Evaluación de los bloques funcionales pertenecientes al CDMA Reverse Traffic Channel IS-95, mediante aproximación

Carlos Alberto Pérez Camacho

Universidad Nacional de Colombia

Sede Manizales

Facultad de Ingeniería y Arquitectura

Departamento de Electricidad, Electrónica y Computación

Manizales, Colombia

2006

Evaluación de los bloques funcionales pertenecientes al CDMA Reverse Traffic Channel IS-95, mediante aproximación

Carlos Alberto Pérez Camacho

Tesis para optar al título de
Magister en Ingeniería Eléctrica

Director
Prof. Julio Cesar García

Universidad Nacional de Colombia
Sede Manizales
Facultad de Ingeniería y Arquitectura
Departamento de Electricidad, Electrónica y Computación
Manizales, Colombia
2006

Evaluation of CDMA IS-95 Reverse Traffic Channel functional blocks, using approximation

by

Carlos Alberto Pérez Camacho

A thesis submitted to the Postgraduate Program "Electrical Engineering"
in partial fulfillment of the requirements for the Master Degree

Thesis Director

Julio Cesar García

Universidad Nacional de Colombia

Sede Manizales

Manizales, Colombia

2006

A Diana,

Oscar,

Carolina,

Gloria y Antonio

Agradezco a Julio César García por su paciencia y apoyo en la realización de este trabajo, a Franklin Sepúlveda por su amistad y a Fanny Camacho por su apoyo incondicional.

Índice general

Contenido II

Lista de figuras IV

Lista de tablas IV

Resumen VII

Abstract 1

1. Diseño de sistemas digitales 3

1.1. Estado del arte y tendencias 3

1.1.1. Desarrollos relacionados 7

2. Metodología de Diseño 10

2.1. Modelo Propuesto 12

2.1.1. Trayectorias de realimentación 13

2.1.2. Funciones de salida 14

2.1.3. Matriz generadora 16

2.2. Representación mediante descriptores UML 18

2.2.1. Diagrama de clases 18

2.2.2. Diagrama de colaboración 19

2.2.3. Diagrama de secuencia 19

3. Aproximaciones del Modelo 21

3.1. Modelos de Markov 21
3.1.1. Modelo Oculto de Markov 22
3.1.2. Aproximación al Modelo Oculto de Markov 23
3.2. Campos de Galois 24
3.2.1. Operaciones vectoriales y matriciales sobre GF(2) 25
3.2.2. Correspondencia del modelo con GF 25

4. Metodología de Validación **27**
4.1. Generación de los datos de referencia 27
4.2. Creación de las entidades VHDL 28
4.3. Creación de archivos gráficos de simulación 28

5. Casos de estudio **31**
5.1. Diseño y validación del Codificador Convolucional 31
5.1.1. Configuración del FEC 31
5.1.2. Validación 32
5.2. Diseño y validación del Repetidor 34
5.2.1. Configuración del Symbol Repeater 34
5.2.2. Validación 35
5.3. Diseño y validación del generador CRC 35
5.3.1. Configuración Generador CRC-3 36
5.3.2. Validación 36

6. Conclusiones **38**

A. Generación de datos de verificación **A–1**

B. Generación del archivo *.vec **A–1**

C. Formato del archivo *.vec **A–1**

Índice de figuras

1.1. *Metamodelo UML-VHDL (a)relaciones (b) características* 6
1.2. *Proceso de co-diseño híbrido* 8

2.1. *Codificador cíclico* 13
2.2. *Codificador convolucional* 15
2.3. *Generalización del Registro de Corrimiento* 17
2.4. *Diagrama de Clases* 18
2.5. *Diagrama de Colaboración* 20
2.6. *Diagrama de Secuencia* 20

4.1. *Modelo de prueba* 27
4.2. *Formato del archivo tipo vector* 29
4.3. *Señales de entrada del sistema* 30
4.4. *Proceso de validación* 30

5.1. *Codificador Convolucional para el canal Reverse: r=1/3, K=9* 32
5.2. *Bloques To Workspace utilizados* 33
5.3. *Resultados de la simulación del bloque FEC* 33
5.4. *Representación del Symbol Repeater mediante Registros de corrimiento* . . 34
5.5. *Resultados de la simulación del bloque Repeater* 35
5.6. *Codificador CRC de orden 3* 36
5.7. *Simulación para el codificador CRC-3* 37

6.1. *Archivo .tbl con los resutados de la simulación del FEC.* 39

Índice de cuadros

5.1. Factores de repetición 34

Resumen

El presente trabajo ilustra los avances realizados para la Tesis de Maestría en Ingeniería Eléctrica desarrollada en el área de simulación de sistemas de comunicación digital y el diseño orientado hacia los conceptos SDR (Software Defined Radio) dentro del Grupo de Control y Procesamiento Digital de Señales.

Este documento presenta una metodología de diseño orientada al modelado de los bloques funcionales CDMA Reverse Traffic Channel y además una metodología de validación dirigida a la comprobación de resultados obtenidos luego de la implementación funcional de los bloques en lenguaje VHDL.

Se propone un procedimiento de generación automática del código VHDL correspondiente a los bloques analizados, los cuales son modelados con una estructura generalizada propuesta, basada en Registros de Corrimiento Realimentados Linealmente (LFSR) modificados con el fin de que permitan generar funcionalidades adicionales de salida, resultantes de la combinación lineal de las posiciones del registro. Se propone una modelo matricial para la configuración de los lazos de realimentación y las funciones de salida. La descripcion del modelo se generaliza haciendo uso de UML y se presenta como un Modelo Oculto de Markov (HMM) operando bajo las teorías de los los Campos de Galois.

Para la metodología de validación se propone la generación de archivos gráficos (*.vec) a partir de los resultados obtenidos con Simulink, los cuales pueden ser utilizados en entornos de desarrollo digital como MAX PLUS II o QUARTUS II. Los archivos generados permiten la simulación directa y la correspondiente verificación de los resultados.

Para la generación de las descripciones VHDL se realiza una aproximación *entity-based* (basada en entidades) la cual se presenta en [27] como un enfoque hacia la descomposición

jerárquica basada en unidades funcionales donde una entidad VHDL se considera como una abstracción de un objeto hardware, en este caso un Bloque Funcional.

Abstract

This work presents advances in the Master Thesis in Electrical Engineering developed in simulation of telecommunication systems and design oriented to SRD (Software Defined Radio) concepts in the Group of Control and Digital Signal Processing.

This document presents a design methodology oriented to modeling of functional CDMA Reverse Traffic Channels blocks and a validation methodology to check results obtained from functional implementation of blocks in VHDL.

A procedure for automatic VHDL code generation corresponding to analyzed blocks, is proposed. Blocks are modeled with a proposed generalized structure based on modified Linear Feedback Shift Registers (LFSR) to allow generation of output functionalities which are result of linear combination of register positions. Is proposed a matrix model to configure feedback loops and output functions. Model description is given as a UML model and is presented as a corresponding Hidden Marvok Model (HMM) operating under Galois Fields theory.

For validation methodology, graphic file generation is proposed, using results from Simulink, which may be used in development environments like MAX PLUS II or QUARTUS II. Generated files allows direct simulation and corresponding results checking.

In generation of VHDL code is made an approach *entity-based* presented in [27] as a focus to hierarchical decomposition based on functional units where a VHDL `entity` is taken as a hardware object abstraction, in this case a functional block.

Introducción

El concepto *Software Defined Radio* (SDR) está enfocado a la construcción de dispositivos de comunicación reconfigurables a nivel de software que permitan cambios funcionales en campo [19] y que faciliten la adaptación a los estándares que surgen ante las cambiantes demandas del mercado. Estos desarrollos van de la mano con nuevas herramientas como el lenguaje de programación VHDL y dispositivos lógicos programables como los FPGA (Arreglos Lógicos Programables en Campo).

En los procesos de diseño y validación de este tipo de sistemas se debe manejar una cantidad significativa de datos binarios que debe corresponder a la descripción de cada uno de los elementos funcionales del sistema a desarrollar. Es por esto que se hace necesario generar procesos formales de validación.

Dentro de la metodología propuesta se trabaja un proceso que hace uso de entornos de desarrollo como *Matlab* y *Max-PLus II*, y que se caracteriza por la generación de herramientas software que permiten la comunicación directa de estos dos, con el fin de facilitar la validación de los diseños realizados en este trabajo.

Se presenta el diseño de un modelo generalizado para sistemas basados en Registros de Corrimiento utilizados en CDMA IS-95, el cual permite lograr modelos estructurales y funcionales específicos a partir del modelo general. Estos registros están principalmente presentes en algunos bloques funcionales del *CDMA Reverse Traffic Channel* tales como el Codificador Convolucional y el Generador CRC del Indicador de Calidad de Trama (FQI).

En el *primer capítulo* se exponen los conceptos fundamentales relacionados con el diseño de sistemas digitales, se realiza un acercamiento a las herramientas de diseño mas comunes, a las diferentes tendencias de desarrollo en este sentido y se presentan algunos trabajos

relacionados que entregan herramientas útiles a esta investigación.

En el *capítulo dos*, se presentan los detalles de la metodología de diseño propuesta. Se expone el modelo generalizado propuesto, detallando las características particulares de los lazos de realimentación y las trayectorias de salida, las cuales se configuran a través de dos matrices básicas, cuyos modelos también son presentados en este capítulo.

El *tercer capitulo* presenta una aproximación matemática del modelo propuesto para el diseño de los bloques considerando su correspondencia con un Modelo Oculto de Markov que funciona en el espacio descrito por los campos Galois.

En el *capítulo cuatro* se brinda una explicación de la metodología de validación propuesta.

El *quinto capítulo* presenta tres casos de estudio en los cuales se aplican y evalúan los procedimientos y modelos propuestos utilizando tres bloques funcionales del *IS-95 CDMA Reverse Traffic Channel*. Inicialmente se presenta el proceso completo para el diseño y la evaluación del bloque Codificador Convolucional, donde se verifica la correspondencia entre los datos generados por el *CDMA Reference Blockset* de *Simulink* y los que entrega el diseño realizado en VHDL. De igual manera se muestran los resultados encontrados en el diseño del bloque Repetidor, el cual contiene funciones adicionales de salida que pueden ser validadas con los datos obtenidos en *Simulink*. Adicionalmente se presenta la implementación un generador CRC, el cual se compara con los resultados obtenidos en algunas referencias bibliográficas. Finalmente en el *capítulo seis* se presentan las conclusiones del trabajo, algunas recomendaciones y se proponen trabajos posteriores.

Capítulo 1

Diseño de sistemas digitales

1.1. Estado del arte y tendencias

Hay una marcada tendencia hacia la reutilización de estructuras para reducir los tiempos de desarrollo [13]. Existen herramientas de diseño hardware a través de software como *SystemC* y *SpecC* que extienden las ventajas de *C* y *C++* hacia el modelado de hardware y que, como en el caso de *SystemC* han sido integradas a una variedad de entornos como *MILAN*, *CoCentric System Studio* de Synopsys, *MaxSim Developer Suite* de Axys Design, *ConvergenSC* de CoWare and *Signal Processing Worksystem (SPW)* de Candence. Este último permite tomar componentes de librerías de diseño optimizadas, especificadas como máquinas de estado, o importadas de Matlab, C++, Verilog o VHDL [13].

Dentro de las posibilidades de reutilización de estructuras existentes para la generación de otras nuevas de aplicación específica, se ha explorado la posibilidad de aplicar técnicas de diseño orientado a objetos (OOD) en el dominio hardware y al diseño de sistemas embebidos, más cuando el diseño de hardware se proyecta como una actividad cada día más orientada a la programación, similar al desarrollo de software. Este nuevo esquema en el que los componentes de hardware pueden ser considerados como clases individuales con atributos y operaciones facilita la orientación a objetos ya que la segmentación del sistema en objetos puede ser aún mas directa que en los sistemas de software [27].

Se asumen tres aproximaciones importantes para el desarrollo de hardware orientado a objetos (OO) que son los lenguajes de hardware OO, la adaptación de lenguajes OO existentes al diseño de hardware y la especificación de alto nivel de hardware y sistemas embebidos

usando lenguajes universales de modelado como UML.

Lenguajes de descripción de hardware OO. En el diseño orientado a objetos se consideran aspectos como especificaciones de alto nivel de modelos de hardware utilizando notaciones formales como Redes Petri y diagramas UML y lenguajes de descripción de hardware orientado a objetos como extensiones VHDL y SystemC [13, 17, 31]. En [27] se propone un esquema de especificación de modelos de hardware usando diagramas de clases UML y un meta-modelo UML-VHDL para la generación automática de componentes VHDL a partir de los diagramas de clases en UML.

Adaptación de lenguajes OO existentes al diseño de hardware. Otra aproximación apunta hacia la adopción de lenguajes de programación orientados a objetos en el dominio del software para ser utilizados en el dominio del hardware, tales como Java o C++. En el caso de C++ que es el más opcionado, y que solo cuenta con dos niveles de abstracción: algorítmico y de objetos, se plantea la generación de otros niveles para acercarlo a los cuatro [26] que maneja una especificación de hardware:

1. Nivel algorítmico
2. Nivel modular
3. Cycle-accurate
4. Nivel de transferencia de registros (RTL)

Esto sería posible a través de la fusión con herramientas como *SystemC-Plus* o *SystemC*, que es básicamente una librería de clases C++ y una metodología que permite el modelado de sistemas embebidos [30], pero que desafortunadamente no está completamente orientado hacia el OOD [8].
Por su parte, *SystemC-Plus* soporta conceptos OOD como encapsulación de datos, herencia, comunicación basada en métodos, polimorfismo y clases. Además, soporta síntesis de algunos principios OO como funciones de clases tipo miembro, constructores, objetos y

arreglos de datos miembro, y herencia de plantillas de clases [27].

Especificación de alto nivel de hardware y sistemas embebidos usando UML. Los diagramas UML proveen soporte OOD y su aplicación para el diseño de hardware se ha enfocado al diseño de sistemas embebidos [16, 18, 25]. Comúnmente se utilizan tres tipos de diagramas UML para describir diseños hardware:

1. *Diagramas de clases*, donde comportamiento y datos son estructurados y encapsulados en clases para implementar la funcionalidad de los sistemas de acuerdo a la interacción que se presenta en los elementos.

2. *Diagramas de objetos*, donde los objetos que son instancias de las clases, representan las entidades del sistema.

3. *Diagramas de estados*, que describen el comportamiento del sistema a través de máquinas de estados finitos (Finite State Machines-FSM).

La forma más explorada para pasar de una especificación UML de alto nivel a una implementación de bajo nivel es el mapeo mediante conceptos OOD utilizando reglas para traducción a través de meta-modelos UML que describen la sintaxis de los diagramas UML [34] para ser traducidos manualmente por el diseñador o automáticamente utilizando una herramienta dedicada de traducción.
McUmber y Cheng [34] proponen una especificación de hardware usando diagramas de estado UML, mediante los cuales representan cada elemento `class` como una pareja `entidad-arquitectura` VHDL, las `asociaciones` entre clases se representan como `señales` VHDL, cada `estado` simple es representado como un `proceso` VHDL. Los `eventos` y `mensajes` intercambiados entre `objetos` y `estados` se definen por declaraciones de `señal` en la declaración de puertos de la entidad. Las `transiciones` de estado son consideradas como `asignaciones de señales` entre procesos, se aplicarán sobre todo en las señales de reloj que llevan los registros de corrimiento a un nuevo estado. Todos los `eventos` del diagrama de estados se representan con `señales`

que asumen un valor temporal y todas las `transiciones` son mapeadas en `ciclos` que contienen sentencias *if* y *wait* esperando por eventos de transición.

Por su parte Damasevicius y Stuikys [27] trabajan sobre la formalización de procesos de diseño de hardware de alto nivel y la especificación de hardware utilizando diagramas de clase UML. Como se obseva en la figura 1.1, desarrollan un proceso de mapeo de `abstract class` (interfaces) en `entidades` VHDL, `class` que generan clases abstractas en `arquitecturas`, `parámetros` de clases en sentencias `genéricas`, `atributos` de clases públicas en `puertos` y los `atributos` de clases privadas en `señales`, `métodos` de clases en `procesos`, relaciones de `composición` en sentencias de mapeo de puertos (`port map`), y las relaciones de `herencia` se representan como puertos de entrada y salida que son heredados de una entidad base por otra entidad.

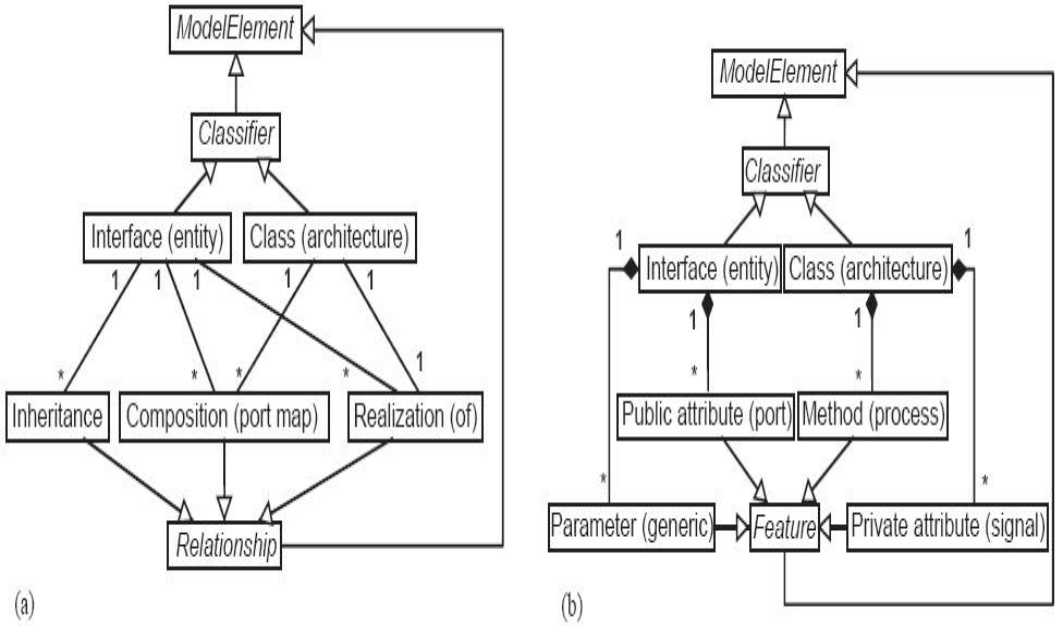

Figura 1.1: *Metamodelo UML-VHDL (a)relaciones (b) características*

Entre estas dos concepciones existen diferencias en el sentido de que la aproximación de Damasevicius y Stuikys orienta el diseño a la reutilización, agregando características de eficiencia al proceso de desarrollo de nuevos diseños, mientras que McUmber presenta un modelo mas genérico concentrado en entidades particulares. A pesar de las ventajas gene-

rales que ofrece la aproximación de Damasevicius y Stuikys, se opta por la de McUmber debido a la correspondencia que tiene con este caso de estudio, ya que el modelo propuesto basado en registros de corrimiento está concentrado básicamente en describir la composición de cada bloque funcional particular como una entidad similar a la de una arquitectura VHDL y los procesos están determinados principalmente por eventos concentrados en las señales de los relojes de entrada y salida, que generan transiciones, las cuales son descritas mediante ciclos basados en sentencias *if*.

1.1.1. Desarrollos relacionados

En [3] se presenta un diseño para generar arquitecturas paralelas de alta velocidad para codificadores convolucionales y su implementación en tarjetas FPGA de la familia Altera Flex10KE . Este tipo de codificadores genera codificación de elementos en adelanto mediante bloques *Anticipated State Generation* (ASG) los cuales calculan el nuevo estado Φ_{k+1} a partir del anterior Φ_k y de p bits de entrada, haciendo que los elementos intermedios sean codificados por estructuras paralelas (*Intermediate State Generation-ISG*), con lo cual se logra que se puedan operar varios bits por cada ciclo de reloj. Utilizando las especificaciones de cada codificador, relacionadas con parámetros como tamaño de la memoria del codificador, nivel de paralelismo, tasa de codificación, polinomios de realimentación, entre otros, se genera una matriz de configuración con un programa desarrollado en lenguaje C que permite describir la configuración del sistema y generar un paquete en VHDL que contiene la configuración completa de la arquitectura.

En [14] se presenta un trabajo que hace parte del proyecto Hybrid Electric Vehicle (HEV), dentro del cual se propone el uso de UML como una herramienta universal que permite integrar en un solo lenguaje descripciones de sistemas físicos desarrolladas en *Modelica*, descriptores de hardware en *VHDL* y sistemas de tiempo continuo descritos en *Simulink*, y que presenta mecanismos adicionales de extensión que permiten interactuar con otros entornos. Por su parte en [12] se presenta una metodología de modelado de sistemas físicos haciendo uso de Cellular Automatas (CAs), los cuales permiten representar sus comportamiento considerando variables propias, del entorno y su interacción. Los algoritmos co-

rrespondientes son implementados mediante un código VHDL, el cual puede ser generado por la herramienta desarrollada. Este código se genera sobre un modelo (plantilla) el cual presenta una serie de variables que pueden ser modificadas de acuerdo a las características particulares del sistema, y esta basado en la generación de una descripción funcional y una estructural de cada sistema, el cual se convierte en una entidad en VHDL.

En cuanto a la integración de varios entornos, en [28] se presenta una metodología de diseño de sistemas híbridos (hardware, software, analogos, radio-frecuencia, electromecánicos), los cuales se describen mediante diferentes lenguajes que son integrados en una sola aplicación en Simulink debido a la dificultad que se presenta para realizar simulaciones del comportamiento del sistema completo y debido a las limitaciones que tienen las herramientas comerciales para casos muy particulares como este. En el diagrama 1.2 se presenta el flujo de diseño propuesto por [28], en el que se integran los diversos entornos.

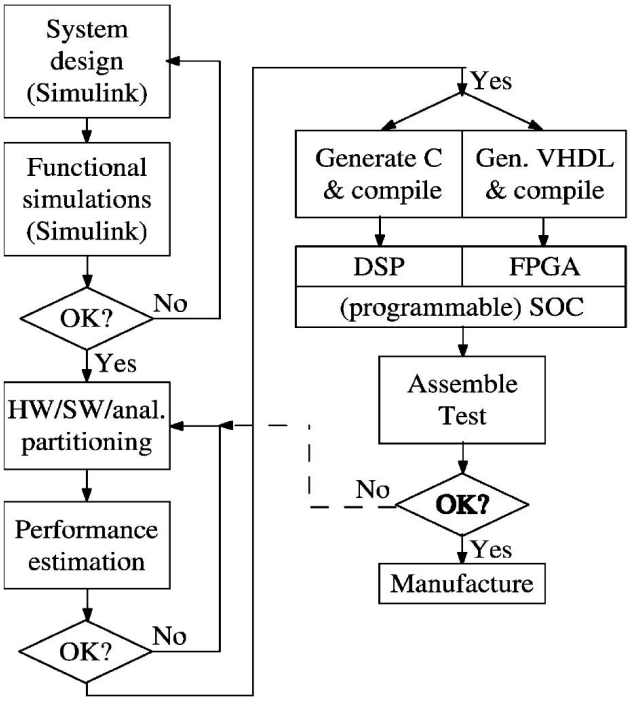

Figura 1.2: *Proceso de co-diseño híbrido*

En [21] se presenta un proceso para acelerar el desarrollo de sistemas neurodifusos con características heterogéneas (hardware/software), desarrollando una herramienta llamada

CodeSimulink, que permite la conexión de los dos entornos y la posterior simulación. Los autores determinan la necesidad de desarrollar una herramienta propia a pesar de que en el mercado existen aplicaciones basadas en *Simulink* que permiten el diseño de sistemas VLSI descritos en VHDL tales como *System Generator* de Xilinx [2] y D*SP Builder* de Altera [1]. Esto se debe a la limitación de estas para interactuar con sistemas software. En el caso particular de este trabajo, también se opta por el desarrollo de herramientas específicas debido a que a pesar de que *System Generator* y *DSP Builder* ofrecen bloques del tipo LFSR y codificadores convolucionales, no ofrecen la facilidad de desarrollar un bloque generalizado como el que se propone, debido a las limitaciones que se dan al momento de realizar una reconfiguración estructural de los mismos.

Respecto al diseño e implementación de sistemas de comunicación haciendo uso de las herramientas VHDL, en Colombia se han desarrollado trabajos como la ”Implementación en VHDL de un detector de Señales Satelitales de baja relación Eb/No y alta desviación en frecuencia Doppler” [11], ”Implementation of an H.263 codec for UMTS using VHDL” [24]. En la Universidad Nacional de Colombia-Sede Manizales se han venido desarrollando algunos trabajos orientados a la implementación de diferentes sistemas de comunicación sobre FPGA, algunos de los cuales se proyectan hacia el futuro desarrollo de técnicas SDR. Entre estos trabajos encontramos el ”Diseño e implementación en VHDL de un Conmutador ATM” [10], el ”Diseño e Implementación de un Conmutador B-ISDN utilizando Descriptores en VHDL” [9], el ”Desarrollo de Herramientas para la construcción de un dispositivo de Conmutación Digital Frame Relay utilizando Descriptores en VHDL” [36], el ”Diseño y Simulación en Lenguaje Descriptor de Hardware (VHDL) de un Multiplexor PDH/E1” [22] y finalmente el ”Diseño y Simulación de una Canal de Tráfico CDMA Reverse por medio de VHDL” [6] que es un trabajo previo a la investigación que se presenta en este documento.

Capítulo 2

Metodología de Diseño

Los métodos clásicos de desarrollo no satisfacen la necesidad de una rápida materialización de ideas en productos [33]. La construcción de un prototipo genera retardos significativos en el desarrollo de nuevos productos. Es por esto que los equipos de diseño frecuentemente prescinden de la construcción de prototipos y sustentan la calidad del diseño basándose en simulaciones y en la experiencia con productos previos [23].

Para los procesos de desarrollo se presentan diversas metodologías que por lo general se concentran dentro de unos marcos de referencia estándar, por ejemplo se proponen metodologías de diseño en las que se realiza un modelo completo del sistema hardware mediante herramientas de software, para ser utilizado como modelo de referencia en posteriores etapas de diseño [4, 32].

Existen algunas herramientas gráficas como *SIMULINK* de The Mathworks, *Signal Processing Workstation (SWP)* de Candence ó $COSSAP/CoCentric$ System Studio de Synopsys con muchas utilidades, que en algunos casos no son completamente aprovechadas por los equipos de investigación, debido a algunas restricciones que presentan sobre todo en las interfaces orientadas a la generación directa de los diseños del hardware [23].

Es importante, además de las herramientas utilizadas, la metodología de trabajo, ya que se convierte en factor clave en el éxito y la eficiencia del desarrollo. En [23] se analiza un modelo tradicional de diseño que utiliza tres equipos diferentes (investigación, diseño y construcción del producto) los cuales están encargados de trabajar en diversas etapas del proyecto. Este modelo puede presentar algunos inconvenientes en cuanto a la posibilidad de corregir errores generados en etapas tempranas de modelado y diseño, lo cual genera

considerables retardos en los procesos de corrección de errores, por esto se propone una metodología de trabajo en la que se opta por formar un solo equipo en el que cada individuo o grupo se encarga de desarrollar una unidad funcional. El modelo está caracterizado por cinco recomendaciones a saber:

- Un entorno
- Una documentación automática por especificación
- Una herramienta de revisión de código con compatibilidad adelante-atrás
- Un código trabajado sobre pasos de refinación
- Un equipo, para mejorar la comunicación

Debido a las ventajas que aporta este modelo, en el presente trabajo se aplican algunas de sus recomendaciones, tal como se describe en adelante.

El entorno. Respecto al entorno, se trabaja con *Matlab* y *Simulink* debido a las ventajas de interacción que existen entre los dos y a la disponibilidad de estas herramientas, lo cual evita la inversión de recursos adicionales relacionados con la compra de licencias. En el caso de *Simulink*, se encuentran dos ventajas adicionales importantes que son la posibilidad de un entorno gráfico de operación y la capacidad de entregar herramientas de simulación relacionadas con el estándar IS-95 a través del *CDMA Reference Blockset*, las cuales permiten generar datos tendientes a la futura validación de los procesos implementados en el entorno de programación de *Matlab* y en VHDL.

En cuanto a *Matlab*, la capacidad de utilizarlo como un entorno de programación, favorece los trabajos de diseño previos ya que permite la implementación y depuración de los algoritmos que describen los bloques funcionales del *CDMA Reverse Traffic Channel* y la posterior generación de las estructuras VHDL correspondientes.

La documentación. En el caso de la documentación, existe la ventaja de tener el estándar IS-95 que define puntualmente aspectos como velocidad de datos y puertos de entrada y salida, permitiendo así una rápida identificación de las interfaces presentes en cada bloque. Adicionalmente se ha definido una caracterización estructural y funcional a través de definición generalizada de los bloques funcionales, lo cual facilita el entendimiento de su funcionamiento en el momento de generar un diseño.

La revisión y refinación. En lo relacionado con la revisión y la refinación del código con compatibilidad hacia adelante y atrás, el entorno de programación de *Matlab*, favorece una eficiente depuración principalmente por la ya mencionada capacidad de conexión con *Simulink* a través de herramientas como los módulos *To Workspace* que permiten la obtención de datos orientados a la validación de los algoritmos. Igualmente se presenta la capacidad de conexión con el entorno *Quartus II* de Altera Corp. a través de una herramienta básica de generación de código VHDL desarrollada en este trabajo, la cual utiliza las estructuras generalizadas propuestas para generar las estructuras VHDL correspondientes a los bloques específicos.

2.1. Modelo Propuesto

Los bloques funcionales *CDMA Reverse Traffic Channel* realizan tratamiento de datos digitales, los cuales son procesados en bloques ó de forma cíclica en la medida en que van llegando, haciendo uso de estructuras tipo buffer que pueden ser consideradas como registros. En este trabajo se presenta un modelo generalizado basado en registros de corrimiento, permite la configuración dinámica a partir de conjuntos de datos binarios organizados como vectores o polinomios que caracterizan los procesos internos del registro, el flujo de datos de salida, y además puede ser aplicado al diseño de otros bloques.

Es posible estructurar un modelo generalizado que herede características particulares de cada uno de los bloques analizados, de tal forma que se puedan configurar mediante la manipulación de dos elementos característicos, como son, las *Trayectorias de Realimentación*

y las *Funciones de Salida*. Estas trayectorias y funciones son implementadas a nivel físico mediante operadores booleanos representados por sumas XOR y productos mediante compuertas AND.

2.1.1. Trayectorias de realimentación

Las trayectorias de realimentación se aplican en las operaciones de codificación de redundancia cíclica principalmente, y están determinadas por polinomios generadores relacionados con los códigos CRC utilizados. Las cajas de la figura 2.1 representan flip-flops o elementos de retardo unitario [15] que mantienen su valor de salida 0 o 1 durante un ciclo de reloj. El contenido de los flip-flops es inicialmente cero y se desplaza en sentido de las flechas con cada pulso de reloj. Los multiplicadores que están presentes en las trayectorias de realimentación están directamente asociados con los coeficientes g_{0i} del polinomio generador que gobierna la realimentación.

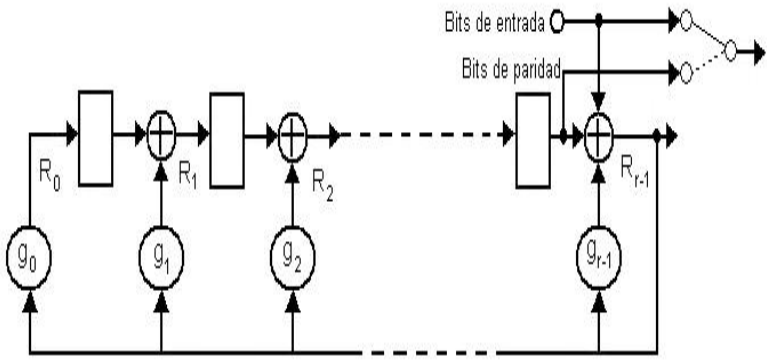

Figura 2.1: *Codificador cíclico*

La función generadora asociada a las trayectorias de realimentación se representa por medio de un polinomio de grado *r-1*, como indica la ecuación 2.1.

$$G_0(X) = g_{00} + g_{01}X + g_{02X^2} + \ldots + g_{0i}X^i + \ldots + g_{0(r-1)}X^{r-1} \tag{2.1}$$

donde *r* es la cantidad de posibles trayectorias de realimentación.
Las ramas que realimentan el valor en el ciclo anterior de la última posición del registro $R_{r-1}(k-1)$ afectan de forma directa los valores en las diferentes posiciones del registro de corrimiento y las operaciones es de desplazamiento para el cálculo de los valores en el presente ciclo (k), que también dependen de los valores de las posiciones anteriores en el ciclo anterior $(k-1)$ y del valor de la entrada actual, tal como se describe en las siguientes ecuaciones:

$$R_0(k) = b_k \oplus [g_{00} R_{r-1}(k-1)] \tag{2.2}$$

$$R_i(k) = R_{i-1}(k-1) \oplus \{g_{0i}[b_k \oplus R_{r-1}(k-1)]\} \qquad \text{i = 1,2,...,r-1} \tag{2.3}$$

donde,

$R_0(k)$ es el valor actual de la primera posición del registro de corrimiento
b_k es la actual entrada
$R_i(k)$ es el valor actual de la posición $i-esima$ del registro de corrimiento
$R_{i-1}(k-1)$es el valor de la posición $i-1$ del registro de corrimiento en el ciclo anterior
$R_{r-1}(k-1)$es el valor de la última posición del registro de corrimiento en el ciclo anterior

2.1.2. Funciones de salida

Las funciones de salida están constituidas por sumadores tipo XOR que realizan convoluciones módulo 2 en cada ciclo de reloj utilizando los datos presentes en el registro de corrimiento, tal como se observa en la figura 2.2.
Las *n* funciones de salida generan *n* dígitos binarios por cada desplazamiento ocurrido en el registro. Las salidas están asociadas a funciones generadoras representadas por los

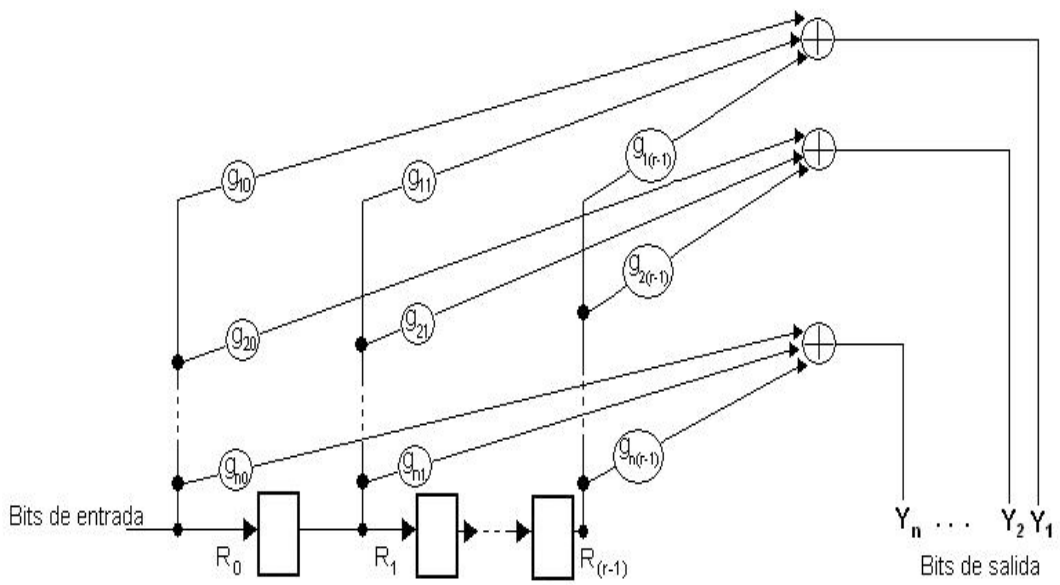

Figura 2.2: *Codificador convolucional*

polinomios de grado *r-1* del sistema de ecuaciones 2.4.

$$
\begin{array}{lclllllll}
G_1(X) & = & g_{10} & +g_{11}X & +g_{12}X^2 & +\dots & +g_{1i}X^i & +\dots & +g_{1(r-1)}X^{r-1} \\
G_2(X) & = & g_{20} & +g_{21}X & +g_{22}X^2 & +\dots & +g_{2i}X^i & +\dots & +g_{2(r-1)}X^{r-1} \\
\vdots & & \vdots & \vdots & \vdots & & \vdots & & \vdots \\
G_j(X) & = & g_{j0} & +g_{j1}X & +g_{j2}X^2 & +\dots & +g_{ji}X^i & +\dots & +g_{0(r-1)}X^{r-1} \\
\vdots & & \vdots & \vdots & \vdots & & \vdots & & \vdots \\
G_n(X) & = & g_{n0} & +g_{n1}X & +g_{n2}X^2 & +\dots & +g_{ni}X^i & +\dots & +g_{n(r-1)}X^{r-1}
\end{array}
\tag{2.4}
$$

El conjunto de ecuaciones 2.5 representa las *n* funciones de salida del sistema.

$$\begin{array}{rcllllll}
Y_1(k) & = & g_{10}R_0(k) & \oplus g_{11}R_1(k) & \oplus g_{12}R_2(k) & \oplus \dots & \oplus g_{1i}R_i(k) & \oplus \dots & \oplus g_{1(r-1)}R_{r-1}(k) \\
Y_2(k) & = & g_{20}R_0(k) & \oplus g_{21}R_1(k) & \oplus g_{22}R_2(k) & \oplus \dots & \oplus g_{2i}R_i(k) & \oplus \dots & \oplus g_{2(r-1)}R_{r-1}(k) \\
\vdots & & \vdots & \vdots & \vdots & & \vdots & & \vdots \\
Y_j(k) & = & g_{j0}R_0(k) & \oplus g_{j1}R_1(k) & \oplus g_{j2}R_2(k) & \oplus \dots & \oplus g_{ji}R_i(k) & \oplus \dots & \oplus g_{j(r-1)}R_{r-1}(k) \\
\vdots & & \vdots & \vdots & \vdots & & \vdots & & \vdots \\
Y_n(k) & = & g_{n0}R_0(k) & \oplus g_{n1}R_1(k) & \oplus g_{n2}R_2(k) & \oplus \dots & \oplus g_{ni}R_i(k) & \oplus \dots & \oplus g_{n(r-1)}R_{r-1}(k)
\end{array} \tag{2.5}$$

A partir de este conjunto de ecuaciones de salida se obtiene la siguiente función generalizada de la ecuación 2.6, descrita como una sumatoria tipo XOR.

$$Y_j(k) = \bigoplus_{i=0}^{r-1} g_{ji}R_i(k) \qquad , j = 1, 2, \dots, n \tag{2.6}$$

2.1.3. Matriz generadora

A partir de la descripción generalizada de los polinomios característicos de las funciones de realimentación y de salida se puede conformar la matriz 2.7 de dimensiones $(n+1)*(r-1)$ que ofrece una representación global de la configuración del sistema correspondiente al sistema de ecuaciones dado por 2.2,2.3 y 2.5.

$$\mathbf{G} = \begin{bmatrix}
g_{00} & g_{01} & g_{02} & \cdots & g_{0i} & \cdots & g_{0(r-1)} \\
g_{10} & g_{11} & g_{12} & \cdots & g_{1i} & \cdots & g_{1(r-1)} \\
g_{20} & g_{21} & g_{22} & \cdots & g_{2i} & \cdots & g_{2(r-1)} \\
\vdots & \vdots & \vdots & & \vdots & & \vdots \\
g_{j0} & g_{j1} & g_{j2} & \cdots & g_{ji} & \cdots & g_{j(r-1)} \\
\vdots & \vdots & \vdots & & \vdots & & \vdots \\
g_{n0} & g_{n1} & g_{n2} & \cdots & g_{ni} & \cdots & g_{n(r-1)}
\end{bmatrix} \tag{2.7}$$

Con estas representaciones se obtiene un modelo generalizado que conjuga las funcionalidades de realimentación y salida tal como lo muestra la figura 2.3.

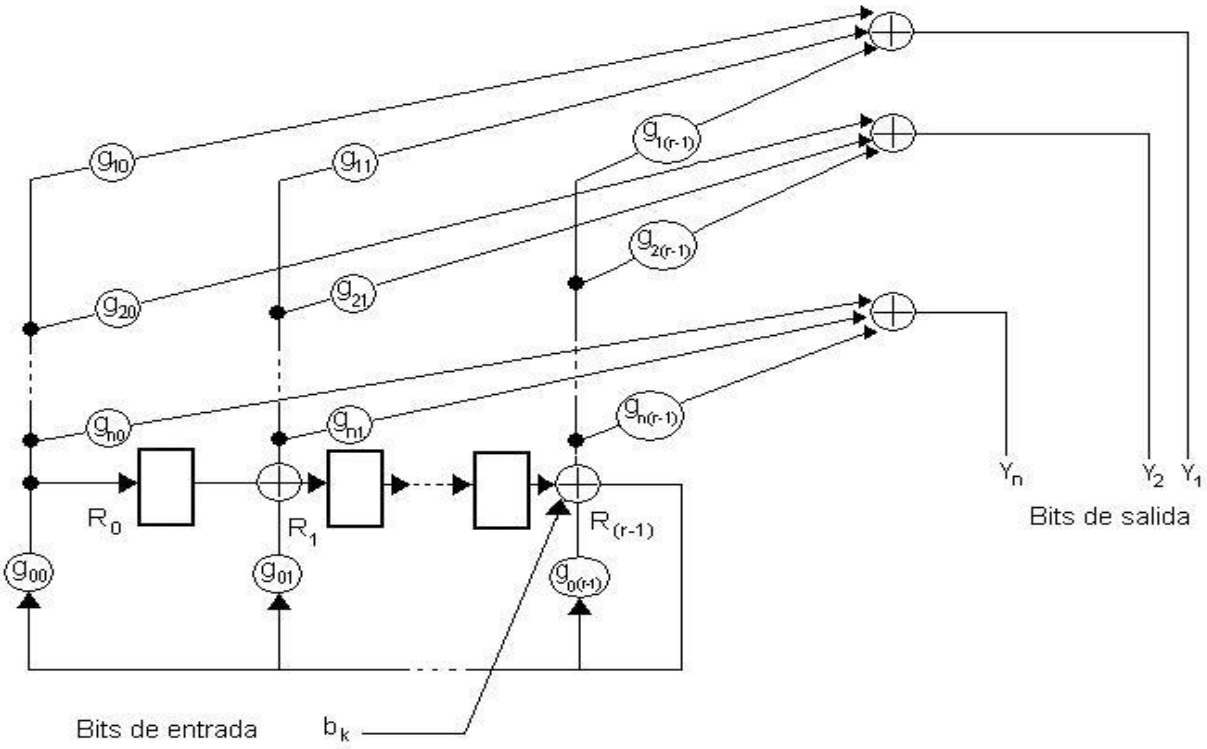

Figura 2.3: *Generalización del Registro de Corrimiento*

Dentro de la metodología de diseño se plantea una herramienta de generación automática de código VHDL, a partir de las descripciones dadas para las funciones de realimentación y salida, las cuales están contenidas en la matriz general, que permite, mediante un programa en *Matlab*, generar un mapeo de la forma descrita en [34] con el fin de generar los objetos y procedimientos dirigidos a la implementación en lenguaje descriptor de hardware. Por ejemplo para un Codificador Convolucional se trabajan solamente funciones de salida, sin generar lazo de realimentación alguno, mientras que para un codificador CRC se configuran los lazos de realimentación gobernados por los datos de la primera fila de la matriz de configuración y una salida correspondiente a la última posición de memoria del registro.

2.2. Representación mediante descriptores UML

Con el fin de entregar una representación universal del modelo propuesto, se presenta un modelado en UML en el que debido a las características estructurales y funcionales de los bloques del *CDMA Reverse Traffic Channel* y a las características del modelo generalizado propuesto se opta por el uso de diagramas de clases, objetos, secuencia e interacción, ya que permiten una buena aproximación estructural y funcional.

2.2.1. Diagrama de clases

Se propone una clase llamada principal `Funct_Block` que representa el modelo descrito en la sección anterior, la cual está asociada a las subclases `Port` y `Shift_Reg` y a la clase `Clock`, que entrega las señales que comandan los cambios de estados y la entrega de los datos de salida. En la figura 2.4 se presenta el diagrama de clases que representa el modelo.

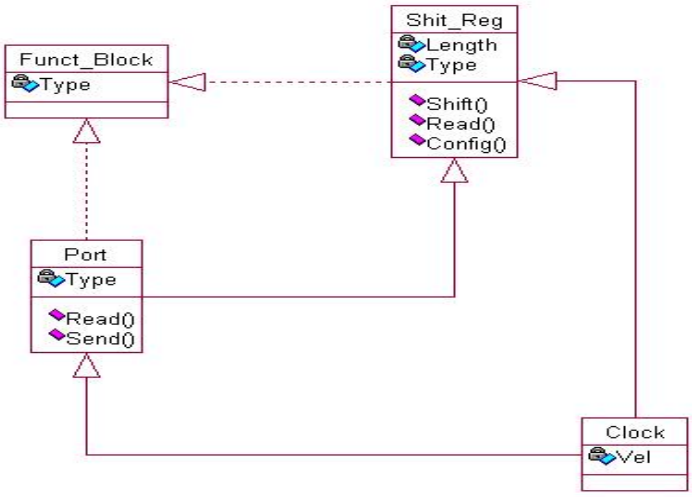

Figura 2.4: *Diagrama de Clases*

Clase `Funct_Block`. Clase que hace uso de las clases funcionales `Port` y `Shift_Reg`.

Clase `Port`.Se encarga de recibir `Read()` o enviar `Send()` los datos de entrada o salida respectivamente. Cada operación esta gobernada por una señal proveniente de una instan-

cia de la clase `Clock`.

Clase `Clock`. Genera los pulsos de onda cuadrados correspondientes a la señal de reloj utilizada para la operación del sistema. Puede ser configurada su velocidad a través del atributo *Vel*.

Clase `Shift_Reg`
Describe las características básicas del Registro de Corrimiento, el cual tiene un longitud *K* caracterizada por el atributo `Length`.
Las operaciones relacionadas con la clase `Shift_Reg` se presentan a continuación:

- `Shift()`, que se refiere al corrimiento (una posición por ciclo)
- `Read()`, lectura del dato
- `Config()`, configuración de realimentación y salida

Respecto a la operación *Config()*, esta se puede caracterizar haciendo uso de una matriz de configuración, la cual determina las posiciones de memoria en las cuales se realimentará el último elemento del mismo y las funciones de salida.

2.2.2. Diagrama de colaboración

En el diagrama de colaboración se presentan las relaciones operativas de los diferentes objetos comprometidos en el bloque funcional. En general estos objetos están sincronizados por medio de una conexión con los relojes de entrada y/o salida para generar un intercambio de mansajes relacionado con las operaciones asociadas a cada uno. La figura 2.5 muestra las relaciones de colaboración que se presentan en cada uno de los elementos del bloque.

2.2.3. Diagrama de secuencia

En el diagrama de secuencia se especifica el orden de las operaciones ordenadas que ejecuta cada objeto y la forma en que este afecta a otro haciendo que inicie una nueva operación.

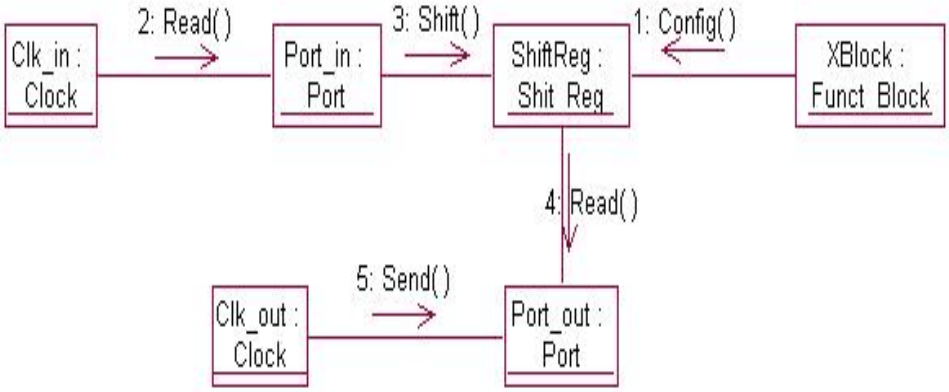

Figura 2.5: *Diagrama de Colaboración*

En el diagrama de la figura 2.6 se presentan las secuencias de operación de los objetos constituyentes de un bloque generalizado.

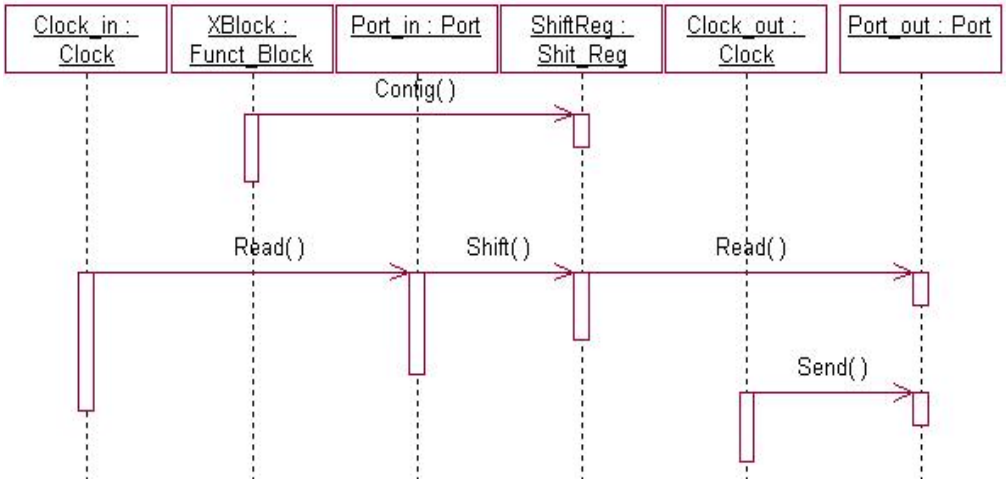

Figura 2.6: *Diagrama de Secuencia*

Capítulo 3

Aproximaciones del Modelo

El funcionamiento de cada uno de los bloques que conforman el *CDMA Reverse Traffic Channel* compromete una serie de conceptos matemáticos que se conservan en el modelo propuesto. En los casos particulares del generador CRC, el codificador convolucional y el repetidor, que son los bloques que se analizan en los casos de estudio, se presentan descripciones matemáticas relacionadas con las características de la información generada, la cual se ve enmarcada dentro de las propiedades de los Campos de Galois. Por esta razón se describe la correspondencia matemática del modelo con estos pricipios. Adicionalemente, la operación del modelo corresponde a procesos estacásticos que están relacionados con aleatoriedad de los datos de entrada, los cuales se pueden modelar como un Proceso Oculto de Markov. Estas aproximaciones se realizan en las siguientes secciones.

3.1. Modelos de Markov

Es una secuencia que representa un proceso estocástico discreto el cual es observable, ya que la salida es el conjunto de estados en cada tiempo. Su característica básica radica en la mínima cantidad de memoria asociada debido a que el estado presente depende solo del estado anterior. Bajo esta condiciones se plantea la *suposición markoviana* dada por la ecuación 3.1.

$$P(X_i|X_1^{i-1}) = P(X_i|X_{i-1}) \tag{3.1}$$

donde $X_1^{i-1} = X_1, X_2, ..., X_{i-1}$. Bajo esta restricción y considerando el *Teorema de Bayes*, se tiene que la probabilidad de que ocurra una secuencia de *n* variables $\mathbf{X} = X_1, X_2, ..., X_n$ está dada por la ecuación 3.1.

$$P(X_1, X_2, ..., X_n) = P(X_1) \prod_{i=1}^{n} P(X_i|X_1^{i-1}) = P(X_1) \prod_{i=1}^{n} P(X_i|X_{i-1}) \tag{3.2}$$

3.1.1. Modelo Oculto de Markov

Un Modelo Oculto de Markov (Hiden Markov Model-HMM) consta de un proceso de Markov no observable y otro observable, que presenta estados estocásticamente dependientes de los estados ocultos pertenecientes al proceso subyacente, el cual solo puede ser asociado probabilísticamente con otro proceso estocástico observable produciendo la secuencia de características que se pueden observar [5].
El modelo oculto de Markov se puede definir mediante las siguientes características:

- $\mathbf{O} = \{o_1, o_2, ..., o_M\}$. Alfabeto de salida o conjunto discreto de M símbolos.
- $\Omega = \{1, 2, ..., N\}$. Conjunto de estados no observavbles.
- $\mathbf{A} = \{a_{ij}\}_{NxN}$. Matriz de probabilidad de transición, donde a_{ij} es la probabilidad de pasar del estado *i* al estado *j*, y se describe como $a_{ij} = P(s_t = j|s_{t-1} = i)$.
- $\mathbf{B} = \{b_i(k)\}$. Matriz de probalilidad de salida, donde $b_i(k)$ es la probabilidad de obtener el símbolo de salida o_k en el estado *i*. Con esto se tiene $b_i(k) = P(X_t = o_k|s_t = i)$
- $\pi = \{\pi_i\}$. Distribución inicial de estados, donde $\pi_i = P(s_0 = i)$, para $1 \leq i \leq N$

Una especificación completa de un Modelo Oculto de Markov está dado por el conjunto $\{\mathbf{N}, \mathbf{M}, \Phi\}$, donde $\Phi = (A, B, \pi)$.
Se manejan dos suposiciones para un modelo de primer orden:

1. *Suposición de independencia a la salida*

$$P(X_t|X_1^t, s_1^t) = P(X_t|s_t) \tag{3.3}$$

que indica que la probabilidad de que un símbolo sea emitido en el tiempo *t* depende solo del estado s_t.

2. *Suposición de Markov*

$$P(s_t|s_1^{t-1}) = P(s_t|s_{t-1}) \tag{3.4}$$

que indica que el estado actual solo depende del anterior.

El proceso que relaciona las salidas con los estados del registro es completamente determinístico y esta dado por el sistema de ecuaciones 2.5, mientras que el proceso asociado a los cambios de estados está condicionado estocasticamente con los cambios en la entrada del sistema y se encuentra descrito por la ecuaciones 2.2 y 2.3.

3.1.2. Aproximación al Modelo Oculto de Markov

El codificador convolucional genera una codificación continua de datos convirtiendose en una máquina de 2^r estados que entrega *n* bits de salida antes de pasar al nuevo estado. La salida generada Y^k y el nuevo estado S^{k+1} dependen del estado presente S^k y de la entrada b_k [29] de acuerdo a

$$Y^k = f(S^k, b_k) \tag{3.5}$$

y

$$S^{k+1} = g(S^k, b_k) \tag{3.6}$$

Estas características se conservan para el generador CRC, el cual maneja lazos de realimentación, y para los demás bloques del sistema CDMA que podrían ser implementados mediante el modelo propuesto.
Considerando estas descripciones, se puede asociar la ecuación 3.6 con la ecuación 3.4 y la ecuación 3.5 con la ecuación 3.3, teniendo en cuenta que los estados en 3.4 y 3.3 son estocásticamente dependientes de los datos de entrada.
A partir del análisis de las caracteristicas del sistema, podemos asociarlo directamente con un proceso subyacente no observable e interno de cada bloque (los estados S del registro) y con un proceso observable (las salidas Y) que es dependiente del anterior, lo cual nos indica que cada bloque corresponde estructural y funcionalmente a un Proceso Oculto de Markov.

3.2. Campos de Galois

Un campo es un conjunto en el que sus elementos cumplen las siguientes condiciones [7]:

1. *Son cerrados bajo adición o multiplicación.*
2. *Satisfacen las propiedades conmutativas bajo adición y multiplicación*
3. *Satisfacen la propiedad distributiva*
4. *Existe una identidad aditiva y un inverso aditivo para cada uno*
5. *Existe una identidad multiplicativa para cada elemento en el campo*
6. *Con la excepción de la identidad aditiva, cada elemento en el campo tiene un inverso multiplicativo*

Los campos de orden (cardinalidad) finito son conocidos como *campos de Galois*. El orden de un campo finito es siempre un primo o una potencia de un primo. Un campo Galois de orden p GF(p) es llamado un *campo primo* de orden p.

3.2.1. Operaciones vectoriales y matriciales sobre GF(2)

Para un campo $GF(2) = \{0, 1\}$ las operaciones de adición y multiplicación están definidas por la lógica XOR y AND, respectivamente. Se pueden formar símbolos a partir de la combinación de símbolos fuente o palabras código mediante operaciones lógicas, con lo cual se obtiene un código lineal que también es una palabra de código [29]. Estas operaciones se describen en adelante.

Dados los vectores en GF(2) $\mathbf{x} = [x_0 \quad x_1 \quad ... \quad x_{k-1}]$ y $\mathbf{y} = [y_0 \quad y_1 \quad ... \quad y_{k-1}]$, la suma se puede definir mediante la ecuación 3.7 [7].

$$\mathbf{x + y} = [x_0 \oplus y_0 \quad x_1 \oplus y_1 \quad ... \quad x_{k-1} \oplus y_{k-1}] \tag{3.7}$$

y la multiplicación mediante la ecuación 3.8

$$\mathbf{x.y} = \sum_{i}^{k-1} x_i y_i \tag{3.8}$$

Dada una matriz **G** de dimensiones $N \times M$ en GF(2), la cual está definida como $[g_0^T \quad g_1^T \quad ... \quad g_{M-1}^T]$, donde $g_0^T \quad g_1^T \quad ... \quad g_{M-1}^T$ son vectores columna de **G** con N elementos en GF(2) y dado un vector para $\mathbf{u} = [u_0 \quad u_1 \quad ... \quad u_{N-1}]$, el producto **uG** puede ser definido usando la ecuación 3.9.

$$\mathbf{uG} = [ug_0^T \quad ug_1^T \quad ... \quad ug_{M-1}^T] \tag{3.9}$$

que genera un nuevo vector de dimensiones $1 \times M$ que igualmente pertenece al campo.

3.2.2. Correspondencia del modelo con GF

Los registros de corrimiento utilizados en CDMA conservan la lógica funcional de un *generador realimentado Galois* [35] que puede moverse entre 2^r estados, los caules tienen

al menos un elemento primitivo a partir del cual se generan potencias en cada ciclo de operación del sistema. Estos estados pertenecen a $GF(2^r)$ y están condicionados por el polinomio característico que describe los lazos de realimentación, cuyos coeficientes corresponden al vector $g_{0i} = [g_{00} \quad g_{01} \quad ... \quad g_{0(r-1)}]$ de la matriz 2.7.
En el modelo propuesto, los estados del registro en el *k-esimo* ciclo se describen en el vector

$$\mathbf{R} = \begin{bmatrix} b_k \oplus g_{00} R_{r-1}(k-1) \\ R_0(k-1) \oplus \{g_{01}[b_k \oplus R_{r-1}(k-1)]\} \\ . \\ . \\ . \\ R_{r-2}(k-1) \oplus \{g_{0(r-1)}[b_k \oplus R_{r-1}(k-1)]\} \end{bmatrix}^T = \begin{bmatrix} R_0(k) \\ R_1(k) \\ . \\ . \\ . \\ R_{r-1}(k) \end{bmatrix}^T \tag{3.10}$$

Para las trayectorias de salida, de acuerdo a la definición de la operación multiplicación entre vectores de un campo finito dada según la ecuación 3.8 y a la definición de salida $Y_j(k)$ dada en la ecuación 2.6, se puede verificar una completa correspondencia si se considera que la suma en GF corresponde a la lógica XOR (ec. 3.7).
El vector de salida $\mathbf{Y}$ (que pertenece a $GF(2^n)$) se puede expresar de la forma

$$\mathbf{Y}(k) \quad = \quad \begin{bmatrix} \bigoplus_{i=0}^{r-1} g_{1i} R_i(k) \\ \bigoplus_{i=0}^{r-1} g_{2i} R_i(k) \\ . \\ . \\ . \\ \bigoplus_{i=0}^{r-1} g_{ji} R_i(k) \\ . \\ . \\ . \\ \bigoplus_{i=0}^{r-1} g_{ni} R_i(k) \end{bmatrix}^T = \begin{bmatrix} Y_1(k) \\ Y_2(k) \\ . \\ . \\ . \\ Y_j(k) \\ . \\ . \\ . \\ Y_n(k) \end{bmatrix}^T \tag{3.11}$$

Capítulo 4

Metodología de Validación

Para la validación de los bloques se considera el modelo generalizado como un sistema de *n* salidas y dos entradas como el de la figura 4.1. La primera entrada corresponde a los datos de prueba que van a ser procesados por el bloque desarrollado en VHDL, mientras que la segunda entrada corresponde a los datos que entrega en bloque simulado en *Simulink*, los cuales permitirán comprobar de forma visual los resultados de la simulación funcional en *Max-Plus II.* En proceso de manejo de datos y conexión de los dos entornos de detalla en las siguientes secciones.

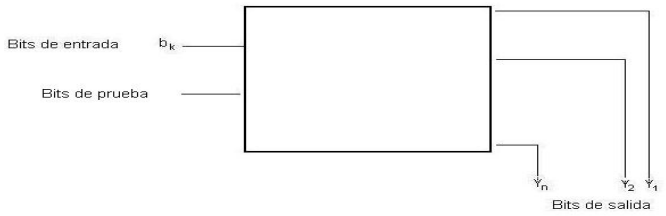

Figura 4.1: *Modelo de prueba*

4.1. Generación de los datos de referencia

A nivel comercial se encuentra un gran número de aplicaciones que permiten el estudio, diseño, desarrollo y evaluación de diversos sistemas de comunicación y, en forma puntual, de los procesos que ocurren dentro de ellos. En cuanto a sistemas CDMA se refiere, existe una herramienta llamada *CDMA Reference Blockset* que está contenida en el entorno

Simulink de *Matlab*, la cual permite visualizar el proceso ocurrido en los canales Reverse y Forward Traffic.
Se obtienen datos de salida del bloque de Simulink a partir de la aplicación de una cadena de bits de entrada, estos datos son comparados con los generados mediante un programa de prueba en *Matlab* que realiza el procesamiento de acuerdo al estándar IS-95. En el anexo A se presenta el código que permite confirmar la validez de los datos entregados por el *CDMA Reference Blockset*, respecto el estándar.

4.2. Creación de las entidades VHDL

Se crean las entidades VHDL mediante el algoritmo de mapeo presentado en la sección anterior y se configuran según sea el caso, aprovechando las ventajas de reconfigurabilidad que brinda el modelo propuesto. Para esto se desarrolla un programa en *Matlab* que recibe la matriz de configuración del sistema de acuerdo a los lineamientos descritos en la metodología de diseño y entrega un código fuente VHDL validado.

4.3. Creación de archivos gráficos de simulación

Las aplicaciones de simulación utilizan archivos de texto para almacenar la información que corresponde a las señales que se presentan en el sistema. Generalmente a cada entrada se asigna una función digital que contiene flancos de subida y/o bajada en tiempos determinados. Para la simulación de sistemas VHDL se maneja un tipo de archivo con extensión **.vec** que presenta un formato como el que aparece en la gráfica 4.2.
Por ejemplo, el anexo B muestra el código fuente del programa que toma los datos de entrada del Codificador Convolucional y los formatea para ser utilizados en la simulación VHDL, convirtiéndose en el puente que conecta los dos entornos.
Aparecen datos como el tiempo de inicio, tiempo final, duración de cada intervalo y los patrones de cambio de cada una de las señales de entrada. Todos estos tiempos se manejan por defecto en *ns* pero pueden ser modificados. En cuanto a los datos de entrada se pueden manejar dos formatos:

Three-Bit Binary Counter

Absolute Time Pattern:

```
UNIT ns;

% equivalent to relative
  time pattern %
INPUTS A1 A2 A3;
PATTERN
0>    0  0  0
100>  0  0  1
200>  0  1  0
300>  0  1  1
400>  1  0  0
500>  1  0  1
600>  1  1  0
700>  1  1  1
800>  0  0  0
900>  0  0  1
1000> X  X  X;
% last pattern is %
% stop time %
```

Relative Time Pattern:

```
UNIT ns;
START 0;
STOP 1000;
INTERVAL 100;
INPUTS A1 A2 A3;
PATTERN
0   0   0
0   0   1
0   1   0
0   1   1
1   0   0
1   0   1
1   1   0
1   1   1;
% repeat until 1000 ns %
```

Figura 4.2: *Formato del archivo tipo vector*

- *Absoluto*, en el que se deben especificar los tiempos de cambio y el valor de las entradas.
- *Relativo*, en el que se establece un intervalo de tiempo fijo al cabo del cual se toma en nuevo bit, contenido en una lista. Debido a las características de los datos de validación obtenidos se opta por utilizar este formato.

Luego de tener el archivo texto, el programa lo convierte en un archivo gráfico con extensión **.scf** como el que aparece en la figura 4.3. A este se le deben agregar las salidas que se quieren visualizar y luego se realiza la simulación y la comprobación.

Como parte del método de validación se hace uso de los datos de salida del correspondiente bloque *Simulink* tomándolos como una entrada adicional del sistema que permite la verificación visual directa de los resultados. La última señal de entrada de la figura 4.3 corresponde a la salida de un bloque FEC.

En la figura 4.4 se describe el procedimiento propuesto para la validación de los bloques diseñados en VHDL.

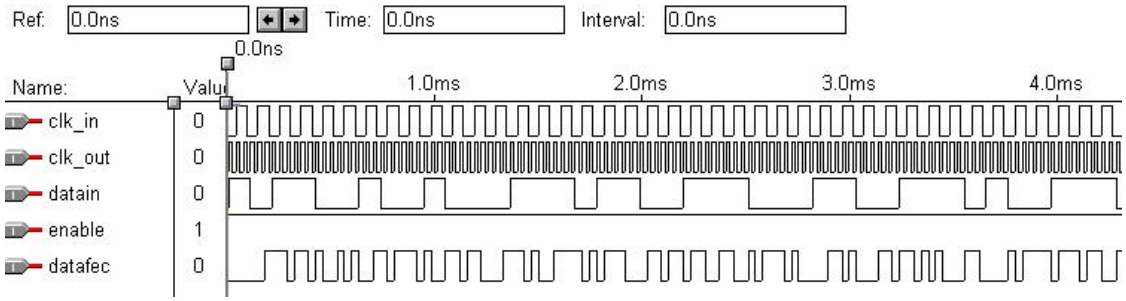

Figura 4.3: *Señales de entrada del sistema*

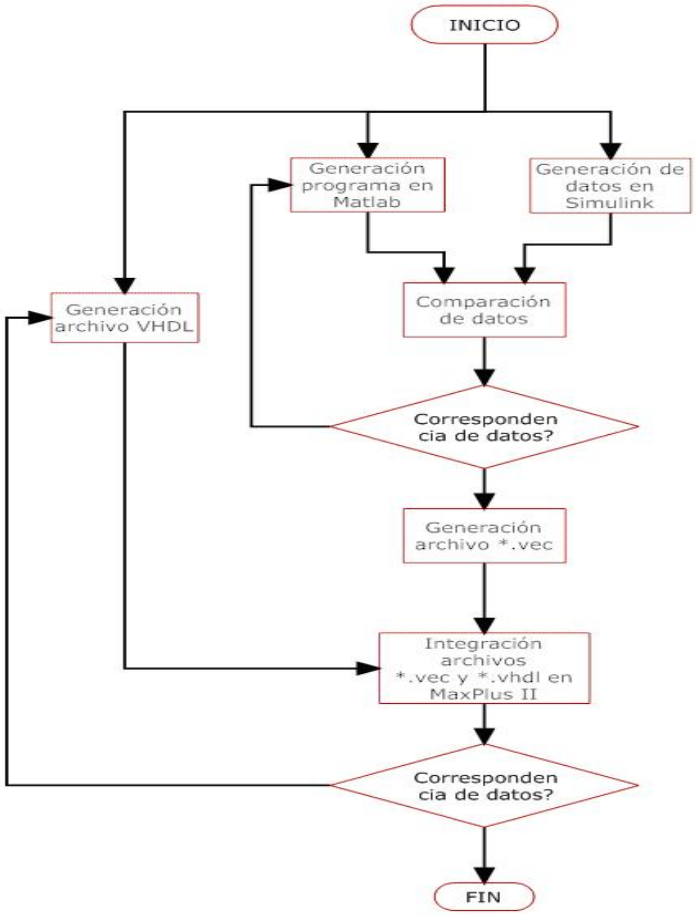

Figura 4.4: *Proceso de validación*

Capítulo 5

Casos de estudio

5.1. Diseño y validación del Codificador Convolucional

Se describe el proceso de validación del bloque Forward Error Correction (FEC) para una velocidad Full Data Rate. Este bloque funcional se encarga de generar un proceso de corrección de errores hacia adelante usando una codificación con tasa $\frac{1}{3}$ que genera información implicita redundante.

5.1.1. Configuración del FEC

Se presenta un Registro de Corrimiento del longitud $K = 9$ y tres sumadores tipo XOR (figura 5.1). Los lazos de realimentación que se pueden configurar con el vector $G_0 = [000000000]$ y tres salidas $(Y_1(t), Y_2(t) y Y_3(t))$ producto de una combinación de las diferentes posiciones del registro de corrimiento y configurables según las siguientes presentaciones del vector de configuración O_j

$$O_1 = G_1 = [101101111]$$
$$O_2 = G_2 = [110110011]$$
$$O_3 = G_3 = [111001001]$$

El flujo de datos de salida está conformado por conjuntos de tres bits que corresponden a la codificación de cada bit de entrada.

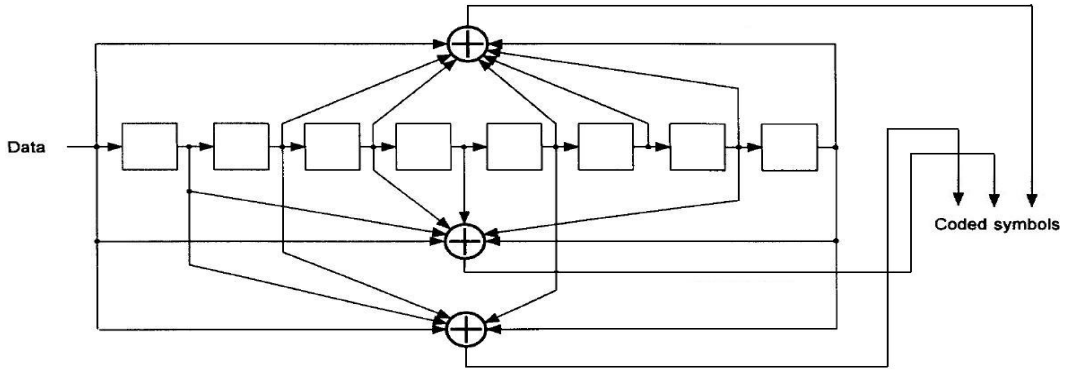

Figura 5.1: *Codificador Convolucional para el canal Reverse: r=1/3, K=9*

5.1.2. Validación

Se utilizó un tiempo de simulación de 20ms, tiempo en el que se transmite una trama. En el caso del bloque *IS-95A Rev Ch Convolutional Encoder*, este recibe un arreglo de datos binarios por parte del bloque *IS-95A Rev Ch CRC Generator* con una longitud de 288 bits, de los cuales el bloque FEC toma los 192 primeros que corresponden a la trama de 20 ms, y los transforma en 576 a través de un proceso de codificación convolucional con relación $\frac{1}{3}$.

Tanto el bloque *IS-95A Rev Ch CRC Generator* como el bloque *IS-95A Rev Ch Convolutional Encoder* están conectados al *Command Window* de Matlab a través de dos bloques *To Workspace* llamados *crct* y *fec*, respectivamente (ver figura 5.2).

Generación del archivo $*$.vec

Se desarrolló un programa en *Matlab* llamado *fec96vec.m* el cual toma el arreglo *crct* y lo transforma según el formato utilizado por los archivos tipo vector de *Max+Plus II*, asignado un tiempo de inicio y un tiempo de finalización de la simulación y asignando los puertos de entrada y los intervalos en que se transmiten los datos.

En este caso el puerto más significativo es el *datain* que recibe los 192 datos procedentes del bloque CRC. En el anexo B se muestra el programa de generación del archivo llamado *fec96.vec*.

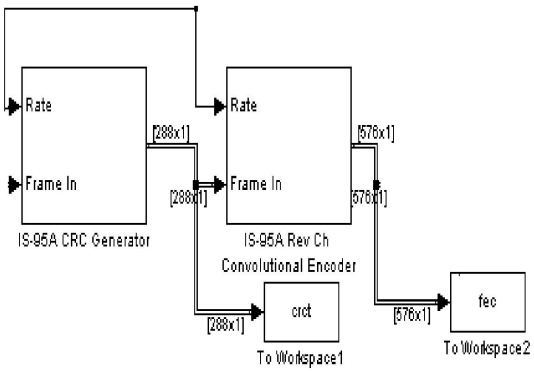

Figura 5.2: *Bloques To Workspace utilizados*

Luego de correr el programa *fec96vec.m* se obtiene el archivo de texto que aparece en el anexo C, el cual contiene la información necesaria para que la aplicación gráfica de simulación de *MaxPlus II* pueda obtener los datos de entrada.

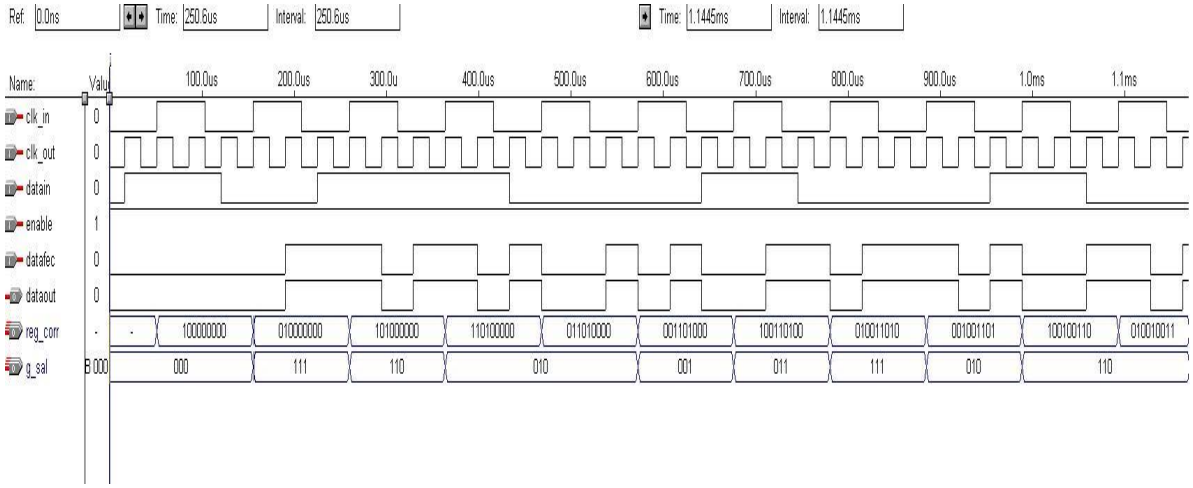

Figura 5.3: *Resultados de la simulación del bloque FEC*

Data Rate (Vel)	Factor de Repetición	Velocidad de Salida
Full (28.8 Kbps)	1	28.8 Kbps
Half (14.4 Kbps)	2	28.8 Kbps
Quarter (7.2 Kbps)	3	28.8 Kbps
One Eighth (3.6 Kbps)	4	28.8 Kbps

Tabla 5.1: Factores de repetición

5.2. Diseño y validación del Repetidor

Para llevar las diferentes velocidades de entrada a un valor estándar, el bloque Symbol Repeater genera una repetición de cada uno de los bits provenientes del FQI Encoder (CRC Generator) de acuerdo a la tabla 5.1.

La figura 5.4 presenta un esquema generalizado del bloque repetidor, el cual, en cualquier caso solo debe contener un Registro unitario a partir del cual se generan las *n* salidas correspondientes al factor *n* de repetición.

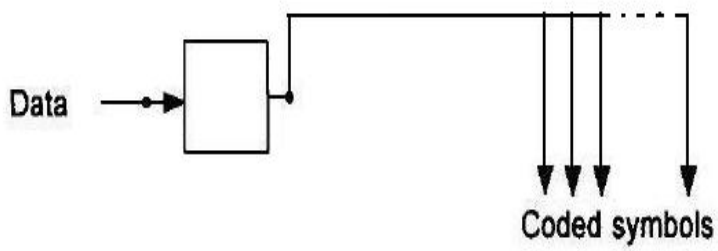

Figura 5.4: *Representación del Symbol Repeater mediante Registros de corrimiento*

5.2.1. Configuración del Symbol Repeater

El registro de corrimiento será de longitud $K = 2$ sin realimentación con $F = [00]$. Aplicando al caso Half Data Rate que duplica los bits de entrada, se tendrá un sistema con dos salidas configuradas por los vectores

$$O_1 = G_1 = [01]$$

$$O_2 = G_2 = [01]$$

5.2.2. Validación

Luego de simular el repetidor para una velocidad Half Data Rate se obtienen los resultados de la figura 5.5, en la cual se aprecia una entrada llamada *datarep* que contiene la información resultante en la simulación en *Simulink*. Esta información fue adicionada a los datos que se aplican a los puertos de entrada en el archivo *.vec* con el fin de permitir una inspección visual de la validez de las simulaciones. Las señales de entrada y salida son prácticamente las mismas, la diferencia básica se marca por los relojes de entrada y salida que tienen una relación de velocidad *1/n*, en este caso *1/2*.

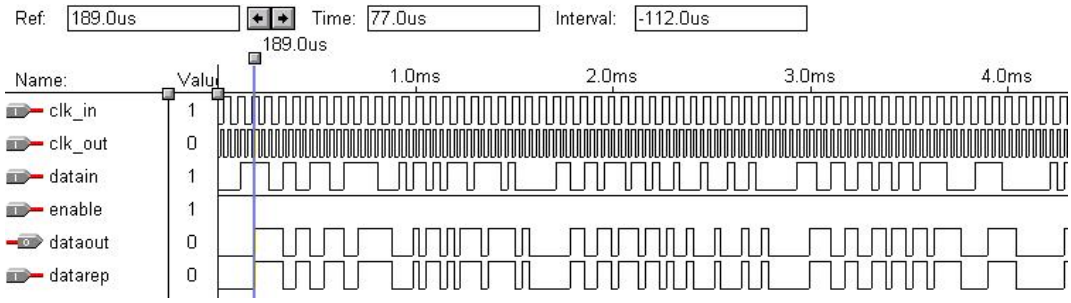

Figura 5.5: *Resultados de la simulación del bloque Repeater*

5.3. Diseño y validación del generador CRC

El CRC utiliza Registros de Corrimiento Realimentados Linealmente (LFSR) configurados de acuerdo a un Polinomio Generador. Dependiendo de la velocidad del canal se utiliza una función generadora representada por un polinomio de grado *n*.

La función de salida se genera por la suma XOR entre el dato de entrada y la última posición $(X_{k-1}(t))$ del registro de corrimiento.

Para la validación del CRC de utilizaron dos ejemplos presentados en el capítulo 5 de la referencia [20] con polinomios de orden 3 y 8.

5.3.1. Configuración Generador CRC-3

Se validó el bloque haciendo uso del generador CRC-3 de la figura 5.6 que utiliza un registro de corrimiento de longitud $K = 3$ con lazos de realimentación dados por $F = [1100]$ y una salida que se configura a partir del vector

$$O_1 = [1000]$$

para $i < 4$, y

$$O_1 = [0001]$$

para $4 \leq i < (4 + 3)$.

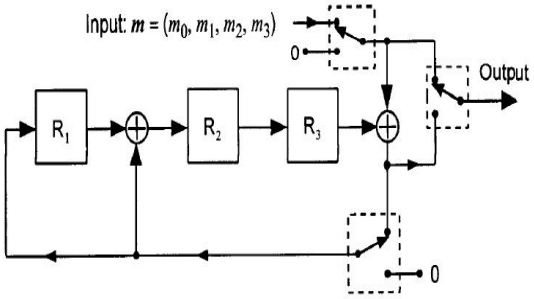

Figura 5.6: *Codificador CRC de orden 3*

5.3.2. Validación

El codificador recibe la cadena de datos de entrada $I = [1110]$ y adiciona en la salida un vector $crc = [100]$ que corresponde a un procesamiento a partir de la ecuación $x^3 + x + 1$. Los resultados de la simulación aparecen en la figura 5.7.

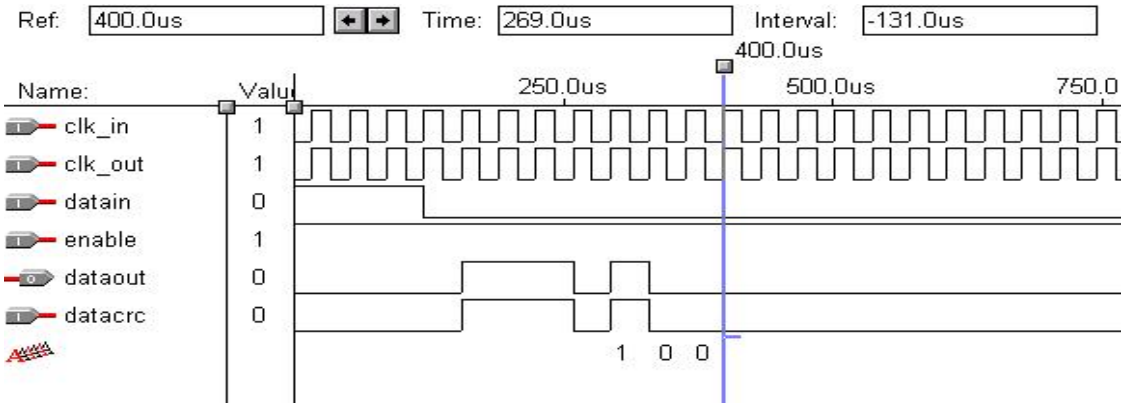

Figura 5.7: *Simulación para el codificador CRC-3*

Capítulo 6

Conclusiones

Mediante la aplicación de la metodología propuesta se puede realizar una validación de los bloques funcionales, tal como se presentó en los casos de estudio del capítulo 5. Esta metodología puede ser aplicada a otro tipo de sistemas de comunicación que puedan ser simulados para obtener datos orientados a la validación de su modelo VHDL. Esto se logra, de forma particular, generando una interfaz entre el *CDMA Reference Blockset* de *Simulink* y la aplicación *Simulator* del entorno de desarrollo *Max+Plus II*, donde se pueden comparar datos por medio de un archivo tipo vector (*.vec), el cual se utiliza para generar la descripción gráfica del comportamiento de los diferentes puertos de una entidad VHDL. Los datos utilizados para la generación del archivo *.vec* son extraídos de *Simulink* y procesados en consola por medio de un programa de *Matlab*.

Diseñando clases como `Funct_Block` se pueden integrar las características funcionales generales basadas en la aplicación de registros de corrimiento con propiedades de realimentación y generación de funciones de salida. Esta clase se asocia a varias clases y objetos relacionados con los bloques funcionales de *CDMA Reverse Traffic Channel* y puede ser extendida a otro tipo de bloques con funcionalidades diferentes.

Al generar especificaciones para la estructuración de una matriz de configuración (similar funcionalmente a la que se propone en [27]) se pueden definir características operacionales relacionadas con las funciones de salida y los procesos de realimentación de los registros de corrimiento.

La herramienta de traducción automática, similar a la que se propone en [34], permite mapear las definiciones de clases correspondientes a los bloques funcionales como parejas

entidad-arquitectura en un código VHDL, ofreciendo la posibilidad de utilizar parámetros configurables, lo cual permite adicionalmente evaluar bloques funcionales con especificaciones diferentes a las del estándar IS-95.

Como trabajo posterior se propone la utilización de los archivos de texto ***.tbl*** que pueden ser obtenidos luego de la simulación en *Max+Plus II*. Estos archivos presentan el comportamiento de los puertos de entrada y salida durante el tiempo de simulación (una muestra de este archivo se presenta en la figura 6.1) y podrían ser utilizados para generar un proceso que permita comparar de forma automática los resultados del *Max+Plus II* con los de *Matlab*, lo cual se constituye como el proceso inverso al que se presenta en este documento.

```
PATTERN
%                          r              %
%              c         d e              %
%           c l d e      a g              %
%           l k a n      t _         g    %
%           k _ t a      a c         _    %
%           _ o a b      o o         s    %
%           i u i l      u r         a    %
%           n t n e      t r         l    %

       0.0> 0 0 0 1 = 0 000000000 000
   17361.0> 0 1 1 1 = 0 000000000 000
   34722.0> 0 0 1 1 = 0 000000000 000
   52083.0> 1 1 1 1 = 0 000000000 000
   52083.1> 1 1 1 1 = 0 100000000 000
   69444.0> 1 0 1 1 = 0 100000000 000
   86805.0> 1 1 1 1 = 0 100000000 000
  104166.0> 0 0 1 1 = 0 100000000 000
  121527.0> 0 1 0 1 = 0 100000000 000
  138888.0> 0 0 0 1 = 0 100000000 000
  156249.0> 1 1 0 1 = 0 100000000 000
  156249.1> 1 1 0 1 = 0 010000000 111
  173610.0> 1 0 0 1 = 0 010000000 111
  190971.0> 1 1 0 1 = 0 010000000 111
  190971.1> 1 1 0 1 = 1 010000000 111
```

Figura 6.1: *Archivo .tbl con los resutados de la simulación del FEC.*

Bibliografía

[1] Altera, dsp builder user manual. www.altera.com.

[2] Xilinx, system generator user manual. www.xilinx.com.

[3] A. D. B. L. A. Msir, F. Monteiro. Design of a high speed parallel encoder for convolutional codes. *Microelectronics Journal*, (35):151–166, 2004.

[4] H. A.Hofmann, T.Kogel. A framework for fast hardware –software co-simulation. *Proceedings of Design Automation and Test in Europe*, pages 760–764, 2001.

[5] J. Baker. Stochastic modeling for automatic speech understanding. *in Readings in Speech Recognition*, pages 297–307, 1990.

[6] D. Blandón. Diseño y simulación de un canal `CDMA` reverse por medio de `VHDL`. Technical report, 2004.

[7] C. Chien. *Digital Radio Systems on a Chip: A systems approach*. Kluwer Academic Publishers, 2000.

[8] T. T. W. C.Schulz-Key, M.Winterholer. Object-oriented modeling and synthesis of systemc specifications. *Proceedings of the Asia South Pacific Design Automation Conference*, pages 238–243, 2004.

[9] G. Céspedes. Diseño e implementación de un conmutador b-isdn utilizando descriptores en vhdl. Technical report, 2004.

[10] A. Cuenca. Diseño e implementación en vhdl de un conmutador atm. Technical report, 2004.

[11] J. C. V. Diaz. Implementación en vhdl de un detector de señales satelitales de baja relación eb/no y alta desviación en frecuencia doppler. Technical report, Fundación Universidad Del Norte, 2003.

[12] A. T. G.Ch. Sirakoulis, I. Karafyllidis. A cad system for the construction and vlsi implementation of cellular automata algorithms using vhdl. *Microprocessors and Microsystems*, (27):381–396, 2003.

[13] M. Gries. Methods for evaluating and covering the design space during early design development. *INTEGRATION,the VLSI journal*, (38):131–183, 2004.

[14] Z. Hai He, Yifnag. A generic framework for modeling heterogeneus real-time systems. *Computer Standars and Interfases*, (28):43–58, 2005.

[15] S. Haykin. *Communication Systems*. John Wiley and Sons, 2001.

[16] H. S. J. Fernandes, R. Machado. Modeling industrial embedded systems with uml. *Proceedings of the Eigth IEEE/IFIP/ACM International Workshop on Hardware/Software Co-Design (CODES'2000)*, pages 18–23, 200.

[17] A. P. M. S. Joachim Fischer, Eckhardt Holz. Tool-based language development. *Computer Networks*, (49):676–688, 2005.

[18] G. D. Jong. A uml-based design methodology for real time and embedded systems. *Proceedings of the Design Automation and Test in Europe DATE 2002*, pages 776–778, 2002.

[19] H. R. Julián Busques. Desarrollo y simulación de una estación base gsm/cdma utilizando software radio. *Revista Facultad de Ingeniería U.T.A.*, (10):3–10, 2002.

[20] J. S. Lee and L. E. Milles. *CDMA Systems Engineering Handbook*. Artech House, 2002.

[21] F. R. L.M. Reyneri. Speeding-up the design of hw/sw implementations of neuro-fuzzy systems using the codesimulink environment. *Applied Soft Computing*, (4):227–240, 2004.

[22] R. D. López. Diseño y simulación en lenguaje descriptor de hardware (vhdl) de un multiplexor pdh/e1. Technical report, 2004.

[23] E. B. Markus Rupp, Andreas Burg. Rapid prototyping for wireless designs: the five-ones approach. *Signal Processing*, (83):1427–1444, 2003.

[24] W. Percybrooks. Implementation of an h.263 codec for umts using vhdl. *XVII Conference on Design of Circuits and Integrated Systems DCIS*, pages 681–686, 2002.

[25] A. M. Q. Zhu. An object-oriented design process for system-on-chip using uml. *Proceedings of the 15th Internationl Symposium on System Synthesis (ISSS 2002)*, pages 249–254, 2002.

[26] D. G. R. Allen. The case for c/c++ hardware design. *EETimes*, 2000.

[27] V. S. R. Damasevicius. Application of the object-oriented principles for hardware and embedded system design. *INTEGRATION,the VLSI journal*, (38):309–339, 2004.

[28] L. Reyneri. A simulink-based hybrid codesign tool for rapid prototyping of fpga's in signal processing systems. *Microprocessors and Microsystems*, (28):273–289, 2004.

[29] R. Soleymani. *Turbo Coding for Satellite and Wireless Communications*. Kluwer Academic Publishers, 2002.

[30] P. S.Swan, D.Vermeersch. Functional specification for systemc 2.0, white paper. Technical report, OSCI, 2001.

[31] M. Y. Takashi Kambe, Akihisa Yamada. Trend of system level design and an approach to c-based design. *Microelectronics Journal*, (33):875–880, 2002.

[32] C. U.Bortfeld. White paper c++system simulation interfaces. Technical report, Infineon, July 2000.

[33] F. S. V.N.Patel, C.K.Wiese. Rapid development and commercialization of products: A business imperative in the global telecommunication landscape. *Bell Labs Technical Journal*, (4):157–171, 2000.

[34] B. H. William McUmber. Uml-based analysis of embedded systems using a mapping to vhdl. *Proceedings of the IEEE International Symposium on High Assurance Software Engineering*, pages 56–63, 1999.

[35] K. Zigangirov. *Theory of Code Division Multiple Access Communication*. Jonh Wiley & Sons, 2004.

[36] L. F. Zuluaga. Desarrollo de herramientas para la construción de un dispositivo de conmutación digital frame relay utilizando descriptores en vhdl. Technical report, 2004.

Apéndice A

Generación de datos de verificación

A continuación se presenta el programa desarrollado en Matlab para determinar la correspondencia de los datos del bloque IS-95A Rev Ch Convolutional Encoder con los lineamientos dados por el estandar. Dentro del programa se utiliza una matriz de nombre GFx que permite al usuario ingresar los datos de configuración de realimentación y salida, ademas de determinar el orden del registro de corrimiento a utilizar.

```
function [] = crc_fec6(crc)

    R = input('Longitud del registro:  ');
    G = input('Cantidad de salidas:  ');
    FGx = input ('Ingrese la matriz [[feedB][G1][G2]...[Gg]]   ');
    li = input ('Cantidad de datos de entrada:  ');
    [f,c]=size(FGx);
    while (f~=(G+1))|(c~=R)
       FGx = input ('Dimensones incorrectas... Matriz [G+1 x R]    ');
       [f,c]=size(FGx)
    end

    FG=FGx

    fec2=zeros(1,li*G);
    shiftR=zeros(1,R);
    penultimo=0;
    for i = 1:li
       crc_i=crc(i)
       penultimo=shiftR(1,R-1)
       for j=2:R
           shiftR(1,R-j+2)= xor(shiftR(1,R-j+1),and(xor(crc(i),penultimo),FG(1,R-j+2)));
       end
       shiftR(1,1)= xor(crc(i),and(penultimo,FG(1,1)))


       for m=1:G
           dato=0;
           for n=1:R
                dato=xor(dato,and(shiftR(1,n),FG(m+1,n)));
           end
           fec2(G*i-(G-m))=dato
       end
    end

    for s=1:R-1
       fec2(G*li+s)=shiftR(s)
    end
```

Apéndice B

Generación del archivo *.vec

El siguiente código corresponde al procedimiento para la generación de los archivos gráficos tipo vector (*.vec) que son utilizados en Max-PlusII para realizar la simulación y validación de las estructuras en VHDL.

```
function[R]=fec96vec(crct)

% Se abre un archivo de texto solo escritura
fid=fopen('fec96.vec','wt');

% Se determinan los rangos de funcionamiento del sistema. En este caso
% el tiempo de fin simulacion es de 19.999872 ms y el intervalo absoluto
% para cada dato es de 34722 ns = 2*17361 ns.
% Es necesario hacer una aproximacion debido a que la relacion entre la
% velocidad del reloj de salida y el reloj de entrada es de 3 a 1, lo cual
% algunas fracciones de ns que no podrian ser obtenidas.
fprintf(fid,'START 0 ; \n STOP 19999872 ; \n INTERVAL 17361 ; \n\n ');

% El reloj de entrada es tres veces mas lento que el reloj de salida.
fprintf(fid,'INPUTS clk_in ; \n PATTERN \n 0 0 0 1 1 1 ;\n\n ');


fprintf(fid,'INPUTS clk_out ; \n PATTERN \n 0 1  ;\n\n ');

% Los datos de entrada en el puerto "datain" se repiten cada tres intervalos
% del reloj de salida debido a relacion 1 a 3 entrada-salida.
fprintf(fid,'INPUTS datain ; \n PATTERN \n 0 ');
for n=1:192
   fprintf(fid,'%d %d %d %d %d %d ',crct(n),crct(n),crct(n),crct(n),crct(n),crct(n));
end

% Se activa la habilitacion "enable" del sistema para t > 0 ns.
fprintf(fid,';\nINPUTS enable ; \n PATTERN \n  0>   1 ;')


fclose(fid)
```

Apéndice C

Formato del archivo *.vec

El siguiente es el formato de los archivos tipo vector, que el Max-Plus II utiliza para crear los archivos gráficos de simulación tipo *.*scf*.

```
START 0 ;
STOP 19999872 ;
INTERVAL 17361 ;

INPUTS clk_in ;
PATTERN
0 0 0 1 1 1 ;

INPUTS clk_out ;
PATTERN
0 1  ;

INPUTS datain ;
PATTERN
0 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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 0 0 0 0 1 1 1 1 1
1 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 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 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 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 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1
1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ;

INPUTS enable ;
PATTERN
0>   1 ;
```