMSK (segundo nivel de jerarquía) generado por *ISE*

En el **Apéndice D** se encuentran los esquemas RTL de los módulos: U1, U2, U3, U4, U5, U6, U7 y U8 generados por *ISE*.

Tabla 5.4: Utilización del dispositivo (valores estimados)

Elementos	Usados	Disponibles	% Utilización
Slice Registers	29	18,224	1 %
Slice LUTs	118	9,112	1 %
LUT-FF pairs	27	120	22 %
IOBs	20	232	8 %
BUFG/BUFGCTRLs	2	16	12 %
DSP48A1s	2	32	6 %

5.2.2. Implementación

El proceso de implementación está dividido en tres fases: *translate, mapping, place y route*

- ☑ En la fase de *translate* (traducción) se combinan el archivo que contiene la información del modelo (.ngc) con el archivo que contiene la información de restricciones (.ucf), en el cual se especifican los pines que conectarán las señales de entrada y salida, así como el tipo de tecnología. Además en este archivo se pueden añadir restricciones temporales como retardos de señales de reloj.

- ☑ En la fase de *mapping* (mapeo) se asignan los componentes físicos del FPGA de acuerdo al proceso de síntesis (slice, LUTs, flip-flops, etc.). En este punto es posible conocer la cantidad de recursos utilizados en el FPGA para la construcción del diseño.

- ☑ En la fase de *place* (colocación) se decide la ubicación de los componentes electrónicos, circuitos, y elementos lógicos dentro del FPGA.

- ☑ En la fase de *route* (enrutamiento) se realiza la conexión de los componentes electrónicos, circuitos, y elementos lógicos.

La tabla 5.5 muestra el reporte de recursos utilizados en el FPGA Spartan 6 tras el proceso de implementación.

Tabla 5.5: Utilización del dispositivo

Elementos	Usados	Disponibles	% Utilización
Slice Registers	29	18,224	1 %
-Flip Flops	29		
-Latches	0		
-Latch-thrus	0		
-AND/OR	0		
Slice LUTs	99	9,112	1 %
Memory	0	2,176	0 %
Slices	36	2,278	1 %
MUXCYs	16	4,556	1 %
LUT-FF pairs	23	102	22 %
IOBs	20	232	8 %
RAMB16BWERs	0	32	0 %
RAMB8BWERs	0	64	0 %
BUFIO2/BUFIO2_2CLKs	0	32	0 %
BUFIO2FB/BUFIO2FB_2CLKs	0	32	0 %
BUFG/BUFGMUXs	2	16	12 %
DCM/DCM_CLKGENs	0	4	0 %
ILOGIC2/ISERDES2s	0	248	0 %
IODELAY2/IODRP2/IODRP2_MCBs	0	248	0 %
OLOGIC2/OSERDES2s	0	248	0 %
BSCANs	0	4	0 %
BUFHs	0	128	0 %
BUFPLLs	0	8	0 %
BUFPLLs_MCBs	0	4	0 %
DSP48A1s	2	32	6 %
ICAPs	0	1	0 %
MCBs	0	2	0 %
PCILOGICSEs	0	2	0 %
PLL_ADVs	0	2	0 %
PMVs	0	1	0 %
STARTUPs	0	1	0 %
SUSPEND_SYNCs	0	1	0 %

5.2.3. Reconfiguración

Una vez implementado el diseño se debe generar el archivo de reconfiguración (.bit) o "bitstream", el cual se cargará al dispositivo lógico. Este proceso se realizó mediante la herramienta de Xilinx iMPACT.

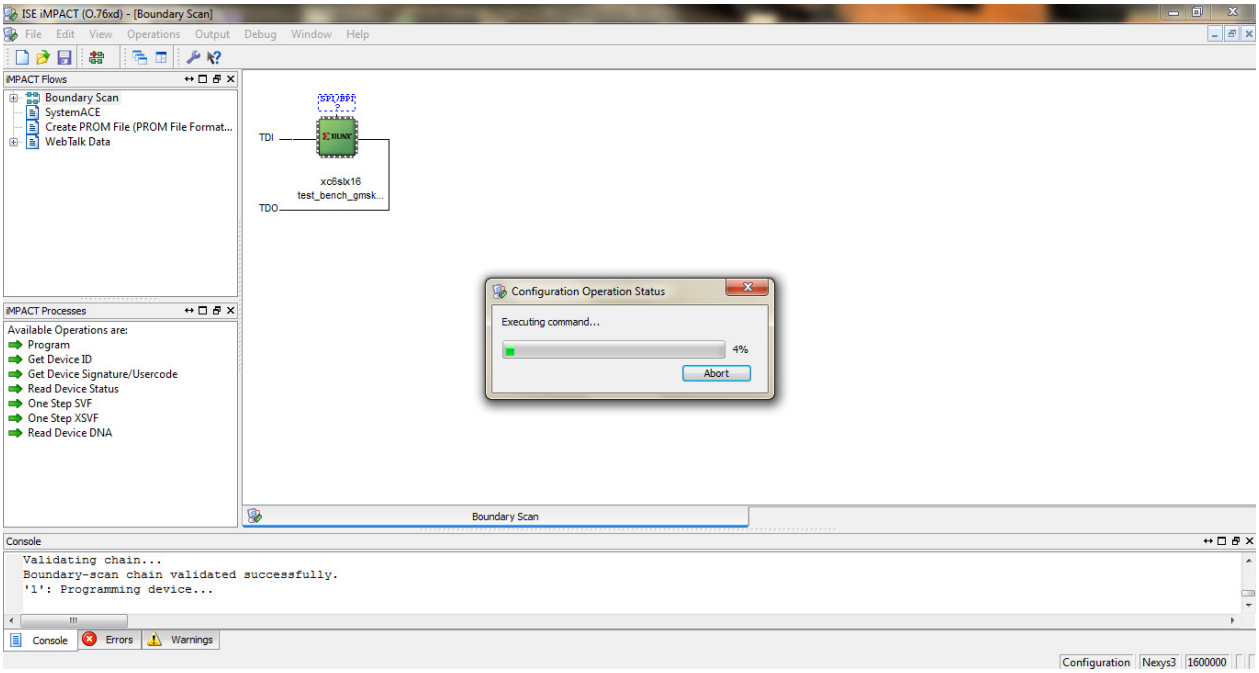


Figura 5.32: Captura de pantalla de la interfaz de usuario de iMPACT al momento de la reconfiguración del FPGA.

Capítulo 6

Resultados experimentales

En este capítulo se muestran los resultados experimentales obtenidos a partir de la simulación lógica y la implementación física de la arquitectura del modulador GMSK.

6.1. Simulaciones realizadas en ModelSim PE Student

Las simulaciones lógicas de la arquitectura del modulador se realizaron mediante **ModelSim PE Student**[1], debido a las opciones de simulación, depuración y visualización con las que cuenta, resaltando la visualización analógica, la cual hace la función de un DAC (convertidor digital-analógico) de manera virtual.

Para realizar la simulación fue necesario crear un *test bench* o banco de pruebas, el cual es una descripción en VHDL que permite aplicar estímulos al circuito que se quiere simular y observar sus respuestas de forma virtual.

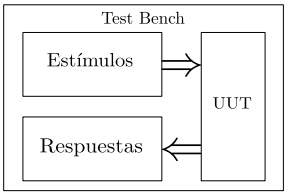


Figura 6.1: Esquema a bloques del test bench.

El archivo *test bench* usado para simular la "arquitectura de validación" se encuentra en el Apéndice C.

Los parámetros de simulación fueron los siguientes:

- $R = \frac{1}{T_b} = \frac{1}{32\mu s} = 31.25\,[\text{kbits/s}]$
- $f_{clk} = \frac{1}{T_b} \cdot 32 = 1\text{MHz}$
- $f_{clk\_fc} = 1\text{MHz}$, para obtener $f_c \approx 38.462\,\text{kHz}$ con $n = 26$ muestras de la función seno y coseno almacenadas en las LUTs del NCO.

$$f_c = \frac{f_{clk\_fc}}{n}$$

[1]Simulador y entorno de depuración unificado para Verilog, VHDL y SystemC. Software de la compañía Mentor Graphics de versión estudiantil.

A continuación se muestran los resultados obtenidos en la simulación.

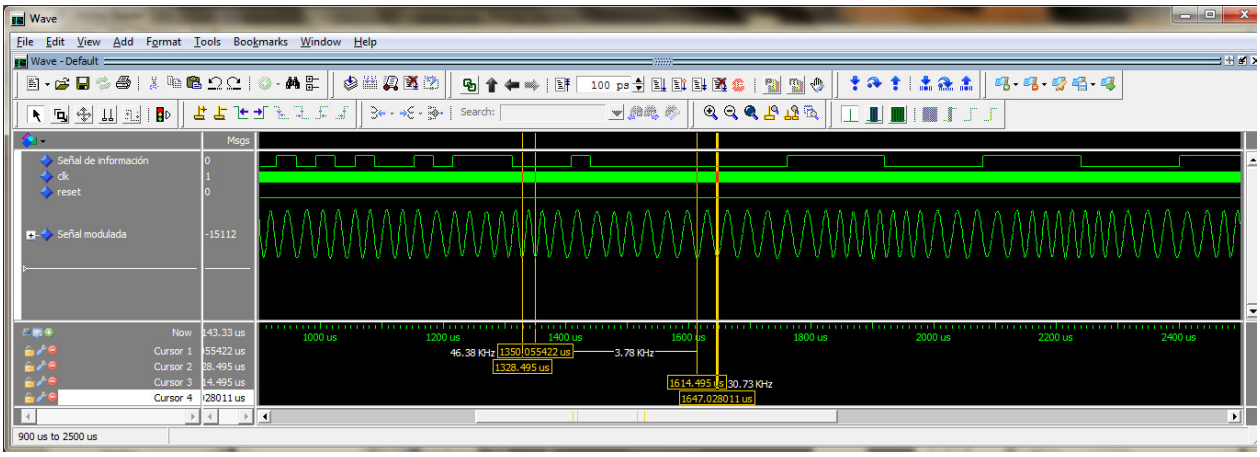


Figura 6.2: Simulación 1: Señal de información $\{a_k\}$ y señal modulada $s(t)$.

En esta figura se muestran la señal de información $\{a_k\}$ y la señal modulada en GMSK $s(t)$ resultantes de la simulación lógica, en la cual se observa de forma cualitativa que la señal modulada posee las características esperadas: señal modulada en frecuencia de fase continua.

De forma cuantitativa se tiene que: $f_1 \simeq 46.38$ kHz y $f_2 \simeq 30.73$ kHz y se sabe que:

$$h = \Delta f \cdot T_b$$

$$h = (f_1 - f_2) \cdot T_b$$

$$h_{sim} = 46.38\,[\text{kHz}] - 30.73\,[\text{kHz}] \cdot 32\,[\mu\text{s}]$$

$$h_{sim} \simeq 0.5008$$

Además:

$$f_c = \frac{f_1 + f_2}{2}$$

$$f_{c_{sim}} = \frac{46.38\,[\text{kHz}] + 30.73\,[\text{kHz}]}{2}$$

$$f_{c_{sim}} \simeq 38.55\,\text{kHz}$$

Debido a que $f_{c_{sim}} \approx f_{clk\_fc}$ y $h \approx 0.5$, se concluye que la señal $s(t)$ es una señal modulada en GMSK.

En la figura 6.3 se observa que la arquitectura del modulador GMSK posee un retardo de $2T_b$, tal como se había estimado. La simulación presenta un retardo de: $2T_b = 64\ \mu s$, de acuerdo a los parámetros de simulación.Por otra parte se muestra además que los cambios de frecuencia ocurren por los cruces por cero de la señal filtrada codificada en NRZ.

En la figura 6.4 se muestran: la señal de fase $\theta(t)$, componente en fase $I(t)$, componente en cuadratura $Q(t)$, $\cos(2\pi f_c t)$, $\text{sen}(2\pi f_c t)$ y señal modulada $s(t)$, observando lo siguiente:

- En la señal de fase $\theta(t)$ se observan los cambios de fase producidos por los símbolos de entrada.

- Se observa que las señales $I(t)$ y $Q(t)$ están desfasadas $\pi/2$ una con respecto a la otra.
- La señales portadoras $\cos(2\pi f_c t)$ y $\text{sen}(2\pi f_c t)$ están desfasadas $\pi/2$ y poseen una frecuencia de 38.55 kHz.

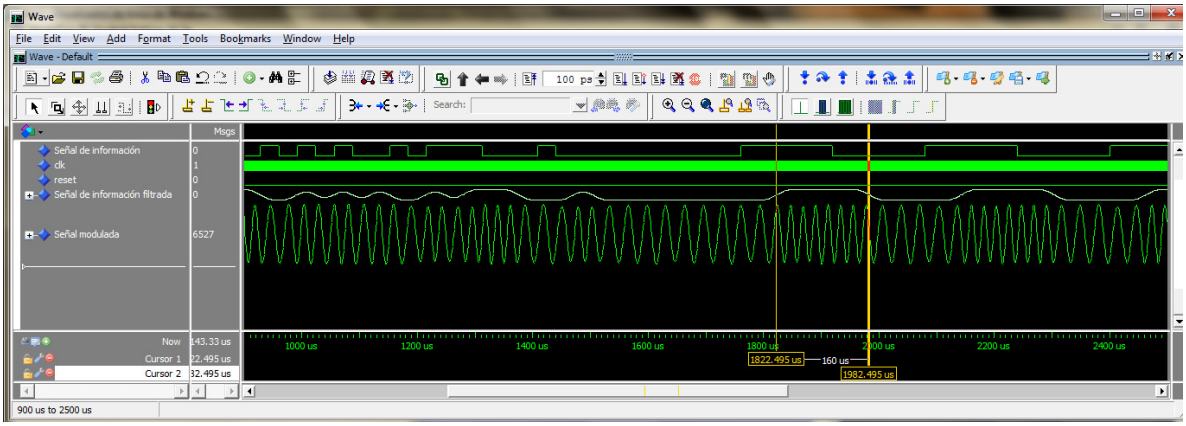


Figura 6.3: Simulación 2: Señal de información $\{a_k\}$, señal de información filtrada $b(t)$ y señal modulada $s(t)$.

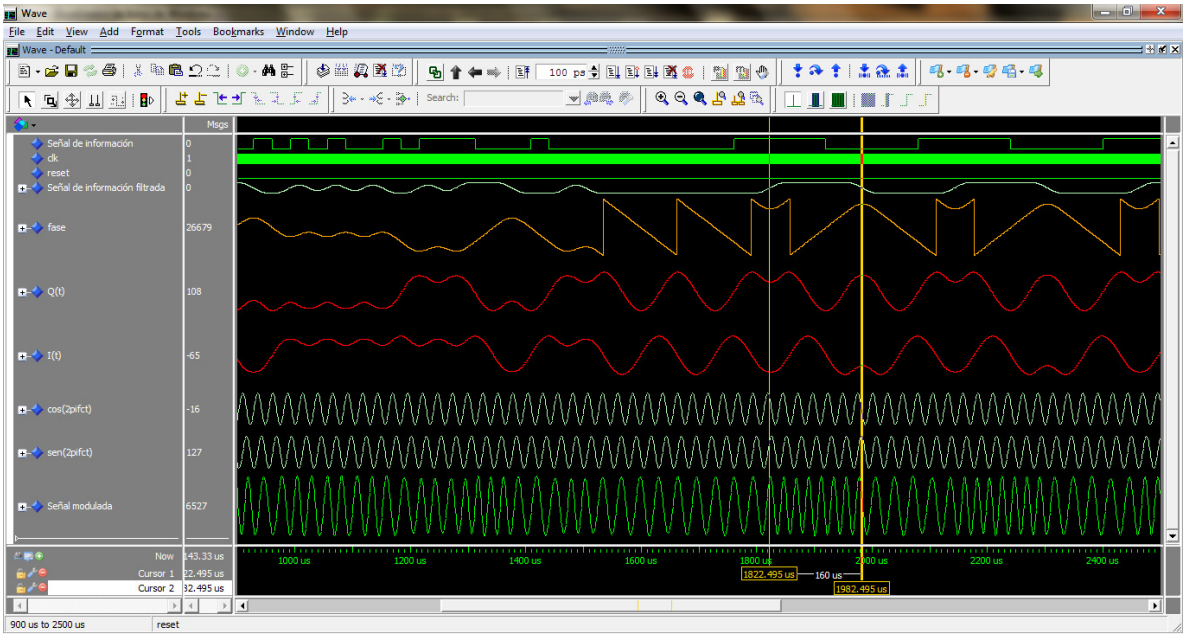


Figura 6.4: Simulación 3: Señal de información $\{a_k\}$, señal de información filtrada $b(t)$, señal de fase $\theta(t)$, componente en fase $I(t)$, componente en cuadratura $Q(t)$, $\cos(2\pi f_c t)$, $\text{sen}(2\pi f_c t)$ y señal modulada $s(t)$.

6.2. Resultados obtenidos tras la implementación física del modulador GMSK

La implementación física de la arquitectura del modulador digital GMSK se realizó en la tarjeta de desarrollo Nexys3^{TM}de Digilent Inc.

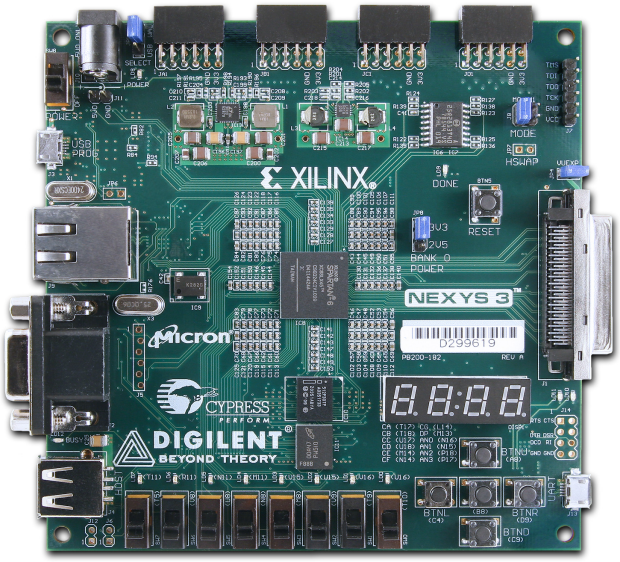


Figura 6.5: Tarjeta de desarrollo Nexys3^{TM}de Digilent Inc.

Esta tarjeta incluye elementos como:

- FPGA Spartan-6 LX16 de Xilinx con 324 pines y empaquetado BGA (XC6SLX16-CSG324)
- Oscilador CMOS de 100 MHz
- GPIOs: 8 LEDs, 5 push-button, 8 switches y 4 displays de 7 segmentos
- 72 puertos de entrada/salida direccionables
- 16 MB en RAM
- 16 MB en PCM-memoria no volátil
- Puerto VGA-8 bits
- Puertos USB-UART y USB-HID
- Ethernet PHY

Además se utilizó un DAC (Convertidor Analógico Digital) para visualizar la señales de interés en forma analógica. El DAC utilizado es el DAC0800 de Texas Instruments [2], cuyas características se citan a continuación:

- 8 bits de resolución
- Error de escala completa: ±1
- Tiempo de asentamiento: 100 ns
- No linealidad sobre temperatura: ±0.1 %
- Interfaz directamente con TTL, CMOS, PMOS y otros
- Amplio margen de alimentación: ±4.5V a ± 18V
- Bajo consumo de potencia: 33 mW a ± 5V
- Bajo costo

El diagrama de conexiones del experimento se muestra a continuación:

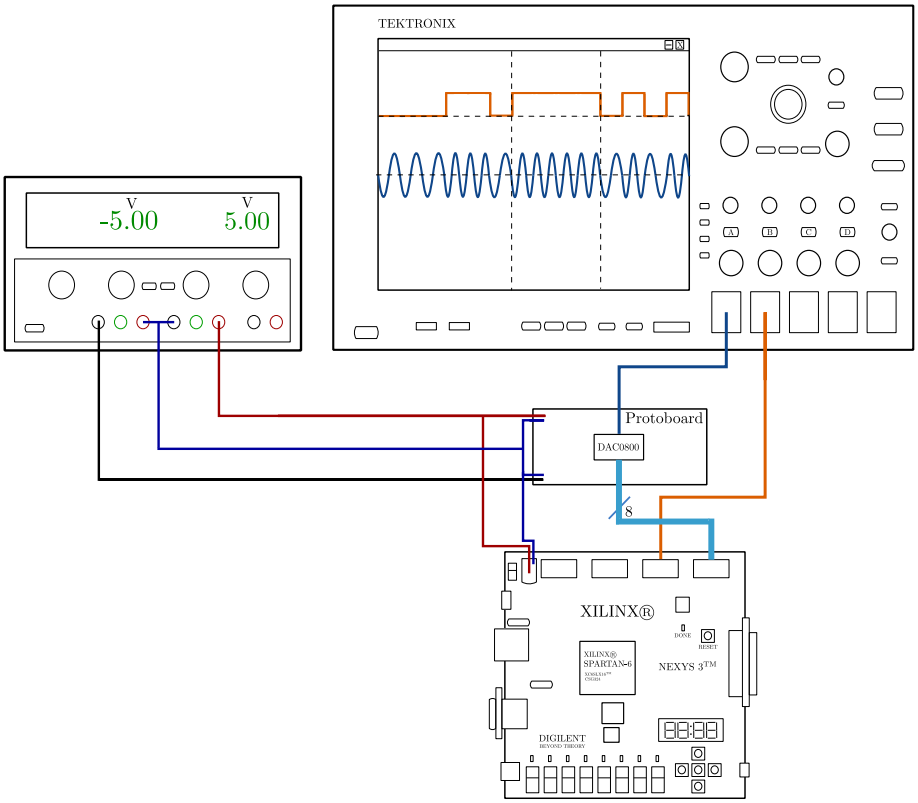


Figura 6.6: Diagrama de conexiones del experimento.

El Osciloscopio utilizado para visualizar las señales es el Tektronix DPO5000.

[2] Hoja de especificaciones del DAC08000 en http://www.ti.com/lit/ds/symlink/dac0800.pdf

El diagrama de conexión del DAC0800 se muestra en la siguiente figura:

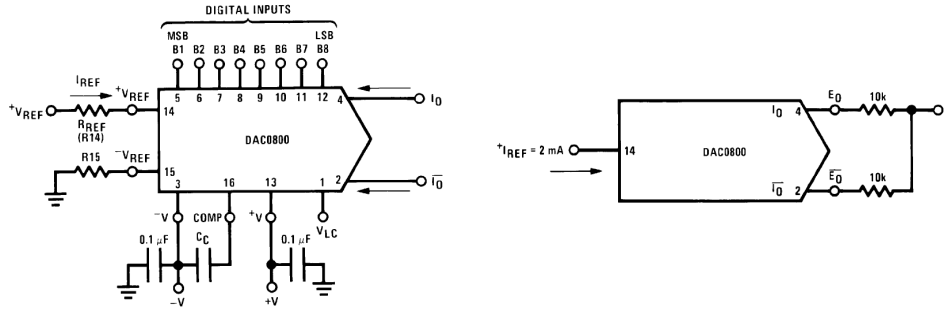


Figura 6.7: Diagrama de conexiones.

El DAC representa su salida mediante un código binario, el cual se aprecia en la tabla 6.1.

$B1$	$B2$	$B3$	$B4$	$B5$	$B6$	$B7$	$B8$	E_0
1	1	1	1	1	1	1	1	-4.96
1	1	1	1	1	1	1	0	-4.92
1	0	0	0	0	0	0	1	-0.040
1	0	0	0	0	0	0	0	0.000
0	1	1	1	1	1	1	1	+0.040
0	0	0	0	0	0	0	1	+4.96
0	0	0	0	0	0	0	0	+5.000

Tabla 6.1: Representación binaria de la salida del DAC.

Se puede observar que el código binario utilizado en el DAC es “exceso-128”.

Por otra parte, la configuración de puertos de entrada/salida de la “arquitectura de pruebas” en el FPGA Spartan 6 fue la siguiente:

	State	Timing Constraints	
1	OK	*#" PlanAhead Generated physical constraints "*	Test_bench_GMSKmodulator.ucf
2	OK	NET "clk_system" LOC = V10	Test_bench_GMSKmodulator.ucf
3	OK	NET "data" LOC = K2	Test_bench_GMSKmodulator.ucf
4	OK	NET "reset_system" LOC = T10	Test_bench_GMSKmodulator.ucf
5	OK	NET "selector[0]" LOC = T5	Test_bench_GMSKmodulator.ucf
6	OK	NET "selector[1]" LOC = V8	Test_bench_GMSKmodulator.ucf
7	OK	NET "selector[2]" LOC = U8	Test_bench_GMSKmodulator.ucf
8	OK	NET "signal_out[0]" LOC = E12	Test_bench_GMSKmodulator.ucf
9	OK	NET "signal_out[1]" LOC = F12	Test_bench_GMSKmodulator.ucf
10	OK	NET "signal_out[2]" LOC = C12	Test_bench_GMSKmodulator.ucf
11	OK	NET "signal_out[3]" LOC = D12	Test_bench_GMSKmodulator.ucf
12	OK	NET "signal_out[4]" LOC = E11	Test_bench_GMSKmodulator.ucf
13	OK	NET "signal_out[5]" LOC = F11	Test_bench_GMSKmodulator.ucf
14	OK	NET "signal_out[6]" LOC = F10	Test_bench_GMSKmodulator.ucf
15	OK	NET "signal_out[7]" LOC = G11	Test_bench_GMSKmodulator.ucf

Figura 6.8: Configuración de puertos de entrada/salida en el FPGA.

Para validar experimentalmente el funcionamiento correcto del modulador se creó una "arquitectura de pruebas", en la cual se agregaron los siguientes módulos:

- ☑ Divisor de frecuencia (frecuency_divider_dg) que da origen a la frecuencia del reloj del generador de datos.
- ☑ Generador de datos (data_generator).
- ☑ Divisor de frecuencia (frecuency_divider_clk) que da origen a la frecuencia del reloj del modulador GMSK (clk).
- ☑ Divisor de frecuencia(frecuency_divider_fc) que da origen a la frecuencia del reloj del NCO (clk_fc), el cual genera las señales portadoras.
- ☑ Modulador GMSK.
- ☑ Multiplexor 7x1, utilizado para visualizar 7 señales con un DAC.
- ☑ módulo "offset", el cual resta una constante de 128 a la señal de salida, logrando así su correcta representación en la salida del DAC.

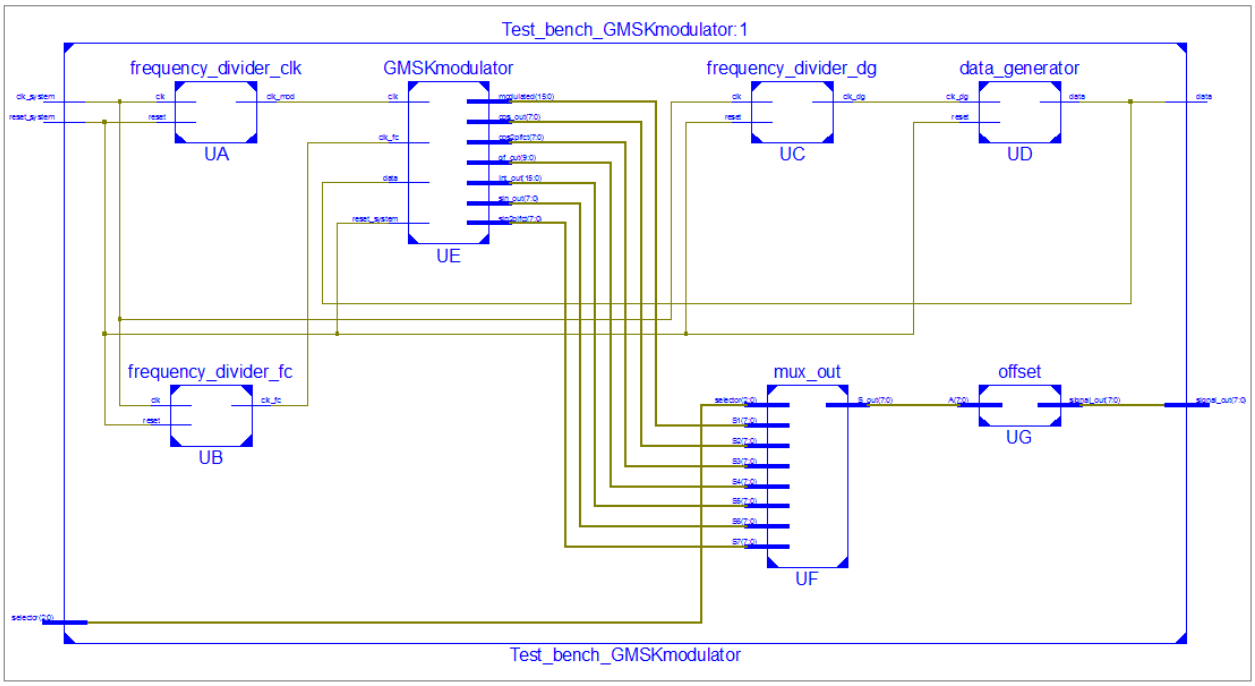


Figura 6.9: Esquema RTL de la "arquitectura de pruebas" generado por *Project Navigator*

El "data_generator" alimenta al modulador GMSK con una tasa de bits (R):

$$R = \frac{1}{T_b} = 1 \cdot f_{clk\_datos} = 31.250\,[\text{kb/s}]$$

Donde $f_{clk\_datos} = f_{clk\_NEXYS3}/3200 = 100\,\text{MHz}/3200 = 31.250\,\text{kHz}$.

El módulo "frecuency_divider_dg" divide la frecuencia del oscilador de la tarjeta de desarrollo "Nexys 3" con un factor de 3200.

El módulo "frecuency_divider_fc" origina a la frecuencia del reloj clk_fc mediante:

$$f_{clk\_fc} = f_{clk\_NEXYS3}/100 = 1\,\text{MHz}$$

La frecuencia de 1 MHz genera las señales portadoras con f_c igual:

$$f_c = \frac{f_{clk\_fc}}{n} = \frac{1\,\text{MHz}}{26} \simeq 38.55\,\text{MHz}$$

Donde n es el número de muestras de las funciones seno y coseno que contiene el NCO.

La frecuencia del reloj del modulador es: $f_{clk} = 32 \cdot f_{clk\_datos} = 32 \cdot 31.250\,\text{kHz} = 1\,\text{MHz}$ y es originada por el módulo "frecuency_divider_clk" mediante:

$$f_{clk} = f_{clk\_NEXYS3}/100 = 1\,\text{MHz}$$

La frecuencia f_{clk} es igual a la frecuencia de muestreo de las señales: $b(t)$, $\theta(t)$, $I(t)$ $Q(t)$, $\cos(2\pi f_c t)$, $\text{sen}(2\pi f_c t)$ y $s(t)$.

Los parámetros: R, f_{clk} y f_c configurados en la "arquitectura de pruebas" son los mismos que se configuraron en el "test bench" para ser simulados, lo cual permite comparar ambos resultados de forma cualitativa y cuantitativa.

A continuación se muestran los oscilogramas obtenidos a partir de la implementación física, en los cuales se hace una descripción de las señales medidas.

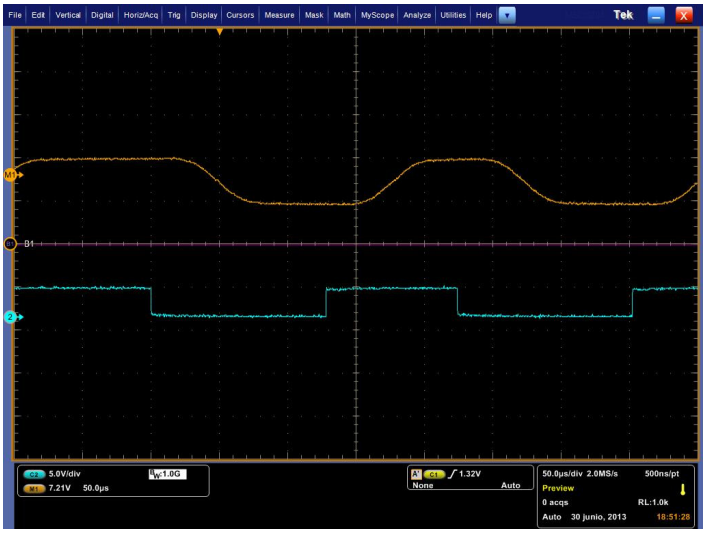


Figura 6.10: Oscilograma 1: Señal de información filtrada $b(t)$ y la señal de información $\{a_k\}$.

En la figura 6.10 se observa la señal de información filtrada $b(t)$ y señal de información digital $\{a_k\}$ Se observa que los cambios bruscos de frecuencia de la señal $\{a_k\}$ son eliminados mediante el filtro pasa bajas Gaussiano; cabe señalar que la señal $b(t)$ esta codificada en NRZ.

La señal $b(t)$ esta retrasada 64 μs $(2T_b)$ respecto a $\{a_k\}$, tiempo de retardo discutido en la subsección 5.1.1. Por lo tanto las señales: $\theta(t)$, $Q(t)$, $\cos(2\pi f_c t)$ y $s(t)$ se observan desfasadas respecto a $\{a_k\}$ en los oscilogramas.

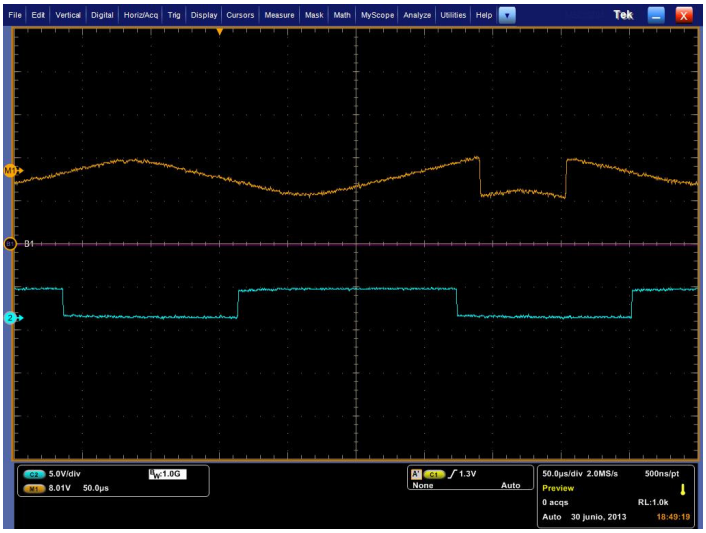


Figura 6.11: Oscilograma 2: Señal de fase $\theta(t)$ y señal de información $\{a_k\}$.

En la figura 6.11 se observa la señal de fase $\theta(t)$, "trellis de fase" o diagrama de fase, donde el cambio de fase por símbolo es $\pm\pi/2$ según el caso.

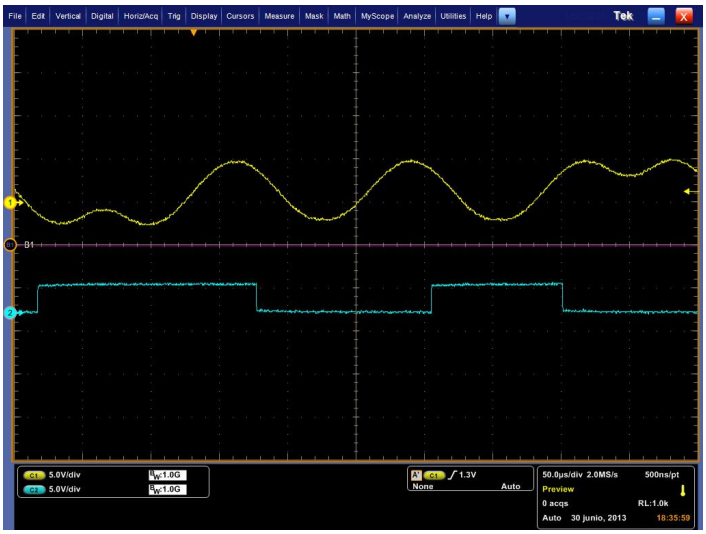


Figura 6.12: Oscilograma 3: Componente en cuadratura $Q(t)$ y señal de información $\{a_k\}$.

En la figura 6.12 se observa la señal en cuadratura $Q(t)$ en banda base.

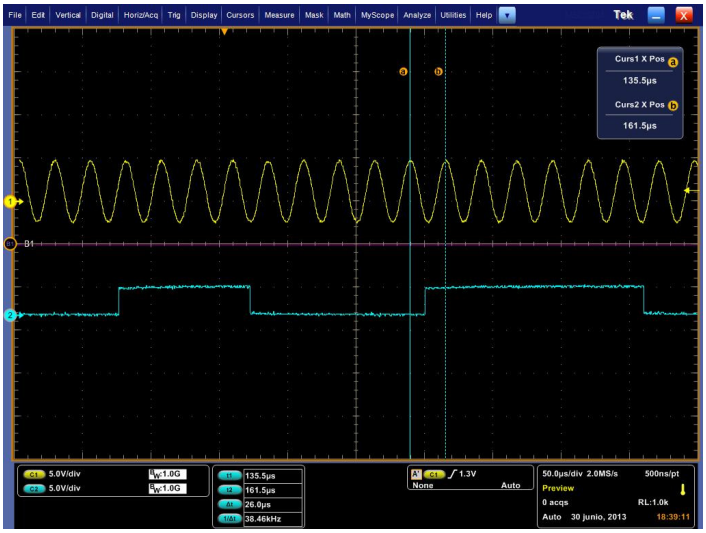


Figura 6.13: Oscilograma 4: $cos(2\pi f_c t)$ y señal de información $\{a_k\}$.

La figura 6.13 se observa la señal portadora $cos(2\pi f_c t)$, donde, f_c medida es de: 38.46 kHz.

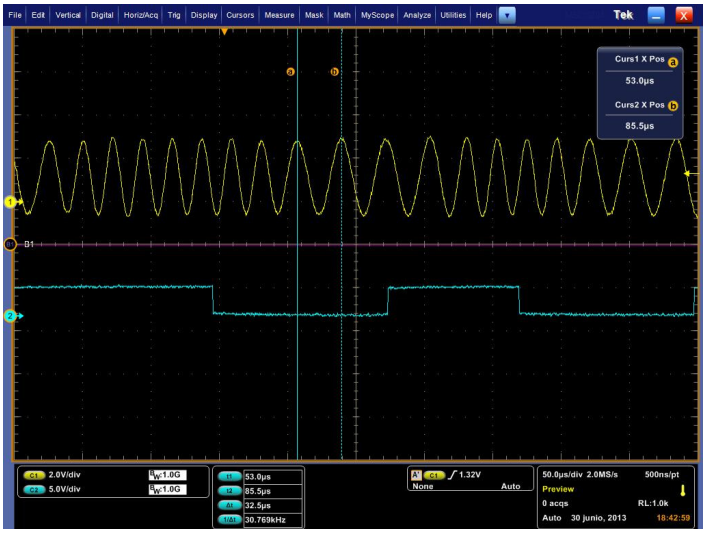


Figura 6.14: Oscilograma 5: Señal de información $\{a_k\}$ y señal modulada $s(t)$.

En la figura 6.14 se observa la señal modulada $s(t)$, donde:
$f_{1_{exp}} \simeq 46.154\,\text{kHz}$ y $f_{2_{exp}} \simeq 30.769\,\text{kHz}$, y sabiendo que:

$$h = \Delta f \cdot T_b$$
$$h = (f_1 - f_2) \cdot T_b$$
$$h_{exp} = 46.154\,[\text{kHz}] - 30.769\,[\text{kHz}] \cdot 32\,[\mu\text{s}]$$
$$h_{exp} \simeq 0.49232$$

Además:

$$f_c = \frac{f_1 + f_2}{2}$$
$$f_{c_{exp}} = \frac{46.154\,[\text{kHz}] + 30.769\,[\text{kHz}]}{2}$$

$$f_{c_{exp}} \simeq 38.46\,\text{kHz}$$

Debido a que $f_{c_{exp}} \approx f_{clk\_fc}$ y $h \approx 0.5$, se concluye que la señal $s(t)$ es una señal modulada en GMSK.

Capítulo 7

Conclusiones, recomendaciones y trabajo a futuro

7.1. Conclusiones

En este trabajo de tesis se propuso la arquitectura de un modulador digital GMSK, la cual se describió en hardware mediante VHDL utilizando la metodología top-down, dicha arquitectura se implementó para un FPGA Spartan 6 y se validó su correcto funcionamiento de manera exitosa en simulaciones lógicas, y de forma experimental tras su implementación física en la tarjeta de desarrollo Nexys 3.

También se concluye que:

- La arquitectura del modulador GMSK puede ser implementado para cualquier FPGA, no importando el fabricante, ya que su descripción en hardware se realizó en VHDL (HDL estándar).
- La arquitectura del modulador se diseñó bajo el enfoque de la optimización de recursos debido a su aplicación en satélites, lo cual se ve reflejado en los reportes de la implementación, resultando el consumo en aproximadamente un 5 % de los recursos del FPGA Spartan 6.
- El tiempo de retardo que posee la arquitectura del modulador GMSK es de $2T_b$, el cual es característico del diseño y no puede ser disminuido.
- Además se puede decir que la arquitectura del modulador GMSK es flexible ante modificaciones, ya que el parámetro BT_b que gobierna el comportamiento del filtro Gaussiano se puede modificar con facilidad cambiando únicamente los valores de las respuestas g_{k_1}, g_{k_2} y g_{k_4} en las LUTs; por lo que se puede obtener un modulador GMSK con $BT_{b\rightarrow}0$ hasta $BT_{b\rightarrow}\infty$ (modulador MSK).

Como conclusión general, este trabajo representa un avance significativo en el desarrollo de sistemas de comunicaciones bajo el concepto de software radio utilizando hardware reconfigurable para procesar señales, el cual puede ser utilizado en diversas aplicaciones que necesiten cierto grado de flexibilidad, grandes velocidades de procesamiento y ahorro en espacio.

7.2. Recomendaciones

Para mejorar la arquitectura del modulador GMSK realizada en este trabajo se recomienda:

- Obtener de manera directa la trayectoria correspondiente en el filtro Gaussiano, unificando el proceso de obtención de la trayectoria negativa a partir de su forma positiva y el multiplexado de dichas trayectorias ; con esto se conseguirá que la arquitectura sea más eficiente en procesos.
- También se recomienda trabajar con un mayor número de muestras por señal, con esto se conseguirá disminuir errores de redondeo y truncamiento.
- Si se decide trabajar con un número mayor de muestras se recomienda implementar un módulo CORDIC para el cálculo de las funciones seno y coseno en lugar de las "LUTs seno y coseno", ya que un mayor número de muestras en las LUTs representa un mayor consumo de recursos en el FPGA.

7.3. Trabajo futuro

Como trabajo a futuro queda la implementación en FPGA de las etapas que integran el sistema de comunicaciones digitales, tales como:

- Codificación de fuente.
- Codificación de canal.
- Demodulador
- Decodificación de fuente.
- Decodificación de canal.

Debido a que la aplicación principal es para uso en satélites se sugiere trabajar a futuro en la implementación de técnicas tolerantes a fallas que ayuden a mitigar los errores producidos por efectos de radiación u otros factores considerables del ambiente espacial, ya que debido a su alta escala de integración el FPGA es un dispositivo vulnerable a dichos efectos.

Referencias

[1] Bava, J. A.; Sanz, A. J. *Microondas y Recepción satelital*. CEILP. Argentina. 1999.

[2] Maral, Gérard; Bousquet, Michel. *Satellite Communications Systems: Systems, Techniques and Technologies*. Wiley. 5ta edición. 2009.

[3] Reed, Jeffrey H. *Software RADIO. A modern Approach to Radio Engineering*. Prentice Hall. E.U.A., 2002.

[4] Maxinez, David G.; Alcalá, Jessica. *VHDL El arte de programar sistemas digitales*. Compañía editorial continental. Primera edición. México, 2002.

[5] Romero T., Rene. *Electrónica digital y lógica programable*. Universidad de Guanajuato. México, 2007.

[6] Herrera Pérez, Enrique. *Comunicaciones I: Señales, modulación y transmisión*. Limusa. México, 2004. pp. 149-150

[7] Sallent, Oriol; Valenzuela, José Luis. Agustí, Ramón. *Principios de comunicaciones móviles*. Ediciones UPC. España 2003. pp. 76-77

[8] Sklar, Bernard. *Digital Communications. Fundamental and Applications*. Prentice Hall. Segunda Edición. E.U.A., 2001. p. 169

[9] Tri, T. Ha. *Theory and design of digital communications systems*. Cambridge University Press. 2011.

[10] Das, Apurba. *Digital Communications Principles and System Modelling*. Springer. 2010.

[11] Adionel Guimarães, Dayan. *Digital Transmisión A Simulation-Aided Introduction with VisSim/Comm*. Springer. 2009. pp. 521-524

[12] Medina, J.F.; Patarroyo, J.F; Muñoz, C. *Implementación en FPGA de un sistema de comunicación con modulación GMSK para un laboratorio modular de comunicaciones digitales*. Universidad del Quindío, Colombia, 2011.

[13] Simon, Marvin K. *Bandwidth-Efficient Digital Modulation with Application to Deep-Space Communications*. JPL Publication 00-17. E.U.A. 2001. pp. 57-67

[14] Turletti, Thierry. *GMSK in a nutshell*. Massachusetts Institute of Technology. 1996.

[15] University of Hull. *Appendix D - Digital Modulation an GMSK*. The LINK Personal Communications Programme (PCP).

[16] Kesoulis, M.J; Kourkoulis, C.S.; Lygouras, J.N.; Soudris, D.; Sahalos, J.N. *Design and implementación of a DDS-based multi-carrier GMSK modulator*. International Journal of communications systems. 2009.

[17] Švedek, T.; Herceg, M.; Matić, T. *A Simple Signal Shaper for GMSK/GFSK and MSK Modulator Based on Sigma-Delta Look-up Table*. University of Osijek, Croatia.

[18] Linz, Alfredo; Hendrickson, Alan. *Efficient Implementation of an IQ GMSK Modulator*. IEEE Transactions on circuits and systems - II Analog and digital signal processing, Vol. 43, No. 1, 1996.

[19] Vankka, Jouko. *Direct Digital Synthesizers: Theory, Design and Applications*. Helsinki University of Technology, Finlandia, 2000.

[20] Meyer-Baese, Uwe. *Digital Signal Processing with Field Programmable Gate Arrays*. Springer. 3ra edición. 2007.

Apéndice A

A.1. Representación binaria de números enteros

A.1.1. Enteros no signados

Un número entero no signado X de N bits esta dado por:

$$X = \sum_{j=0}^{N-1} 2^j b_j$$

Donde b_j representa el j bit del número binario X_2 y b_0 es el bit menos significativo (LSB) y b_{N-1} es el bit más significativo (MSB).

El rango de datos que pueden ser representados con N bits de forma entera no signada es:

$$0 \leq X \leq 2^N - 1$$

A continuación se presenta una tabla que muestra la representación entera no signada de un número binario de $N = 4$ bits.

$[b_3 b_2 b_1 b_0]$	X
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Tabla A.1: Representación entera no signada de un número binario de $N = 4$ bits.

A.1.2. Enteros signados

A.1.2.1 Representación magnitud-signo

En la representación de números enteros signados en forma binaria existe la representación "signo-magnitud", en la cual el signo y la magnitud son representados de manera separada. El primer bit b_{N-1} (es decir, el MSB) representa el signo, y los bits restantes representan la magnitud.

$$X = \begin{cases} \sum_{j=0}^{N-2} 2^j b_j, & X \geq 0 \\ \\ -\sum_{j=0}^{N-2} 2^j b_j, & X < 0 \end{cases}$$

El rango de datos que pueden ser representados con N bits de forma "signo-magnitud" es:

$$-(2^{N-1} - 1) \leq X \leq 2^{N-1} - 1$$

A continuación se presenta una tabla que muestra la representación "signo-magnitud" de un número binario de $N = 4$ bits.

$[b_3 b_2 b_1 b_0]$	X
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-0
1001	-1
1010	-2
1011	-3
1100	-4
1101	-5
1110	-6
1111	-7

Tabla A.2: Representación entera signada "signo-magnitud" de un número binario de $N = 4$ bits.

Ventajas y desventajas de la representación en signo-magnitud

- ☑ La mayor ventaja de esta representación es la prevención de *desbordamiento* (overflow).
- ☒ Su desventaja es que resulta muy complejo operar aritméticamente.

A.1.2.2. Representación en complemento a uno

En la representación de números enteros signados en forma binaria existe la representación "complemento a uno", la cual esta dada por:

$$X = \begin{cases} \sum_{j=0}^{N-2} 2^j b_j, & X \geq 0 \\ \\ -2^{N-1} + 1 + \sum_{j=0}^{N-2} 2^j b_j, & X < 0 \end{cases}$$

El rango de datos que pueden ser representados con N bits en "complemento a uno" es:

$$-(2^{N-1} + 1) \leq X \leq 2^{N-1} - 1$$

A continuación se presenta una tabla que muestra la representación en "complemento a uno" de un número binario de $N = 4$ bits.

$[b_3 b_2 b_1 b_0]$	X
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-7
1001	-6
1010	-5
1011	-4
1100	-3
1101	-2
1110	-1
1111	0

Tabla A.2: Representación entera signada "complemento a uno" de un número binario de $N = 4$ bits.

Ventajas y desventajas de la representación en complemento uno

- ✓ La mayores ventajas de esta representación es que posee un rango asimétrico y que permite operar aritméticamente.[1]
- ⊠ Posee doble representación del cero.

[1] Al operar con números representados con complemento a uno se debe sumar el acarreo obtenido al final de la suma/resta realizadas en caso de haberlo obtenido, para conseguir el resultado correcto.

A.1.2.3. Representación en complemento a dos

En la representación de números enteros signados en forma binaria existe la representación "complemento a dos", la cual esta dada por:

$$X = \begin{cases} \sum_{j=0}^{N-2} 2^j b_j, & X \geq 0 \\ \\ -2^{N-1} + \sum_{j=0}^{N-2} 2^j b_j, & X < 0 \end{cases}$$

El rango de datos que pueden ser representados con N bits en "complemento a dos" es:

$$-2^{N-1} \leq X \leq 2^{N-1} - 1$$

A continuación se presenta una tabla que muestra la representación en "complemento a dos" de un número binario de $N = 4$ bits.

$[b_3 b_2 b_1 b_0]$	X
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

Tabla A.2: Representación entera signada "complemento a dos" de un número binario de $N = 4$ bits.

Ventajas y desventajas de la representación en complemento dos

- ☑ La mayores ventajas de esta representación es que no posee doble representación del cero y que permite operar aritméticamente.
- ☒ Su desventaja es su rango asimétrico.

A.2. Representación de números reales mediante punto fijo

La representación de números en punto fijo es una generalización de la representación decimal de un número. En esta notación los dígitos de la izquierda del punto decimal representan la parte entera del número, y los dígitos de la derecha representan la parte fraccionaria del número. Por lo que un número real X, puede representarse como:

$$X = \{b_{-A}, ..., b_1 \bullet b_0, b1, ..., b_B\}_r = \sum_{i=-A}^{B} b_i r^{-i}, \qquad 0 \leq b_1 \leq (r-1)$$

Donde b_i representa el dígito, r es la base, A es el número de dígitos enteros, y B es el número de dígitos fraccionarios. Por ejemplo, si queremos representar en decimal el número 432.28, realizamos:

$$(432.28)_{10} = 4 \cdot 10^2 + 3 \cdot 10^1 + 2 \cdot 10^0 + 2 \cdot 10^-1 + 8 \cdot 10^-2$$

La representación de punto fijo consiste en que dado espacio de N bits para almacenar un número, se reservan I bits para almacenar la parte entera del número, F bits para almacenar la parte fraccionaria y un bit de signo (sí el número fraccionario es "signado"), donde $N = I + F$ (número no signado) ó $N = I + F + 1$ (número signado) . De esta forma el punto de la representación fraccionaria queda fijo en la *t-ésima* posición de la secuencia de bits.

0	010	1110
signo	I	F

A.2.1. Representación de números no signados en punto fijo

La representación de números fraccionarios en punto fijo requiere que el punto decimal sea colocado entre dos bits, dividiendo así, una parte entera y una parte fraccionaria:

$$IQF$$

Donde I es el número de bits que representa la parte entera y F es el número de bits que representan la parte fraccionaria. La suma de I y F es igual al tamaño de la palabra N.

El valor de un número fraccionario decimal no signado D representado por un número binario de n bits en formato IQF está dado por:

$$D = \frac{1}{2^F} \sum_{j=0}^{N-1} 2^j b_j$$

Donde b_j representa el j bit del número binario X_2 y b_0 es el bit menos significativo (LSB). Obsérvese que $\sum_{j=0}^{N-1} 2^j b_j$ es la expresión para convertir un número en formato binario"X_2" a formato decimal "X_{10}". Por lo que la ecuación anterior se puede reescribir como:

$$D = \frac{1}{2^F} X_{10}$$

El rango de datos que pueden ser representados con el formato IQF esta dado por:

$$0 \leq D \leq 2^I - 2^{-F}$$

$$o$$

$$0 \leq D \leq \frac{(2^n - 1)}{2^F}$$

Ejemplo: En una representación con formato $2Q6$, se tienen dos bits para representar la parte entera y seis bits para representar la parte fraccionaria, por lo que la representación binaria es:

$$b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 \text{ Representación binaria en 8 bits}$$

$$b_1 b_0 . b_{-1} b_{-2} b_{-3} b_{-4} b_{-5} b_{-6} \text{ Representación binaria en formato } 2Q6$$

El rango de datos que es posible que es posible representar con este formato es:

$$0 \leq D \leq 2^2 - 2^{-6}$$

$$\boxed{0 \leq D \leq 3.9844}$$

Si se tiene el número binario $X = \{11001001\}_2$, la representación del número en punto fijo mediante el formato $2Q6$ es:

$$\boxed{D = \frac{210}{2^6} = 3.1406}$$

Donde "210" es la representación decimal del número binario X.

Para realizar el paso inverso, es decir la conversión de un número decimal a un número binario en formato IQN, se hace lo siguiente:

- ✓ Realizar el producto: $D \cdot 2^F$
- ✓ Redondear a un número entero (Es inevitable un error de cuantización y redondeo)
- ✓ Convertir el número entero a un número binario

Ejemplo: Convertir el número $D = 3.1406$ a un número binario en formato IQN

$$3.1406 \cdot 2^6 = 200.9984$$

$$200.9984 \approx 201$$

$$201_{10} = 11001001_2$$

A.2.2. Representación de números signados en punto fijo

La representación de números signados mediante punto fijo requiere un bit adicional (bit de signo) en comparación a la "representación de números no signados",por lo que, su tamaño de palabra es:

$$N = I + F + 1$$

El valor de un número fraccionario decimal signado D representado por un número binario de N bits en formato IQN está dado por:

$$D = \frac{1}{2^F}\left[-2^{N-1} + \sum_{j=0}^{N-2} 2^j b_j \right]$$

Donde b_j representa el j bit del número binario X_2 y b_0 es el bit menos significativo (LSB). Obsérvese que $-2^{N-1} + \sum_{j=0}^{N-2} 2^j b_j$ es la expresión para convertir un número en formato binario representado en *complemento a dos* "X_2" a formato decimal "X_{10}". Por lo que la ecuación anterior se puede reescribir como:

$$D = \frac{1}{2^F} X_{10}$$

El rango de datos que pueden ser representados con el formato IQF esta dado por:

$$-2^I \leq D \leq 2^I - 2^{-F}$$

Ejemplo: En una representación con formato $2Q6$, se tienen dos bits para representar la parte entera, seis bits para representar la parte fraccionaria y un bit para representar el signo, por lo que la representación binaria es:

$$b_8 b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 \text{ Representación binaria en 9 bits}$$

$$b_s b_1 b_0 . b_{-1} b_{-2} b_{-3} b_{-4} b_{-5} b_{-6} \text{ Representación binaria en formato } 2Q6$$

El rango de datos que es posible que es posible representar con este formato es:

$$-2^2 \leq D \leq 2^2 - 2^{-6}$$

$$\boxed{-4 \leq D \leq 3.9844}$$

Si se tiene el número binario $X = \{100110111\}_2$, la representación del número en punto fijo mediante el formato $2Q6$ es:

$$\boxed{D = \frac{-210}{2^6} = -3.1406}$$

Donde "-210" es la representación decimal del número binario X representado en complemento a dos.

Para realizar el paso inverso, es decir la conversión de un número decimal a un número binario en formato IQN, se hace lo siguiente:

- ✓ Realizar el producto: $D \cdot 2^F$
- ✓ Redondear a un número entero (Es inevitable un error de cuantización y redondeo)
- ✓ Convertir el número entero a un número binario representado en complemento a dos

Ejemplo: Convertir el número $D = -3.1406$ a un número binario en formato IQN

$$-3.1406 \cdot 2^6 = -200.9984$$

$$-200.9984 \approx -201$$

$$-201_{10} = 100110111_{2_{c2}}$$

Ventajas y desventajas de la representación en punto fijo

- ☑ Una de las ventajas que posee la notación en punto fijo es la facilidad de realizar operaciones aritméticas, pues se usa toda la palabra como si fuera un entero y el proceso de colocar el punto decimal se hace al final.
- ☒ La mayor desventaja que posee esta notación es la poca flexibilidad para representar números que tengan muchos dígitos en la parte fraccionaria.

Apéndice B

B.1. Valores almacenados en la LUT seno y LUT coseno del modulador GMSK.

Tabla de valores que fueron almacenados en las LUT seno y LUT coseno implementadas en el modulador GMSK, $n = 128$

i	$\phi = \frac{2\pi i}{n}[rad]$	$\cos(\phi)$	$\text{sen}(\phi)$
0	0	1	0
1	0.049	0.999	0.049
2	0.098	0.995	0.098
3	0.147	0.989	0.147
4	0.196	0.981	0.195
5	0.245	0.970	0.243
6	0.294	0.957	0.290
7	0.344	0.942	0.337
8	0.392	0.924	0.382
9	0.441	0.904	0.427
10	0.491	0.882	0.471
11	0.540	0.858	0.514
12	0.589	0.831	0.556
13	0.638	0.803	0.596
14	0.687	0.773	0.634
15	0.736	0.741	0.672
16	0.785	0.707	0.707
17	0.834	0.672	0.741
18	0.884	0.634	0.773
19	0.933	0.596	0.803
20	0.981	0.556	0.831
21	1.031	0.514	0.858
22	1.078	0.471	0.882
23	1.129	0.428	0.904
24	1.178	0.383	0.924
25	1.227	0.337	0.942
26	1.276	0.290	0.957

i	$\phi = \frac{2\pi i}{n}[rad]$	$\cos(\phi)$	$\text{sen}(\phi)$
27	1.325	0.243	0.970
28	1.374	0.195	0.981
29	1.426	0.147	0.989
30	1.473	0.098	0.995
31	1.522	0.049	0.999
32	1.571	0	1
33	1.620	-0.049	0.999
34	1.669	-0.098	0.995
35	1.718	-0.146	0.989
36	1.761	-0.195	0.981
37	1.816	-0.242	0.97
38	1.865	-0.290	0.957
39	1.914	-0.336	0.942
40	1.963	-0.382	0.924
41	2.013	-0.427	0.904
42	2.062	-0.471	0.882
43	2.111	-0.514	0.858
44	2.160	-0.555	0.831
45	2.209	-0.595	0.803
46	2.258	-0.634	0.773
47	2.307	-0.671	0.741
48	2.356	-0.707	0.707
49	2.405	-0.740	0.672
50	2.454	-0.773	0.634
51	2.503	-0.803	0.596
52	2.553	-0.831	0.556
53	2.602	-0.857	0.514
54	2.651	-0.881	0.471
55	2.670	-0.903	0.428
56	2.749	-0.923	0.383
57	2.798	-0.941	0.337
58	2.847	-0.956	0.290
59	2.896	-0.970	0.243
60	2.945	-0.980	0.195
61	2.994	-0.989	0.147
62	3.043	-0.995	0.098
63	3.0925	-0.998	0.049
64	3.1415	-1	0
65	3.191	-0.998	-0.049
66	3.240	-0.995	-0.098
67	3.289	-0.989	-0.146
68	3.338	-0.980	-0.195

i	$\phi = \frac{2\pi i}{n}[rad]$	$\cos(\phi)$	$\text{sen}(\phi)$
69	3.387	-0.970	-0.242
70	3.436	-0.956	-0.290
71	3.485	-0.941	-0.336
72	3.534	-0.923	-0.382
73	3.583	-0.903	-0.427
74	3.632	-0.881	-0.471
75	3.682	-0.857	-0.514
76	3.731	-0.831	-0.555
77	3.780	-0.803	-0.595
78	3.829	-0.773	-0.634
79	3.878	-0.740	-0.671
80	3.927	-0.707	-0.707
81	3.976	-0.671	-0.740
82	4.025	-0.634	-0.773
83	4.074	-0.595	-0.803
84	4.123	-0.555	-0.831
85	4.172	-0.514	-0.857
86	4.222	-0.471	-0.881
87	4.271	-0.427	-0.903
88	4.320	-0.382	-0.923
89	4.369	-0.336	-0.941
90	4.418	-0.290	-0.956
91	4.467	-0.242	-0.970
92	4.516	-0.195	-0.98
93	4.565	-0.146	-0.989
94	4.614	-0.098	-0.995
95	4.663	-0.049	-0.998
96	4.712	0	-1
97	4.761	0.049	-0.998
98	4.810	0.098	-0.995
99	4.860	0.147	-0.989
100	4.909	0.195	-0.98
101	4.958	0.243	-0.97
102	5.007	0.0.290	-0.956
103	5.056	0.337	-0.941
104	5.105	0.383	-0.923
105	5.154	0.428	-0.903
106	5.203	0.471	-0.881
107	5.252	0.514	-0.857
108	5.301	0.556	-0.831
109	5.351	0.596	-0.803
110	5.400	0.6343	-0.773

i	$\phi = \frac{2\pi i}{n}[rad]$	$\cos(\phi)$	$\text{sen}(\phi)$
111	5.449	0.672	-0.740
112	5.498	0.707	-0.707
113	5.547	0.741	-0.671
114	5.596	0.773	-0.634
115	5.645	0.803	-0.595
116	5.694	0.831	-0.555
117	5.743	0.858	-0.514
118	5.792	0.882	-0.471
119	5.841	0.904	-0.427
120	5.890	0.9239	-0.382
121	5.940	0.942	-0.336
122	5.989	0.957	-0.29
123	6.038	0.97	-0.242
124	6.087	0.9807	-0.195
125	6.136	0.989	-0.146
126	6.185	0.995	-0.098
127	6.234	0.999	-0.049

B.2. Valores almacenados en la LUT seno y LUT coseno del NCO.

Tabla de valores que fueron almacenados en las LUT seno y LUT coseno implementadas en el NCO, $n = 26$

i	$\phi = \frac{2\pi i}{n}[rad]$	$\cos(\phi)$	$\text{sen}(\phi)$
0	0	1	0
1	0242	0.971	0.239
2	0.483	0.885	0.465
3	0.725	0.749	0.663
4	0.967	0.568	0.823
5	1.201	0.355	0.935
6	1.445	0.121	0.993
7	1.692	-0.120	0.993
8	1.993	-0.354	0.935
9	2.175	-0.568	0.823
10	2.412	-0.748	0.663
11	2.6582	-0.885	0.465
12	2.900	-0.970	0.239
13	3.142	-1	0
14	3.383	-0.970	-0.239
15	4.108	-0.885	-0.464
16	3.867	-0.748	-0.663
17	4.108	-0.568	-0.822
18	4.350	-0.354	-0.935
19	4.592	-0.120	-0.992
20	4.833	0.121	-0.992
21	5.075	0.355	-0.935
22	5.317	0.568	-0.882
23	5.559	0.749	-0.633
24	5.800	0.886	-0.464
25	6.0415	0.971	-0.239

Apéndice C

C.1. Código VHDL de la arquitectura del modulador GMSK.

```
1  ----------------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:    04/02/2013
5  -- Design Name:    GMSK Modulator
6  -- Module Name:    GMSKmodulator - Behavioral
7  -- Project Name:   GMSK Modulador
8  ----------------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11
12 entity GMSKmodulator is
13 port (
14      data           :   in  std_logic;
15      clk            :   in  std_logic;
16      reset_system   :   in  std_logic;
17      clk_fc         :   in  std_logic;
18      modulated      :  out  std_logic_vector(15 downto 0)
19      );
20 end GMSKmodulator;
21
22 architecture Behavioral of GMSKmodulator is
23
24 component gaussian_filter
25 port (
26      gf_in      :   in  std_logic;
27      clk_gf     :   in  std_logic;
28      reset_gf   :   in  std_logic;
29      gf_out     :  out  std_logic_vector(9 downto 0)
30      );
31 end component;
32
33
34
```

```
35
36 component integrator
37 port (
38       int_in      :    in  std_logic_vector(9 downto 0);
39       clk_int     :    in  std_logic;
40       reset_int   :    in  std_logic;
41       int_out     :   out  std_logic_vector(15 downto 0)
42       );
43 end component;
44
45 component cos_phi
46 port (
47       phase_in    :    in  std_logic_vector (6 downto 0);
48       cos_out     :   out  std_logic_vector (7 downto 0)
49       );
50 end component;
51
52 component sin_phi
53 port (
54       phase_in    :    in  std_logic_vector (6 downto 0);
55       sin_out     :   out  std_logic_vector (7 downto 0)
56       );
57 end component;
58
59 component NCO
60 port (
61       clk_nco     :    in  std_logic;
62       reset_nco   :    in  std_logic;
63       cos2pifct   :   out  std_logic_vector(7 downto 0);
64       sin2pifct   :   out  std_logic_vector(7 downto 0)
65       );
66 end component;
67
68 component multiplier
69 port (
70       mult_in1    :    in  std_logic_vector(7 downto 0);
71       mult_in2    :    in  std_logic_vector(7 downto 0);
72       mult_out    :   out  std_logic_vector(15 downto 0)
73       );
74 end component;
75
76 component adder
77 port (
78       adder_in1   :    in  std_logic_vector(15 downto 0);
79       adder_in2   :    in  std_logic_vector(15 downto 0);
80       adder_out   :   out  std_logic_vector(15 downto 0)
81       );
82 end component;
83
```

```
84
85  signal gf_to_int          :   std_logic_vector(9 downto 0);
86  signal aux_int_out        :   std_logic_vector(15 downto 0);
87  signal phase              :   std_logic_vector(6 downto 0);
88  signal cos_to_mult1       :   std_logic_vector(7 downto 0);
89  signal sin_to_mult2       :   std_logic_vector(7 downto 0);
90  signal cos_nco_to_mult1   :   std_logic_vector(7 downto 0);
91  signal sin_nco_to_mult2   :   std_logic_vector(7 downto 0);
92  signal mult1_to_adder     :   std_logic_vector(15 downto 0);
93  signal mult2_to_adder     :   std_logic_vector(15 downto 0);
94
95  begin
96
97  U1 : gaussian_filter  port map(data,clk_data,reset_system,gf_to_int);
98  U2 : integrator       port map(gf_to_int,clk_data,reset_system,
99                        aux_int_out);
100 U3 : cos_phi          port map(phase,cos_to_mult1);
101 U4 : sin_phi          port map(phase,sin_to_mult2);
102 U5 : NCO              port map(clk_fc,reset_system,cos_nco_to_mult1,
103                       sin_nco_to_mult2);
104 U6 : multiplier       port map(cos_to_mult1,cos_nco_to_mult1,
105                       mult1_to_adder);
106 U7 : multiplier       port map(sin_to_mult2,sin_nco_to_mult2,
107                       mult2_to_adder);
108 U8 : adder            port map(mult1_to_adder,mult2_to_adder,
109                       modulated);
110
111 phase <=  aux_int_out(15 downto 9);
112
113 end Behavioral;
```

```
1  ----------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:     04/02/2013
5  -- Design Name:     GMSK Modulator
6  -- Module Name:     gaussian_filter - Behavioral
7  -- Project Name:    GMSK Modulador
8  ----------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11
12 entity gaussian_filter is
13 port (
14      gf_in      :   in  std_logic;
15      clk_gf     :   in  std_logic;
16      reset_gf   :   in  std_logic;
17      gf_out     :  out  std_logic_vector(9 downto 0)
18      );
19 end gaussian_filter;
20
21 architecture Behavioral of gaussian_filter is
22
23 component frequency_divider
24 port (
25      clk_fd     :   in  std_logic;
26      reset_fd   :   in  std_logic;
27      clk_fd_32  :  out  std_logic
28      );
29 end component;
30
31 component shift_register_sipo
32 port (
33      data_si    :      in  std_logic;
34      clk_sipo   :      in  std_logic;
35      reset_sipo :      in  std_logic;
36      data_po    :   inout  std_logic_vector(2 downto 0)
37      );
38 end component;
39
40 component counter
41 port (
42      clk_counter    :   in  std_logic;
43      reset_counter  :   in  std_logic;
44      counter_out    :  out  std_logic_vector(4 downto 0)
45      );
46 end component;
47
48
49
```

```
50 component address_decoder
51 port (
52       decoder_in             :  in   std_logic_vector(2 downto 0);
53       trajectory_sign        :  out  std_logic;
54       trajectory_selection   :  out  std_logic_vector(1 downto 0);
55       trajectory_sense       :  out  std_logic
56       );
57 end component;
58
59 component XOR_sense
60 port (
61       data_in_xor    :   in  std_logic_vector(4 downto 0);
62       sense_in       :   in  std_logic;
63       data_out_xor   :  out  std_logic_vector(4 downto 0)
64       );
65 end component;
66
67 component GT_LUT
68 port (
69       LUT_select   :   in  std_logic_vector (1 downto 0);
70       address      :   in  std_logic_vector (4 downto 0);
71       LUT_out      :  out  std_logic_vector (9 downto 0)
72       );
73 end component;
74
75 component comp2
76 port (
77       data_in_c2    :   in  std_logic_vector (9 downto 0);
78       data_out_c2   :  out  std_logic_vector (9 downto 0)
79       );
80 end component;
81
82 component mux
83 port (
84       sign_select      : in  std_logic;
85       negative_in      : in  std_logic_vector (9 downto 0);
86       positive_in      : in  std_logic_vector (9 downto 0);
87       trajectory_out   : out  std_logic_vector (9 downto 0)
88       );
89 end component;
90
91 signal fq_to_sipo             :    std_logic;
92 signal sipo_to_ad             :    std_logic_vector(2 downto 0);
93 signal counter_to_xorsense    :    std_logic_vector(4 downto 0);
94 signal ad_to_xorsense         :    std_logic;
95 signal xorsense_to_GTLUT      :    std_logic_vector(4 downto 0);
96 signal ad_to_GTLUT            :    std_logic_vector(1 downto 0);
97 signal GTLUT_to_c2            :    std_logic_vector(9 downto 0);
98 signal c2_to_mux              :    std_logic_vector(9 downto 0);
```

```
 99 signal ad_to_mux              :     std_logic;
100
101 begin
102
103 U11 : frequency_divider      port map(clk_gf,reset_gf,fq_to_sipo);
104 U12 : shift_register_sipo    port map(gf_in,fq_to_sipo,reset_gf,
105                               sipo_to_ad);
106 U13 : counter                port map(clk_gf,reset_gf,
107                               counter_to_xorsense);
108 U14 : address_decoder        port map(sipo_to_ad,ad_to_mux,ad_to_GTLUT,
109                               ad_to_xorsense);
110 U15 : XOR_sense              port map(counter_to_xorsense,
111                                ad_to_xorsense,xorsense_to_GTLUT);
112 U16 : GT_LUT                 port map(ad_to_GTLUT,xorsense_to_GTLUT,
113                               GTLUT_to_c2);
114 U17 : comp2                  port map(GTLUT_to_c2,c2_to_mux);
115 U18 : mux                    port map(ad_to_mux,c2_to_mux,GTLUT_to_c2,
116                               gf_out);
117
118 end Behavioral;
```

```
1  ----------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:     04/02/2013
5  -- Design Name:     GMSK Modulator
6  -- Module Name:     (gaussian_filter) frequency_divider - Behavioral
7  -- Project Name:    GMSK Modulador
8  ----------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11
12 entity frequency_divider is
13 port (
14      clk_fd      :   in  std_logic;
15      reset_fd    :   in  std_logic;
16      clk_fd_32   :  out  std_logic
17                           );
18 end frequency_divider;
19
20 architecture Behavioral of frequency_divider is
21
22 signal temporal : std_logic:='0';
23 signal count    : integer range 0 to 15 := 0;
24
25 begin
26
27   process(reset_fd,clk_fd)
28   begin
29      if(reset_fd='1') then
30         temporal <= '0';
31         count <= 0;
32      elsif rising_edge(clk_fd) then
33         if(count = 15) then
34            temporal <= not(temporal);
35            count <= 0;
36         else
37            count <= count + 1;
38         end if;
39      end if;
40    end process;
41    clk_fd_32 <= temporal;
42
43 end Behavioral;
```

```
1  ----------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:     04/02/2013
5  -- Design Name:     GMSK Modulator
6  -- Module Name:     (gaussian_filter) shift_register_sipo - Behavioral
7  -- Project Name:    GMSK Modulador
8  ----------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11
12 entity shift_register_sipo is
13 port (
14      data_si     :      in  std_logic;
15      clk_sipo    :      in  std_logic;
16      reset_sipo  :      in  std_logic;
17      data_po     :   inout  std_logic_vector(2 downto 0)
18      );
19 end shift_register_sipo;
20
21 architecture Behavioral of shift_register_sipo is
22 begin
23
24   process(reset_sipo,clk_sipo)
25   begin
26      if (reset_sipo = '1') then
27         data_po <= "000";
28      elsif(clk_sipo='0' and clk_sipo'event) then
29          data_po(2 downto 1) <= data_po(1 downto 0);
30          data_po(0) <= data_si;
31      end if;
32   end process;
33
34 end Behavioral;
```

```
1  ----------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:     04/02/2013
5  -- Design Name:     GMSK Modulator
6  -- Module Name:     (gaussian_filter) counter - Behavioral
7  -- Project Name:    GMSK Modulador
8  ----------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11 use ieee.numeric_std.all;
12
13 entity counter is
14 port (
15      clk_counter     :   in  std_logic;
16      reset_counter   :   in  std_logic;
17      counter_out     :  out  std_logic_vector(4 downto 0)
18      );
19 end counter;
20
21 architecture Behavioral of counter is
22
23 signal count_int : unsigned(4 downto 0):=(others => '0');
24
25 begin
26
27   process (reset_counter,clk_counter)
28   begin
29      if (reset_counter = '1') then
30         count_int <= "00000";
31      elsif(clk_counter='1' and clk_counter'event) then
32         count_int <= count_int + 1;
33      end if;
34   end process;
35   counter_out <= std_logic_vector(count_int);
36
37 end Behavioral;
```

```
1  ----------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:     04/02/2013
5  -- Design Name:     GMSK Modulator
6  -- Module Name:     (gaussian_filter) address_decoder - Behavioral
7  -- Project Name:    GMSK Modulador
8  ----------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11
12 entity address_decoder is
13 port (
14      decoder_in             :  in   std_logic_vector(2 downto 0);
15      trajectory_sign        :  out  std_logic;
16      trajectory_selection   :  out  std_logic_vector(1 downto 0);
17      trajectory_sense       :  out  std_logic
18      );
19 end address_decoder;
20
21 architecture Behavioral of address_decoder is
22 begin
23
24 trajectory_selection(1) <=
25      (decoder_in(2)   and   decoder_in(1)   and   decoder_in(0))   or
26 (not(decoder_in(2))) and (not(decoder_in(1))) and (not(decoder_in(0)));
27
28 trajectory_selection(0) <=
29 ((not(decoder_in(2))) and decoder_in(1) and (not(decoder_in(0))))  or
30      (decoder_in(2) and not (decoder_in(1)) and decoder_in(0)) ;
31
32 trajectory_sense <=
33 ((not(decoder_in(2))) and (not(decoder_in(1))) and decoder_in(0))  or
34 (decoder_in(2) and decoder_in(1) and (not(decoder_in(0))));
35
36 trajectory_sign <= not(decoder_in(1));
37
38 end Behavioral;
```

```
1  ----------------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:     04/02/2013
5  -- Design Name:     GMSK Modulator
6  -- Module Name:     (gaussian_filter) XOR_sense - Behavioral
7  -- Project Name:    GMSK Modulador
8  ----------------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11
12 entity XOR_sense is
13 port (
14       data_in_xor    :    in  std_logic_vector(4 downto 0);
15       sense_in       :    in  std_logic;
16       data_out_xor   :   out  std_logic_vector(4 downto 0)
17       );
18 end XOR_sense;
19
20 architecture Behavioral of XOR_sense is
21 begin
22
23   process(sense_in,data_in_xor)
24   begin
25      if(sense_in = '0') then
26         data_out_xor <= data_in_xor;
27      else
28         data_out_xor <= not(data_in_xor);
29      end if;
30   end process;
31
32 end Behavioral;
```

```
1  ----------------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:    04/02/2013
5  -- Design Name:    GMSK Modulator
6  -- Module Name:    (gaussian_filter) GT_LUT - Behavioral
7  -- Project Name:   GMSK Modulador
8  ----------------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11
12 entity GT_LUT is
13 port (
14      LUT_select  :   in  std_logic_vector (1 downto 0);
15      address     :   in  std_logic_vector (4 downto 0);
16      LUT_out     :  out  std_logic_vector (9 downto 0)
17      );
18 end GT_LUT;
19
20 architecture Behavioral of GT_LUT is
21 begin
22
23   process(LUT_select,address)
24   begin
25      if(LUT_select="00") then
26         case address is
27            when "00000" =>  LUT_out <= "0000000000";
28            when "00001" =>  LUT_out <= "0000011110";
29            when "00010" =>  LUT_out <= "0000111100";
30            when "00011" =>  LUT_out <= "0001011010";
31            when "00100" =>  LUT_out <= "0001111000";
32            when "00101" =>  LUT_out <= "0010010101";
33            when "00110" =>  LUT_out <= "0010110010";
34            when "00111" =>  LUT_out <= "0011001101";
35            when "01000" =>  LUT_out <= "0011101000";
36            when "01001" =>  LUT_out <= "0100000001";
37            when "01010" =>  LUT_out <= "0100011010";
38            when "01011" =>  LUT_out <= "0100110001";
39            when "01100" =>  LUT_out <= "0101000111";
40            when "01101" =>  LUT_out <= "0101011011";
41            when "01110" =>  LUT_out <= "0101101110";
42            when "01111" =>  LUT_out <= "0110000000";
43            when "10000" =>  LUT_out <= "0110010001";
44            when "10001" =>  LUT_out <= "0110100000";
45            when "10010" =>  LUT_out <= "0110101101";
46            when "10011" =>  LUT_out <= "0110111010";
47            when "10100" =>  LUT_out <= "0111000101";
48            when "10101" =>  LUT_out <= "0111001111";
49            when "10110" =>  LUT_out <= "0111011000";
```

```
50              when "10111" =>  LUT_out <= "0111100000";
51              when "11000" =>  LUT_out <= "0111100111";
52              when "11001" =>  LUT_out <= "0111101101";
53              when "11010" =>  LUT_out <= "0111110010";
54              when "11011" =>  LUT_out <= "0111110110";
55              when "11100" =>  LUT_out <= "0111111010";
56              when "11101" =>  LUT_out <= "0111111100";
57              when "11110" =>  LUT_out <= "0111111110";
58              when "11111" =>  LUT_out <= "0111111111";
59              when others=>null;
60          end case;
61
62      elsif(LUT_select="01") then
63          case address is
64              when "00000" =>  LUT_out <= "0000000000";
65              when "00001" =>  LUT_out <= "0000011010";
66              when "00010" =>  LUT_out <= "0000110011";
67              when "00011" =>  LUT_out <= "0001001101";
68              when "00100" =>  LUT_out <= "0001100110";
69              when "00101" =>  LUT_out <= "0001111011";
70              when "00110" =>  LUT_out <= "0010010100";
71              when "00111" =>  LUT_out <= "0010101001";
72              when "01000" =>  LUT_out <= "0010111000";
73              when "01001" =>  LUT_out <= "0011001000";
74              when "01010" =>  LUT_out <= "0011010111";
75              when "01011" =>  LUT_out <= "0011100110";
76              when "01100" =>  LUT_out <= "0011110001";
77              when "01101" =>  LUT_out <= "0011110110";
78              when "01110" =>  LUT_out <= "0011111011";
79              when "01111" =>  LUT_out <= "0100000000";
80              when "10000" =>  LUT_out <= "0100000000";
81              when "10001" =>  LUT_out <= "0011111011";
82              when "10010" =>  LUT_out <= "0011110110";
83              when "10011" =>  LUT_out <= "0011110001";
84              when "10100" =>  LUT_out <= "0011100110";
85              when "10101" =>  LUT_out <= "0011010111";
86              when "10110" =>  LUT_out <= "0011001000";
87              when "10111" =>  LUT_out <= "0010111000";
88              when "11000" =>  LUT_out <= "0010101001";
89              when "11001" =>  LUT_out <= "0010010100";
90              when "11010" =>  LUT_out <= "0001111011";
91              when "11011" =>  LUT_out <= "0001100110";
92              when "11100" =>  LUT_out <= "0001001101";
93              when "11101" =>  LUT_out <= "0000110011";
94              when "11110" =>  LUT_out <= "0000011010";
95              when "11111" =>  LUT_out <= "0000000000";
96              when others=>null;
97          end case;
98
```

```
 99      elsif(LUT_select="10") then
100         case address is
101            when "00000" =>  LUT_out <= "0111111111";
102            when "00001" =>  LUT_out <= "0111111111";
103            when "00010" =>  LUT_out <= "0111111111";
104            when "00011" =>  LUT_out <= "0111111111";
105            when "00100" =>  LUT_out <= "0111111111";
106            when "00101" =>  LUT_out <= "0111111111";
107            when "00110" =>  LUT_out <= "0111111111";
108            when "00111" =>  LUT_out <= "0111111111";
109            when "01000" =>  LUT_out <= "0111111111";
110            when "01001" =>  LUT_out <= "0111111111";
111            when "01010" =>  LUT_out <= "0111111111";
112            when "01011" =>  LUT_out <= "0111111111";
113            when "01100" =>  LUT_out <= "0111111111";
114            when "01101" =>  LUT_out <= "0111111111";
115            when "01110" =>  LUT_out <= "0111111111";
116            when "01111" =>  LUT_out <= "0111111111";
117            when "10000" =>  LUT_out <= "0111111111";
118            when "10001" =>  LUT_out <= "0111111111";
119            when "10010" =>  LUT_out <= "0111111111";
120            when "10011" =>  LUT_out <= "0111111111";
121            when "10100" =>  LUT_out <= "0111111111";
122            when "10101" =>  LUT_out <= "0111111111";
123            when "10110" =>  LUT_out <= "0111111111";
124            when "10111" =>  LUT_out <= "0111111111";
125            when "11000" =>  LUT_out <= "0111111111";
126            when "11001" =>  LUT_out <= "0111111111";
127            when "11010" =>  LUT_out <= "0111111111";
128            when "11011" =>  LUT_out <= "0111111111";
129            when "11100" =>  LUT_out <= "0111111111";
130            when "11101" =>  LUT_out <= "0111111111";
131            when "11110" =>  LUT_out <= "0111111111";
132            when "11111" =>  LUT_out <= "0111111111";
133            when others=>null;
134         end case;
135
136      else
137         case address is
138            when "00000" =>  LUT_out <= "0000000000";
139            when "00001" =>  LUT_out <= "0000000000";
140            when "00010" =>  LUT_out <= "0000000000";
141            when "00011" =>  LUT_out <= "0000000000";
142            when "00100" =>  LUT_out <= "0000000000";
143            when "00101" =>  LUT_out <= "0000000000";
144            when "00110" =>  LUT_out <= "0000000000";
145            when "00111" =>  LUT_out <= "0000000000";
146            when "01000" =>  LUT_out <= "0000000000";
147            when "01001" =>  LUT_out <= "0000000000";
```

```
148             when "01010" =>  LUT_out <= "0000000000";
149             when "01011" =>  LUT_out <= "0000000000";
150             when "01100" =>  LUT_out <= "0000000000";
151             when "01101" =>  LUT_out <= "0000000000";
152             when "01110" =>  LUT_out <= "0000000000";
153             when "01111" =>  LUT_out <= "0000000000";
154             when "10000" =>  LUT_out <= "0000000000";
155             when "10001" =>  LUT_out <= "0000000000";
156             when "10010" =>  LUT_out <= "0000000000";
157             when "10011" =>  LUT_out <= "0000000000";
158             when "10100" =>  LUT_out <= "0000000000";
159             when "10101" =>  LUT_out <= "0000000000";
160             when "10110" =>  LUT_out <= "0000000000";
161             when "10111" =>  LUT_out <= "0000000000";
162             when "11000" =>  LUT_out <= "0000000000";
163             when "11001" =>  LUT_out <= "0000000000";
164             when "11010" =>  LUT_out <= "0000000000";
165             when "11011" =>  LUT_out <= "0000000000";
166             when "11100" =>  LUT_out <= "0000000000";
167             when "11101" =>  LUT_out <= "0000000000";
168             when "11110" =>  LUT_out <= "0000000000";
169             when "11111" =>  LUT_out <= "0000000000";
170             when others=>null;
171         end case;
172 
173     end if;
174   end process;
175 
176 end Behavioral;
```

```
1  ----------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:     04/02/2013
5  -- Design Name:     GMSK Modulator
6  -- Module Name:     (gaussian_filter) comp2 - Behavioral
7  -- Project Name:    GMSK Modulador
8  ----------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11
12 entity comp2 is
13 port (
14      data_in_c2    :   in  std_logic_vector (9 downto 0);
15      data_out_c2   :  out  std_logic_vector (9 downto 0)
16      );
17 end comp2;
18
19 architecture Behavioral of comp2 is
20
21 component comp1
22 port (
23      data_in_c1    :  in  std_logic_vector(9 downto 0);
24      );
25 end component;
26
27 component adderc2
28 port (
29      adder_in    :  in   std_logic_vector(9 downto 0);
30      adder_out   :  out  std_logic_vector(9 downto 0)
31      );
32 end component;
33
34 signal comp1_to_adder    :   std_logic_vector(9 downto 0);
35
36 begin
37
38 U171  : comp1     port map(data_in_c2,comp1_to_adder);
39 U172  : adderc2   port map(comp1_to_adder,data_out_c2);
40
41 end Behavioral;
```

```
1  ----------------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:     04/02/2013
5  -- Design Name:     GMSK Modulator
6  -- Module Name:     (gaussian_filter) comp1 - Behavioral
7  -- Project Name:    GMSK Modulador
8  ----------------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11
12 entity comp1 is
13 port (
14       data_in_c1    :  in  std_logic_vector(9 downto 0);
15       data_out_c1   :  out std_logic_vector(9 downto 0)
16       );
17 end comp1;
18
19 architecture Behavioral of comp1 is
20 begin
21
22    data_out_c1 <= not(data_in_c1);
23
24 end Behavioral;
```

```
1  ----------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:     04/02/2013
5  -- Design Name:     GMSK Modulator
6  -- Module Name:     (gaussian_filter) adderc2 - Behavioral
7  -- Project Name:    GMSK Modulador
8  ----------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11 use ieee.numeric_std.all;
12
13 entity adderc2 is
14 port (
15      adder_in    :   in  std_logic_vector(9 downto 0);
16      adder_out   :  out  std_logic_vector(9 downto 0)
17      );
18 end adderc2;
19
20 architecture Behavioral of adderc2 is
21
22 signal temp    : signed(9 downto 0):= "0000000000";
23
24 begin
25
26         temp <= signed(adder_in) + "0000000001";
27    adder_out <= std_logic_vector(temp);
28
29 end Behavioral;
```

```
1  ----------------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:    04/02/2013
5  -- Design Name:    GMSK Modulator
6  -- Module Name:    (gaussian_filter) mux - Behavioral
7  -- Project Name:   GMSK Modulador
8  ----------------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11
12 entity mux is
13 port (
14       sign_select     :  in  std_logic;
15       negative_in     :  in  std_logic_vector (9 downto 0);
16       positive_in     :  in  std_logic_vector (9 downto 0);
17       trajectory_out  : out  std_logic_vector (9 downto 0)
18       );
19 end mux;
20
21 architecture Behavioral of mux is
22 begin
23
24   process(sign_select, positive_in,negative_in)
25   begin
26      case sign_select is
27         when '0' => trajectory_out <= positive_in;
28         when '1' => trajectory_out <= negative_in;
29         when others => trajectory_out <= "0000000000";
30      end case;
31   end process;
32
33 end Behavioral;
```

```
1  ----------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:     04/02/2013
5  -- Design Name:     GMSK Modulator
6  -- Module Name:     integrator - Behavioral
7  -- Project Name:    GMSK Modulador
8  ----------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11 use ieee.numeric_std.all;
12
13 entity integrator is
14 port (
15      int_in      :   in  std_logic_vector(9 downto 0);
16      clk_int     :   in  std_logic;
17      reset_int   :   in  std_logic;
18      int_out     :  out  std_logic_vector(15 downto 0)
19      );
20 end integrator;
21
22 architecture Behavioral of integrator is
23
24 signal tmp  : signed(15 downto 0):= "0000000000000000";
25
26 begin
27
28   process (reset_int,clk_int)
29   begin
30      if (reset_int='1') then
31         tmp <= "0000000000000000";
32      elsif (clk_int'event and clk_int='1') then
33         tmp <= tmp + signed(int_in);
34      end if;
35   end process;
36   int_out <= std_logic_vector(tmp);
37
38 end Behavioral;
```

```
1  ----------------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:    04/02/2013
5  -- Design Name:    GMSK Modulator
6  -- Module Name:    cos_phi - Behavioral
7  -- Project Name:   GMSK Modulador
8  ----------------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11
12 entity cos_phi is
13 port (
14      phase_in        :   in  std_logic_vector (6 downto 0);
15      cos_out         :  out  std_logic_vector (7 downto 0)
16      );
17 end cos_phi;
18
19 architecture Behavioral of cos_phi is
20 begin
21
22   process(phase_in)
23   begin
24      case phase_in is
25         when "0111111" =>  cos_out <= "01111111";
26         when "0111110" =>  cos_out <= "01111110";
27         when "0111101" =>  cos_out <= "01111110";
28         when "0111100" =>  cos_out <= "01111101";
29         when "0111011" =>  cos_out <= "01111100";
30         when "0111010" =>  cos_out <= "01111011";
31         when "0111001" =>  cos_out <= "01111001";
32         when "0111000" =>  cos_out <= "01110111";
33         when "0110111" =>  cos_out <= "01110101";
34         when "0110110" =>  cos_out <= "01110010";
35         when "0110101" =>  cos_out <= "01101111";
36         when "0110100" =>  cos_out <= "01101100";
37         when "0110011" =>  cos_out <= "01101001";
38         when "0110010" =>  cos_out <= "01100101";
39         when "0110001" =>  cos_out <= "01100001";
40         when "0110000" =>  cos_out <= "01011101";
41         when "0101111" =>  cos_out <= "01011001";
42         when "0101110" =>  cos_out <= "01010100";
43         when "0101101" =>  cos_out <= "01001111";
44         when "0101100" =>  cos_out <= "01001010";
45         when "0101011" =>  cos_out <= "01000101";
46         when "0101010" =>  cos_out <= "01000000";
47         when "0101001" =>  cos_out <= "00111010";
48         when "0101000" =>  cos_out <= "00110101";
49         when "0100111" =>  cos_out <= "00101111";
```

```
50          when "0100110" =>  cos_out <= "00101001";
51          when "0100101" =>  cos_out <= "00100011";
52          when "0100100" =>  cos_out <= "00011101";
53          when "0100011" =>  cos_out <= "00010111";
54          when "0100010" =>  cos_out <= "00010001";
55          when "0100001" =>  cos_out <= "00001010";
56          when "0100000" =>  cos_out <= "00000100";
57          when "0011111" =>  cos_out <= "11111111";
58          when "0011110" =>  cos_out <= "11111100";
59          when "0011101" =>  cos_out <= "11110110";
60          when "0011100" =>  cos_out <= "11101111";
61          when "0011011" =>  cos_out <= "11101001";
62          when "0011010" =>  cos_out <= "11100011";
63          when "0011001" =>  cos_out <= "11011101";
64          when "0011000" =>  cos_out <= "11010111";
65          when "0010111" =>  cos_out <= "11010001";
66          when "0010110" =>  cos_out <= "11001011";
67          when "0010101" =>  cos_out <= "11000110";
68          when "0010100" =>  cos_out <= "11000000";
69          when "0010011" =>  cos_out <= "10111011";
70          when "0010010" =>  cos_out <= "10110110";
71          when "0010001" =>  cos_out <= "10110001";
72          when "0010000" =>  cos_out <= "10101100";
73          when "0001111" =>  cos_out <= "10100111";
74          when "0001110" =>  cos_out <= "10100011";
75          when "0001101" =>  cos_out <= "10011111";
76          when "0001100" =>  cos_out <= "10011011";
77          when "0001011" =>  cos_out <= "10010111";
78          when "0001010" =>  cos_out <= "10010100";
79          when "0001001" =>  cos_out <= "10010001";
80          when "0001000" =>  cos_out <= "10001110";
81          when "0000111" =>  cos_out <= "10001011";
82          when "0000110" =>  cos_out <= "10001001";
83          when "0000101" =>  cos_out <= "10000111";
84          when "0000100" =>  cos_out <= "10000101";
85          when "0000011" =>  cos_out <= "10000100";
86          when "0000010" =>  cos_out <= "10000011";
87          when "0000001" =>  cos_out <= "10000010";
88          when "0000000" =>  cos_out <= "10000010";
89
90          when "1111111" =>  cos_out <= "10000001";
91          when "1111110" =>  cos_out <= "10000000";
92          when "1111101" =>  cos_out <= "10000001";
93          when "1111100" =>  cos_out <= "10000010";
94          when "1111011" =>  cos_out <= "10000010";
95          when "1111010" =>  cos_out <= "10000011";
96          when "1111001" =>  cos_out <= "10000100";
97          when "1111000" =>  cos_out <= "10000101";
98          when "1110111" =>  cos_out <= "10000111";
```

```
99          when "1110110" =>  cos_out <= "10001001";
100         when "1110101" =>  cos_out <= "10001011";
101         when "1110100" =>  cos_out <= "10001110";
102         when "1110011" =>  cos_out <= "10010001";
103         when "1110010" =>  cos_out <= "10010100";
104         when "1110001" =>  cos_out <= "10010111";
105         when "1110000" =>  cos_out <= "10011011";
106         when "1101111" =>  cos_out <= "10011111";
107         when "1101110" =>  cos_out <= "10100011";
108         when "1101101" =>  cos_out <= "10100111";
109         when "1101100" =>  cos_out <= "10101100";
110         when "1101011" =>  cos_out <= "10110001";
111         when "1101010" =>  cos_out <= "10110110";
112         when "1101001" =>  cos_out <= "10111011";
113         when "1101000" =>  cos_out <= "11000000";
114         when "1100111" =>  cos_out <= "11000110";
115         when "1100110" =>  cos_out <= "11001011";
116         when "1100101" =>  cos_out <= "11010001";
117         when "1100100" =>  cos_out <= "11010111";
118         when "1100011" =>  cos_out <= "11011101";
119         when "1100010" =>  cos_out <= "11100011";
120         when "1100001" =>  cos_out <= "11101001";
121         when "1100000" =>  cos_out <= "11101111";
122         when "1011111" =>  cos_out <= "11110110";
123         when "1011110" =>  cos_out <= "11111100";
124         when "1011101" =>  cos_out <= "11111111";
125         when "1011100" =>  cos_out <= "00000100";
126         when "1011011" =>  cos_out <= "00001010";
127         when "1011010" =>  cos_out <= "00010001";
128         when "1011001" =>  cos_out <= "00010111";
129         when "1011000" =>  cos_out <= "00011101";
130         when "1010111" =>  cos_out <= "00100011";
131         when "1010110" =>  cos_out <= "00101001";
132         when "1010101" =>  cos_out <= "00101111";
133         when "1010100" =>  cos_out <= "00110101";
134         when "1010011" =>  cos_out <= "00111010";
135         when "1010010" =>  cos_out <= "01000000";
136         when "1010001" =>  cos_out <= "01000101";
137         when "1010000" =>  cos_out <= "01001010";
138         when "1001111" =>  cos_out <= "01001111";
139         when "1001110" =>  cos_out <= "01010100";
140         when "1001101" =>  cos_out <= "01011001";
141         when "1001100" =>  cos_out <= "01011101";
142         when "1001011" =>  cos_out <= "01100001";
143         when "1001010" =>  cos_out <= "01100101";
144         when "1001001" =>  cos_out <= "01101001";
145         when "1001000" =>  cos_out <= "01101100";
146         when "1000111" =>  cos_out <= "01101111";
147         when "1000110" =>  cos_out <= "01110010";
```

```
148         when "1000101" =>  cos_out <= "01110101";
149         when "1000100" =>  cos_out <= "01110111";
150         when "1000011" =>  cos_out <= "01111001";
151         when "1000010" =>  cos_out <= "01111011";
152         when "1000001" =>  cos_out <= "01111100";
153         when "1000000" =>  cos_out <= "01111101";
154         when others=>null;
155
156      end case;
157   end process;
158
159 end Behavioral;
```

```
1  ----------------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:    04/02/2013
5  -- Design Name:    GMSK Modulator
6  -- Module Name:    sin_phi - Behavioral
7  -- Project Name:   GMSK Modulador
8  ----------------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11
12 entity sin_phi is
13 port (
14       phase_in  :   in  std_logic_vector (6 downto 0);
15       sin_out   :  out  std_logic_vector (7 downto 0)
16       );
17 end sin_phi;
18
19 architecture Behavioral of sin_phi is
20 begin
21
22   process(phase_in)
23   begin
24      case phase_in is
25         when "0111111" =>  sin_out <= "00000000";
26         when "0111110" =>  sin_out <= "11111010";
27         when "0111101" =>  sin_out <= "11110100";
28         when "0111100" =>  sin_out <= "11101110";
29         when "0111011" =>  sin_out <= "11101000";
30         when "0111010" =>  sin_out <= "11100010";
31         when "0111001" =>  sin_out <= "11011100";
32         when "0111000" =>  sin_out <= "11010110";
33         when "0110111" =>  sin_out <= "11010000";
34         when "0110110" =>  sin_out <= "11001010";
35         when "0110101" =>  sin_out <= "11000101";
36         when "0110100" =>  sin_out <= "10111111";
37         when "0110011" =>  sin_out <= "10111010";
38         when "0110010" =>  sin_out <= "10110101";
39         when "0110001" =>  sin_out <= "10110000";
40         when "0110000" =>  sin_out <= "10101011";
41         when "0101111" =>  sin_out <= "10100111";
42         when "0101110" =>  sin_out <= "10100010";
43         when "0101101" =>  sin_out <= "10011110";
44         when "0101100" =>  sin_out <= "10011010";
45         when "0101011" =>  sin_out <= "10010111";
46         when "0101010" =>  sin_out <= "10010100";
47         when "0101001" =>  sin_out <= "10010000";
48         when "0101000" =>  sin_out <= "10001110";
49         when "0100111" =>  sin_out <= "10001011";
```

```
50          when "0100110" =>  sin_out <= "10001001";
51          when "0100101" =>  sin_out <= "10000111";
52          when "0100100" =>  sin_out <= "10000101";
53          when "0100011" =>  sin_out <= "10000100";
54          when "0100010" =>  sin_out <= "10000011";
55          when "0100001" =>  sin_out <= "10000010";
56          when "0100000" =>  sin_out <= "10000010";
57          when "0011111" =>  sin_out <= "10000001";
58          when "0011110" =>  sin_out <= "10000010";
59          when "0011101" =>  sin_out <= "10000010";
60          when "0011100" =>  sin_out <= "10000011";
61          when "0011011" =>  sin_out <= "10000100";
62          when "0011010" =>  sin_out <= "10000101";
63          when "0011001" =>  sin_out <= "10000111";
64          when "0011000" =>  sin_out <= "10001001";
65          when "0010111" =>  sin_out <= "10001011";
66          when "0010110" =>  sin_out <= "10001110";
67          when "0010101" =>  sin_out <= "10010000";
68          when "0010100" =>  sin_out <= "10010100";
69          when "0010011" =>  sin_out <= "10010111";
70          when "0010010" =>  sin_out <= "10011010";
71          when "0010001" =>  sin_out <= "10011110";
72          when "0010000" =>  sin_out <= "10100010";
73          when "0001111" =>  sin_out <= "10100111";
74          when "0001110" =>  sin_out <= "10101011";
75          when "0001101" =>  sin_out <= "10110000";
76          when "0001100" =>  sin_out <= "10110101";
77          when "0001011" =>  sin_out <= "10111010";
78          when "0001010" =>  sin_out <= "10111111";
79          when "0001001" =>  sin_out <= "11000101";
80          when "0001000" =>  sin_out <= "11001010";
81          when "0000111" =>  sin_out <= "11010000";
82          when "0000110" =>  sin_out <= "11010110";
83          when "0000101" =>  sin_out <= "11011100";
84          when "0000100" =>  sin_out <= "11100010";
85          when "0000011" =>  sin_out <= "11101000";
86          when "0000010" =>  sin_out <= "11101110";
87          when "0000001" =>  sin_out <= "11110100";
88          when "0000000" =>  sin_out <= "11111010";
89
90          when "1111111" =>  sin_out <= "00000000";
91          when "1111110" =>  sin_out <= "00000110";
92          when "1111101" =>  sin_out <= "00001100";
93          when "1111100" =>  sin_out <= "00010010";
94          when "1111011" =>  sin_out <= "00011000";
95          when "1111010" =>  sin_out <= "00011110";
96          when "1111001" =>  sin_out <= "00100100";
97          when "1111000" =>  sin_out <= "00101010";
98          when "1110111" =>  sin_out <= "00110000";
```

```
99          when "1110110" =>  sin_out <= "00110110";
100         when "1110101" =>  sin_out <= "00111011";
101         when "1110100" =>  sin_out <= "01000001";
102         when "1110011" =>  sin_out <= "01000110";
103         when "1110010" =>  sin_out <= "01001011";
104         when "1110001" =>  sin_out <= "01010000";
105         when "1110000" =>  sin_out <= "01010101";
106         when "1101111" =>  sin_out <= "01011001";
107         when "1101110" =>  sin_out <= "01011110";
108         when "1101101" =>  sin_out <= "01100010";
109         when "1101100" =>  sin_out <= "01100110";
110         when "1101011" =>  sin_out <= "01101001";
111         when "1101010" =>  sin_out <= "01101100";
112         when "1101001" =>  sin_out <= "01110000";
113         when "1101000" =>  sin_out <= "01110010";
114         when "1100111" =>  sin_out <= "01110101";
115         when "1100110" =>  sin_out <= "01110111";
116         when "1100101" =>  sin_out <= "01111001";
117         when "1100100" =>  sin_out <= "01111011";
118         when "1100011" =>  sin_out <= "01111100";
119         when "1100010" =>  sin_out <= "01111101";
120         when "1100001" =>  sin_out <= "01111110";
121         when "1100000" =>  sin_out <= "01111110";
122         when "1011111" =>  sin_out <= "01111111";
123         when "1011110" =>  sin_out <= "01111110";
124         when "1011101" =>  sin_out <= "01111110";
125         when "1011100" =>  sin_out <= "01111101";
126         when "1011011" =>  sin_out <= "01111100";
127         when "1011010" =>  sin_out <= "01111011";
128         when "1011001" =>  sin_out <= "01111001";
129         when "1011000" =>  sin_out <= "01110111";
130         when "1010111" =>  sin_out <= "01110101";
131         when "1010110" =>  sin_out <= "01110010";
132         when "1010101" =>  sin_out <= "01110000";
133         when "1010100" =>  sin_out <= "01101100";
134         when "1010011" =>  sin_out <= "01101001";
135         when "1010010" =>  sin_out <= "01100110";
136         when "1010001" =>  sin_out <= "01100010";
137         when "1010000" =>  sin_out <= "01011110";
138         when "1001111" =>  sin_out <= "01011001";
139         when "1001110" =>  sin_out <= "01010101";
140         when "1001101" =>  sin_out <= "01010000";
141         when "1001100" =>  sin_out <= "01001011";
142         when "1001011" =>  sin_out <= "01000110";
143         when "1001010" =>  sin_out <= "01000001";
144         when "1001001" =>  sin_out <= "00111011";
145         when "1001000" =>  sin_out <= "00110110";
146         when "1000111" =>  sin_out <= "00110000";
147         when "1000110" =>  sin_out <= "00101010";
```

```
148         when "1000101" =>  sin_out <= "00100100";
149         when "1000100" =>  sin_out <= "00011110";
150         when "1000011" =>  sin_out <= "00011000";
151         when "1000010" =>  sin_out <= "00010010";
152         when "1000001" =>  sin_out <= "00001100";
153         when "1000000" =>  sin_out <= "00000110";
154         when others=>null;
155
156      end case;
157   end process;
158
159 end Behavioral;
```

```
1  ----------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:     04/02/2013
5  -- Design Name:     GMSK Modulator
6  -- Module Name:     NCO - Behavioral
7  -- Project Name:    GMSK Modulador
8  ----------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11
12 entity NCO is
13 port (
14      clk_nco     :   in  std_logic;
15      reset_nco   :   in  std_logic;
16      cos2pifct   :  out  std_logic_vector(7 downto 0);
17      sin2pifct   :  out  std_logic_vector(7 downto 0)
18      );
19 end NCO;
20
21 architecture Behavioral of NCO is
22
23 component counter_nco
24 port (
25      clk_counter     :  in  std_logic;
26      reset_counter   :  in  std_logic;
27      counter_out     :  out std_logic_vector(4 downto 0)
28      );
29 end component;
30
31 component cos_nco
32 port (
33      address_lut     :  in   std_logic_vector (4 downto 0);
34      LUT_cos_out     :  out  std_logic_vector (7 downto 0)
35      );
36 end component;
37
38 component sin_nco
39 port (
40      address_lut     :  in   std_logic_vector (4 downto 0);
41      LUT_sin_out     :  out  std_logic_vector (7 downto 0)
42      );
43 end component;
44
45 signal counter_to_address     :     std_logic_vector(4 downto 0);
46
47
48
49
```

```
50 begin
51
52 U51: counter_nco    port map(clk_nco,reset_nco,counter_to_address);
53 U52: cos_nco        port map(counter_to_address,cos2pifct);
54 U53: sin_nco        port map(counter_to_address,sin2pifct);
55
56 end Behavioral;
```

```
1  ----------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:     04/02/2013
5  -- Design Name:     GMSK Modulator
6  -- Module Name:     (NCO) counter_nco - Behavioral
7  -- Project Name:    GMSK Modulador
8  ----------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11 use ieee.numeric_std.all;
12
13 entity counter_nco is
14 port(
15     clk_counter     :  in  std_logic;
16     reset_counter   :  in  std_logic;
17     counter_out     :  out std_logic_vector(4 downto 0)
18     );
19 end counter_nco;
20
21 architecture Behavioral of counter_nco is
22
23 signal Qp, Qn : unsigned(4 downto 0):=(others => '0');
24
25 begin
26
27   combinacional : process(Qp)
28   begin
29       if (Qp = "11001") then
30          Qn <= "00000";
31       else
32          Qn <= Qp + 1;
33       end if;
34       counter_out <= std_logic_vector(Qp);
35   end process combinacional;
36
37   secuencial : process (reset_counter, clk_counter )
38   begin
39       if(reset_counter = '1') then
40          Qp <= "00000";
41       elsif(clk_counter'event and clk_counter = '1') then
42          Qp <= Qn;
43       end if;
44   end process secuencial;
45
46 end Behavioral;
```

```
1  ----------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:     04/02/2013
5  -- Design Name:     GMSK Modulator
6  -- Module Name:     (NCO) cos_nco - Behavioral
7  -- Project Name:    GMSK Modulador
8  ----------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11
12 entity cos_nco is
13 port (
14      address_lut  :  in   std_logic_vector (4 downto 0);
15      LUT_cos_out  :  out  std_logic_vector (7 downto 0)
16      );
17 end cos_nco;
18
19 architecture Behavioral of cos_nco is
20 begin
21
22   process(address_lut)
23   begin
24      case address_lut is
25         when "00000" =>  LUT_cos_out <= "01111111";
26         when "00001" =>  LUT_cos_out <= "01111100";
27         when "00010" =>  LUT_cos_out <= "01110001";
28         when "00011" =>  LUT_cos_out <= "01100000";
29         when "00100" =>  LUT_cos_out <= "01001001";
30         when "00101" =>  LUT_cos_out <= "00101110";
31         when "00110" =>  LUT_cos_out <= "00010000";
32         when "00111" =>  LUT_cos_out <= "11110000";
33         when "01000" =>  LUT_cos_out <= "11010010";
34         when "01001" =>  LUT_cos_out <= "10110111";
35         when "01010" =>  LUT_cos_out <= "10100000";
36         when "01011" =>  LUT_cos_out <= "10001111";
37         when "01100" =>  LUT_cos_out <= "10000100";
38         when "01101" =>  LUT_cos_out <= "10000001";
39         when "01110" =>  LUT_cos_out <= "10000100";
40         when "01111" =>  LUT_cos_out <= "10001111";
41         when "10000" =>  LUT_cos_out <= "10100000";
42         when "10001" =>  LUT_cos_out <= "10110111";
43         when "10010" =>  LUT_cos_out <= "11010010";
44         when "10011" =>  LUT_cos_out <= "11110000";
45         when "10100" =>  LUT_cos_out <= "00010000";
46         when "10101" =>  LUT_cos_out <= "00101110";
47         when "10110" =>  LUT_cos_out <= "01001001";
48         when "10111" =>  LUT_cos_out <= "01100000";
49         when "11000" =>  LUT_cos_out <= "01110001";
```

```
50          when "11001" =>  LUT_cos_out <= "01111100";
51          when "11010" =>  LUT_cos_out <= "00000000";
52          when "11011" =>  LUT_cos_out <= "00000000";
53          when "11100" =>  LUT_cos_out <= "00000000";
54          when "11101" =>  LUT_cos_out <= "00000000";
55          when "11110" =>  LUT_cos_out <= "00000000";
56          when "11111" =>  LUT_cos_out <= "00000000";
57          when others=>null;
58
59      end case;
60    end process;
61
62 end Behavioral;
```

```
1  ----------------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:    04/02/2013
5  -- Design Name:    GMSK Modulator
6  -- Module Name:    (NCO) cos_nco - Behavioral
7  -- Project Name:   GMSK Modulador
8  ----------------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11
12 entity sin_nco is
13 port (
14      address_lut  :  in   std_logic_vector (4 downto 0);
15      LUT_sin_out  :  out  std_logic_vector (7 downto 0)
16      );
17 end sin_nco;
18
19 architecture Behavioral of sin_nco is
20 begin
21
22   process(address_lut)
23   begin
24      case address_lut is
25         when "00000" =>  LUT_sin_out <= "00000000";
26         when "00001" =>  LUT_sin_out <= "00011111";
27         when "00010" =>  LUT_sin_out <= "00111100";
28         when "00011" =>  LUT_sin_out <= "01010101";
29         when "00100" =>  LUT_sin_out <= "01101001";
30         when "00101" =>  LUT_sin_out <= "01110111";
31         when "00110" =>  LUT_sin_out <= "01111111";
32         when "00111" =>  LUT_sin_out <= "01111111";
33         when "01000" =>  LUT_sin_out <= "01110111";
34         when "01001" =>  LUT_sin_out <= "01101001";
35         when "01010" =>  LUT_sin_out <= "01010101";
36         when "01011" =>  LUT_sin_out <= "00111100";
37         when "01100" =>  LUT_sin_out <= "00011111";
38         when "01101" =>  LUT_sin_out <= "00000001";
39         when "01110" =>  LUT_sin_out <= "11100001";
40         when "01111" =>  LUT_sin_out <= "11000100";
41         when "10000" =>  LUT_sin_out <= "10101011";
42         when "10001" =>  LUT_sin_out <= "10010111";
43         when "10010" =>  LUT_sin_out <= "10001001";
44         when "10011" =>  LUT_sin_out <= "10000001";
45         when "10100" =>  LUT_sin_out <= "10000001";
46         when "10101" =>  LUT_sin_out <= "10001001";
47         when "10110" =>  LUT_sin_out <= "10010111";
48         when "10111" =>  LUT_sin_out <= "10101011";
49         when "11000" =>  LUT_sin_out <= "11000100";
```

```
50          when "11001" =>  LUT_sin_out <= "11100001";
51          when "11010" =>  LUT_sin_out <= "00000000";
52          when "11011" =>  LUT_sin_out <= "00000000";
53          when "11100" =>  LUT_sin_out <= "00000000";
54          when "11101" =>  LUT_sin_out <= "00000000";
55          when "11110" =>  LUT_sin_out <= "00000000";
56          when "11111" =>  LUT_sin_out <= "00000000";
57          when others=>null;
58
59      end case;
60    end process;
61
62  end Behavioral;
```

```
1  ----------------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:    04/02/2013
5  -- Design Name:    GMSK Modulator
6  -- Module Name:    multiplier - Behavioral
7  -- Project Name:   GMSK Modulador
8  ----------------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11 use ieee.numeric_std.all;
12
13 entity multiplier is
14 port (
15       mult_in1  :   in  std_logic_vector(7 downto 0);
16       mult_in2  :   in  std_logic_vector(7 downto 0);
17       mult_out  :  out std_logic_vector(15 downto 0)
18       );
19 end multiplier;
20
21 architecture Behavioral of multiplier is
22
23 signal temp  :  signed(15 downto 0):= "0000000000000000";
24
25 begin
26
27         temp <= signed(mult_in1) * signed(mult_in2);
28     mult_out <= std_logic_vector(temp);
29
30 end Behavioral;
```

```
1  ----------------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  --
5  -- Create Date:     04/02/2013
6  -- Design Name:     GMSK Modulator
7  -- Module Name:     adder - Behavioral
8  -- Project Name:    GMSK Modulador
9  --
10 ----------------------------------------------------------------------------------
11 library ieee;
12 use ieee.std_logic_1164.all;
13 use ieee.numeric_std.all;
14
15 entity adder is
16 port (
17       adder_in1   :    in  std_logic_vector(15 downto 0);
18       adder_in2   :    in  std_logic_vector(15 downto 0);
19       adder_out   :   out  std_logic_vector(15 downto 0)
20       );
21 end adder;
22
23 architecture Behavioral of adder is
24
25 signal temp   :  signed(15 downto 0):= "0000000000000000";
26
27 begin
28
29       temp <= signed(adder_in1) -  signed(adder_in2);
30  adder_out <= std_logic_vector(temp);
31
32 end Behavioral;
```

C.2. Test bench utilizado para simular la arquitectura del modulador GMSK.

```
1  ----------------------------------------------------------------------------------
2  -- National Autonomous University of Mexico
3  -- Undergraduate Engineer: Miguel Angel Alvarado Zaragoza
4  -- Create Date:    04/02/2013
5  -- Design Name:    GMSK Modulator
6  -- Module Name:    modulator_tb - Behavioral
7  -- Project Name:   GMSK Modulador
8  ----------------------------------------------------------------------------------
9  library ieee;
10 use ieee.std_logic_1164.all;
11
12 entity modulator_tb is
13 end modulator_tb;
14
15 architecture behavior of modulator_tb is
16
17 -- Component Declaration for the Unit Under Test (UUT)
18
19 component Test_bench_GMSKmodulator
20 port (
21       data            : in  std_logic;
22       clk_system      : in  std_logic;
23       reset_system    : in  std_logic;
24       modulated       : out std_logic_vector(15 downto 0)
25       );
26 end component;
27
28 --Inputs
29 signal data               :   std_logic := '0';
30 signal clk_system         :   std_logic := '0';
31 signal reset_system       :   std_logic := '0';
32
33 --Outputs
34 signal modulated         :   std_logic_vector(15 downto 0);
35
36 -- Clock period definitions
37 constant clk_system_period : time := 10 ns;
38 begin
39
40    -- Instantiate the Unit Under Test (UUT)
41
42    uut: Test_bench_GMSKmodulator
43    port map(
44            data => data,
45            clk_system => clk_system,
```

```
46              reset_system => reset_system,
47              modulated => modulated
48              );
49
50      -- Clock process definitions
51
52      clk_system_process :process
53      begin
54          clk_system <= '0';
55          wait for clk_system_period/2;
56          clk_system <= '1';
57          wait for clk_system_period/2;
58      end process;
59
60      -- Stimulus process
61
62      stim_proc: process
63      begin
64
65          -- insert stimulus here
66          reset_system <= '1';
67          wait for clk_system_period*0.01;
68          reset_system <= '0';
69          data <= '1';
70          wait for clk_system_period*32*100;
71          data <= '0';
72          wait for clk_system_period*32*100;
73          data <= '1';
74          wait for clk_system_period*32*100;
75          data <= '1';
76          wait for clk_system_period*32*100;
77          data <= '0';
78          wait for clk_system_period*32*100;
79          data <= '0';
80          wait for clk_system_period*32*100;
81          data <= '1';
82          wait for clk_system_period*32*100;
83          data <= '0';
84          wait for clk_system_period*32*100;
85          data <= '1';
86          wait for clk_system_period*32*100;
87          data <= '0';
88          wait for clk_system_period*32*100;
89          data <= '1';
90          wait for clk_system_period*32*100;
91          data <= '1';
92          wait for clk_system_period*32*100;
93          data <= '1';
94          wait for clk_system_period*32*100;
```

```
95        data <= '0';
96        wait for clk_system_period*32*100;
97        data <= '0';
98        wait for clk_system_period*32*100;
99        data <= '1';
100       wait for clk_system_period*32*100;
101       data <= '0';
102       wait for clk_system_period*32*100;
103       data <= '1';
104       wait for clk_system_period*32*3*100;
105       data <= '0';
106       wait for clk_system_period*32*3*100;
107       data <= '1';
108       wait for clk_system_period*32*100;
109       data <= '0';
110       wait for clk_system_period*32*100;
111       data <= '1';
112       wait for clk_system_period*32*100;
113       data <= '1';
114       wait for clk_system_period*32*100;
115       data <= '0';
116       wait for clk_system_period*32*100;
117       data <= '0';
118       wait for clk_system_period*32*100;
119       data <= '1';
120       wait for clk_system_period*32*100;
121       data <= '0';
122       wait for clk_system_period*32*100;
123       data <= '1';
124       wait for clk_system_period*32*100;
125       data <= '0';
126       wait for clk_system_period*32*100;
127       data <= '1';
128       wait for clk_system_period*32*100;
129       data <= '0';
130       wait for clk_system_period*32*100;
131       data <= '0';
132       wait for clk_system_period*32*100;
133       data <= '1';
134       wait for clk_system_period*32*100;
135       data <= '0';
136       wait for clk_system_period*32*100;
137       data <= '1';
138       wait for clk_system_period*32*3*100;
139       data <= '0';
140       wait for clk_system_period*32*3*100;
141       data <= '1';
142       wait for clk_system_period*32*100;
143       data <= '0';
```

```
144         wait for clk_system_period*32*5*100;
145         data <= '0';
146         wait for clk_system_period*32*5*100;
147         data <= '1';
148         wait for clk_system_period*32*5*100;
149         data <= '0';
150         wait for clk_system_period*32*5*100;
151         data <= '1';
152         wait for clk_system_period*32*5*100;
153         data <= '0';
154         wait for clk_system_period*32*5*100;
155         data <= '1';
156         wait;
157     end process;
158 end;
```

Apéndice D

D.1. Esquemas RTL del modulador GMSK generados por ISE después del proceso de síntesis

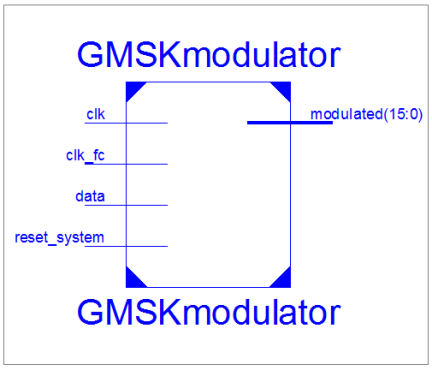


Figura D.1: Esquema RTL del modulador GMSK generado por *ISE* (primer nivel de jerarquía)

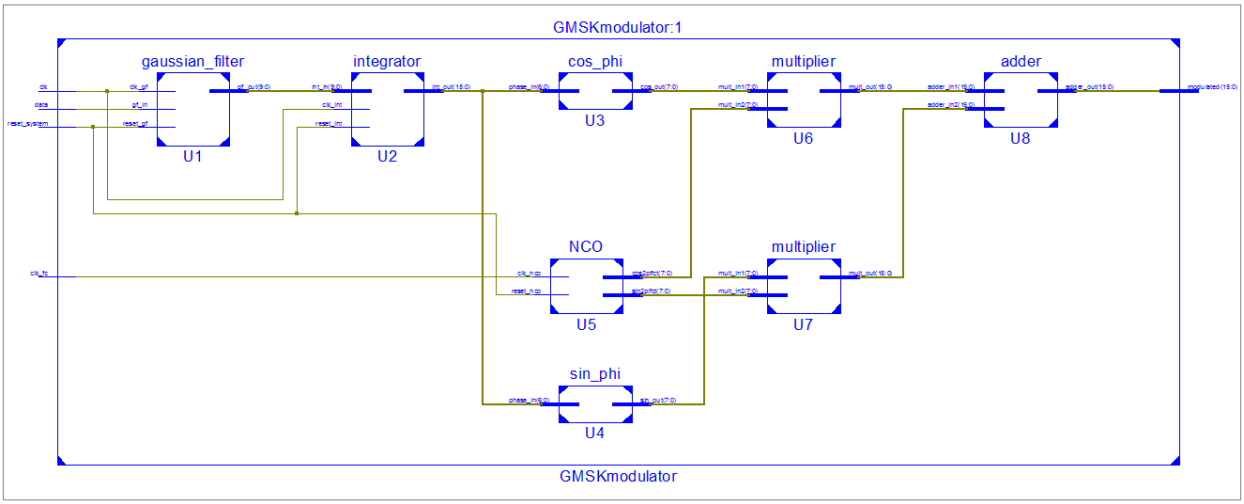


Figura D.2: Esquema RTL del modulador GMSK generado por *ISE* (segundo nivel de jerarquía)

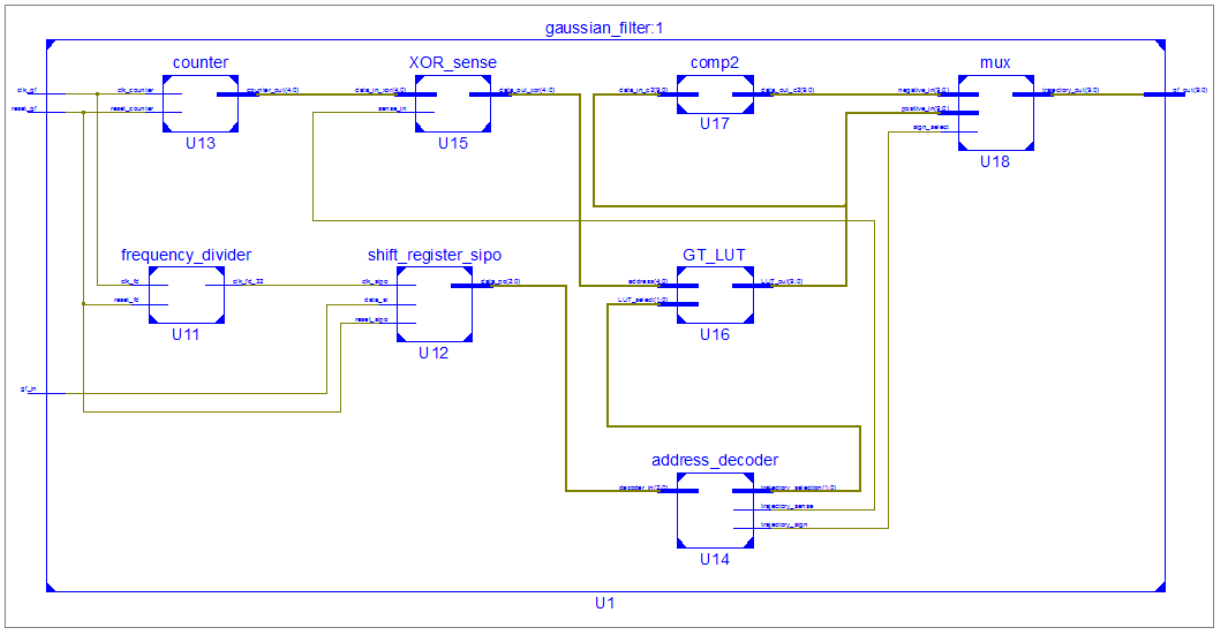


Figura D.3: Esquema RTL del filtro Gaussiano generado por *ISE*

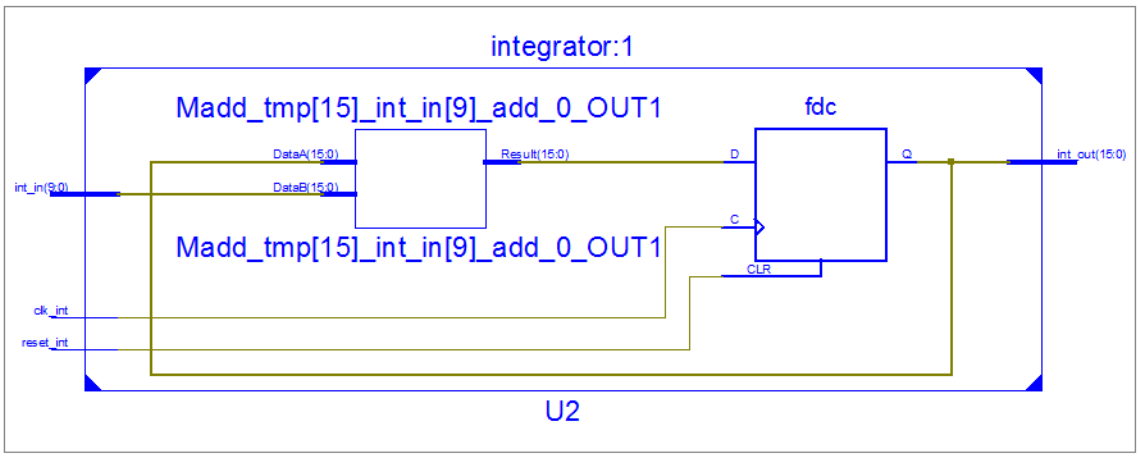


Figura D.4: Esquema RTL del integrador generado por *ISE*

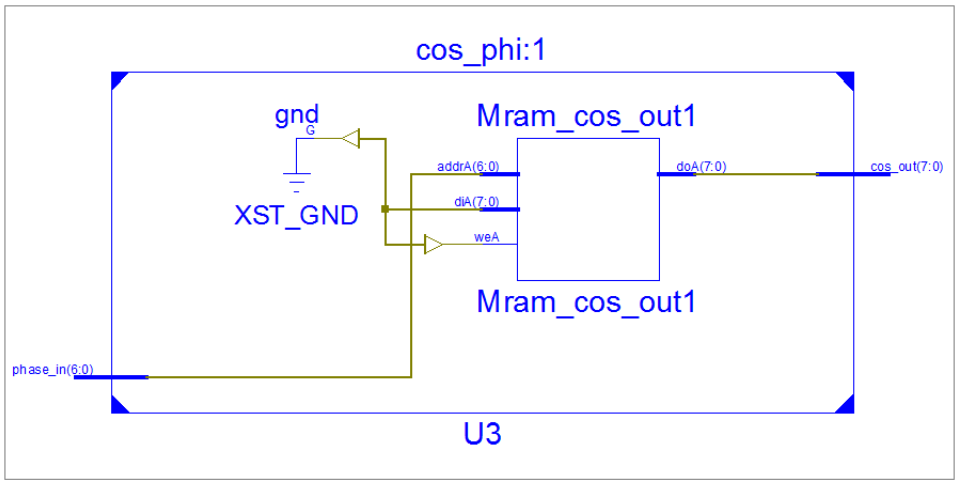


Figura D.5: Esquema RTL de la LUT coseno generado por *ISE*

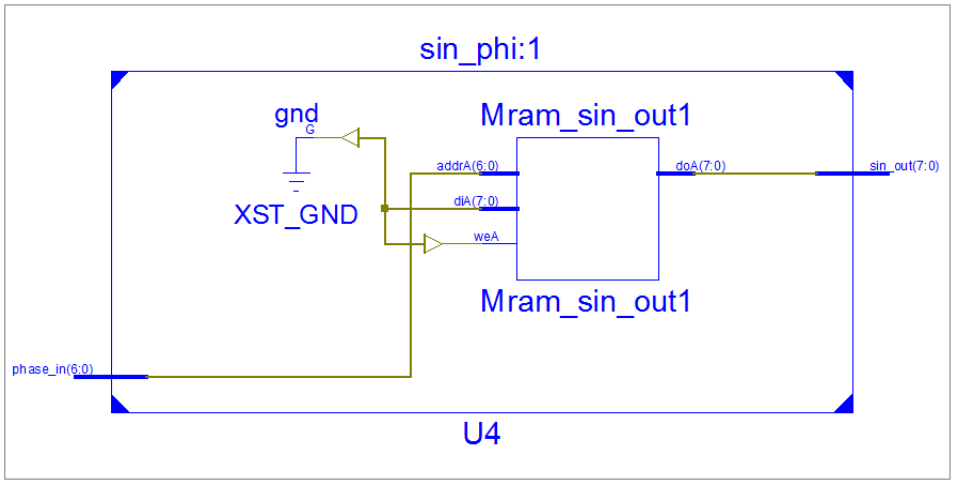


Figura D.6: Esquema RTL de la LUT seno generado por *ISE*

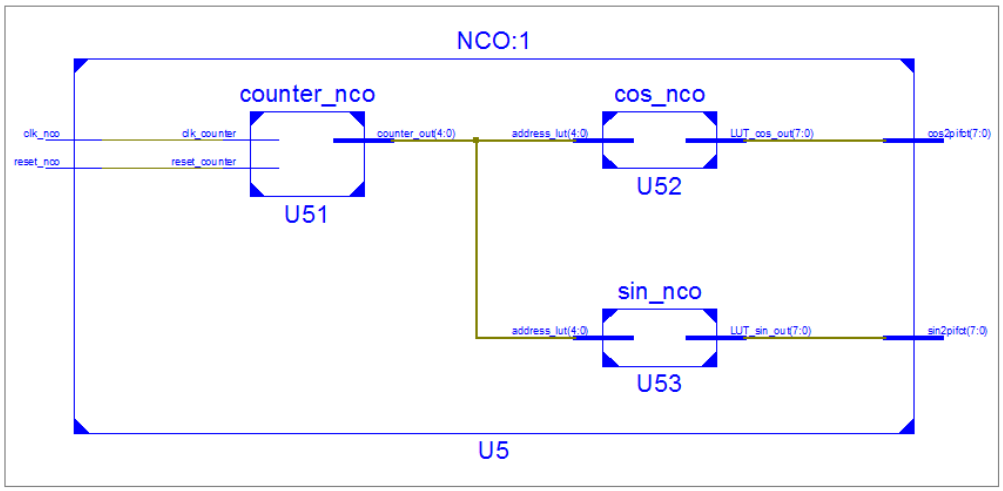


Figura D.7: Esquema RTL del NCO generado por *ISE*

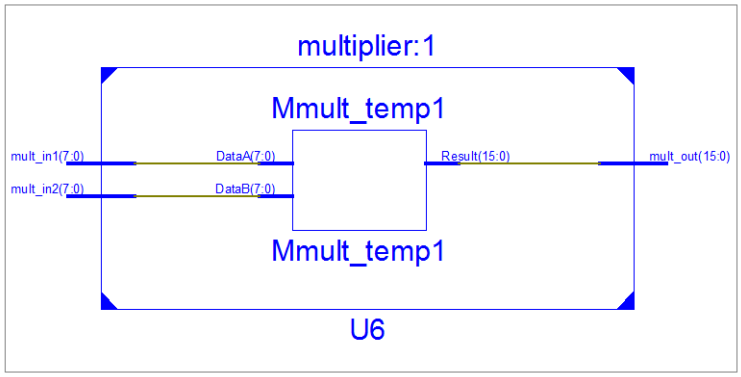


Figura D.8: Esquema RTL del multiplicador generado por *ISE*

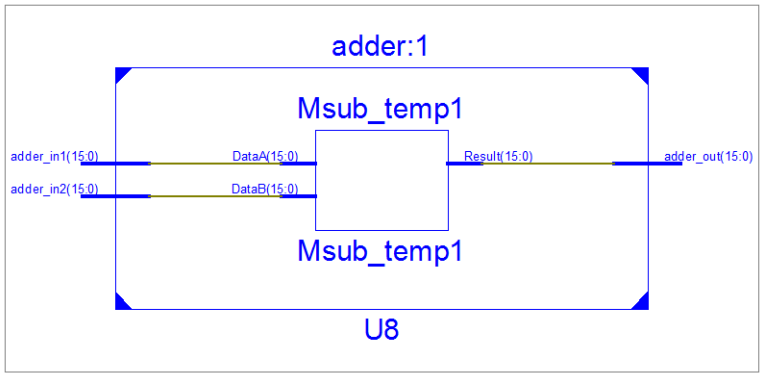


Figura D.9: Esquema RTL del sumador generado por *ISE*