UNIVERSIDAD DE EL SALVADOR

FACULTAD DE INGENIERÍA Y ARQUITECTURA

ESCUELA DE INGENIERÍA ELÉCTRICA

TRABAJO DE GRADUACIÓN:

"Diseño e implementación de un instrumento virtual de monitoreo de señalización CAS R2 digital"

PRESENTADO POR:

ALEXANDER VLADIMIR CONDE JACOBO

DAVID ALBERTO RIVAS PREZA

PARA OPTAR AL TÍTULO DE:

INGENIERO ELECTRICISTA

CIUDAD UNIVERSITARIA, DICIEMBRE DE 2004

UNIVERSIDAD DE EL SALVADOR

RECTORA :
Dra. María Isabel Rodríguez

SECRETARIA GENERAL :
Alicia Margarita Rivas de Recinos

FACULTAD DE INGENIERIA Y ARQUITECTURA

DECANO :
Ing. Mario Roberto Nieto Lovo

SECRETARIO :
Ing. Oscar Eduardo Marroquín Hernández

ESCUELA DE INGENIERÍA ELECTRICA

DIRECTOR :
Ing. Luis Roberto Chévez Paz

UNIVERSIDAD DE EL SALVADOR
FACULTAD DE INGENIERÍA Y ARQUITECTURA
ESCUELA DE INGENIERÍA ELÉCTRICA

Trabajo de Graduación previo a la opción al Grado de:
INGENIERO ELECTRICISTA

Título :

"Diseño e implementación de un instrumento virtual de monitoreo de señalización CAS R2 digital"

Presentado por :

Alexander Vladimir Conde Jacobo
David Alberto Rivas Preza

Trabajo de Graduación aprobado por:

Docente Director :

Ing. Werner David Meléndez Valle

Docente Director :

Ing. René Naúm Clímaco Cortez

San Salvador, Diciembre de 2004

Trabajo de Graduación Aprobado por:

Docentes Directores:

Ing. Werner David Meléndez Valle

Ing. René Naúm Clímaco Cortez

Dedicado a nuestras familias, por su apoyo incondicional.

PREFACIO

La señalización R2 digital es un sistema de señalización de canal asociado creado en los años 60 por el CCITT (ahora UIT-T), que en la actualidad es utilizada especialmente en la conexión de las centrales telefónicas privadas (PBX) con la red pública de telefonía conmutada. Aunque originalmente formaba parte de la señalización inter centrales de la red pública, su uso para este propósito ha sido sustituido a raíz de la aparición de otros sistemas, tales como la señalización CCS7.

A pesar del hecho que buena parte de las centrales privadas del país utilizan esta señalización, existe una escasez de equipo de monitoreo, el cual es indispensable para el diagnóstico y corrección de errores en este tipo de redes.

En base a esta necesidad se planteó el siguiente trabajo de graduación en el cual se realiza el diseño e implementación de un instrumento virtual de monitoreo de señalización R2 digital. El instrumento que aquí se desarrolla no posee la capacidad de monitorear de forma completa la señalización R2, sino solo la parte de línea. Sin embargo, debe destacarse que este constituye un avance en el área de la investigación y desarrollo de equipo de telecomunicaciones dentro de la universidad, y pretende servir de base a trabajos posteriores relacionados a este y otros sistemas de señalización.

El instrumento virtual consta de dos partes: un circuito de interfaz y un programa para computadora personal con la capacidad de interpretar y presentar la información recibida. Este instrumento virtual se encamina funcionalmente a cumplir con el monitoreo de señalización de canal asociado. Un paso posterior importante será cumplir fielmente con las especificaciones y recomendaciones pertinentes sobre aparatos de medición que vienen detalladas en el compendio de Recomendaciones de la Serie O de la UIT-T, además de las Rec. Q.400-490 relacionadas a la señalización R2 digital.

RESUMEN

El objetivo principal del presente trabajo es el diseño y la implementación de un instrumento virtual para monitoreo de señalización R2 digital de línea. Para dar cumplimiento a este objetivo, el trabajo se ha desarrollado con la siguiente estructura: Generalidades e información básica del sistema de señalización R2 digital, diseño del circuito eléctrico interfaz, desarrollo del programa de decodificación de señalización R2 de línea, y análisis e interpretación de resultados obtenidos.

El primer capítulo es un resumen de los aspectos teóricos, protocolos, normas y conceptos necesarios para el planteamiento del diseño y la posterior implementación del instrumento. Este se desarrolla a partir de la estructura de la red pública de telefonía conmutada (PSTN) y la señalización formando parte de esta red. La teoría involucrada está normalizada bajo los estándares de la UIT-T, lo cual constituye la base para las etapas posteriores.

El diseño del circuito eléctrico interfaz con la PC se plantea en el segundo capítulo. A pesar del hecho de ser una interfaz en la que se requiere tener un programa que pueda interpretar la información recibida, esta etapa toma en cuenta, de manera inicial, los conceptos y normas desarrollados en el capítulo anterior. La información presentada incluye diagramas de bloques, circuitos y diagramas de tiempo, además de explicaciones detalladas de todos estos.

El tercer capítulo es el desarrollo del programa que ha de implementar todos los principios y normativa del capítulo I. Primero se presenta un marco teórico de las técnicas de programación y consideraciones necesarias para su implementación, para luego mostrar la estructura de los algoritmos resultantes. El recurso principal utilizado para explicar tales procesos son los flujogramas, dejando el código fuente completo para los anexos.

Finalmente en el cuarto capítulo se presentan los resultados obtenidos, los cuales incluyen la descripción de las pruebas que garantizaron el funcionamiento confiable del aparato y corroboraron el diseño, algunos resultados de estas pruebas, detalles técnicos del instrumento implementado y propuestas de mejora.

TABLA DE CONTENIDOS

Capítulo **Página**

LISTA DE FIGURAS

LISTA DE CUADROS

CAPÍTULO I

GENERALIDADES E INFORMACIÓN BÁSICA DEL SISTEMA DE SEÑALIZACIÓN R2 DIGITAL

INTRODUCCIÓN

En este capítulo se presenta el contexto, teoría básica y especificaciones de las interfaces y protocolos relacionados al diseño del instrumento de monitoreo. El capítulo se divide en tres partes principales: primero, se muestra a la señalización como un elemento de la red pública telefónica conmutada. Se da, además, una reseña histórica y estructura básica de la red y algunos aspectos de la transmisión digital de la voz.

En la siguiente sección se muestra información detallada de la interfaz E1, que es la interfaz de transmisión para la que se ha diseñado el instrumento. Esta información incluye especificaciones eléctricas, esquemas de trama y algoritmos de alineación.

Finalmente se presenta la señalización R2 digital, objeto del trabajo, con sus partes de línea e inter registro. Se hace énfasis en la señalización de línea, ya que el instrumento está limitado a la decodificación de ésta parte del código.

El texto está basado principalmente en las recomendaciones del Sector de Normalización de las Telecomunicaciones de la Unión Internacional de las Telecomunicaciones (UIT-T, antes CCITT), a las que se apegan los sistemas de comunicaciones a nivel mundial.

1.1 LA RED PÚBLICA DE TELEFONÍA CONMUTADA

La Red Pública Telefónica Conmutada (Public Switched Telephone Network, PSTN) es la interconexión de las redes públicas telefónicas conmutadas por circuito de todo el mundo. Su uso hoy en día resulta algo tan cotidiano, que difícilmente se logran percibir la cantidad de procesos que se ejecutan para la simple operación de realizar una llamada.

Su historia comienza el 10 de Marzo de 1876, cuando Alexander Graham Bell logra transmitir su voz a través del primer teléfono. A partir de ese momento nació una nueva industria que a través del tiempo ha ido evolucionando permanentemente.

Hoy en día la telefonía está extendida por todo el mundo, lo cual ha provocado mucha diversidad en cuanto a las diferentes técnicas que interactúan dentro de la PSTN, sin embargo se mantiene una estructura básica que logra mantener armonía aún con las variantes tecnológicas que pudieran presentarse. La estructura básica está compuesta de los siguientes elementos:

a) Conmutación
b) Señalización
c) Transmisión
d) Gestión
e) Datos
f) Equipos Terminales
g) Servicios
h) Tecnología Inalámbrica.

Cada uno de ellos constituye un tema amplio y con mucho contenido. El presente trabajo está enfocado hacia un elemento específico, la señalización, y más específicamente a la señalización R2 digital, tema que es desarrollado con mayor detalle en secciones posteriores.

1.1.1 Estructura de la PSTN

En sus inicios, el servicio telefónico consistía en una conexión directa hacia cada uno de los teléfonos que conformaban la red local. A medida que pasó el tiempo, y la demanda de usuarios empezó a crecer exponencialmente, esta técnica ya no era factible. Fue entonces cuando surgió la idea de una central telefónica (Central Office, CO), en donde se enruta cada una de las llamadas. De esta manera surge el concepto de la conmutación, el cual se define como el establecimiento de un circuito o canal de comunicaciones entre el punto de origen de la llamada y el punto donde finaliza la llamada. Esta idea se desarrolló y también ha estado evolucionado con el pasar del tiempo. El cuadro 1.1 ilustra algunos momentos importantes.

Sistema de conmutación	Operación	Método de Conmutación	Tipo de control	Tipo de red
1878 operador manual	Manual	Contactos/Analógica	Manual	Cables desmontables
1892 Paso a paso	Electromecánica	Contactos/Analógica	Distribución etapa por etapa	Secuencia de pulsaciones
1918 Barra cruzada	Electromecánica	Contactos/Analógica	Control común	Interruptor de barras
1960 Conmutación Electrónica. Primera Generación	Semi-Electrónica	Contactos/Analógica	Control común	Interruptor digital
1972 Conmutación Electrónica. Segunda Generación	Semi-Electrónica	Contactos/Analógica	Control programado	Interruptor digital
1976 Conmutación Electrónica. Tercera Generación	Electrónica	Sincronizada/Digital	Control programado	Modulación de impulsos Codificados. PCM

Cuadro 1.1 Evolución de la conmutación telefónica.

Como se observa en el cuadro 1.1, la conmutación se ha desarrollado hasta tal punto que hoy en día se cuenta con un sistema automático y completamente electrónico. También, el circuito telefónico es digital, es decir no se establece físicamente un circuito eléctrico.

El desarrollo de las técnicas de conmutación obedeció a la creciente demanda del servicio y por consiguiente al alcance geográfico que era necesario establecer entre los usuarios. Lo que empezó por un sistema que controlaba las operaciones para una pequeña localidad, pronto se expandió hasta brindar cobertura nacional, y posteriormente a nivel internacional.

La expansión de la cobertura telefónica dio origen a la red telefónica, en la cual el nivel básico de interconexión se encuentra en las centrales telefónicas finales (End Office), las cuales representan el punto más próximo al usuario (abonado). La red posee una topología de tipo estrella, siendo la CO, tal como se menciono anteriormente el punto de llegada de todos los usuarios ubicados geográficamente en el mismo sector.

Cuando un enlace se quiere establecer hacia puntos más lejanos, las centrales se conectan entre sí, dentro de otras centrales de más alta jerarquía llamadas centrales tandem. De esta manera, la red telefónica puede extenderse hasta dar cobertura internacional. La figura 1.1 esquematiza la manera en que está estructurada la red PSTN. En esta estructura, las CO de más bajo nivel son llamadas CO clase 5, y las más altas, CO clase 1.

Dentro de la red telefónica el último eslabón lo conforman los abonados. Es posible que un abonado pueda poseer más de una línea. Sin embargo puede haber situaciones donde exista una alta demanda de líneas telefónicas. Este es el caso de las empresas con la necesidad de redes de voz completas dentro de sus instalaciones. Usualmente estas empresas optan por conectarse a la PSTN, a través de pequeñas centrales llamadas PBX (Public Branch Exchange) [3].

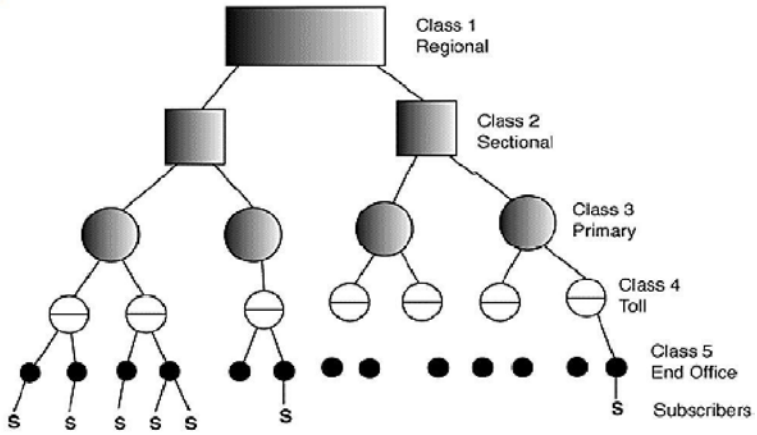


Figura 1.1 Estructura básica de la PSTN.

1.1.2 Digitalización de las señales de voz

Hoy en día los sistemas PCM predominan dentro de la estructura de la PSTN. PCM (Pulse Code Modulation) es una técnica de modulación digital en la que la señal analógica se prueba y se convierte a una longitud fija, es decir, un número binario serial adecuado para la transmisión. El valor de este número varía de acuerdo a la amplitud de la señal analógica.

Para transmitir voz en un medio digital, la señal analógica de la voz debe ser transformada en un formato binario, para el caso, utilizando la modulación PCM. Las señales de voz poseen una frecuencia máxima de aproximadamente 4000Hz. A partir de esto y tomando en cuenta el teorema de Nyquist, la señal de voz debe muestrearse al menos a una frecuencia de 8000 muestras / segundo. La velocidad de muestreo utilizada en las redes PCM es, justamente, 8 kHz.

Los sistemas PCM convierten cada muestra en una palabra de 13 ó 14 bits, sin embargo en el momento de la transmisión la muestra es una palabra de 8 bits. Esta conversión del tamaño en bits de las palabras obedece a una técnica llamada compansión, la cual se define como el proceso de comprimir y después expandir. La compansión se realiza cuando las señales analógicas muestreadas se procesan usando una formula no lineal que, efectivamente, transforma la muestra en una palabra de 8 bits, afectando a las magnitudes mayores y con mayor resolución para las amplitudes menores. El resultado no distorsiona el tono de la voz en forma significativa, ya que de hecho, en una transmisión de voz, las amplitudes menores predominan. Los sistemas PCM europeo y estadounidense utilizan dos fórmulas distintas para realizar la compansión, conocidas como ley A y ley μ, respectivamente.

Debido a que las señales de voz digitalizadas poseen 8 bits y se muestrean a una velocidad de 8 kHz, el flujo de datos resultante para la transmisión de voz es de 64 kbps, que

representa un canal telefónico básico, que en el sistema estadounidense se conoce como DS0. El proceso de digitalización se ilustra en la figura 1.2.

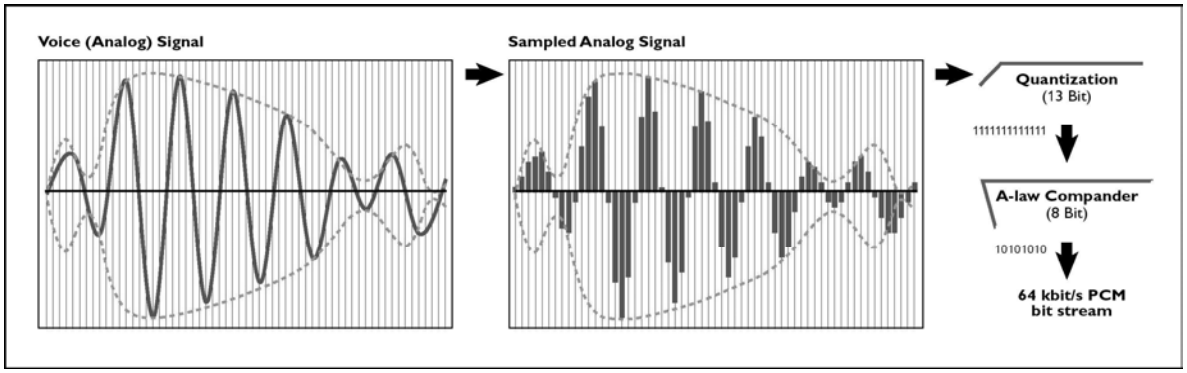


Figura 1.2. Esquema de conversión de voz a transmisión PCM.

Los datos luego de ser digitalizados, se encuentran listos para ser transmitidos. Al momento de la transmisión; los datos binarios producto de la modulación, pueden representarse bajo distintas formas en el nivel físico. Las distintas formas de representación eléctrica se conocen como códigos de línea. Existen varias razones para la existencia de estos códigos, por ejemplo, mejor aprovechamiento de ancho de banda y para sincronización.

Uno de estos códigos de uso común es el HDB3 (High Density Bipolar-3 Zeros). Este código está basado en una variación del código AMI, en donde los unos se representan de manera alternada por pulsos negativos y positivos y los ceros se representan por ausencia de señal o pulsos que representan violaciones a la alternancia de los unos [4].

1.1.3 Transporte de los canales de voz dentro de la PSTN

Los canales telefónicos de 64 kbps son la forma más básica en que la conmutación toma lugar en una central telefónica. Estos canales son conocidos también como espacios de tiempo (timeslots) ya que se multiplexan utilizando la división por tiempo. Múltiples DS0 se unen para formar circuitos de mayor capacidad, así, en el sistema estadounidense 24 DS0 forman un DS1, que cuando se transporta en líneas de cobre se conoce como T1. En el sistema europeo, de uso en el país, 32 canales de 64kbps forman un E1 (Figura 1.3). En las redes modernas, esta multiplexación es llevada tan cerca del usuario como sea posible, colocando el equipo en gabinetes próximos a la vía pública en las áreas residenciales o dentro de locales comerciales grandes. La multiplexación tiene lugar también dentro de las PBX.

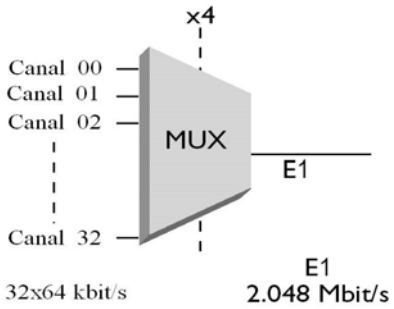


Figura 1.3. Multiplexado de primer orden.

Los timeslots son transportados del multiplexor inicial hasta la central telefónica a través de un conjunto de equipos conocido como red de acceso. La red de acceso y el transporte inter-centrales de la PSTN utilizan tecnología de transmisión óptica asíncrona (SONET y SDH), aunque en algunas partes se usa todavía la antigua tecnología PDH.

La red telefónica todavía utiliza tecnología analógica únicamente para el enlace de última milla con el usuario (la digitalización toma lugar en los multiplexores), pero está siendo desplazada poco a poco por servicios tales como DSL e ISDN. [4]

1.1.4 Señalización

Después de un breve recorrido por temas relacionados a la transmisión digital y la telefonía, es necesaria una introducción del tema en cuestión del presente trabajo, esto se refiere a la señalización como uno de los elementos que integran la PSTN y el funcionamiento de los equipos PBX (Centrales empresariales). La señalización se define como el intercambio de información entre los componentes de una llamada; dicha información es requerida para proveer y mantener dicha llamada. Como ejemplo a ello es posible mencionar el marcado de dígitos, tono de marcados, línea ocupada, etc.

Siendo usuarios de la PSTN, esta información es intercambiada con los elementos de la red en todo momento.

Aunque la señalización ha ido variando también con el pasar del tiempo, hoy en día predominan 2 tipos:

a) *Señalización por canal común (CCS):* Forma parte de la llamada; en términos generales, señalización fuera de banda. En esta estructura la señalización viaja por un canal independiente al de voz y datos; el canal de señalización es común para varios enlaces, lo que facilita acceder a esta información cuando es solicitada. El sistema CCS7 pertenece a este grupo, y es el que predomina en la actualidad.

b) *Señalización por canal asociado (CAS):* La señalización trabaja por el mismo canal de voz / datos, es decir no existe un circuito aparte que realiza la señalización. A pesar de estar cayendo en obsolescencia todavía existen sistemas telefónicos que utilizan esta forma de señalización, en el país CAS predomina sobretodo en los PBX, por lo cual su importancia aún es vigente, y será ahora el punto de atención.

La señalización CAS existe en varias formas, las más frecuentes son R-2 analógica y R-2 digital. La señalización R2 digital forma parte de los contenidos esenciales de este documento, por lo que será un tema ampliado en la sección 1.3.

1.2 LA INTERFAZ E1 (2,048 KBIT/S)

La interfaz E1 es la base de los sistemas de transmisión digital de datos utilizados en Europa y buena parte de América Latina, incluido El Salvador. Tal como se mencionó anteriormente, la interfaz E1 forma parte de la estructura de la PSTN. Posee una capacidad de 30 canales de 64 kbit/s para voz y datos, más un canal de servicio y otro de señalización, que se multiplexan para formar un flujo de datos de 2048 kbit/s.

En esta sección se presentan las características más importantes de la interfaz E1; estas características constituyen los aspectos eléctricos, estructura de trama, multitrama de señalización CAS, monitoreo de errores usando VRC-4 y sincronismo. Esta información constituye la base del diseño del circuito de interfaz del sistema de monitoreo.

1.2.1 Características Eléctricas de la Interfaz E1

La recomendación G.703 de la UIT-T (CCITT) [7] establece las características eléctricas que deben presentar las interfaces E1. Estas características deben cumplirse para asegurar el buen funcionamiento de los equipos a través de toda la red.

1.2.1.1 Características generales

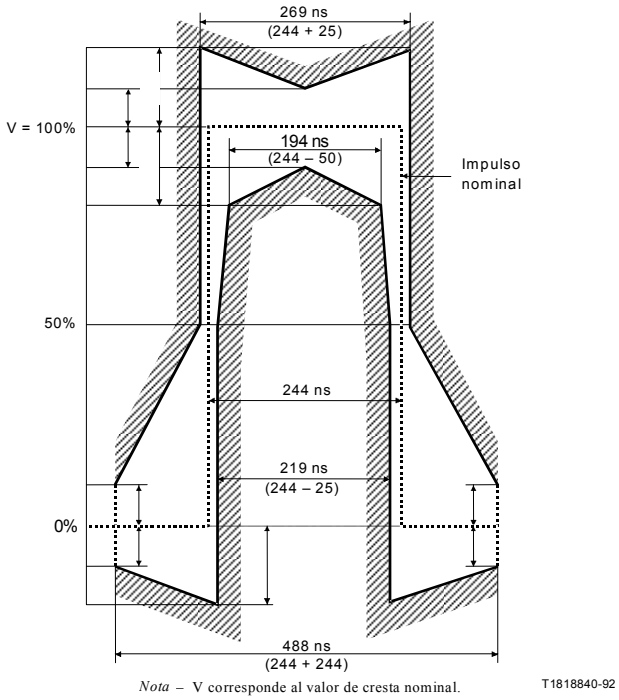
Velocidad binaria: 2048 kbit/s ± 50ppm
Código de línea: HDB3 (Bipolar de alta densidad de orden 3).

1.2.1.2 Especificaciones en los accesos de salida

Las interfaces de salida deben cumplir con las características presentadas en el cuadro 1.2.

Forma del impulso (forma nominal rectangular)	Todas las marcas de una señal válida deberán ajustarse a la plantilla (figura 1.4), independientemente del signo. El valor V corresponde al valor nominal de cresta	
Par(es) en cada sentido de transmisión	Un par coaxial	Un par simétrico
Impedancia de carga de prueba	75 ohmios, resistiva	120 ohmios, resistiva
Tensión nominal de cresta de una marca (impulso)	2.37 V	3 V
Tensión de cresta de un espacio (ausencia de impulso)	0 ± 0.237 V	0 ± 0.3 V
Anchura nominal del impulso	244 ns	
Relación entre la amplitud de los impulsos positivos y la de los negativos en el punto medio del intervalo del impulse	De 0.95 a 1.05	
Relación entre la anchura de los impulsos positivos y la de los negativos en los puntos de semiamplitud nominal	De 0.95 a 1.05	
Máxima fluctuación de fase cresta a cresta en un acceso de salida	Véase el cuadro 1.3.	

Cuadro 1.2. Características de salida de la interfaz a 2048kbit/s.



Nota – V corresponde al valor de cresta nominal.

Figura 1.4. Plantilla para el impulso en el caso de una interfaz a 2048 kbit/s.

Los límites presentados en el cuadro 1.3 representan los niveles máximos admisibles de fluctuación de fase (jitter) en interfaces a 2048 kbit/s. El montaje para la medición de la fluctuación de fase a la salida de una interfaz digital se ilustra en la figura 1.5. La respuesta en frecuencia de los filtros asociados a los aparatos de medida debe tener un régimen de decremento de 20 dB/década.

Límite de red		Anchura de banda del filtro de medición		
B_1 Intervalo unitario cresta a cresta	B_2 Intervalo unitario cresta a cresta	Filtro paso banda con una frecuencia de corte inferior f_1 o f_3 y una frecuencia de corte superior f_4		
		f_1	f_3	f_4
1.5	0.2	20 Hz	18 kHz	100 kHz

Cuadro 1.3. Fluctuación de fase máxima admisible en una interfaz a 2048 kHz

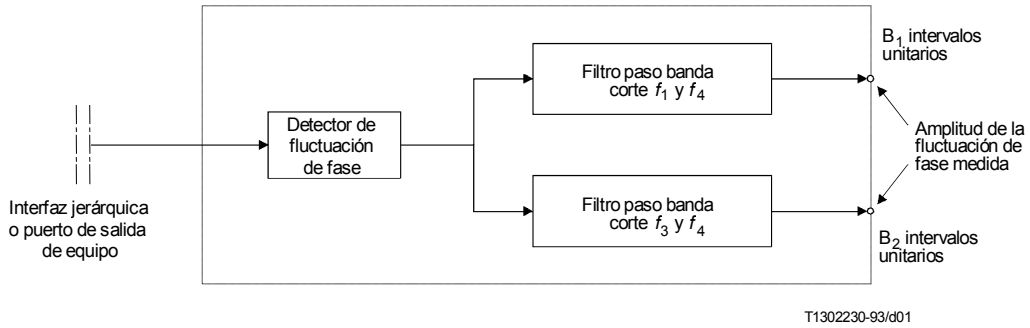


Figura 1.5. Disposición para medir la fluctuación de fase procedente de un acceso de salida de equipo.

1.2.1.3 Especificaciones en los accesos de entrada

La señal de entrada debe corresponder con las características mencionadas anteriormente, con las modificaciones que introduzcan los pares de interconexión. La atenuación de tales pares debe seguir la ley $\sqrt{f}$ y la atenuación a frecuencia de 1024 kHz debe estar entre 0 y 6dB.

La tolerancia de fluctuación de fase en los puertos de entrada se define en función de la amplitud y la frecuencia de una fluctuación de fase sinusoidal que, al modular una señal de prueba, no causa una degradación apreciable del funcionamiento del equipo. Todos los puertos de entrada deben estar en condiciones de tolerar una señal digital cuyas características eléctricas satisfacen los requerimientos y requisitos mencionados en la sección 1.2, pero modulada por una fluctuación lenta de fase y una fluctuación de fase sinusoidales que tienen una relación amplitud-frecuencia definida en la figura 1.6. Para efectos de prueba, el contenido binario equivalente de la señal modulada por la fluctuación

de fase debe ser una secuencia binaria seudo aleatoria. Los límites y valores apropiados se definen en el cuadro 1.4.

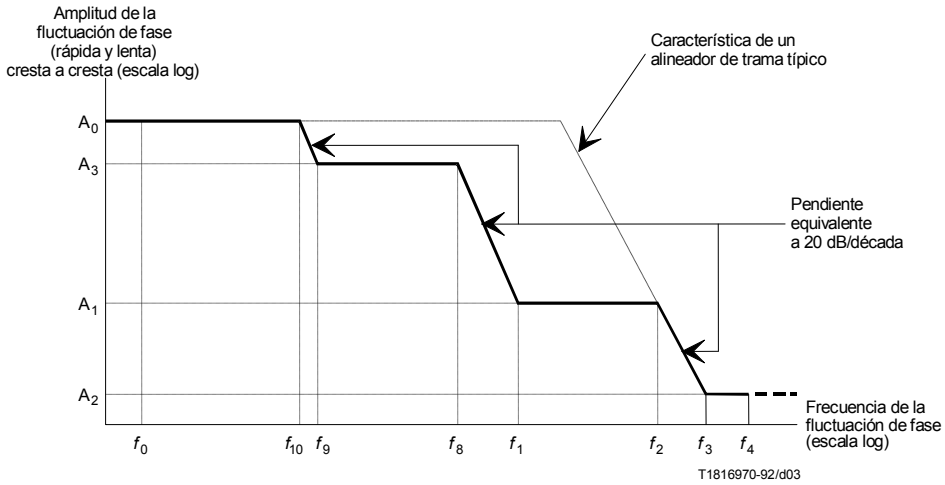


Figura 1.6. Límite inferior de fluctuación de fase y fluctuación lenta de fase máximas admisibles de entrada.

Intervalo unitario cresta a cresta (1 IU = 488ns)	A_0	36.9
	A_1	18
	A_2	1.5
	A_3	0.2
Frecuencia	f_0	1.2×10^{-5} Hz
	f_{10}	4.88×10^{-3} Hz
	f_9	0.01 Hz
	f_8	1.667 Hz
	f_1	20 Hz
	f_2	2.4 kHz
	f_3	18 kHz
	f_4	100 kHz
Señal de prueba seudo aleatoria	$2^{15} - 1$	

Cuadro 1.4. Valores de los parámetros a la tolerancia de fluctuación de fase y fluctuación lenta de fase de entrada para la interfaz a 2048 kbit/s.

1.2.1.4 Puesta a tierra del conductor exterior o del blindaje

En la salida, y de considerarse necesario en la entrada, debe conectarse a tierra el conductor exterior del par coaxial o el blindaje del par simétrico.

1.2.2 Estructura de la trama E1

1.2.2.1 Estructura de trama básica

La trama básica de la interfaz a 2,048 kbit/s, definida en la Rec. UIT-T G.704, [8] tiene una longitud 256 bits y una duración de 125µs, lo que hace un total de 8000 tramas por segundo. Esta se divide en 32 octetos ó "timeslots" numerados del 0 a 31. Cada timeslot constituye un canal a 64 kbit/s. Los bits de la trama se numeran del 1 al 256, siendo el primero el bit más significativo del timeslot 0, y los de cada timeslot individual del 1 (MSB) al 8 (LSB). Es importante tener en cuenta esta numeración para evitar errores y malentendidos. El orden de transmisión se corresponde con la numeración, es decir, se envía primero el bit 1 de la trama (bit 1, MSB, timeslot 0), luego el bit 2 y así sucesivamente, hasta llegar al bit 256 (bit 8, LSB, timeslot 31).

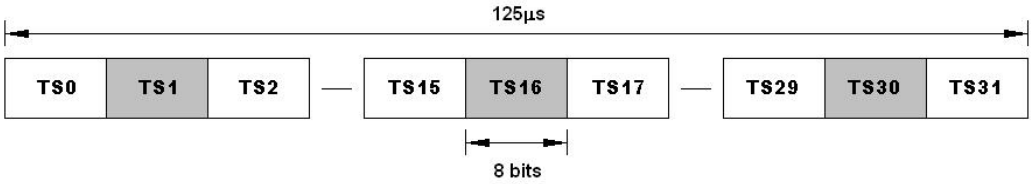


Figura 1.7. Trama de la interfaz a 2048 kbit/s

El timeslot 0 está reservado para llevar información de alineación de trama, mantenimiento, detección de errores y alarmas. Dependiendo de la información que contiene el timeslot 0, a las tramas se les denomina pares o impares, las cuales son transmitidas en forma alternada (ver Figura 1.8).

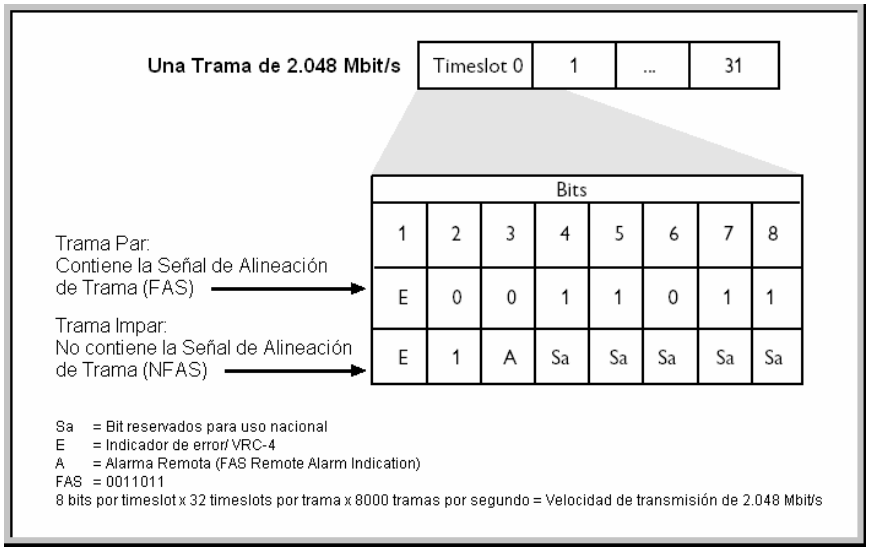


Figura 1.8. Asignación de los bits de la trama numerados del 1 al 8

Las tramas pares contienen la señal de alineación de trama (Frame Alignment Signal, FAS) en los bits del 2 al 8, que consiste en la secuencia 0011011, y que se utiliza para sincronización, es decir, para que los equipos terminales sean capaces de distinguir el inicio de la trama y, por tanto, los timeslots individuales.

Las tramas impares no contienen la señal de alineación de trama, por lo que se le llama NFAS. El bit 2 de esta trama se pone a 1 para evitar simulaciones de la FAS. El bit 3 (bit A) se utiliza como indicación de una alarma distante, en condiciones normales puesto a 0 y en condición de alarma puesto a 1. Los bits 4 al 8, llamados $S_{a4} - S_{a8}$, son bits adicionales de reserva que pueden ser utilizados en aplicaciones punto a punto, como enlace de datos basado en mensaje para operaciones de mantenimiento y monitoreo de la calidad de servicio, o para uso nacional. Si los bits $S_{a4} - S_{a8}$ no se utilizan, deben ponerse a 1.

En ambos casos, el bit 1 de la trama se denomina S_i y esta reservado para uso internacional. En algunos casos es usado para llevar la información de verificación de redundancia cíclica VRC-4, de la cual se hablará más adelante. Si no se utiliza, este bit debe ponerse a 1.

El timeslot 16 es utilizado normalmente para señalización, ya sea por canal común (CCS) o por canal asociado (CAS)

En los sistemas PCM-31, los timeslots 1 al 31 constituyen los canales 1 al 31. En cambio, en los sistemas PCM-30, los timeslots 1 al 15 constituyen los canales 1 al 15 y los timeslots 17 al 31 constituyen los canales 16 al 30, y el timeslot 16 es utilizado para llevar la señalización por canal asociado.

La interfaz a 2,048Kbit/s también puede transportar canales de n x 64 kbit/s (Rec. UIT-T G.704 sección 5.2).

1.2.2.2 Estructura de Multitrama de Señalización

El concepto de multitrama esta relacionado a la señalización por canal asociado de los sistemas PCM-30, el cual permite utilizar un solo timeslot para llevar la información de señalización de todos los canales de la interfaz.

Una multitrama (Rec. UIT-T G.704) está formada por 16 tramas básicas consecutivas, numeradas del 0 al 15. La información de señalización esta incluida en el timeslot 16 de cada una de las tramas que forman la multitrama (Ver figura 1.9).

La señal de alineación de multitrama (MultiFrame Alignment Signal, MFAS) es la secuencia 0000 que utiliza los bits 1 al 4 del timeslot 16 de la trama 0. Los bits 5 al 8 se denominan NMFAS y se les asigna las letras XYXX, respectivamente. El bit Y (bit 6) se utiliza para indicación de alarma de multitrama distante, teniendo un valor 0 en estado normal y 1 en condición de alarma. Los bits X son bits de reserva, que si no son utilizados deben ponerse a 1.

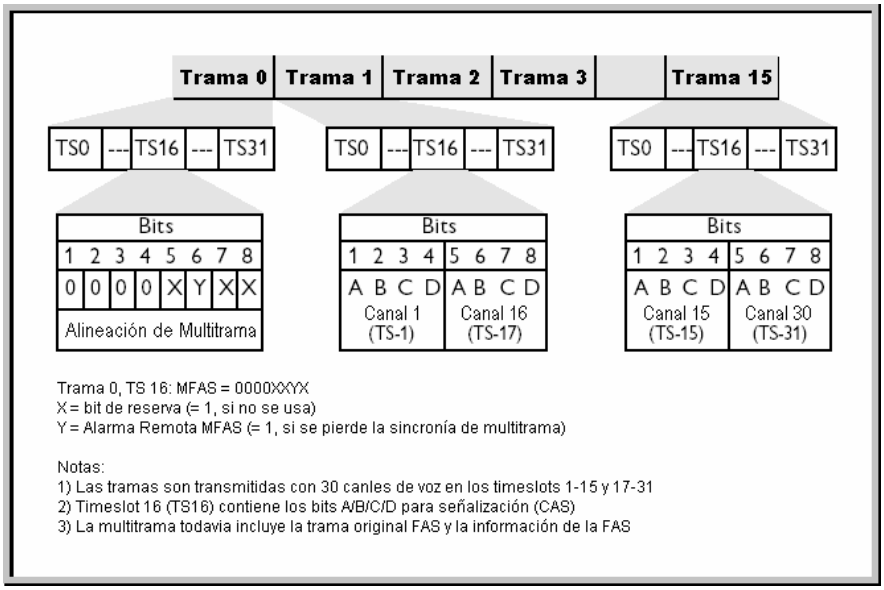


Figura 1.9. Estructura de Multitrama de Señalización CAS

Cada uno de los treinta canales de la trama tiene asociados cuatro bits de señalización, llamados A, B, C y D. Cada timeslot 16 de las tramas 1 a la 15 posee la información de dos canales. La trama n lleva la señalización de los canales n y n + 15. Los bits ABCD asociados al canal n son los bits 1 al 4, respectivamente, del timeslot 16 y los bits ABCD asociados al canal n + 15 son los bits 5 al 8.

Si alguno de los bits B, C, o D no se utiliza, deberán tener los siguientes valores: B = 1, C = 0 y D = 1.

1.2.3 Verificación de Redundancia Cíclica (VRC-4)

La Verificación de Redundancia Cíclica (Cyclic Redundancy Check, CRC) es un proceso que provee protección adicional contra simulaciones de la señal de alineación de trama básica y mayor capacidad de monitoreo de errores durante la operación de los equipos, a través del uso del bit 1 de la trama básica.

El proceso utiliza una multitrama formada por 16 tramas básicas, divididas en dos sub-multitramas de 8 tramas cada una, denominadas SMT-I y SMT-II. Esta multitrama no esta relacionada a la multitrama de señalización. En la figura 1.10 se muestra la asignación de bits de la multitrama VRC-4.

Multitrama	Sub-multitrama	Trama#	Timeslot 0 Bits bit1	bit2	bit3	bit4	bit5	bit6	bit7	bit8
	SMT - I	0	c1	0	0	1	1	0	1	1
		1	0	1	A	Sa4	Sa5	Sa6	Sa7	Sa8
		2	c2	0	0	1	1	0	1	1
		3	0	1	A	Sa4	Sa5	Sa6	Sa7	Sa8
		4	c3	0	0	1	1	0	1	1
		5	1	1	A	Sa4	Sa5	Sa6	Sa7	Sa8
		6	c4	0	0	1	1	0	1	1
		7	0	1	A	Sa4	Sa5	Sa6	Sa7	Sa8
	SMT - II	8	c1	0	0	1	1	0	1	1
		9	1	1	A	Sa4	Sa5	Sa6	Sa7	Sa8
		10	c2	0	0	1	1	0	1	1
		11	1	1	A	Sa4	Sa5	Sa6	Sa7	Sa8
		12	c3	0	0	1	1	0	1	1
		13	E	1	A	Sa4	Sa5	Sa6	Sa7	Sa8
		14	c4	0	0	1	1	0	1	1
		15	E	1	A	Sa4	Sa5	Sa6	Sa7	Sa8

SMT-I: Sub-multitrama I
Sa: Bit reservado para uso nacional
A: Alarma Remota (FAS Remote Alarm Indication)
Señal de Alineación de Trama: 0011011
Señal de Alineación de Multitrama VRC-4: 001011
La Mutitrama VRC-4 no esta alineada con la Multitrama de Señalización

SMT-II: Sub-multitrama II
E: Indicador de error
c1, c2, c3, c4: Bits VRC

Figura 1.10. Estructura de Multitrama VRC-4

Cada sub-multitrama forma un bloque de 2048 bits. Este bloque se multiplica por x^4 y luego se divide entre el polinomio generador $x^4 + x + 1$. El residuo de esta operación de división corresponde a una palabra VRC-4, de 4 bits.

El lado de transmisión genera una palabra VRC-4 de cada sub-multitrama y la envía en la siguiente sub-multitrama a través de los bits C_1, C_2, C_3 y C_4. Los bits 1 de las tramas pares, es decir, las que poseen la FAS, se utilizan para transportar estos bits. Los bits 1 de las tramas impares se usan para llevar la secuencia 001011 de alineación de multitrama VRC-4 y los bits E.

En el lado de recepción, cada vez que se recibe una sub-multitrama se realiza el mismo cálculo, poniendo a cero los bits VRC-4. El resultado se compara con el valor transmitido en la siguiente sub-multitrama y si son exactamente iguales entonces se supone que esta no posee errores. La existencia de una diferencia entre las palabras indica que la trama posee por lo menos un bit erróneo. Este método no indica el número de errores dentro del bloque, además, existe la posibilidad de que cierta combinación de errores genere una palabra VRC-4 idéntica a la correcta, pero la probabilidad de que esto ocurra es baja (la exactitud del método es de 93.75%).

Los bits E sirven para indicar errores en las sub-multitramas. Cada bit E cambia de estado de 1 a 0 por cada sub-multitrama con errores. El retraso entre la detección de una multitrama con errores y la indicación en los bits E debe ser menor a un segundo. Los bits E deben ser tomados en cuenta incluso si la trama que los contiene posee errores, ya que la probabilidad de que estos tengan errores es baja. En caso de no ser utilizados, deben ponerse en 1.

Los sistemas PCM que utilizan la verificación de redundancia cíclica VRC-4 se conocen como PCM-31C y PCM-30C.

1.2.4 Procedimientos de alineación de trama y multitrama VRC-4

La recomendación G.706 de la UIT-T (CCITT) [9] establece los procedimientos que deben seguirse para lograr y recuperar la sincronía tanto de trama básica como de multitrama VRC-4 en una interfaz de 2048 kbit/s. Estos procedimientos son especialmente importantes ya que a través de estos podemos detectar errores causados por simulaciones de la señal de alineación de trama o multitrama VRC, y otros causados por señales espurias o de interferencia. Solo después que los procedimientos de alineación han sido realizados, puede considerarse que los equipos están funcionando en sincronía. Si no existe sincronía de trama, los equipos no consideran válidos los datos recibidos.

1.2.4.1 Procedimientos de alineación de trama básica

Los procedimientos de alineación de trama básica son los primeros que deben realizarse. Si las condiciones para considerar que existe sincronía de trama básica no se cumplen, las multitramas de señalización y VRC-4 no puede considerarse en sincronía.

1.2.4.1.1 Pérdida de alineación de trama

Se considera que se ha perdido la alineación de trama básica cuando se reciban 3 FAS incorrectas consecutivas. Además, se considera que hay pérdida de la alineación si el bit 2 de la NFAS se recibe con error en tres ocasiones consecutivas. Aparte de estas dos condiciones, la alineación de trama básica puede considerarse perdida a partir de procedimientos relacionados a la VRC-4 que se mencionarán posteriormente.

1.2.4.1.2 Recuperación de alineación de trama

Para recuperar la alineación de trama básica debe realizarse el siguiente procedimiento:

a) Detectar una señal FAS válida.
b) Verificar que el bit 2 de la siguiente trama esta en 1.
c) Encontrar la FAS en la siguiente trama.

Si alguno de los pasos no se cumple, debe iniciarse una nueva búsqueda de la FAS en la trama consecutiva al error. Este procedimiento se realiza también al momento de iniciar la operación de los equipos.

1.2.4.2 Procedimientos de alineación de multitrama VRC-4

Dado que la alineación de trama básica ha sido lograda, es considerado que se ha logrado la alineación de multitrama VRC-4 cuando por lo menos dos señales de alineación de multitrama válidas puedan localizarse en un plazo de 8ms, las cuales se reciben cada 2ms.

Si esta no se logra en el plazo de 8ms se supondrá que la alineación de trama básica era una simulación y se iniciará de nuevo el proceso de alineación de trama básica.

1.2.4.3 Monitoreo de los bits VRC para detectar una falsa alineación de trama básica.

El monitoreo de errores que proveen los bits VRC-4 pueden utilizarse también en la detección de falsa alineación de trama. La recomendación G.706 indica que debe detectarse una falsa alineación de trama en un plazo de un segundo y con una probabilidad mayor a 0.99.

Para una tasa de errores aleatorios de 10^{-3}, la probabilidad de que se inicie de manera indebida una búsqueda de alineación de trama como consecuencia de un número excesivo de bloques VRC-4 con errores debe ser inferior a 10^{-4} durante un segundo. La figura 1.11 muestra la forma en que ha de pasarse de la búsqueda de alineación de trama básica a la supervisión de errores mediante la VRC.

Para cumplir los límites mencionados anteriormente, se toma un valor de 915 bloques VRC erróneos de entre 1000 como umbral, lo que indica que un número superior a este será indicación de una falsa alineación de trama básica.

La información de monitoreo de VRC-4 debe presentarse de dos formas:

a) Información directa: Debe indicarse cuando exista un bloque erróneo.
b) Información integrada: Se debe indicar el número de bloques erróneos encontrados en un segundo.

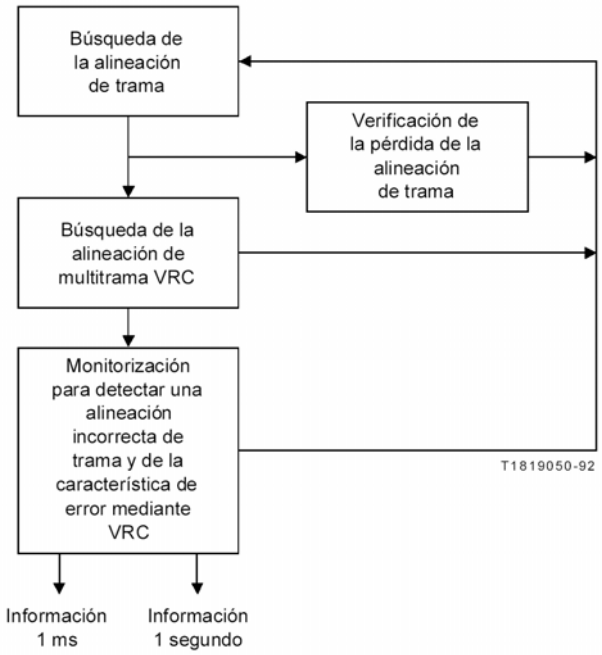


Figura 1.11. Procedimiento para pasar de la búsqueda de señal de alineación de trama básica al monitoreo de errores mediante la verificación por redundancia cíclica.

1.3 SEÑALIZACIÓN R-2 DIGITAL

Tal como se ha mencionado anteriormente, este trabajo esta orientado a la señalización R-2 digital, y en esta sección se entrara más en detalle.

La señalización R-2 pertenece al sistema de señalización por canal asociado (CAS), el cual fue desarrollado en los años 60, pero que aún es utilizado en Europa, América Latina, Australia y Asia [2]. Aunque en cada una de estas regiones existen variantes de esta técnica, la base común está regulada por la CCITT-R2 (UIT-T). Las recomendaciones que contienen las definiciones de la señalización R-2 están entre la Q.400 a la Q.490 [11].

1.3.1 Tipos de señalización R-2

Hay 2 elementos que componen la señalización R-2, la señalización de línea (señales de monitoreo y supervisión) y la señalización inter registro (señales para el control de inicio de llamada). A pesar de la normativa de estas formas de señalización, en algunos países existen variantes, sobre todo en la señalización inter registro.

La señalización de línea es definida según los siguientes grupos [2]:

1- *R-2 digital*: Definida sobre todo en la norma UIT. Q.421, típicamente usada en sistemas PCM.

2- *R-2 analógica*: Definida sobre todo en la norma UIT. Q.411, típicamente usada en sistemas de mensajería donde los bits de señalización se representan mediante tonos.

3- *R-2 de pulsos*: Definida en el suplemento 7 de la UIT, típicamente usada para sistemas con enlaces satelitales.

La señalización inter registros se divide en los siguientes tipos:

1- *R-2 forzada*: Se envían un par de tonos, los cuales se mantienen hasta que el receptor envía la señal de ACK, por medio de otro par de tonos.

2- *R-2 no forzada*: Similar a la anterior, excepto que los tonos se envían en forma de pulsos, sin mantenerse, pero siempre en espera de la respuesta hacia atrás. Esta es la forma más común de señalización Inter. Registro.

3- *R-2 semi-forzada*: En esta forma solo los tonos hacia delante son mantenidos hasta esperar la respuesta hacia atrás, esta respuesta viene como un pulso similar a la R-2 no forzada.

1.3.2 Señalización de línea

La señalización de línea utiliza los bits ABCD del timeslot 16 de la trama E1 para propósitos de supervisión del estado de la línea. La señalización CCITT R2 digital solo utiliza los bits A y B, mientras que los otros se dejan fijos (C = 0, D = 1). La recomendación Q.421 [12] denomina "canales de señalización" a cada uno de estos bits.

Existen dos grupos de señales:

a) Señales hacia adelante: son las que envía la parte que llama (equipo de salida). A los bits de señalización de este grupo se les denomina A_f y B_f.
b) Señales hacia atrás: son las que envía la parte que ha sido llamada (equipo de llegada). Los bits se denominan A_b y B_b.

En condiciones normales, los bits de señalización tienen el siguiente significado:

- A_f: Identifica la condición de funcionamiento del equipo de conmutación de salida y refleja la condición de la línea del abonado que llama (colgado o descolgado).
- B_f: Sirve para identificar la existencia de una avería en el sentido hacia delante del equipo de conmutación de llegada.
- A_b: Indica la condición de la línea del abonado llamado (gancho conmutador colgado o descolgado).
- B_b: Indica si el equipo de conmutación de llegada se halla en estado de reposo o de conmutación.

1.3.2.1 Código digital de señalización

El cuadro 1.5 muestra el código de señalización R2 digital y los estados del circuito que representa.

Estado del circuito	Código de señalización			
	Hacia adelante		Hacia atrás	
	a_f	b_f	A_b	b_b
Reposo/liberado	1	0	1	0
Toma	0	0	1	0
Acuse de recibo de toma	0	0	1	1
Respuesta	0	0	0	1
Abonado llamado cuelga	0	0	1	1
Señal de fin	1	0	0	1
			o	
	1	0	1	1
Bloqueo			1	1

Cuadro 1.5. Código de señalización R2 de línea.

1.3.2.2 Identificación de una transición del código de señalización

1.3.2.2.1 Transición en un canal (bit) de señalización

La recomendación Q.422 [13] dice que para reconocer una transición de estado de un bit de señalización, ya sea de 0 a 1 o viceversa, esta debe de poseer un tiempo de identificación de 20 ± 10 ms. Esto asegura protección contra señales espurias.

El tiempo de identificación se define como la duración que deben poseer las señales que representan el estado 0 o el estado 1 en la salida del equipo terminal de un canal de señalización a fin de que sean reconocidas por el equipo de una central.

1.3.2.2.2 Cambio del código de señalización

Un cambio del código de señalización se define de acuerdo a cualquiera de los dos siguientes casos:

a) Identificación de una transición detectada en una canal (bit) de señalización sin ninguna transición en el otro bit durante el tiempo de identificación.
b) Identificación de una transición detectada en el segundo bit de señalización durante el período de identificación que ya se esta aplicando en el primer bit. Si esto ocurre, solo se reconoce el cambio transcurridos ambos tiempos de identificación.

1.3.2.3 Estados y procedimientos normales

Estos son los procedimientos de señalización de línea a seguir en el establecimiento de llamadas.

1.3.2.3.1 Estado de reposo

En este estado, el extremo de salida envía $A_f = 1$, $B_f = 0$. En el extremo de llegada, esta señal produce el envío de $A_b = 1$, $B_b = 0$ en el sentido hacia atrás, siempre que el equipo de llegada este en reposo.

1.3.2.3.2 Procedimiento de toma

a) Toma: Una toma solo tiene lugar si se identifica $A_b = 1$, $B_b = 0$. El extremo de salida cambia $A_f = 0$, $B_f = 0$, este estado debe mantenerse hasta que se identifica la señal de acuse de recibo de toma. El equipo de salida solo podrá emitir la señal de fin después de identificar la señal de acuse de recibo de toma.

b) Acuse de recibo de toma: Una vez que se identifica la señal de toma, se envía la señal hacia atrás $A_b = 1$, $B_b = 1$ como acuse de recibo.

1.3.2.3.3 Respuesta

Cuando se contesta del lado de la persona a quien se esta llamando, se provoca que el equipo de conmutación de llegada envíe $A_b = 0$, $B_b = 1$. Después de esto el enlace debe ponerse en estado de repuesta.

1.3.2.3.4 Abonado llamado cuelga

Cuando se cuelga del lado de la persona a quien se esta llamando, el equipo de conmutación de llegada envía $A_b = 1$ $B_b = 1$. El estado de gancho conmutador colgado debe establecerse en el enlace.

1.3.2.3.5 Procedimiento de fin

La liberación del equipo de conmutación de salida produce normalmente el envío de $A_f = 1$, $B_f = 0$. El equipo de conmutación de salida no pasará al estado de reposo hasta el reconocimiento del código $A_b = 1$, $B_b = 0$.

1.3.2.3.6 Procedimiento de liberación

La identificación de la señal de fin en el equipo de llegada tiene por efecto la liberación del enlace, incluso si se ha producido la respuesta o liberación por parte de la persona a quien se llama. Una vez que se libera el equipo de conmutación de llegada, se establece el código $A_b = 1$, $B_b = 0$. Esto hace que el equipo de salida este disponible para otra comunicación.

1.3.2.3.7 Procedimiento de bloqueo y desbloqueo

El bloqueo de un circuito en reposo para nuevas llamadas que puedan introducirse en el extremo de salida debe tener lugar tan pronto se identifica $A_b = 1$, $B_b = 1$. Al identificar Ab = 1, Bb = 0 se vuelve al estado de reposo.

1.3.2.4 Disposiciones correspondientes a distintas condiciones de señalización

Los cuadros 1.6 y 1.7 muestran los procedimientos ha realizarse en las distintas condiciones de funcionamiento, ya sean normales o en caso de averías, en ambos sentidos de la comunicación.

Cuadro 1.6. Extremo de salida

Estado normal en el extremo de salida	Código enviado	Código recibido			
		$a_b = 0, b_b = 0$	$a_b = 0, b_b = 1$	$a_b = 1, b_b = 0$	$a_b = 1, bb = 1$
Reposo/liberación	$A_f = 1$, bf = 0	Anormal, véase nota 1	Anormal, véase nota 1	Reposo	Bloqueado
Toma	$A_f = 0, b_f = 0$	Anormal, véase nota 2	Anormal, véase nota 2	Toma, véase nota 2	Acuse de recibo de toma
Acuse de recibo de toma	$A_f = 0, b_f = 0$	Anormal, véase nota 3	Respondido	Anormal, véase nota 3	Acuse de recibo de toma
Respondido	$A_f = 0, b_f = 0$	Anormal, véase nota 4	Respondido	Anormal, véase nota 4	Colgar
Colgar	$A_f = 0, b_f = 0$	Anormal, véase nota 4	Respondido	Anormal, véase nota 4	Colgar
Fin	$A_f = 1, b_f = 0$	Anormal, véase nota 1	Fin	Liberado = reposo	Fin
Bloqueado	$A_f = 1, b_f = 0$	Anormal, véase nota 1	Anormal, véase nota 1	Reposo	Bloqueado

Nota 1 – En estas condiciones anormales, el extremo de salida debe evitar una nueva toma del circuito. También debe enviarse una señal de alarma tardía.

Nota 2 – La ausencia de identificación de la señal de acuse de recibo de toma 100 ms-200 ms después del envío de la señal de toma en un enlace terrenal o 1-2 segundos después del envío de la señal de toma en un enlace por satélite provoca una alarma y el envío hacia atrás de la indicación de congestión o la repetición del intento de establecimiento de la llamada. El extremo de salida debe evitar una nueva toma del circuito. Cuando se identifica la señal de acuse de recibo de toma después de transcurrido el periodo de temporización, debe enviarse la señal de fin.

Nota 3 – La recepción $b_b = 0$ por el equipo de conmutación de salida durante 1-2 segundos después de la identificación de la señal de acuse de recibo de toma y antes de la identificación de la señal de respuesta, produce una alarma y el envío hacia atrás de la indicación de congestión o la repetición del intento de establecimiento de la llamada. El extremo de salida debe evitar nuevas tomas del circuito. Cuando b_b vuelve a 1 después de transcurrido el periodo de temporización de 1-2 segundos, debe enviarse la señal de fin.

Nota 4 – En el caso de identificación de $b_b = 0$ mientras el circuito se halla en el estado de respuesta o de colgar, no es necesaria una acción inmediata. Al recibir la señal de liberación del enlace precedente, no se enviará la señal de fin ($a_f = 1, b_f = 0$), hasta que b_b queda restablecido a 1. También debe enviarse una señal de alarma tardía.

Cuadro 1.7. Extremo de llegada

Estado normal en el extremo de llegada	Código enviado	Código recibido			
		$a_f = 0, b_f = 0$	$a_f = 0, b_f = 1$	$a_f = 0, b_f = 1$	$a_f = 1, b_f = 0$
Reposo/liberación	$a_b = 1, b_b = 0$	Toma	Avería véase nota 1	Reposo	Avería véase nota 1
Acuse de recibo de toma	$a_b = 1, b_b = 1$	Acuse de recibo de toma	Avería véase nota 2	Fin	Avería véase nota 2
Respuesta	$a_b = 0, b_b = 1$	Respuesta	Avería véase nota 3	Fin	Avería véase nota 3
Colgar	$a_b = 1, b_b = 1$	Colgar	Avería véase nota 4	Fin	Avería véase nota 4
Fin	$a_b = 0, b_b = 1$ ó $a_b = 1, b_b = 1$	Toma anormal, véase nota 7	Avería véase nota 5	Fin véase nota 7	Avería véase nota 7
Bloqueado	$a_b = 1, b_b = 1$	Toma anormal véase nota 5	Avería véase nota 6	Bloqueado	Avería véase nota 6

Nota 1 – Cuando el estado de reposo/liberación b_f pasa a 1, b_b deberá pasar a 1.

Nota 2 – En estos casos entra en funcionamiento un dispositivo de temporización que, después de cierto intervalo, libera la conexión más allá del circuito defectuoso: este dispositivo de temporización puede ser el especificado en la Recomendación Q.118, § 4.3.3. Si se identifica la señal de respuesta durante el periodo de temporización, se detiene el temporizador pero la señal de respuesta no se envía al enlace precedente hasta la identificación de $a_f = 0$, $b_f = 0$. Si se identifica la señal de colgar mientras persiste la avería, la conexión situada más allá del circuito defectuoso debe liberarse inmediatamente. Además, cuando el registro de llegada no ha empezado a transmitir la última señal hacia atrás, puede utilizarse el procedimiento de liberación rápida descrito en la nota 5.

Nota 3 – En estos casos no se adopta ningún procedimiento hasta la identificación de la señal de colgar, y en ese momento se libera inmediatamente la conexión situada más allá del circuito defectuoso.

Nota 4 – En estas condiciones debe liberarse inmediatamente el enlace subsiguiente.

Nota 5 – En este caso no se necesita ninguna medida inmediata. Sin embargo, la liberación rápida del circuito debe producirse si el extremo de llegada simula una respuesta enviando $a_b = 0$, $b_b = 1$.

Nota 6 – En estas condiciones no se necesita ninguna medida.

Nota 7 – Después de identificar la señal de fin y hasta que se envía el código $a_b = 1$, $b_b = 0$, se ignorarán todas las transiciones en el sentido hacia adelante.

1.3.3 Señalización Inter Registro

La señalización inter registro se realiza mediante el envío de señales de multifrecuencia a través del canal de voz / datos en ambas direcciones. Se les denomina inter registro debido a que están asociadas con los registradores, que son equipos utilizados para control de los procesos de conmutación. Para generar cada señal multifrecuencia, se realiza una selección de 2 tonos (frecuencias) de entre un conjunto de 6. La información que se envía mediante esta señalización incluye los dígitos marcados, categoría de usuario, control de índice de registro, etc.

Las señales enviadas por los puntos de origen son las señales hacia delante, los que se generan en el punto terminal son las señales hacia atrás. Cada uno de estos grupos de señales posee dos tablas de asignación con las cuales se codifica la señal transmitida, es decir, cada multitono posee dos significados distintos en cada sentido.

Los valores de frecuencia válidos y sus combinaciones se muestran en el cuadro 1.8. A cada señal multitono se le asigna un número del 1 al 15 que sirve para identificarla dentro de las tablas de asignación.

Combinaciones		Frecuencias (Hz)						
N.°	Valor numérico = $x + y$	Hacia delante (señales de los grupos I y II)	1380	1500	1620	1740	1860	1980
		Hacia atrás (señales de los grupos A y B)	1140	1020	900	780	660	540
		Índice (x)	$f0$	$f1$	$f2$	$f3$	$f4$	$f5$
		Peso (y)	0	1	2	4	7	11
1	0 + 1		x	y				
2	0 + 2		x		y			
3	1 + 2			x	y			
4	0 + 4		x			y		
5	1 + 4			x		y		
6	2 + 4				x	y		
7	0 + 7		x				y	
8	1 + 7			x			y	
9	2 + 7				x		y	
10	3 + 7					x	y	
11	0 + 11		x					y
12	1 + 11			x				y
13	2 + 11				x			y
14	3 + 11					x		y
15	4 + 11						x	y

Cuadro 1.8. Combinaciones de multifrecuencia.

La recomendación UIT Q.441 define las tablas de asignación de las señales multifrecuencia en ambos sentidos de transmisión. Para las señales hacia delante se definen las tablas del grupo I y II (cuadros 1.9 y 1.10, respectivamente).

Combi-nación (a)	Designación de la señal (b)	Significado de la señal (c)	(d)	Observaciones (e)
1	I-1	Cifra de idioma: francés	Cifra 1	Col. (c) - Estas señales constituyen la primera señal transmitida por un enlace internacional cuando este circuito termina en el país de llegada de la llamada. Sin embargo, si un circuito termina en un centro de tránsito internacional, estas señales pueden transmitirse por este enlace después del indicador de distintivo de país y del distintivo de país. Véase también la Recomendación Q.107.
2	I-2	Cifra de idioma: inglés	Cifra 2	
3	I-3	Cifra de idioma: alemán	Cifra 3	
4	I-4	Cifra de idioma: ruso	Cifra 4	
5	I-5	Cifra de idioma: español	Cifra 5	
6	I-6	En reserva (cif. de idioma)	Cifra 6	
7	I-7	En reserva (cif. de idioma)	Cifra 7	
8	I-8	En reserva (cif. de idioma)	Cifra 8	
9	I-9	En reserva (cif. de discriminación)	Cifra 9	
10	I-10	Cifra de discriminación	Cifra 0	
11	I-11	Indicador de indicativo de país, semisupresor de eco de salida necesario	Acceso a operadora de llegada (código 11)	Col. (c) - Primera señal en un enlace internacional cuando éste termina en un centro de tránsito internacional. Col. (d) - Señal distinta de la primera en un enlace internacional.
12	I-12	Indicador de indicativo de país, supresor de eco innecesario	i) Acceso a operadora de tráfico diferido (código 12)	
13	I-13	Indicador de llamada de prueba (Llamada del aparato de pruebas automáticas)	ii) Petición no aceptada	
14	I-14	Indicador de indicativo de país, semisupresor de eco de salida insertado	i) Acceso al aparato de pruebas (código 13)	
15	I-15	Señal no utilizable	ii) Enlace por satélite no incluido	
			i) Semisupresor de eco de llegada necesario	
			ii) Enlace por satélite incluido	
			i) Fin de numeración (código 15)	
			ii) Fin de identificación	

Cuadro 1.9. Señales hacia delante del grupo I.

Combinación (a)	Designación de la señal (b)	Significado de la señal (c)	Observaciones (d)
1	II-1	Abonado sin prioridad	Estas señales se usan sólo en explotación nacional
2	II-2	Abonado con prioridad	
3	II-3	Equipo de mantenimiento	
4	II-4	Reservada	
5	II-5	Operadora	
6	II-6	Transmisión de datos	
7	II-7	Abonado (u operadora que no tiene la facilidad de intervención)	Estas señales se usan en explotación internacional
8	II-8	Transmisión de datos	
9	II-9	Abonado con prioridad	
10	II-10	Operadora que tiene la facilidad de intervención	
11	II-11	Reserva para el servicio nacional	
12	II-12		
13	II-13		
14	II-14		
15	II-15		

Cuadro 1.10. Señales hacia delante del grupo II.

El siguiente grupo de señales definidas en tablas son los llamados grupos A (cuadro 1.11) y B (cuadro 1.12), los cuales son los grupos de multitonos enviados hacia atrás en respuesta a las señales de los grupos I y/o II.

Combinación (a)	Designación de la señal (b)	Significado de la señal (c)
1	A-1	Envíese la cifra siguiente (n + 1)
2	A-2	Envíese la penúltima cifra (n + 1)
3	A-3	Dirección completa, paso a la recepción de las señales del grupo B
4	A-4	Congestión en la red nacional
5	A-5	Envíese la categoría del abonado que llama
6	A-6	Dirección completa con tasación, paso a la posición de conversación
7	A-7	Envíese la antepenúltima cifra (n - 2)
8	A-8	Envíese la cifra que precede a la antepenúltima (n - 3)
9	A-9	Reserva para uso nacional
10	A-10	
11	A-11	Envíese el indicador de indicativo de país
12	A-12	Envíese la cifra de idioma o la de discriminación
13	A-13	Envíese la naturaleza del circuito
14	A-14	Petición de información sobre el empleo de supresor de eco (¿Se precisa un semisupresor de eco de llegada?)
15	A-15	Congestión en una central internacional o a su salida

Cuadro 1.11. Señales hacia atrás del grupo A.

ggCombinación (a)	Designación de la señal (b)	Significado de la señal (c)
1	B-1	Reserva para uso nacional
2	B-2	Envío de tono especial de información
3	B-3	Línea de abonado ocupada
4	B-4	Congestión (después del paso de las señales del grupo A a las señales
5	B-5	del grupo B)
6	B-6	Número no asignado
7	B-7	Línea de abonado libre, con tasación
8	B-8	Línea de abonado libre, sin tasación
9	B-9	Línea de abonado fuera de servicio
10	B-10	
11	B-11	
12	B-12	
13	B-13	Reserva para uso nacional
14	B-14	
15	B-15	

Cuadro 1.12. Señales hacia atrás del grupo B.

Una llamada se origina con el envío de la cifra correspondiente en el grupo I de las señales hacia delante, la repuesta hacia atrás se codifica por la cifra correspondiente del grupo A de señales hacia atrás. Esta señal podría requerir una respuesta hacia delante proveniente del grupo II, en cuyo caso la repuesta hacia atrás corresponde a una cifra del grupo B, y así sucesivamente hasta que la llamada sea establecida. Aunque en la realidad el proceso de establecimiento de una llamada es un poco complejo en lo referente a la señalización inter registro, básicamente se trata de una serie de comandos de envió y respuesta, ello a través de la codificación en multitonos.

1.3.4 Ejemplo de establecimiento de llamadas

La figura 1.12 muestra ejemplos del establecimiento de llamadas normales utilizando señalización R2 digital usada entre una central y un PBX para llamadas locales.

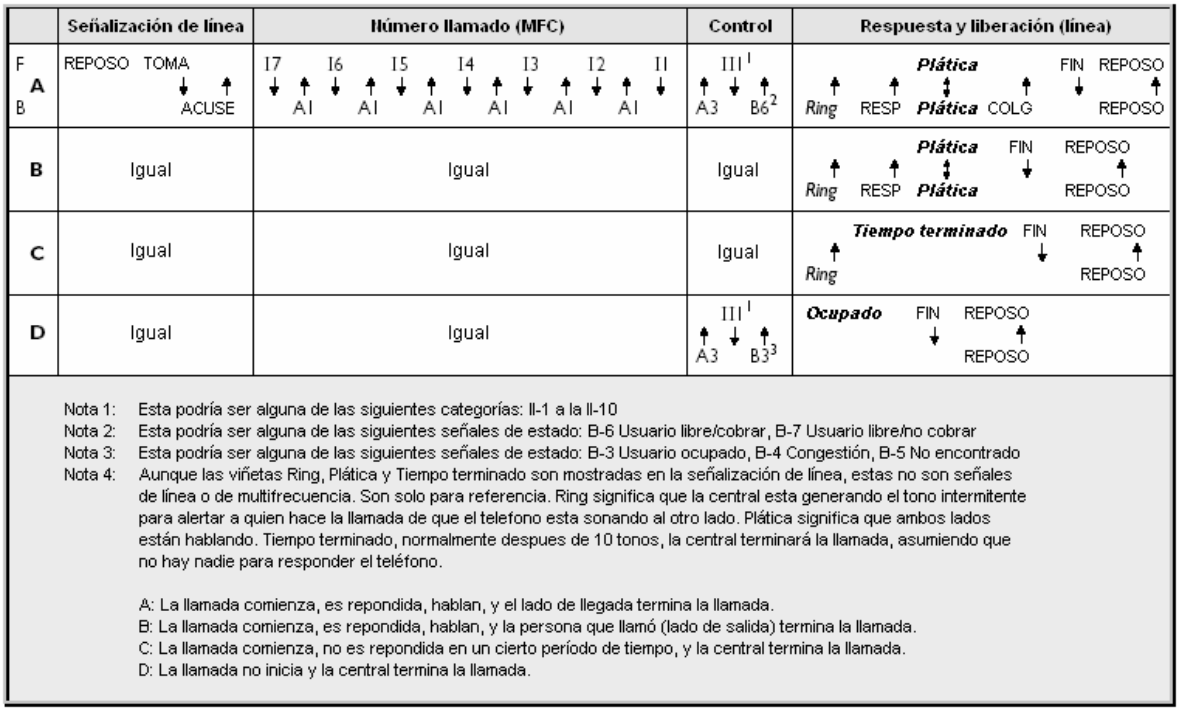


Figura 1.12. Ejemplo de establecimiento de llamadas locales.

1.4 REFERENCIAS BIBLIOGRÁFICAS

[1] Calderón Osorio, Daniel Alberto. "Diseño de instrumentos virtuales para medir señalización siete por canal común de la Unión Internacional de Telecomunicaciones (CCS7UIT) en tiempo real". Universidad de El Salvador. 2003.

[2] Cisco Systems Inc. "E1 R2 Signaling Theory". Tech notes. http://www.cisco.com/warp/customer/788/signalling/e1_r2_sig.html. 2000.

[3] IEC. "Fundamentals of Telecommunications". Web ProForum Tutorials. http://www.iec.org.

[4] Sunrise Telecom Inc. "Technology Series. Introduction to E1/2.048 Mbit/s". Publication Number TEC-GEN-001 Rev.B. 2001.

[5] Sunrise Telecom Inc. "Technology Series. Introduction to MFC-R2 Signaling". Publication Number TEC-GEN-00 2 Rev.B. 2001.

[6] Tibbs, John. "E1 Pocket Guide. The World of E1". Vo. 5. Wavetek Wandel Goltermann. .

[7] UIT-T. “Características físicas y eléctricas de los interfaces digitales jerárquicos”. Rec. G.703. 1991.

[8] UIT-T. “Estructuras de trama síncrona utilizadas en los niveles jerárquicos primario y secundario”. Rec. G.704. 1991.

[9] UIT-T. “Procedimientos de alineación de trama y de verificación por redundancia cíclica (VRC) relativos a las estructuras de trama básica definidas en la recomendación G.704”. Rec. G.706. 1991.

[10] UIT-T. “Control de la fluctuación de fase y de la fluctuación lenta de fase en las redes digitales basadas en la jerarquía de 2048 kbit/s”. Rec. G.823. 1993.

[11] UIT-T. “Especificaciones del sistema de señalización R2. Definición y función de las señales”. Rec. Q.400. 1993.

[12] UIT-T. “Especificaciones del sistema de señalización R2. Señalización de línea, versión digital. Código digital de señalización de línea”. Rec. Q.421. 1993.

[13] UIT-T. “Especificaciones del sistema de señalización R2. Señalización de línea, versión digital. Cláusulas relativas al equipo de señalización de línea de las centrales”. Rec. Q.422. 1993.

[14] UIT-T. “Especificaciones del sistema de señalización R2. Señalización entre registradores. Consideraciones generales”. Rec. Q.440. 1993.

[15] UIT-T. “Especificaciones del sistema de señalización R2. Señalización entre registradores. Código de señalización”. Rec. Q.441. 1993.

[16] UIT-T. “Especificaciones de los aparatos de medida. Aparato para efectuar la supervisión en servicio de las señales de 2048, 8448, 34368 139264 Kbit/s”. Rec. O.162 1992.

CAPÍTULO II

DISEÑO DEL CIRCUITO DE INTERFAZ

INTRODUCCIÓN

La siguiente etapa del trabajo consiste en el diseño de un circuito de interfaz que, cumpliendo con las recomendaciones UIT-T presentadas anteriormente, permita conectase a una interfaz E1 y trasladar los datos que esta transporta a una computadora personal, para posterior procesamiento.

El dispositivo debe tener capacidad de conectarse en forma transparente a una línea E1, es decir, que su conexión se realice directamente a una línea en la cual existen dos aparatos de comunicación conectados (transmisor-receptor), sin perturbar el funcionamiento del sistema.

En esta sección se presenta el diagrama general de bloques, explicaciones detalladas de configuración y diseño, circuitos esquemáticos, diagramas de tiempo y de estado. Al final se presentan las especificaciones técnicas del aparato. Las hojas técnicas de todos los circuitos integrados utilizados se incluyen en los anexos.

2.1 DIAGRAMA DE BLOQUES GENERAL

Como ya se mencionó, el circuito de interfaz tiene como propósito llevar la información que transporta una línea E1 hacia una PC. Para cumplir con esta función, el circuito debe realizar distintos procesos, entre los cuales están acoplamiento de impedancias, decodificación de HDB3 a TTL, recuperación de reloj, sincronización con la trama E1 y acceso a un puerto específico de la computadora.

El trabajo de diseño del circuito se inició tomando como base el diagrama que se muestra en la figura 2.1, resultado de investigaciones previas[1]. Este diagrama toma en cuenta la mayoría de procesos mencionados, pero resulta limitado.

Entre las limitantes del diseño base podemos mencionar:

a) *Ignora la dependencia de la señal de indicación de lectura de 256 kHz respecto a la señal de indicación de inicio de trama*: El proceso de sincronización con la trama E1 se refiere al hecho de poder distinguir la trama de entre los datos serie de entrada y esto se logra utilizando la FAS, o sea, el inicio de trama. Una vez que se ha detectado esta secuencia de entre los datos, pueden distinguirse individualmente los timeslots y ser enviados a la PC. Ambas señales deben estar sincronizadas y el inicio del envío de la indicación de la lectura debe depender de la indicación de inicio de trama.

b) *Falta de un medio de almacenamiento intermedio*: Las computadoras personales poseen en su mayoría sistemas operativos de multitarea. Esto implica que reparten el tiempo de trabajo entre distintas tareas, una de las cuales es la lectura de datos de los distintos puertos. Por tanto, es de esperar que los datos puedan sufrir retrasos en la lectura si la PC se encuentra realizando procedimientos que demandan mucho tiempo de procesador. El problema está en que los datos que llegan no esperan a que se realice la lectura, por tratarse de monitoreo, y no se tiene ningún control sobre el proceso de transmisión. Ante esto, es necesario tener un medio para almacenar la información de manera temporal y así asegurar que se están recibiendo todos los datos.

c) *Uso del puerto paralelo como puerto de entrada*: Este puerto no es el más adecuado para el manejo de la cantidad de datos que se reciben de la línea E1. Este puerto posee una velocidad nominal máxima de 2Mbytes/s, pero para alcanzar esta tasa de transferencia utiliza algoritmos de compresión implementados en circuitos. La tasa real de transferencia es mucho menor. Se hace necesario también de la implementación de circuitos de handshaking, lo que limita además las líneas disponibles para entrada de información. Otro problema es que no todos los fabricantes soportan los modos de transferencia de alta velocidad. A todo esto se une el hecho de que este puerto se esta volviendo obsoleto con la aparición de puertos, como el USB, que lo superan con creces en velocidad.

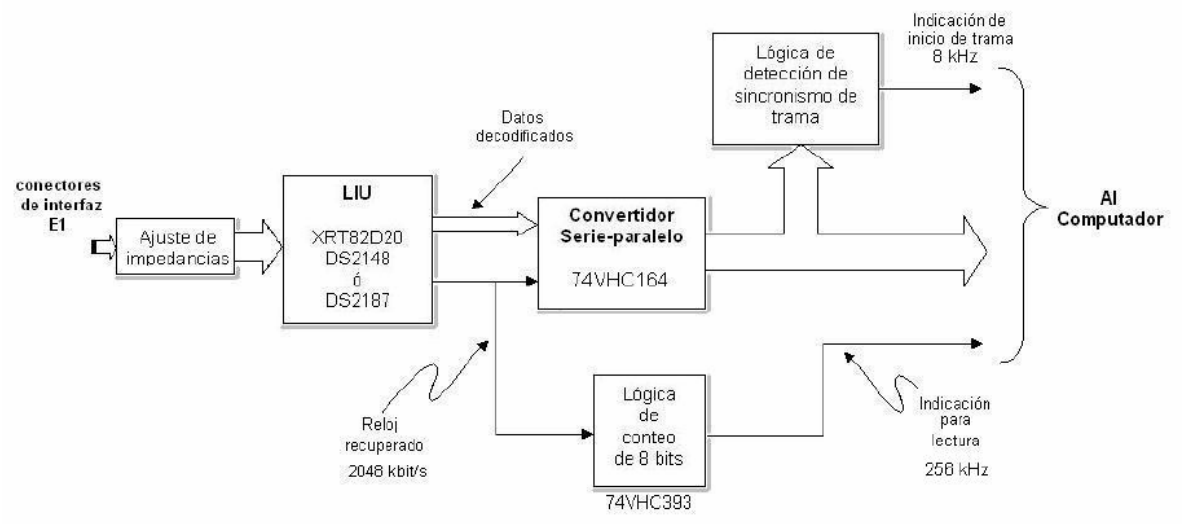


Figura 2.1. Diagrama de bloques base del circuito de interfaz

Basándose en los procesos y las observaciones antes realizadas, se presenta el siguiente diseño para el circuito de interfaz. Como lo muestra la figura 2.2, el circuito se divide en las siguientes etapas funcionales:

a) Circuito de acoplamiento
b) Interfaz de línea
c) Lógica de sincronización
d) Etapa de buffer
e) Interfaz paralelo/USB

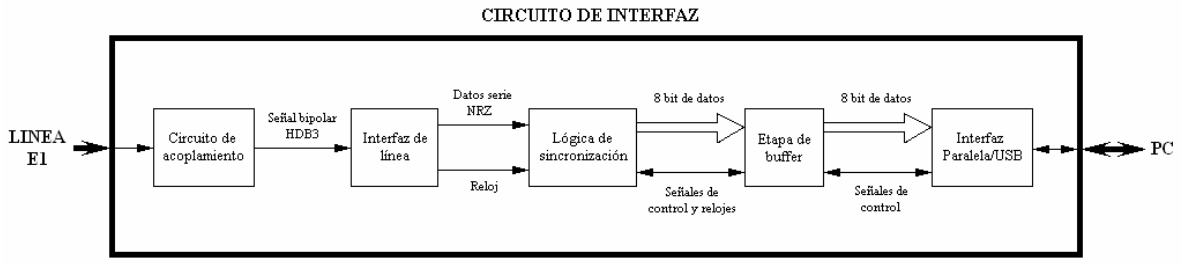


Figura 2.2 Diagrama de bloques general.

En las siguientes secciones se explicará en detalle cada parte.

2.2 CIRCUITO DE ACOPLAMIENTO

La etapa de acoplamiento, tal como se aprecia en el esquema de bloques, está en contacto directo con la línea E1. Su importancia radica no solo en el hecho que es precisamente aquí donde los datos son adquiridos, sino también en que es la parte que garantiza que el equipo cumple la característica de ser transparente, es decir, que no perturbe la información entre el transmisor y receptor.

Las señales de entrada y salida en esta etapa están codificadas en HDB3 y deben cumplir con la máscara presentada en la norma G.703 [4]. El circuito de acoplamiento se ha diseñado para trabajar con una línea de 75Ω desbalanceada, común en los sistemas de transmisión.

Esta etapa posee dos componentes: un conector BNC tipo T y un circuito de terminación de línea.

El conector BNC tipo T posee tres terminales en las que pueden conectarse líneas de 75Ω, y que éstas sigan viendo la misma impedancia y no el paralelo de dos líneas. Para colocar este conector es necesario interrumpir el tráfico, colocar un terminal al transmisor, otro al receptor y el último a la línea que va al circuito terminal.

El circuito de terminación de línea (figura 2.3) fue tomado de las hojas técnicas del IC XRT82D20. Está compuesto por un transformador y un arreglo de resistencias. El transformador provee de aislamiento magnético, es decir, elimina cualquier componente de corriente continua que pueda haber en la línea y filtra componentes de alta frecuencia. Los valores de las resistencias son tales que la impedancia de entrada de la etapa de interfaz de línea sea vista por la línea (a través del conector BNC tipo T) como una impedancia de un valor aproximado a 75Ω. El transformador utilizado en la implementación de este circuito es el Schott 32136, especialmente diseñado para líneas E1/T1.

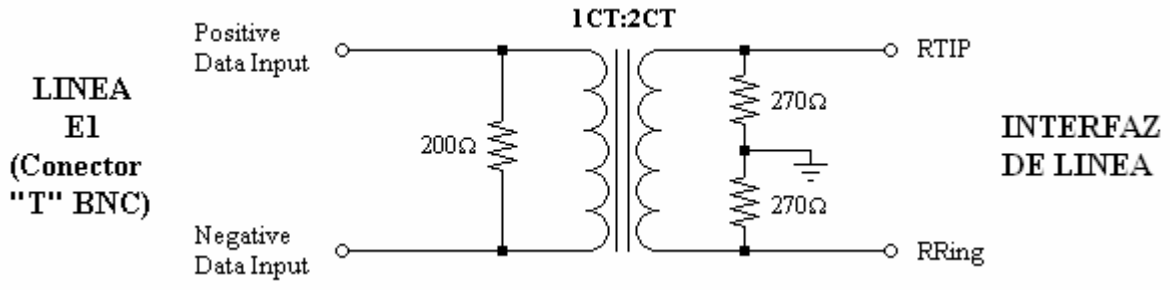


Figura 2.3 Circuito de terminación de línea.

2.3 INTERFAZ DE LÍNEA E1

Esta etapa se encarga del proceso más complejo dentro del aparato, que incluye recuperación de reloj y decodificación de los datos serie HDB3 a niveles TTL/CMOS y formato NRZ. Estas funciones son implementadas por el circuito integrado XRT82D20 (EXAR). La figura 2.4 muestra el diagrama de bloques interno de este circuito integrado.

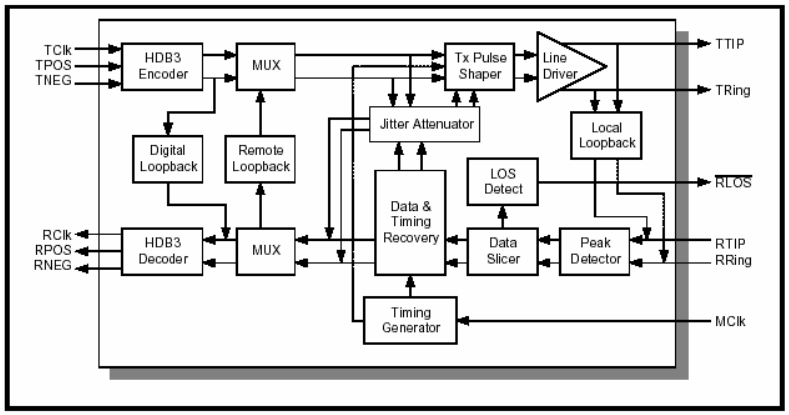


Figura 2.4 Diagrama de bloques interno del XRT82D20.

No se hablará de la compleja teoría sobre la cual funciona el XRT82D20, ya que esto excede el alcance de este trabajo, sino más bien de la forma en que es utilizado.

Para poder realizar la recuperación de reloj, el XRT82D20 necesita una señal de referencia de 2.048 MHz ± 50ppm en el pin MClk. El XRT82D20 no funciona si no posee esta señal. Para proveer esta señal se utiliza un oscilador B550SE 2.048 MHZ (Brookdale Electronics) compatible TTL que opera a 5V.

El XRT82D20 recibe la señal HDB3 en las terminales RTIP (positiva) y RRing (negativa). A estos pines se conecta la salida del circuito de acoplamiento. El integrado puede también trabajar con señales en código AMI, por lo que es necesario configurarlo para trabajar en código HDB3 poniendo en alto (= 1) el pin TNEG/CODE. Además, el XRT82D20 tiene la capacidad de indicar la pérdida de señal y reloj de entrada a través de los pines RLOS (= 0) y ClkLOS (=0), respectivamente.
Otra característica del XRT82D20 es que posee un atenuador de jitter que puede ser deshabilitado y es configurable para funcionar en transmisión o recepción. En la interfaz se ha habilitado (pin JAEN = 1) y configurado en recepción (pin JATx/Rx = 0).

Los datos TTL/CMOS se obtienen a través de los pines RPOS y RNEG, y el reloj recuperado en RClk. Este integrado puede trabajar en modo Single Rail y Dual Rail. Cuando trabaja en modo Single Rail, en RPOS se obtienen los datos serie en formato NRZ y RNEG toma un valor de 1 cada vez que se da una violación del código HDB3 en la señal entrante. En modo Dual Rail, se obtienen las señales negativas (del código HDB3) a través

de RNEG y las positivas a través de RPOS. El circuito de interfaz funciona en modo Single Rail (TNEG/CODE = 1), debido a que en esta forma es sencillo utilizarlo con lógica TTL, por lo que las salidas hacia la siguiente etapa son RPOS y RClk. Hay que mencionar también que los datos se actualizan en el flanco de bajada del reloj recuperado (pin DIGI =0).

Además del modo de operación normal (transmisión/recepción), el XRT82D20 posee distintos modos de prueba, de entre los cuales podemos mencionar especialmente el modo Local Loop back (pin LLopp = 0). En este modo, la salida de transmisión, TTIP y TRing se retroalimenta hacia las entradas RTIP y RRing e inhiben cualquier entrada externa. Esta configuración permitió probar el XRT82D20 inicialmente sin que fuera necesario un generador de señales HDB3, sino solo datos TTL/CMOS a 2.048 Mbit/s.

El XRT82D20 tiene la capacidad de trabajar a 5V ± 5% y a 3.3V ± 5%. En el circuito de interfaz, se trabaja éste y todos los demás circuitos en 5V (TTL).

El diagrama de la etapa de interfaz de línea se presenta en la figura 2.5 (se han omitido los pines de configuración).

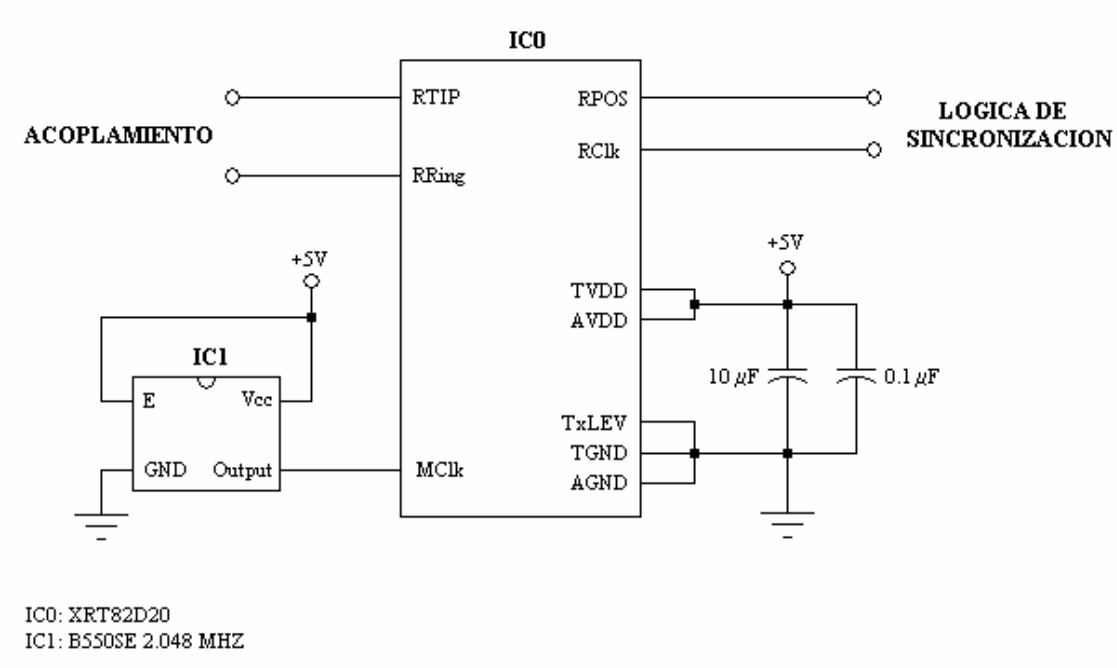
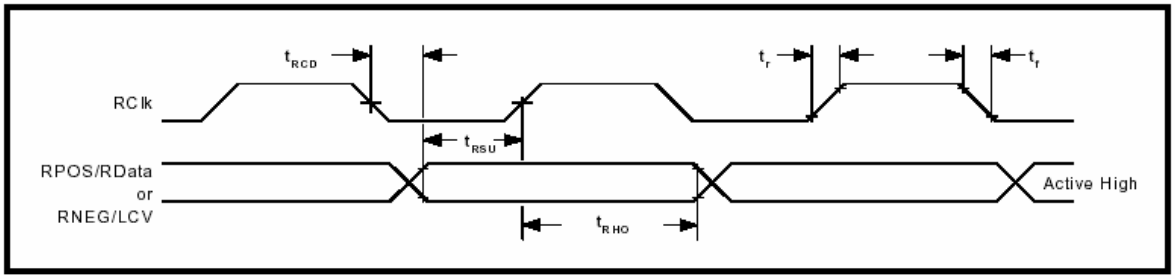


Figura 2.5 Diagrama del circuito de Interfaz de línea.

La figura 2.6 muestra el diagrama de tiempo de la salida de datos en el pin RPOS y el reloj recuperado en RClk, en modo Single Rail. Estas son las señales que recibe la siguiente etapa del aparato. Los valores típicos de los tiempos mostrados se encuentran en la tabla 5 de las hojas técnicas del XRT82D20 (revisión 1.0.6).



t_{RSU} : Tiempo de establecimiento de los datos recibidos (Receive Data Setup Time)
t_{RCD}: Retraso entre RClk y los datos (RClk to Data Delay)
t_{RHO}: Tiempo de sostenimiento de los datos recibidos (Receive Data Hold Time)
t_r : Tiempo de levantamiento de RClk (RClk Rise Time 10%-90%)
t_f : Tiempo de caída de RClk (RClk Fall Time 90%-10%)

Figura 2.6 Diagrama de tiempo de las señales de salida RPOS y RClk del XRT82D20.

2.4 LÓGICA DE SINCRONIZACIÓN

Esta etapa recibe el reloj recuperado y los datos serie NRZ provenientes de la interfaz de línea, convierte los datos a paralelo y los sincroniza con los octetos de la trama E1. Esta sincronización incluye la generación de relojes de valores fraccionarios del reloj recuperado, para ser usados en la etapa del buffer. Cabe mencionar que en esta etapa no se implementan ni los algoritmos de alineación de trama básica ni de multitrama, esto se hace en el programa.

La figura 2.7 muestra el diagrama de circuito de la presente etapa. Se muestran diferenciadas las distintas partes funcionales que lo componen. A continuación se explica el funcionamiento de cada una de ellas.

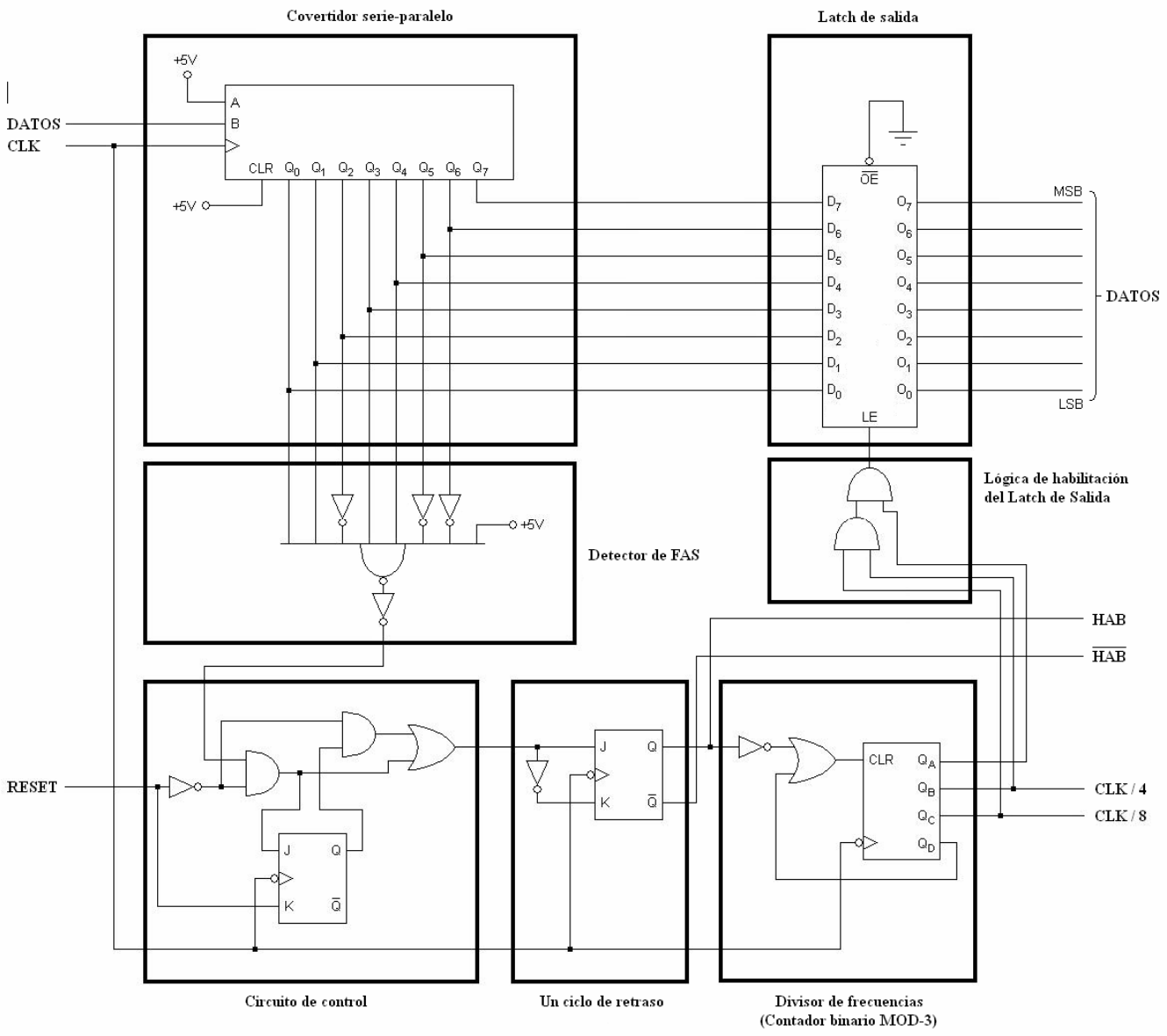


Figura 2.7 Diagrama de circuito de la etapa de sincronización.

2.4.1 Convertidor serie-paralelo

Como su nombre lo dice, esta parte se encarga de convertir los datos serie en paralelos de 8 bits, para lo cual se utiliza el IC 74VHC164. La razón por la cual se realiza este proceso es que es más fácil trabajar con datos paralelos que con los serie, debido a que normalmente en los protocolos tal como el RS-232 se necesita agregar bits extras al flujo de datos, además de la existencia de interfaces paralelas a otros protocolos, como USB.

El 74VHC164 posee dos entradas de datos, A y B. Cualquiera de estas puede servir como entrada para el flujo de datos, únicamente debe ponerse la otra en 1 para que funcione. La

entrada de reloj es activa en flanco de subida, y presenta el primer bit recibido en la entrada Q_0 para luego irse desplazando a Q_1, Q_2, hasta Q_7 a media que se introducen nuevos bits. La entrada de reloj es, por supuesto, el reloj recuperado, el cual se identifica en el diagrama como CLK, como será llamado por el resto del documento.
Las salidas Q_0-Q_7 se conectan al detector de FAS y al latch de salida. En la interfaz E1, los bits de los octetos se envían siempre del más significativo al menos significativo, por tanto el bit más significativo (MSD) es Q_7 y el menos significativo (LSD) es Q_0.

2.4.2 Detector de FAS

Esta parte detecta la secuencia de 7 bits 0011011 de la FAS[3] presente en el timeslot 0 de la trama par de la interfaz E1. El circuito se implementa mediante una NOR de 8 entradas (74HCT30) y algunos inversores (74VHC04). Como puede observarse en la figura 2.7, la secuencia se busca en los bits menos significativos. Solo cuando la secuencia aparece en las salidas Q_0-Q_6 el estado de la salida se pondrá en 1.

2.4.3 Circuito de control

Aquí es donde se controla la sincronización de los octetos. La función de esta parte es disparar la generación de relojes fraccionarios al detectar la primera FAS y detenerlo en caso de que el programa así lo requiera, debido, por ejemplo, a perdida de alineación.

La implementación está hecha a través de un circuito secuencial sincrónico que posee dos entradas: una que viene del detector de FAS y la otra es RESET, que proviene de la computadora. Posee además dos estados, por lo tanto un solo Flip-Flop. La salida va hacia el pin de puesta a cero (CLR) del divisor de frecuencia. La figura 2.8 muestra el diagrama de estados del circuito.

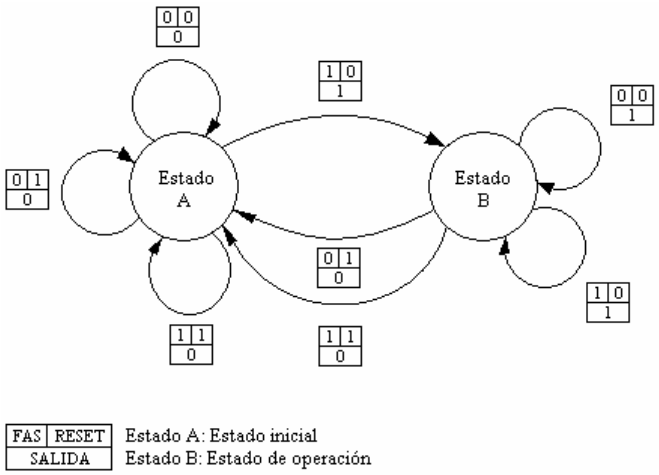


Figura 2.8 Diagrama de estados del circuito de control.

El circuito funciona de la siguiente manera: El estado A se toma como estado inicial. En este estado el circuito puede estar ya sea esperando que llegue una señal del detector de FAS (RESET = 0, FAS = 0) o en estado de reset, en el cual no responderá a la señal del detector (RESET = 1, FAS = X). En ambos casos, la salida se mantiene a cero.

El estado B se considera el estado de operación. La única forma de pasar del estado inicial al de operación es si se recibe una señal del detector mientras espera (RESET = 0, FAS = 1), la salida se pone a 1 para iniciar la generación de los relojes. Una vez en el estado de operación, se ignora la entrada de la FAS y se mantiene la salida en 1. No se regresa al estado inicial sino hasta que la señal de RESET toma el valor de 1.

Es de hacer notar que existe un ciclo de retraso en la salida (Flip-Flop D), y es necesario debido a que la salida del circuito, como en todo circuito secuencial, cambia inmediatamente a la llegada de la señal, o sea no es síncrona, lo que provocaría un deslizamiento de un bit al habilitar el divisor de frecuencias. Esta situación se explica con mayor detalle más adelante. A la señal de salida retrasada se le llama HAB.

La existencia de la señal de RESET permite que los algoritmos de alineación de trama básica, multitrama o VRC definidos en la norma UIT-T G.706 puedan ser implementados por un programa en la PC, pues permite una nueva sincronización, ya sea porque la secuencia que detectó no era la FAS, si existe pérdida de alineación o si hay errores en la trama de origen. Además, esta señal debe usarse para asegurarse que el estado inicial sea A (en los circuitos secuenciales no se puede saber cual es el estado inicial) y para permitir algún tipo de configuración que requiera cierto tiempo sin que se reciban datos.

Los ICs utilizados para la implementación de este circuito y el ciclo de retraso son inversores (74VHC04), OR de dos entradas (74VHC32), AND de dos entradas (74VHC08) y Flip-Flop J-K (74VHC112).

2.4.4 Divisor de frecuencias

El divisor de frecuencias está formado por un contador binario MOD-3, es decir, hace 2^3 conteos. Por tratarse de un número de conteos que es una potencia de 2, este sirve como divisor de frecuencias, siendo la salida menos significativa de frecuencia $f_{CLK}/2$, la siguiente f_{CLK} /4 (la que se llamará CLK/4) y la más significativa $f_{CLK}/8$ (CLK/8).

Se utiliza para la implementación el IC 74VHC393, que un contador doble de 4 bits con reloj activo en flanco de bajada. Este IC posee una entrada de puesta a cero CLR activa alta, la que manejada por dos señales: la salida del circuito de control retrasada (HAB) y el bit de conteo más significativo. Como se observa en la figura 2.7, cualquiera de las dos que se encuentre en 1 hace que el conteo vuelva a cero. El bit de conteo provoca que se hagan únicamente 2^3 conteos. La señal HAB sirve de habilitador para el inicio de conteo para que las señales de frecuencias fraccionarias CLK/4 y CLK/8 estén sincronizadas con los

timeslots u octetos para ser utilizadas en las siguientes etapas del circuito de interfaz y también por el habilitador del latch de salida.

2.4.5 Habilitador del latch de salida

Las salidas del convertidor serie-paralelo cambian con cada bit nuevo que llega, por lo que, para lograr distinguir entre cada octeto de la trama, se ha colocado el latch de salida (IC 74VHC373) cuya función es mantener en su salida el octeto durante el mayor tiempo posible, evitando que se presenten los valores intermedios.
El 74VHC373 posee la entrada LE (Latch Enable), que captura el dato presente en las entradas en el flanco de bajada, y lo mantiene en las salidas mientras LE = 0. Si LE = 1, entonces las salidas siguen a las entradas.

LE se conecta a la salida del habilitador del latch de salida, que consiste en una AND de tres entradas (implementada con compuertas de dos entradas 74VHC08) que se conectan a las 3 salidas del contador MOD-3. Con esta configuración, el latch hará la captura de datos cada 8 ciclos de reloj, que iniciarán cuando se presente el octeto posterior al que contenía la FAS que provocó la habilitación del conteo. Además, mantendrá el dato durante 7/8 del período de la señal CLK/8, el resto del tiempo las salidas contendrán datos no válidos, situación que se da justo antes de la captura.

2.4.6 Diagrama de tiempo

Para ilustrar mejor el funcionamiento del circuito de la lógica de sincronización como conjunto se presenta el diagrama de tiempo de la figura 2.9.

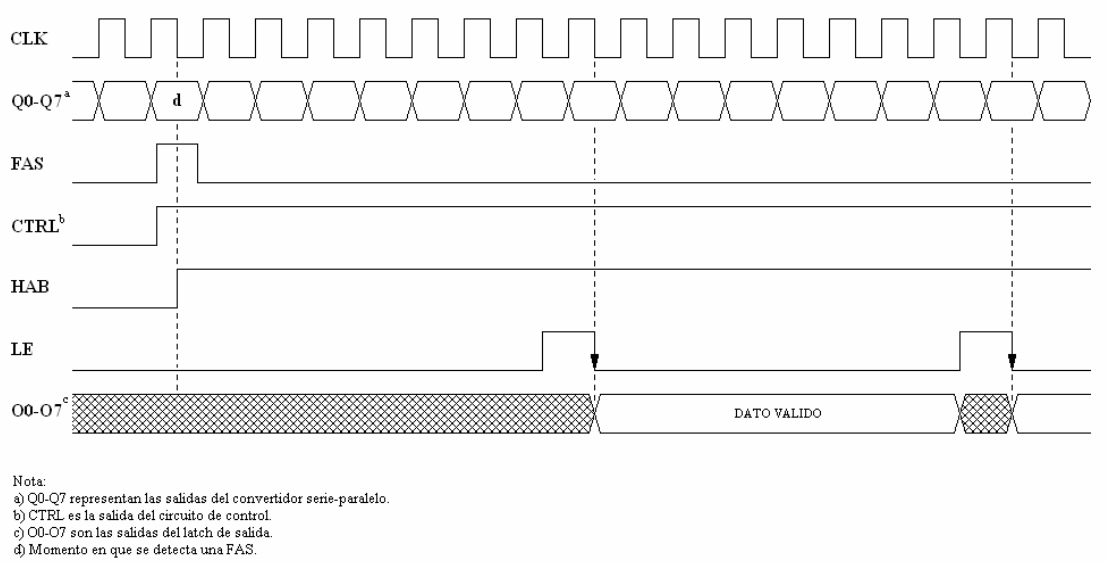


Figura 2.9 Diagrama de tiempo de la lógica de sincronización.

En el diagrama de tiempo se muestra lo que ocurre en el momento en que se detecta una FAS en las salidas menos significativas del convertidor serie-paralelo, y la reacción posterior del circuito hasta llegar al estado de estabilidad. Puede verse que si la señal que controlara la puesta a cero del contador fuera CTRL, entonces habría un deslizamiento de un bit, ya que el conteo iniciaría inmediatamente y no al inicio del siguiente octeto.

También puede observarse el tiempo en que se mantiene un dato válido en las salidas del latch en funcionamiento estable y como las salidas no tienen sentido un ciclo de CLK antes de la captura (flanco de bajada en LE).

2.5 INTERFAZ USB

La siguiente parte a tratar será la interfaz USB, debido a que la etapa de buffer está concebida en base a consideraciones relacionadas a ésta.

El uso del protocolo USB como medio de comunicación con la PC en el instrumento se debe a que posee ventajas como portabilidad (independencia del hardware de la PC y soporte por parte de los sistemas de ventanas más populares), tasas de transferencias altas (1.5Mbit/s, 12 Mbit/s y 480 Mbit/s) y la existencia de controladores (ICs) que proveen interfaces a protocolos más simples como RS-232 o paralelo, tal como la interfaz DLP-USB245M, que fue utilizada en la implementación. A continuación se describe esta interfaz.

2.5.1 DLP-USB245M. Descripción general

La interfaz DLP-USB245M permite enviar y recibir mediante un bus multiplexado 8 bit de datos paralelos hacia o desde la PC a través del puerto USB, sin necesidad de conocer en profundidad el funcionamiento de este protocolo, que es implementado por el controlador FT245BM. Posee además una memoria EEPROM (programable vía software) para guardar configuraciones, un reloj de 6 MHz y un conector tipo B. La figura 2.10 muestra un diagrama esquemático de la interfaz. Una importante ventaja de esta interfaz es que posee librerías y drivers proporcionados por el fabricante, lo que facilita en gran medida el desarrollo de aplicaciones. Este tópico se tratará más adelante en el documento.

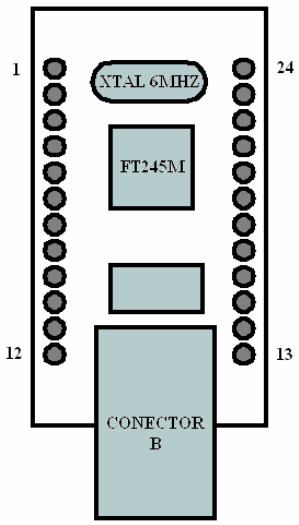


Figura 2.10 Interfaz DLP-USB245M

Algunas de las características más importantes de la interfaz son:

- Capacidad de transferencia máxima nominal de 8 Mbit/s (1 MByte/s).
- Buffer de transmisión FIFO de 384 bytes.
- Compatible USB 1.1 y 2.0.
- Modos de transferencia Bulk e Isócrono.

2.5.2 Configuraciones de energía

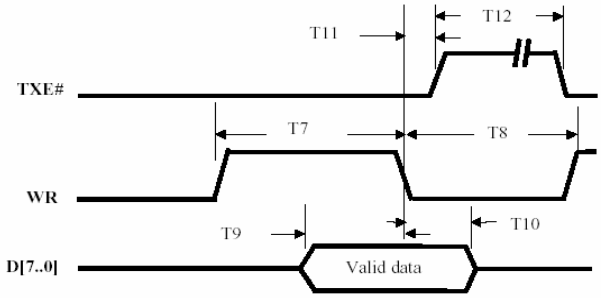
El USB245BM puede configurarse para recibir la energía del bus (bus powered) conectando las entradas RESET# (pin 3), VCC-IO (pin 10), EXTVCC (pin 11) a PORTVCC (pin 12), con un consumo máximo de 100mA. Además, puede configurarse para recibir energía de una fuente propia (self powered) conectando los pines 3,10 y 11 a la fuente de 5V. En ambos casos se trabaja con niveles TTL, si se desea trabajar con niveles CMOS el pin VCC-IO debe ponerse a 3.3V.

Debido a la cantidad de circuitos integrados utilizados en el circuito, y como medida de protección para el controlador de la PC, se ha utilizado el modo "self powered", es decir, la energía para la interfaz USB se toma de una fuente externa de 5V.

2.5.3 Operaciones de lectura-escritura

La interfaz posee 4 pines relacionados a las operaciones de lectura y escritura: WR, RD#, TXE# y RXF#.

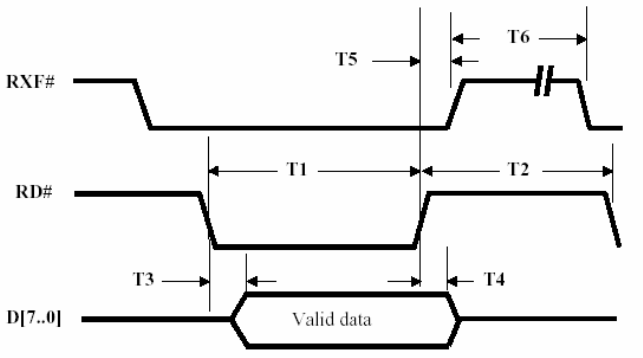
WR (pin 15) escribe el dato presente en los pines de entrada-salida (D0-D7) en el buffer de transmisión cuando se hace una transición de nivel alto a bajo (flanco de bajada). Para que la operación pueda llevarse a cabo, es necesario que la bandera TXE# (pin 14) se encuentre en 0, en caso contrario, no debe intentarse hacer una escritura. TXE# puede ponerse en uno si el controlador está ocupado guardando el último dato o si el buffer de transmisión está lleno. La figura 2.11 muestra el diagrama de tiempo del ciclo de escritura.



Tiempos:
T7: Pulso activo de WR, min. 50 ns.
T8: Pre-carga entre cada WR, min. 50 ns.
T9: Establecimiento de los datos antes que WR esté inactivo, max. 20 ns.
T10: Sostenimiento de los datos desde que WR está inactivo, min. 10 ns.
T11: Desde WR inactivo hasta TXE#, entre 5 y 25 ns.
T12: TXE inactivo despues de ciclo de WR, min. 80 ns.

Figura 2.11 Diagrama de tiempo del ciclo de escritura del DLP-USB245M

La bandera RXF# avisa si se ha recibido uno o más bytes, poniéndose a cero. La lectura del buffer de recepción se realiza poniendo RD# a cero. Después de esto, los pines de E/S (D0-D7) pasan al estado de alta impedancia para luego presentar el primer dato recibido. Al poner RD# de nuevo en uno, los pines de E/S vuelven al estado de alta impedancia. La figura 2.12 ilustra el proceso de lectura.



Tiempos:
T1: Pulso activo de RD, min. 50 ns.
T2: Precarga entre cada RD, min. 50 ns.
T3: De RD activo a datos válidos, max. 30 ns.
T4: Sostenimiento de datos válidos desde que RD está inactivo, min. 10 ns.
T5: Desde RD inactivo a RXF#, entre 5 y 25 ns.
T6: RXF incativo después de ciclo de RD, min. 80 ns.

Figura 2.12 Diagrama de tiempo del ciclo de lectura del DLP-USB245M

En el aparato, el ciclo de lectura no es utilizado, ya que únicamente se necesita recibir un bit (la señal RESET). Lo que se hace es enviar un byte al buffer del DLP-USB245M y el cambio de RXF# es el que se utiliza como RESET (negado).

2.5.4 Envío de datos a la PC

Los datos presentes en el buffer de transmisión se envían de dos formas: utilizando el latency timer o a través del pin SND.

Por defecto, los datos presentes en el buffer de transmisión se envían cierto tiempo después de haber completado la última transacción USB de envío de datos. Este tiempo esta definido por el valor del latency timer, que puede estar entre 1 y 255ms (no puede configurarse en valores fraccionarios). Este valor se puede configurar vía software. El latency timer no puede ser deshabilitado. Esta forma de envío tiene el problema de que realmente no se tiene control de cuantos bytes se han de mandar, especialmente si el flujo de datos es grande, ya que la computadora puede retrasar la petición de transacción y en el buffer podrían existir más o menos datos de los esperados.

La otra forma es utilizar la señal SND (pin 9). Si la interfaz no está en ciclo USB suspend, un flanco de subida provocará que se envíen en la siguiente transacción todos los datos presentes en el buffer en ese momento. Este método es más conveniente ya que se sabe que esos datos serán enviados, aunque sea en transacciones distintas. Si se desea utilizar este método, es necesario cambiar el valor de latency timer a un tiempo mayor que el período de SND, para que no haya transmisiones no deseadas. Debido a sus ventajas, en la implementación se utiliza este método para enviar los datos.

Cabe mencionar el hecho de que si el buffer de transmisión se llena, esto no significa que los datos han de enviarse automáticamente, sino que se esperará al próximo evento (vencimiento del latency timer o flanco de subida en SND).

2.5.5 Modos de transferencia y eficiencia en la transmisión de datos

El controlador FT245BM puede transmitir datos tanto en modo isócrono como en modo bulk. De ambos modos de transferencia de datos, el modo isócrono es el más conveniente, debido a que no existe retransmisión, garantiza el ancho de banda, posee esperas cortas y las transacciones se hacen continuamente. El problema con el modo isócrono es que la librería estándar (D2XX) del controlador no lo soporta, por tanto sería necesario escribir nuevos drivers y librerías, lo cual es un proceso complicado, sobre todo si se trabaja en Windows XP y NT. Además, los fabricantes no proveen mucha documentación para trabajar en este modo.

Por estas razones el modo escogido fue el Bulk, cuyas características principales se presentan a continuación:

- Usado para transmitir paquetes grandes de datos.
- Corrección vía VRC, con garantía de entrega.
- Sin garantía de ancho de banda o de tiempos de espera.
- Para velocidades full speed y high speed.
- Datos transferidos por milisegundo (transacción): 1216 bytes en paquetes de 64 bytes.

Debido a que este modo de transmisión no garantiza el ancho de banda, lo más conveniente es que, en el caso de nuestro aparato, la interfaz no sea usada junto a otros dispositivos USB.

Por ser un aparato de monitoreo, es imposible hacer peticiones de retrasmisión si se presentara un error en la línea USB. Esta situación provocaría errores, posiblemente una pérdida de alineación y por consecuencia, la interrupción de una prueba, pero debido a que la distancia entre interfaz y PC no es muy grande (6 pies), esta situación difícilmente ha de presentarse.

La velocidad de los datos recibidos por el aparato es de 2.048Mbit/s, es decir, 256 bytes/ms, que es menor a la velocidad máxima nominal en el modo Bulk. Los datos son enviados en paquetes de 64 bytes, 19 como máximo por transacción. Si se envía una cantidad de datos menor a 64 bytes en alguno de estos paquetes, el protocolo detiene la transacción, por tanto, si se quiere mandar más datos después, debe iniciarse una nueva transacción. Cada transacción tiene un período aproximado de 1 ms. Esta situación reduce la capacidad de transmisión de la interfaz. Además, el controlador FT245BM utiliza 2 bytes de cada paquete de 64 bytes para enviar información de mantenimiento, lo que hace que la información de usuario sea de 62 bytes.

Teniendo en cuenta esto, y como se menciona en la nota de aplicación AN-232B-03 “Optimizing D2XX Data Throughput” publicada por FTDI Chip, para lograr una transferencia óptima de datos es necesario que se utilice el menor número de transacciones. Esto quiere decir que cada vez que se envíe información utilizando el pin SND, por ejemplo, la cantidad de datos efectivos debe ser múltiplo de 62 bytes.

Debido a los aspectos mencionados y al tamaño del buffer del controlador (384 bytes), en nuestro aparato se hacen envíos de paquetes de 372 bytes por transacción, utilizando el pin SND para mayor control. Esto permite que exista tiempo intermedio entre cada transacción.

2.6 ETAPA DE BUFFER

La necesidad de esta etapa surge debido a una limitación de la interfaz USB: el tamaño del buffer del FT245BM no es suficiente para evitar la pérdida de datos debido a los tiempos de espera relacionados al modo de transferencia Bulk. Si la PC se encuentra ocupada en alguna tarea que requiere muchos recursos, el envío de los datos USB es retrasado.

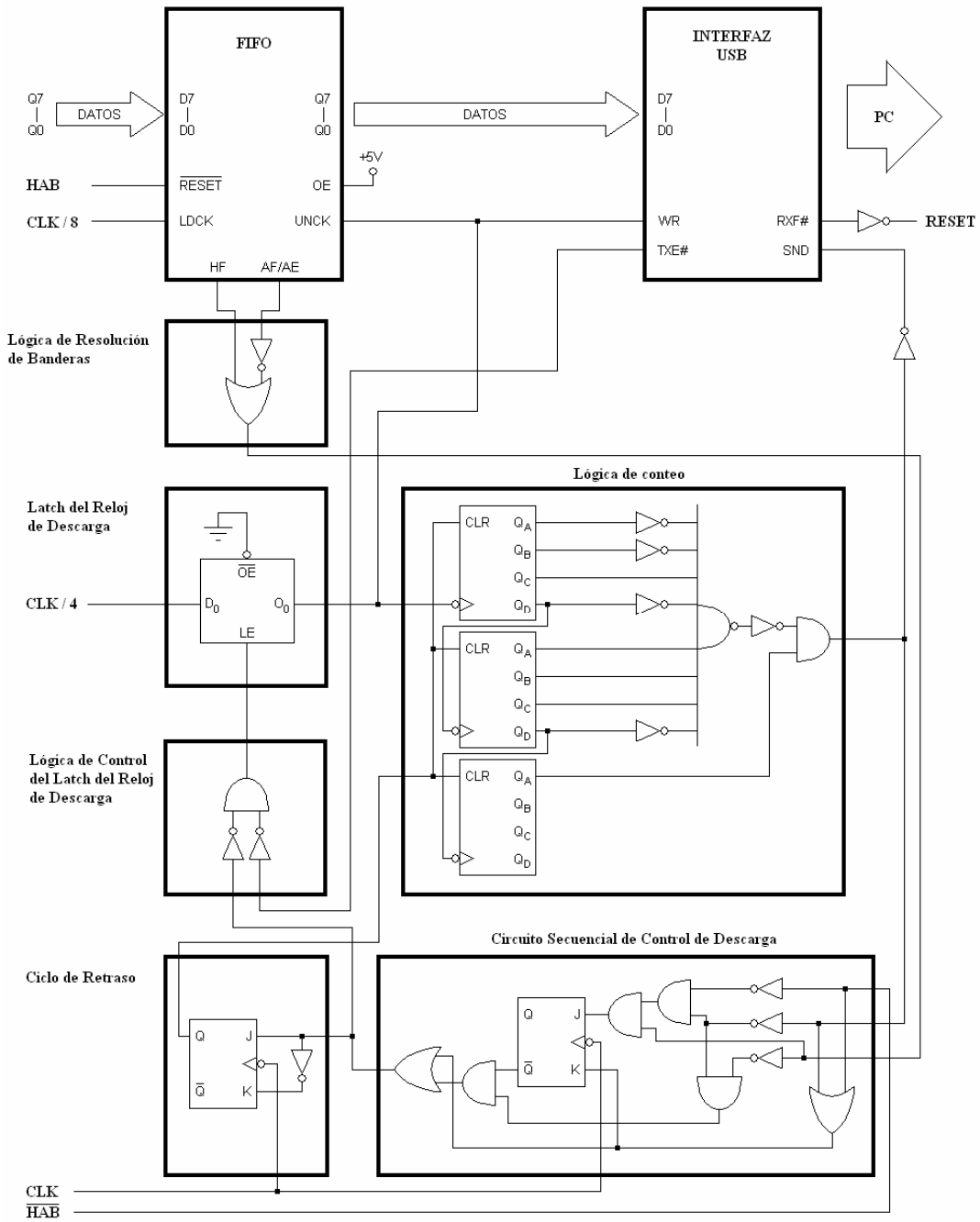


Figura 2.13 Circuito de la etapa de buffer.

La etapa de buffer esta constituida por una memoria FIFO (First-In First-Out) asíncrona (relojes de carga y descarga independientes), en la que se cargan los datos provenientes de la lógica de sincronización y que los descarga en paquetes de 372 bytes al doble de la velocidad de carga (CLK/4) hacia el buffer de la interfaz USB. La descarga es inhibida si la señal TXE# pasa al estado alto. A continuación se presenta una explicación detallada del funcionamiento de esta etapa. El circuito se muestra en la figura 2.13.

2.6.1 Memoria FIFO

La memoria FIFO utilizada en la implementación del circuito es la SN74ACT7802. Tiene una capacidad de 1024 palabras de 18 bits, buses de datos unidireccionales y puede trabajar a velocidades de hasta 40 MHz con tiempos de acceso de 30ns. Posee además banderas que indican el estado de la memoria.

La memoria necesita un pulso bajo en la entrada Reset (pin 1, se escribirá así para distinguirla de la señal RESET proveniente de la PC) antes de iniciar la operación, para colocar los punteros de carga y descarga en sus posiciones iniciales.

La carga de datos presentes en el bus de entrada D_0-D_{17} (pines 7-15, 17, 19-26, respectivamente) ocurre cuando existe una transición de bajo a alto en la señal del reloj de carga LDCK (Load Clock, pin 29). La memoria se llena cuando el número de palabras cargadas excede las 1024. A partir de ese momento, la FIFO ignora cualquier flanco de subida presente en LDCK hasta que no se vacíe por lo menos una palabra, además la señal FULL (pin 35) pasa a cero.

La descarga de cada palabra se hace con un flanco de subida aplicado a la señal de reloj de descarga UNCK (pin 5), después de lo cual aparecerá en el bus de salida Q_0-Q_{17}. Cuando la memoria esta vacía, se ignora la señal UNCK y la bandera EMPTY se pone a cero. Es de resaltar que el primer dato que se carga en la memoria, después de un ciclo de reset, aparece inmediatamente en las salidas, sin necesidad de transición en UNCK.

Además de las banderas ya mencionadas, la SN74ACT7802 posee la bandera HF (Half Full) que cambia a nivel alto si la cantidad de palabras en memoria es de 512 o más. Posee también otra bandera, AF/AE (Almost Full/Almost Empty) que se pone en alto si en la memoria hay X o menos palabras, o si existen 1024-X o más palabras. El valor de X es programable y tiene un valor por defecto de 256. El offset X tomará el valor presente en los pines D0-D7 cuando ocurra un flanco de bajada en la entrada DAF (Define Almos Full, pin 27) mientras se mantiene Reset = 0. Para que X vuelva a tener el valor por defecto (256), debe mantenerse DAF en nivel alto mientras Reset esta en cero.

La bandera AF/AE es una ventaja de esta familia de memorias (Texas Instruments), respecto a otras en el mercado, que usualmente poseen solo las banderas EMPTY, FULL y HF (Half-Full).

Una desventaja de esta memoria es que no puede conectarse en cascada, es decir, no puede aumentarse la profundidad, sino solo el ancho de la palabra. Como una alternativa esta memoria puede utilizarse la FIFO SN74ACT7808, que tiene una capacidad de 2048 palabras por 9 bits y es expandible en profundidad y ancho, además, posee todas las banderas, señales y características de la SN74ACT7802, por lo que puede reemplazarla sin cambios al diseño.

En el circuito, el pin Reset se conecta a la señal HAB proveniente de la lógica de sincronización, por tanto, la memoria se encontrará en estado reset hasta que una FAS sea detectada e inicie generación de los relojes, asegurando su funcionamiento correcto inicial y asegurando la eliminación de datos cuando la señal RESET proveniente de la PC se activa.

Como se indica en la figura 2.13, en el diseño se utilizan únicamente los 8 bits menos significativos de los buses de entrada (D0-D7) y salida (Q0-Q7). El reemplazo de la memoria SN74ACT7802 por la SN74ACT7808 (o un conjunto de estas) es preferible a la multiplexación del bus, si se quiere aumentar la capacidad del buffer.

2.6.2 Proceso de carga

En la figura 2.14 se muestra el diagrama de tiempo de la carga de datos en la memoria FIFO. La señal CLK/8 se utiliza como entrada del reloj de carga LDCK y los datos se actualizan en el flanco de bajada de LE del latch de salida. La utilidad del latch de salida puede notarse, ya que se provee un tiempo de estabilidad amplio para que los datos sean leídos.

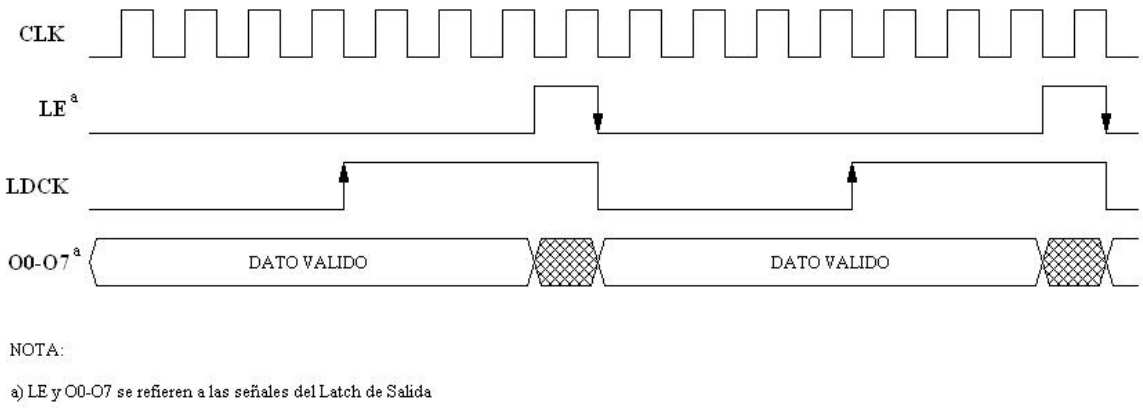


Figura 2.14 Diagrama de tiempo del proceso de carga.

2.6.3 Lógica de resolución de banderas

Esta lógica provee la señal de disparo (nivel activo alto) para iniciar la descarga cuando las banderas indican que la memoria ha llegado al nivel indicado, siendo esta recibida por el circuito secuencial de control de descarga.

La descarga se hace en paquetes de 372 bytes, pero esta inicia cuando las banderas indican un nivel de 256 o más palabras en la memoria. Este límite se debe a que, dado que la carga de datos no se detiene en ningún momento, durante el tiempo en que se enviaron los 256 bytes se habrán cargado otros 128 bytes, pudiendo completarse el paquete. Con este proceso la memoria nunca se vacía completamente, sino que habrá un remanente de por lo menos 12 bytes. No es recomendable vaciar completamente la memoria, debido a que podrían surgir problemas con el primer byte.

El inicio de la descarga en X = 256 bytes evita tener que programar un nuevo valor para AF/AE, proceso que necesitaría de lógica extra.

En la figura 2.15 se muestra el mapa de memoria de la SN74ACT7802, y los valores que toman las banderas en cada parte. El área sombreada corresponde a la parte en que la salida de la lógica de resolución de banderas esta inactiva (nivel bajo). En las otras partes, esta deberá estar activa y propiciar una descarga.

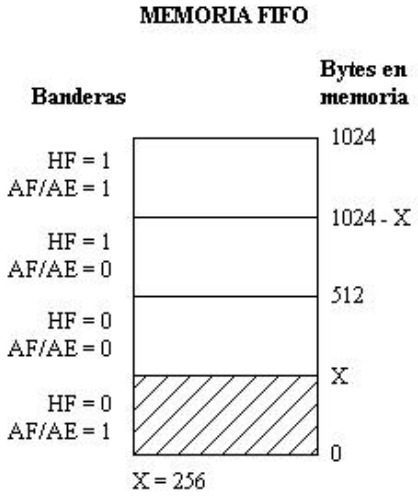


Figura 2.15 Mapa de memoria de la FIFO.

2.6.4 Latch del reloj de descarga y su lógica de control

La descarga, como ya se ha mencionado, se hace al doble de la velocidad de llegada de los datos, es decir, se utiliza la señal CLK/4 proveniente de la lógica de sincronización para manejar UNCK de la FIFO. La misma señal se utiliza para realizar la escritura al buffer de

la interfaz USB, a través de la entrada WR. Esto es posible debido a que ambas entradas utilizan distintos flancos activos.

El latch de descarga (74VHC373) sirve para inhibir la llegada del reloj CLK/4 a UNCK y WR, pudiendo así controlar la descarga y escritura.

La lógica de control permite que dos señales, TXE# y la salida del circuito secuencial de control de descarga (Z), manejen el latch. Cuando TXE# = 1 o Z = 1, se detiene el paso de CLK/4 (LE = 0).

2.6.5 Lógica de conteo

Esta parte consta de 3 contadores de 4 bits (74VHC393) conectados para construir un contador binario de 9 bits y un arreglo de compuertas lógicas cuya salida cambia a nivel alto al detectar el número 372_{10} ($1\ 0111\ 0100_2$).
La señal de salida lógica de conteo sirve al circuito de control de descarga para saber en que momento se ha terminado de descargar y escribir un paquete. Esta misma señal negada se conecta a SND provocar el envío del contenido del buffer de la interfaz USB hacia la PC al finalizar la escritura de un paquete.

La señal de reloj del contador es la misma que para UNCK y WR, el reloj de descarga. Por ser flanco de bajada activo, el conteo se incrementa una vez que el dato ha sido escrito al buffer de la interfaz USB.

La entrada CLR (activa alta) de los contadores esta conectada a la salida del circuito de control de descarga, pero con un retraso un ciclo de CLK. Entonces, cuando Z = 1, el contador de 9 bits se pone a cero y esta listo para contar un nuevo paquete. Los motivos del ciclo de retraso serán ilustrados en la siguiente sección.

2.6.6 Circuito secuencial de control de descarga

Este circuito controla el inicio y finalización de la descarga de la memoria y escritura al buffer de la interfaz USB de los paquetes de 372 bytes. Se trata, como su nombre lo dice, de un circuito secuencial sincrónico de dos estados (un Flip-Flop J-K), cuyo diagrama de estados se presenta en la figura 2.16. El circuito utiliza como reloj la señal CLK para que existan un tiempo de respuesta adecuado.

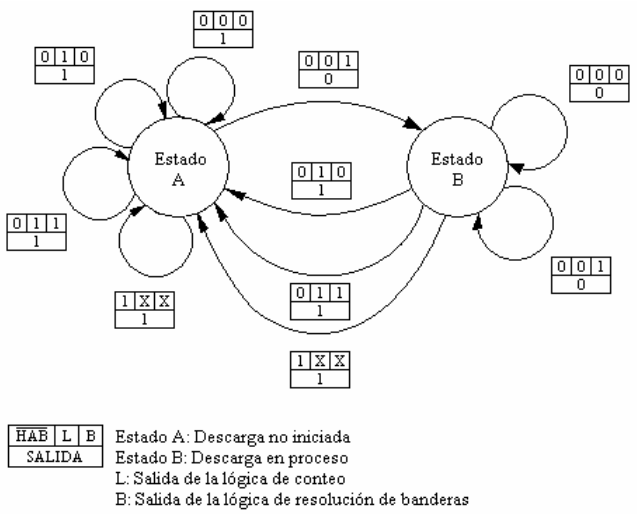


Figura 2.16 Diagrama de estados del circuito de control de descarga.

El circuito posee 3 entradas: La salida de la lógica de conteo (L), la salida de la lógica de resolución de banderas (B) y la señal HAB negada proveniente de la lógica de sincronización.

La señal HAB negada produce que la salida tome el valor de uno y que se mantenga o regrese al estado A, cuando toma valor de uno. Esta señal tiene prioridad sobre las otras, es decir, si está en nivel alto, actúa sin importar el estado de las demás. Es una clase de "reset".

Mientras el circuito se encuentra en el estado A, el conteo, la descarga y la escritura están detenidos, ya que se mantiene la salida Z en nivel alto, inhibiendo el reloj y manteniendo el CLR del contador activo. Como la salida (retrasada un ciclo por un Flip-Flop D) controla la puesta a cero del contador, esta también hace que L tome el valor de cero. Por tanto, mientras se está en el estado A, L será cero la mayor parte del tiempo, y solo tomará un valor distinto durante el ciclo de retraso.

La única combinación de señales que permite el paso al estado B es HAB = 0, L = 0 y B = 1, es decir, cuando el contador esta en cero y la lógica de banderas se ha activado, indicando que se ha alcanzado el nivel adecuado en la memoria. La salida es puesta a 0 para iniciar el conteo, habilitando el reloj de descarga y poniendo CLR a cero.

Una vez en el estado B, o de descarga en proceso, la señal B es ignorada, el circuito se mantiene en el presente estado y la salida en cero hasta que ya sea HAB negada pase a uno o que L = 1. Cuando salida de la lógica de conteo esta en uno, le indica al circuito que se ha alcanzado el conteo número 372 y que la descarga ha terminado. El circuito cambia entonces la salida a cero y pasa al estado A.

Como en los circuitos secuenciales la salida no es sincrónica, si no se utilizara un ciclo de retraso el valor de L = 1 provocaría que la salida cambiara inmediatamente a uno, y esto, a su vez, que L retornara a nivel bajo. La señal de L habría desaparecido antes de la llegada del flanco de bajada del reloj, impidiendo el cambio de estado, es decir, provocaría que el circuito ignorara el hecho de que el conteo ha finalizado. El circuito de retraso afecta también el inicio del conteo, pero debido a que este trabaja con CLK como reloj, el retraso es despreciable.

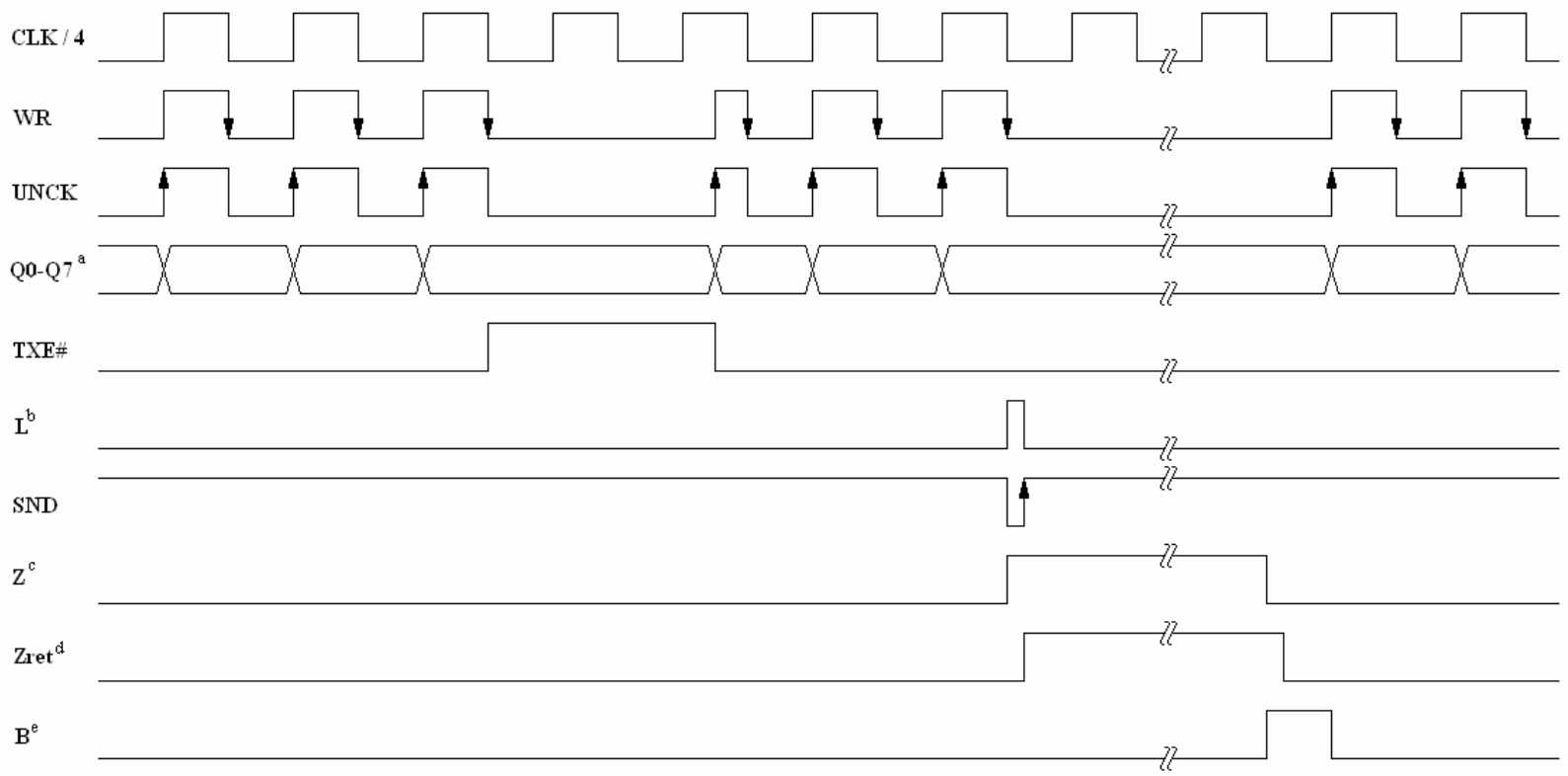
2.6.7 Proceso de descarga

En la figura 2.17 se muestra el diagrama de tiempo del proceso de descarga y escritura de datos. En este se retratan los estados de las señales en tres momentos distintos.

En el primer momento, mientras se realiza la descarga de un paquete, la señal TXE# proveniente de la interfaz USB cambia a uno después de una escritura, debido a que el buffer de la interfaz está lleno, lo que provoca que se detengan los procesos. Luego que el buffer se vacía, TXE# vuelve a cero, y se reanuda la descarga.

Seguidamente, se muestra el momento en que se activa la señal de la lógica de conteo, indicando que ya detecto el número 372, o sea, el fin del paquete. Puede notarse el retraso entre la señal L y la salida del circuito de control de descarga retrasada (Z_{ret}). Este retraso también hace que el pulso SND posea un ancho mayor y el envío hacia la PC ocurra después de haber escrito el último dato.

Al final, se muestra el momento en que se inicia la descarga, gracias al pulso proveniente de la lógica de banderas.



NOTA:

a) Q0-Q7 se refiere a las salidas de la memoria FIFO
b) L se refiere a la salida de la lógica de conteo
c) Z es la salida del circuito de control de descarga
d) Zret es la salida Z retrasada un ciclo de CLK
e) B es la salida de la lógica de resolución de banderas

Figura 2.17 Diagrama de tiempo del proceso de descarga.

2.7 CONCLUSIONES Y RECOMENDACIONES AL CAPÍTULO II

- El diseño del circuito de interfaz que se ha presentado en este capítulo puede manejar los datos provenientes de una interfaz E1, es decir 2.048 Mbps, y llevarlos satisfactoriamente hasta la PC. Por la forma en que está diseñado, este circuito puede servir no solo para decodificar señalización R2 digital, sino también CCS7 o cualquiera que se transporte sobre E1, además puede obtenerse la información de cualquiera de los canales de voz / datos.

- Una forma de mejorar la robustez del circuito respecto a los retrasos en la lectura de datos por parte de la computadora, que dependen de la velocidad de los buses y del procesador además de la cantidad de aplicaciones que se encuentren corriendo en un instante dado, es el aumento de la capacidad de la memoria FIFO. En la sección 2.6.1, se mencionó un tipo específico de memoria, porque posee características muy similares a la SN74ACT7802. Las memorias de otros fabricantes poseen normalmente más capacidad que las de Texas Instruments, pero no poseen las banderas AF/AE (Almost Full / Almost Empty), por lo que el único nivel intermedio identificable es cuando se encuentra a la mitad (bandera normalmente llamada HF, Half Full). Esta situación haría innecesario el uso de la lógica de resolución de banderas, debiendo conectarse la bandera HF directamente al circuito de control de descarga, además, se desperdiciaría la mitad de la capacidad de la memoria. Este último hecho puede compensarse colocando una memoria de mucha más capacidad, por ejemplo, de 64 kbytes, que está muy por encima de la SN74ACT7802.

2.8 REFERENCIAS BIBLIOGRÁFICAS

[1] Calderón Osorio, Daniel Alberto. “Diseño de instrumentos virtuales para medir señalización siete por canal común de la Unión Internacional de Telecomunicaciones (CCS7UIT) en tiempo real”. Universidad de El Salvador. 2003.

[2] Sunrise Telecom Inc. “Technology Series. Introduction to E1/2.048 Mbit/s”. Publication Number TEC-GEN-001 Rev.B. 2001.

[3] Tibbs, John. “E1 Pocket Guide. The World of E1”. Vo. 5. Wavetek Wandel Goltermann. .

[4] UIT-T. “Características físicas y eléctricas de los interfaces digitales jerárquicos”. Rec. G.703. 1991.

[5] UIT-T. “Estructuras de trama síncrona utilizadas en los niveles jerárquicos primario y secundario”. Rec. G.704. 1991.

[6] UIT-T. “Procedimientos de alineación de trama y de verificación por redundancia cíclica (VRC) relativos a las estructuras de trama básica definidas en la recomendación G.704”. Rec. G.706. 1991.

CAPÍTULO III

DESARROLLO DEL PROGRAMA DE DECODIFICACIÓN DE SEÑALIZACIÓN R2 DIGITAL DE LÍNEA

INTRODUCCIÓN

En esta etapa del trabajo se presenta el desarrollo del programa de decodificación de señalización R2 digital de línea, como parte del instrumento virtual de monitoreo. El programa tiene como objetivo tomar los datos recibidos a través del puerto USB, provenientes del circuito de interfaz, y decodificar la información de señalización R2 digital de línea de los 30 canales de voz que transporta la interfaz E1, para luego almacenarlos en archivos de texto y presentar uno en pantalla.

Los procesos de alineación asociados a la interfaz E1, mencionados en el capítulo I y que son parte de la recomendación G.706 de la UIT-T, son realizados como parte del programa. Estos procesos incluyen alineación de trama básica y multitrama de señalización. Esta característica permite expandir el tipo de funciones de servicio, por ejemplo agregar la verificación de redundancia cíclica, que no serían posibles si estuvieran implementados en el circuito de interfaz, aunque agregan carga a la PC.

El programa ha sido desarrollado en Visual C++ 6.0, que es un lenguaje de gran versatilidad, y utiliza las técnicas de programación de multitareas que proveen las librerías Win32 API y MFC (Microsoft Foundation Class Library), propias de todos los sistemas operativos Windows, aunque este ha sido desarrollado y probado en el sistema Windows XP.

La información presentada a continuación incluye la programación de la interfaz DLP-USB245M utilizando las librerías que provee el fabricante, una pequeña revisión de la programación multitarea y luego la explicación funcional del programa. El código fuente completo se incluye en el anexo B.

3.1 PROGRAMACIÓN DE LA INTERFAZ DLP-USB245M

En el capítulo anterior fue explicada la manera en que, gracias al circuito de interfaz, los datos provenientes de la línea E1 son convertidos de un tren del pulsos serie a octetos sincronizados con los timeslots de la trama E1, para luego ser transferidos hacia la PC a través de la interfaz paralela / USB.

USB es un protocolo bastante complejo. Afortunadamente la interfaz DLP-USB245M implementa este protocolo a través del chip controlador FT245BM y el driver FTD2XX. A partir de este momento el interés se dirige hacia la estructura y el manejo de las librerías del driver FTD2XX.

Cabe mencionar que existe una forma alternativa al driver FTD2XX para comunicación con el controlador, utilizando un VCP (Virtual COM Port), pero esta no será discutida pues es más complicada, ya que deben utilizarse librerías estándar de Windows.

3.1.1 El driver FTD2XX

El driver FTD2XX tiene como finalidad establecer comunicación directa entre la aplicación y el controlador FT245BM a través del protocolo USB, usando como base el sistema operativo Windows en sus versiones 98, 2000 o XP.

Su arquitectura consta de un driver WDM (Windows Device Manager) que interactúa directamente entre el chip controlador y el USB STACK de Windows; así mismo consta de una librería DLL (Dynamic Link Library) que crea un enlace entre el driver WDM y la aplicación. Este tipo de librería posibilita el desarrollo del programa en distintos lenguajes (Visual C++, Visual Basic, Delphi, C++ Builder, etc).

La existencia de esta librería facilita en gran medida la programación, ya que el usuario interactúa con un conjunto de funciones previamente definidas en forma amigable. En el caso de programar utilizando C++, el fabricante proporciona un archivo de cabecera (ftd2xx.h) que contiene definiciones de variables y tipos de datos, con nombres que hacen más sencillo el manejo de las funciones.

3.1.1.1 Interfaces de programación del driver FTD2XX

Como ya se ha mencionado, el driver proporciona una librería de funciones necesarias para la programación y adquisición de datos. Estas funciones están clasificadas en cuatro grupos de acuerdo a sus características, para tener mayor claridad.

- *Funciones de la interfaz clásica:* Este grupo clasifica aquellas funciones que ejercen la parte básica del funcionamiento del controlador, como la identificación del

mismo, lectura y escritura simple, revisión de estatus, inicialización, reinicialización, etc.

- *Funciones de la interfaz de programación de la EEPROM:* El módulo cuenta con una memoria EEPROM, la cual contiene parámetros y características del chip controlador. Este grupo de funciones permiten leer y modificar estos parámetros, tales como la viñeta del controlador, tipo de transferencia, alimentación, etc.

- *Funciones extendidas de la API:* Son un complemento a las funciones básicas, con ellas es posible modificar otros parámetros que pertenecen al protocolo USB en sí, como son el tamaño de transferencia, latency timer, etc.

- *Funciones de la FT-Win32 API:* Es una alternativa un poco más sofisticada a las funciones de la interfaz clásica, basada en las llamadas a la Win32 API, pero básicamente posee la misma utilidad. Es de hacer notar que no deben mezclarse ambos grupos de funciones dentro de un mismo programa.

3.1.2 Flujo de datos a través del puerto USB

La interfaz recibe un flujo de datos constante, sobre el que se tiene poco control. Surge entonces la necesidad de realizar todos los ajustes necesarios para conseguir una transferencia de forma eficiente, y de esta manera evitar pérdidas de datos. Para ello es necesario primero entender la manera en que los datos son transferidos desde el dispositivo USB hasta la aplicación.

En el protocolo USB, el intercambio de los datos se da entre dos IC controladores, de los cuales hay uno que maneja todo el proceso, al que se denomina Host, y que se encuentra normalmente del lado de la PC.

Cuando se desea enviar datos en una u otra dirección, ambos controladores intercambian primero información de control de la transferencia (handshaking) y luego envían los datos de usuario. A este proceso se le denomina transacción. Estas transacciones pueden ser de 2 tipos: de entrada y salida, desde el punto de vista del host. La información de control y de usuario se envía organizada en forma de tramas, pues USB es un protocolo serial. A la cantidad máxima de información de usuario que puede intercambiarse en cada transacción se le denomina Maximum Transfer Size.

Existen distintos modos de transferencia de información, que poseen distinto formato de tramas, manejo del ancho de banda, información de control, control de errores y tiempo de entrega de la información. Entre estos modos pueden mencionarse el Bulk, Isócrono, Interrupt y Control.

El controlador FT245BM puede trabajar en modo Bulk y en modo Isócrono. Las librerías del driver D2XX soportan, en su versión actual v2.01, únicamente el modo Bulk.

En este modo, los datos de usuario están organizados en paquetes de 64 bytes, pudiendo soportar un máximo de 64 kbytes por transacción. Si la información enviada no es un múltiplo de 64 bytes, y por consiguiente alguno de estos paquetes no posee 64 bytes, la transacción se toma como terminada, y para enviar más datos, debe iniciarse otra transacción. Si se utiliza en FT245BM, la carga efectiva de usuario por paquete es de 62 bytes, ya que este utiliza 2 bytes para enviar cierta información de control.

Hablando específicamente de la interfaz paralelo / USB, cuando se manda un flanco de bajada a través del pin WR, se escribe el byte presente en las entradas en un buffer FIFO interno, que posee un tamaño de 384 bytes.

El envío de los datos presentes en el buffer interno puede propiciarse de dos formas:

- La expiración de un temporizador del controlador llamado latency timer. Su valor puede ser ajustado entre 1 y 255 milisegundos, con una resolución de 1 ms, siendo por defecto de 16 milisegundos.

- Un flanco de subida en la entrada SND del controlador del USB.

Una vez enviados, los datos son recibidos en otro buffer dentro de la PC, llamado buffer de transferencia, que posee un tamaño igual al valor del Maximum Transfer Size. La aplicación no posee acceso a este buffer. Una vez se completa la transacción, ya sea por que se llenó el buffer o porque se ha recibido un paquete menor a 64 bytes, los datos se transfieren otro buffer, conocido como buffer de lectura. Sólo cuando los datos llegan a este buffer están disponibles para la aplicación, que puede leerlos en la cantidad que estime conveniente.

La figura 3.1 esquematiza el proceso.

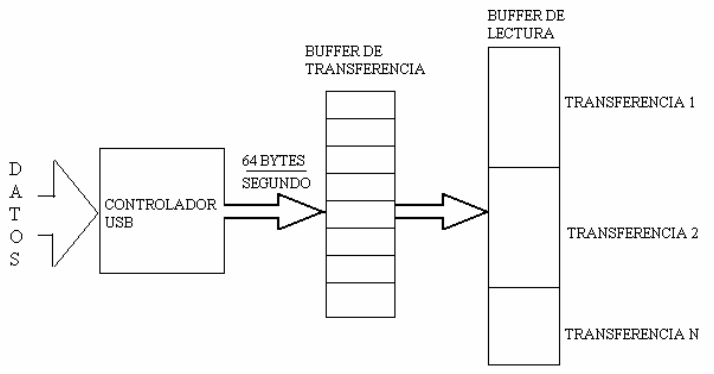


Figura 3.1 Transferencias por el USB

Hay que aclarar que el número de datos disponibles para la aplicación es menor al número de datos transferidos, ya que, como se había mencionado, solo 62 bytes son datos efectivos en cada paquete.

3.1.3 Optimización del flujo de datos

Cuando se tiene un flujo de datos bajo o esporádico, no es necesario determinar las condiciones en que se están transfiriendo los datos hacia la aplicación, pues no existe temor a pérdidas. Pero por los requerimientos de este trabajo, en donde se necesita una transferencia constante de 2.048 Mbps, es necesario tomar en cuenta todos los parámetros posibles para lograr una transmisión de datos eficiente.

Existen varios elementos que influyen dentro de la transmisión de datos. Uno de ellos es el Maximum Transfer Size, la cantidad de datos enviados por transacción. Este influye porque determina cuantos datos serán enviados desde el controlador hacia la PC antes de renegociar otro envío, lo cual toma un tiempo mínimo de 1 ms[2]. En el circuito se reciben 256 bytes por ms, por lo que no puede enviarse menos que esta cantidad por transacción.

El Maximum Transfer Size influye, además, porque determina la cantidad de datos que estarán disponibles para la aplicación por lectura. Cada lectura hecha por la aplicación toma un tiempo mínimo, independiente de la cantidad de datos que se capturen. Por lo tanto, es más eficiente realizar lecturas de bloques grandes de datos.

Otro factor a tomar en cuenta es que en el tipo de transferencia Bulk los datos son enviados en paquetes de 64 bytes. Si la cantidad de datos enviados por el controlador es un número que no es múltiplo de 64, se provocaría la cancelación de la transacción, antes de cumplir con el Maximum Transfer Size. Como el paquete de 64 bytes tiene 62 bytes de información de usuario y 2 bytes reservados, la cantidad de datos que el circuito de interfaz debe enviar tiene que ser múltiplo de 62.

A los elementos ya mencionados debe agregarse el latency timer. Si el valor de este timer es muy reducido, podría enviar datos antes de lo esperado. Para evitar esta situación, el latency timer puede ponerse en el máximo, 255 ms.

Tomando en cuenta todos estos factores, y de pruebas experimentales, se ajustaron las características del circuito y parámetros del USB de la siguiente manera:

- El circuito de interfaz hace descargas al buffer del controlador FT245BM de 372 bytes, máximo número múltiplo de 62 que puede ser colocado en el buffer.

- El envío es provocado a través de la señal de entrada SND, que coloca los datos en la transacción activa.

- Latency timer en 255 ms.

- Maximum Transfer Size igual a 4608 bytes. De estos, el buffer de lectura recibirá 4464, que es un múltiplo de 372 (372*12). Se probaron valores superiores, pero estos provocan retraso debido a que la aplicación se tarda mucho en leerlos.

- La aplicación lee todos los datos presentes en el buffer de lectura, que han de ser como mínimo 4464 bytes en condiciones normales. Las pruebas demostraron que, si el programa es eficiente, la PC es capaz de procesar todos los datos de una transacción antes de la llegada de otra, evitando el rebasamiento del buffer de lectura, cuyo tamaño es de aproximadamente 64kbytes, determinado de forma experimental. Además, manteniendo el buffer en el nivel mínimo, se da robustez a la transferencia de datos frente a retrasos provocados por terceros programas.

3.2 PROGRAMACIÓN MULTITAREA

Típicamente los programas se desarrollan en un solo bloque conteniendo todas las instrucciones, esta forma suele dar resultados satisfactorios en la mayoría de aplicaciones. Pero cuando se necesita adquirir muchos datos y a su vez procesarlos, este tipo de implementación puede resultar problemática, sobre todo si se trata de un programa con muchas líneas de instrucciones. Para superar esta situación se puede distribuir las tareas, con un programa encargado de la adquisición de los datos y otro del procesamiento corriendo de manera simultánea. Esto es posible utilizando multithreading, una capacidad propia de los sistemas operativos multitarea.

El término thread se refiere una secuencia de ejecución de código de programa. Cada programa posee por defecto un thread principal o primario, pero puede poseer más, a los que se denomina secundarios. Multithreading es cuando una aplicación corre más de un thread simultáneamente.

A partir del sistema Windows 95 es posible utilizar multithreading en un programa. El hecho de ejecutar procesos simultáneos es algo relativo, ya que en realidad el sistema operativo esta intercambiando el control entre cada uno de ellos en una forma rápida, dándonos la sensación de un proceso simultáneo. Aún cuando el diseño de los programas se vuelve más complicado, este método crea programas más eficientes en cuanto a ejecución y utilización de recursos.

3.2.1 Implementación de threads

Para Windows 95 y sistemas posteriores, las funciones para la creación de threads están dentro de la Win32 API. Existe además la librería MFC, basada en la Win32 API, que facilita la estructuración de threads. Aunque la MFC se basa en la Win32 API, no se recomienda la mezcla de ambas para implementar threads dentro de un mismo programa. La MFC fue escogida para el desarrollo del programa.

La MFC contiene dos tipos de threads: User Interface Threads (UI Threads) y Worker Threads. Los UI Threads son esenciales en el desarrollo de aplicaciones con interfaz gráfica

que necesitan información de entrada del usuario, por otra parte, los worker threads se utilizan en aplicaciones que contienen tareas de fondo, por ejemplo: lecturas de periféricos, generación de reportes, cálculos secundarios, e incluso procesos paralelos.

Por lo tanto, los worker threads se aproximan más a los requerimientos de la aplicación que se está desarrollando, siendo así, en adelante se explica el modelo de construcción de un worker thread utilizando la MFC y desarrollado en Visual C++ 6.0.

Para utilizar las funciones y clases pertenecientes a la MFC, la librería debe ser incluida en las configuraciones generales del proyecto de Visual C++, ya sea para enlace estático o dinámico (DLL). Además, debe incluirse en el programa la librería stdafx.h.

3.2.1.1 Creación y manejo de threads

Para crear un nuevo thread es necesario seguir los siguientes pasos:

1) Definir el código que ejecutará el worker thread. Para esto se debe crear una función con las instrucciones y variables a utilizar, que debe ajustarse a la estructura siguiente, definida por la MFC:

```
UINT nombre_del_proceso(LPVOID Pparam){
      /*instrucciones
            *
            *
      */
      return 0;
}
```

   Al valor de retorno la función, de tipo UINT, se le conoce como código de salida, que normalmente es cero para indicar una salida normal, pero puede ajustarse cualquier valor que se desee para indicar otros estados, con la única excepción del valor STILL_ACTIVE (0x00000103L).

   Si bien el código de esta función se ejecuta en forma independiente al resto del programa, también puede compartir variables con otras funciones, si estas se declaran como globales.

2) Crear el thread secundario. La MFC maneja los threads a través de objetos de la clase CWinThread. Para crear un thread, asociarlo al objeto que ha de manejarlo, enlazarlo con la función que posee el código que ha de ser ejecutado e inicializar algunos parámetros se utiliza la función AfxBeginThread. El llamado a esta función se hace preferentemente desde la función main(), que es el thread principal de cualquier programa en C++, aunque puede hacerse desde un thread secundario.

   La función AfxBeginThread posee el siguiente prototipo:

```
CWinThread* AfxBeginThread(
                  AFX_THREADPROC pfnThreadProc,
                  LPVOID pParam,
                  int nPriority = THREAD_PRIORITY_NORMAL,
                  UINT nStackSize = 0,
                  DWORD dwCreateFlags = 0,
                  LPSECURITY_ATTRIBUTES lpSecurityAttrs = NULL
                  );
```

El significado de los parámetros de AfxBeginThread se presenta en el cuadro 3.1.

Parámetro	Descripción
pfnThreadProc	Nombre de la función del thread. Esta función debe ser declarada así: UINT nombre_del_thread(LPVOID pParam);
pParam	Parámetro similar al que se encuentra dentro de la función del thread.
nPriority	Prioridad de ejecución del thread, pueden ser varios tipos; los más comunes son: THREAD_PRIORITY_NORMAL, THREAD_PRIORITY_BELOW NORMAL, THREAD_PRIORITY_ABOVE_NORMAL.
nStackSize	Especifica el tamaño en bytes de la pila o STACK, del thread; si toma valor 0, corresponderá al tamaño en memoria del thread.
dwCreateFlags	Esta bandera acepta dos valores: CREATE_SUSPENDED y 0. Si es 0, el thread se activa inmediatamente, de lo contrario se activa solo con la llamada a "ResumeThread"
lpSecurityAttrs	Apunta a la estructura SECURITY_ATTRIBUTES. La cual determina si el objeto que maneja el thread puede ser heredado. Por defecto su valor es NULL.

Cuadro 3.1. Parámetros de la función AfxBeginThread.

Los valores por defecto de los últimos cuatro parámetros se usan comúnmente. En el caso de dwCreateFlags, si se utiliza el valor por defecto 0, el thread comenzará a correr inmediatamente se haga la llamada a AfxBeginThread. En caso contrario, dwCreateFlags = CREATE_SUSPENDED, es necesario llamar a la función ResumeThread(), miembro de la clase CWinThread, en el momento en que se desee iniciar la ejecución.

A continuación se muestra un ejemplo de creación de un thread:

```
//Declarando el puntero PWinThread1
CWinThread *PWinThread1;

//Creando el thread y asociándolo a PWinThread1
PWinThread1 = AfxBeginThread( thread1,
              Pparam,
              THREAD_PRIORITY_ABOVE_NORMAL,
```

```
                0,
                CREATE_SUSPENDED);
// ...

//Iniciando la ejecución del thread
PWinThread1->ResumeThread();
```

Existen otras funciones miembros de CWinThread, con las que puede manejarse un thread una vez este se ha creado. Estas permiten entre otras cosas, suspender la ejecución (SuspendThread), reiniciarla (ResumeThread), obtener el código de salida y cambiar la prioridad (SetThreadPriority)[11].

El uso de estas funciones puede ocasionar problemas. Por defecto, cuando un thread termina de ejecutarse se destruye el puntero CWinThread asociado y, dado que no se sabe con certeza en que momento ocurrirá, puede hacerse un llamado a las funciones cuando el puntero ya ha desaparecido. Para evitar esta situación debe cambiarse el valor del miembro m_bAutoDelete a FALSE, antes de que el thread comience a correr, y al dejar de utilizar el puntero, debe borrarse con la instrucción delete.

La figura 3.2 muestra la estructura típica de un programa que utiliza multithreading.

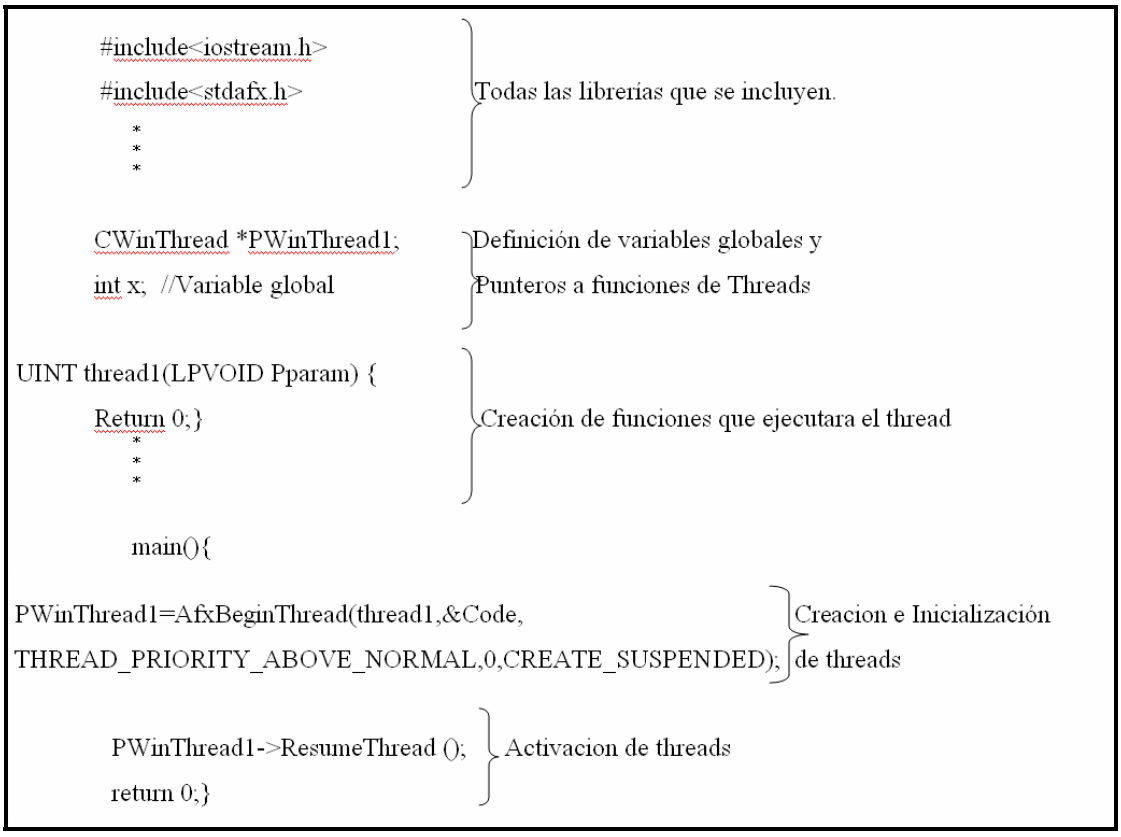


Figura 3.2 Esquema básico de un programa con threads.

3.2.2 Sincronismo de threads

Cuando se implementan programas con multithreading, cada secuencia de ejecución es independiente de las demás. Además, en Windows, el control pasa de un programa a otro sin que se pueda saber cual se está ejecutando en un momento determinado y cual ha de seguir. Esta situación crea un gran problema cuando los threads de un programa comparten recursos, por ejemplo, variables globales. Podría darse el caso de que una variable cambiara de valor, debido a un thread, antes de que sea procesado el valor anterior en otro.

De esta situación surge la necesidad de mecanismos de sincronización de threads. La Win32 API y la MFC proveen varias herramientas para este propósito, entre las que podemos mencionar: eventos, mutex y semáforos. Todas poseen distintas capacidades y características y su uso depende de la necesidad específica de la aplicación. Todos estos métodos posee una herramienta común: las funciones de espera.

En las siguientes secciones se presenta en detalle la información de los métodos de sincronización de threads utilizados en el programa.

3.2.2.1 Funciones de espera

Las funciones de espera permiten detener la ejecución del thread que las llamó, hasta que se cumplan ciertas condiciones. En la Win32 API, las funciones de espera son WaitForSingleObject() y WaitForMultipleObjects(). Estas funciones retornan cuando se han cumplido cualquiera de dos condiciones: los objetos especificados como entrada se encuentran señalizados (signaled state) o se cumple el intervalo de tiempo de espera escogido. La diferencia entre ambas es el numero de objetos que reciben, uno en el caso de WaitForSingleObject y varios para la otra.

El prototipo de WaitForSingleObject es:

```
WaitForSingleObject(
            HANDLE hHandle,
            DWORD dwTiempo
            );
```

hHandle: Es una variable de tipo HANDLE que hace referencia al objeto de sincronización, que puede ser un evento o un mutex.

dwTiempo: Es una variable de tipo DWORD (Double Word) que define el tiempo en milisegundos durante el cual la función estará en estado de espera antes de retornar. Si dwTiempo = INFINITE, la función esperará por tiempo indefinido.

Las funciones esperan mientras el objeto que ha sido pasado como parámetro se encuentra “non signaled”, consumiendo muy poco tiempo de procesador. Una vez el objeto se

encuentra “signaled”, permiten que se continúe la ejecución del thread. Antes de retornar, la función modifica el estado de algunos objetos de sincronización al estado “non signaled”. Pueden usarse también como temporizadores.

3.2.2.2 Mutex

El término mutex se refiere a la expresión “mutual exclusion”, y se refiere a un objeto de sincronización que hace que un recurso compartido sea accesible por un solo thread a la vez. Cuando un thread específico tiene el permiso de acceder al recurso sincronizado, se dice que el thread es propietario del mutex y que este último se encuentra en estado “non signaled”. Una vez el thread libera el recurso, y si no hay ningún otro utilizándolo, el mutex no tiene propietario y se encuentra en estado “signaled”.

Para utilizar los mutex, primero debe crearse uno utilizando la función ::CreateMutex de la Win32 API, cuyo prototipo es:

```
HANDLE ::CreateMutex(
                  LP_SECURITY_ATRIBUTES lpMutexAttributes,
                  BOOL bInitialOwner,
                  LPCTSTR lpName
                  );
```

lpMutexAttributes: Determina si el objeto puede ser heredado por un proceso hijo. Si el valor es NULL el handle no puede heredarse, lo cual es el caso común.

bInitialOwner: Especifica el propietario inicial del mutex. Si su valor es TRUE, el thread que llamó a la función tiene la propiedad del mutex, y está, por lo tanto, “non signaled”. Si su valor es FALSE, no posee propietario y el mutex está “signaled”.

lpName: Puntero a un string terminado en NULL, donde se especifica el nombre del mutex. Si su valor es NULL, el mutex es creado sin nombre.

Valor de retorno: Una variable de tipo HANDLE que hace referencia al objeto mutex que se crea. Esta se utiliza para referencia en las funciones de espera.

Para proteger un recurso compartido, una variable por ejemplo, lo que se hace es utilizar una función de espera al inicio de los bloques de código de los thread donde se accede o modifica el recurso. Si el mutex está “signaled”, la función de espera de alguno de los thread continúa y cambia inmediatamente el estado del mutex a “non signaled”, deteniendo la ejecución de los demás threads cuando estos alcancen las respectivas funciones de espera, es decir, toma la propiedad del mutex.

Una vez se ha procesado la información, se libera la propiedad del mutex utilizando la función ::ReleaseMutex(), que recibe como único parámetro el handle del mutex que se

desea liberar. Al hacer esto, el mutex vuelve a estar "signaled" y otro thread que se encuentre esperando puede hacerse con la propiedad.

El siguiente ejemplo comprende un programa aplicado a la lectura de un puerto con dos threads, e ilustra las ideas que se mencionan anteriormente:

```
HANDLE mlectura  //variable global tipo HANDLE

//Se crea el mutex dentro del main()
mlectura = ::CreateMutex(NULL, FALSE, NULL);

//Instrucciones dentro de un thread que leen desde un puerto de la
//dirección 378h y almacenan los datos en un arreglo
WaitForSingleObject(mlectura, NULL);
for(i=0; i<=2000; i++){
   data_recibido[i]=_inp(0x378);
}
ReleaseMutex(mlectura);

//Instrucciones dentro de un Thread que copian los datos
//almacenados en el arreglo anterior a otro arreglo donde serán
procesados.
WaitForSingleObject(mlectura, NULL);
for(j=0; j<i; j++){
   dato_para_proceso[j]= data_recibido[j];
}
ReleaseMutex(mlectura);
{
//Cualquier proceso
}
```

En el primer thread, los datos de una dirección están siendo almacenados en un arreglo, al mismo tiempo hay otro thread que los copia a otro arreglo para su posterior proceso; a pesar de que cada thread es independiente, la lectura del puerto y la copia de los datos a otro arreglo están encerrados entre la función de espera WaitForSingleObject(mlectura,NULL) y ReleaseMutex(mlectura), de esta manera se asegura que ninguno de los dos procesos accedan al mismo arreglo simultáneamente, ya que sus accesos son restringidos a través del mutex común que comparten.

3.2.2.3 Eventos

La palabra evento se usa mucho en la programación, por ejemplo para denominar una acción del usuario (GUI) o una señal de alerta proveniente del hardware. Aquí, cuando se habla de un evento se refiere a un objeto de sincronización cuyo estado puede ser explícitamente puesto a "signaled", a través del uso de ciertas funciones. Esto sirve en el caso de que quiera comunicarse la ocurrencia de alguna situación, de la cual dependa el inicio o continuación de procesos en otros threads, que posean funciones de espera enlazadas al objeto evento.

Para crear un objeto evento, se utiliza la función CreateEvent, cuyo prototipo es:

```
HANDLE CreateEvent(
                LPSECURITY_ATTRIBUTES lpEventAttributes,
                BOOL bManualReset,
                BOOL bInitialState,
                LPCTSTR lpName
                );
```

lpEventAttributes: Determina si el handle al evento puede ser heredado por procesos hijos. Si su valor es NULL, el handle no puede ser heredado.

bManualReset: Esta es una variable booleana que nos indica si el evento vuelve al estado non signaled de forma manual o automática. Un valor "true" indica que el "reset" es manual, y un valor FALSE indica reset automático.

bInitialState: Indica el estado inicial del evento. Si su valor es "true", el evento está en "signaled" inicialmente, de lo contrario estará "non signaled".

Valor de retorno: Una variable de tipo HANDLE que apunta al objeto evento, y a través de la cual se hace referencia en las funciones.

El estado del objeto evento se pone en estado signaled a través de la función SetEvent(HANDLE). Además, un objeto puede ser de reset manual o automático. En la forma manual, se debe hacer una llamada explicita a la función ResetEvent(HANDLE) para que este vuelva al estado non signaled, en cambio, cuando es automático, la función de espera se encarga del reset en el momento en que esta retorna.

En el siguiente ejemplo se tiene un thread en el que se leen datos desde un puerto, y cuando se tiene cierta cantidad de datos recibidos, otro thread empieza a procesar estos datos.

```
HANDLE heventolectura  //variable global tipo HANDLE
//Creamos el evento dentro del main()
heventolectura = ::CreateEvent(NULL, FALSE, FALSE, NULL);

//Instrucciones dentro del thread que lee desde un puerto de la
dirección 379h y almacena los datos en un arreglo //
for(i=0; i<=2000; i++)
   data_recibido[i]=_inp(0x379);
if(i>=500)
   SetEvent(heventolectura);

//Instrucciones en el Thread que inicia un proceso cualquiera
cuando se haya leído 500 datos.
WaitForSingleObject(heventolectura, NULL);
/*{PROCESO}*/
```

Los mutex y los eventos, son los objetos de sincronización que se utilizan para el desarrollo del programa. Al inicio su manejo es problemático y complicado, pero constituyen herramientas sin las cuales la programación con multithreading se haría casi imposible.

3.3 PROGRAMA DE DECODIFICACIÓN DE SEÑALIZACIÓN R2 DIGITAL DE LÍNEA

Como se ha mencionado anteriormente, el programa que se desarrolla tiene como fin capturar los datos provenientes del circuito de interfaz, decodificar la información de señalización R2 para los 30 canales de voz, y luego presentarla al usuario en forma de archivos de texto, para todos los canales, y salida a pantalla, para un canal.

Para lograr su objetivo, el programa debe realizar varios procesos:

- Lectura de datos
- Alineación de trama básica
- Alineación de multitrama de señalización
- Decodificación de señalización R2 digital de línea
- Salida hacia archivos y pantalla

La implementación de estos procesos ha requerido el uso de varias herramientas, como las librerías FTD2XX del USB y librerías de programación con multithreading presentadas con anterioridad en este capítulo, y se basa las recomendaciones y especificaciones del capítulo I.

Aunque se trabaja en un sistema operativo de ventanas, el programa se ha desarrollado para utilizarse en línea de comando, es decir, no posee una interfaz gráfica, además recibe como únicos datos de entrada la duración que ha de tener la prueba y el canal que ha de desplegarse en pantalla.

En las siguientes secciones se muestra la estructura general del programa y luego el detalle de las partes que lo componen.

3.3.1 Estructura general del programa

En este programa la eficiencia es un factor muy importante, debido a que no hay control sobre el flujo de datos de 2 Mbps, pues en caso de generar mucho retraso se provocaría una pérdida de datos. Existen algunas funciones más delicadas que otras, y por esto es mejor aislarlas lo más posible. Una de esas actividades es la lectura de los datos del buffer de lectura del USB. El acceso a este buffer es lento, y la existencia de gran cantidad de cálculos intermedios entre las lecturas provocaría grandes retrasos. Existen además otras funciones que no poseen mayor prioridad y que toman mucho tiempo, por ejemplo la escritura a disco y la salida a pantalla.

Con estas consideraciones en mente, las distintas actividades que realiza el programa se han distribuido en varios threads, de la siguiente manera:

1. *Thread principal*: Se encarga de inicializar los thread, eventos y mutex globales, configura la interfaz y parámetros USB, recibe los datos del usuario y se maneja el temporizador de la prueba.

2. *Thread de lectura del buffer USB*: Saca los datos del buffer de lectura USB y los envía al thread de alineación de trama básica.

3. *Thread de alineación de trama básica:* Se encarga de procesar la información del TS0 e implementar los procesos mencionados en la recomendación G.706[8], para trama básica sin VRC.

4. *Thread de multitrama y de distribución de información de señalización:* Recibe los datos del thread de trama básica, extrae el TS16 para alinear la multitrama y luego envía los bits ABCD de señalización para que sean procesados por los threads de decodificación.

5. *Threads de decodificación de señalización R2 de línea y presentación:* Son los encargados de la decodificación, escritura de archivos y envío a pantalla. Son 30 threads en total, uno por canal de voz.

La comunicación de los datos entre threads se realiza a través de arreglos unidimensionales de números de ocho bits (OCTETO o unsigned char), que se han denominado también buffers, y que se sincronizan utilizando eventos y mutex. Existen también banderas, generalmente variables booleanas globales, que sirven para la comunicación entre threads.

Debe mencionarse que la totalidad de datos provenientes de la línea E1 llegan únicamente hasta el thread de multitrama, de ahí en adelante se utilizan solo los bits de señalización y algunos bits adicionales y la información de canales de voz desaparece.

3.3.2 Thread principal

Como ya se mencionó, este thread se encarga de la creación de variables y objetos globales, de inicializar los threads secundarios, recibir datos del usuario y de temporizar la prueba. El flujograma correspondiente al thread principal se muestra en la figura 3.3.

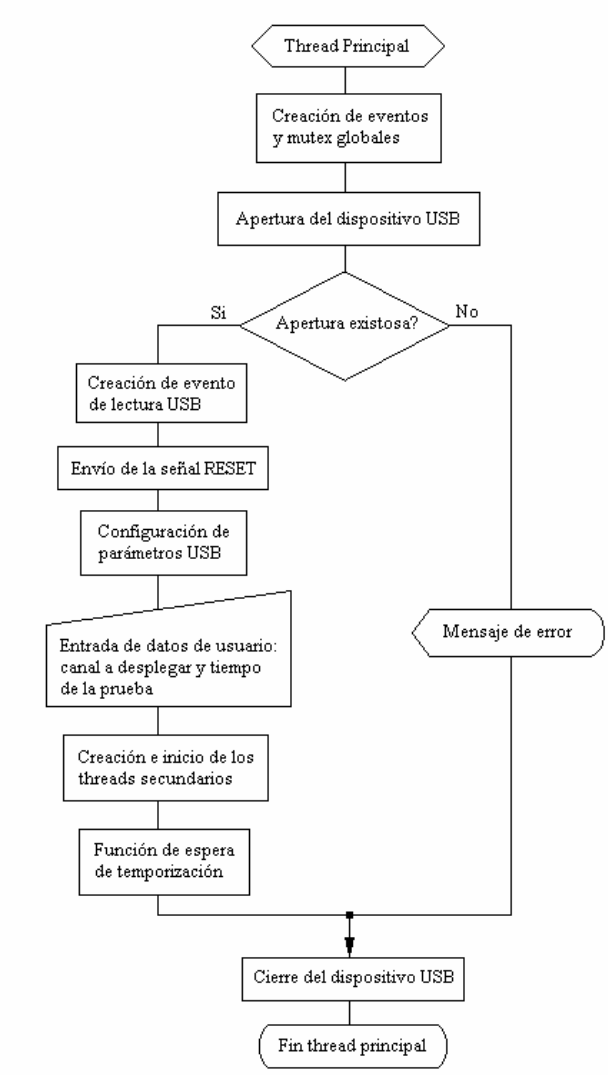


Figura 3.3 Flujograma del thread principal.

Como lo muestra la figura 3.3, a parte de inicializar algunas variables locales, el primer procedimiento es crear eventos y mutex globales. En el caso de los eventos, estos se crean utilizando la función CreateEvent(). Los eventos creados en este momento son:

- hEvent1: Sirve para notificar la escritura de datos en el buffer intermedio entre el thread de lectura y el de trama básica (buffer[]). Este evento tiene las siguientes características: reset manual, estado inicial non signaled, handle global y no posee nombre.

- hEvent2: Evento que sirve únicamente como parámetro de la función de espera que temporiza la ejecución del programa. Características: Reset automático, estado inicial non signaled, sin nombre.

- hEvFinEspera: Evento que notifica el fin del tiempo de espera de búsqueda de FAS. Su uso se explica en secciones posteriores. Características: Reset automático, estado inicial non signaled, sin nombre.

- hEvInicioEspera: Evento que comunica el comunica el inicio de espera de búsqueda de FAS. Como el anterior, se explica con más detalles después. Características: Reset automático, estado inicial non signaled, sin nombre.

En el caso de los mutex, la función que se usa para inicializarlos es CreateMutex(). Los mutex creados son:

- hmutex: Mutex que sincroniza el acceso al buffer intermedio entre los threads de trama básica y multitrama. Características: Sin propietario inicial.

- mLoF: Protege la bandera de notificación de pérdida de alineación de multitrama (bool LoF). Sin propietario inicial.

- mTimer: Protege la bandera bool TiempoTerminado que detiene la lectura de datos del USB. Sin propietario inicial.

El siguiente procedimiento consiste en la apertura del dispositivo USB y asignación del control al handle hInterfaz. Esto se hace con la función FT_Open() utilizando el número de dispositivo 0, asumiendo que solo la interfaz está conectada al host USB, como se ha recomendado. Esta función retorna un valor de tipo FT_STATUS []. Si el valor de retorno es igual a FT_OK, quiere decir que el dispositivo ha sido inicializado sin ningún problema (apertura exitosa). Si el valor difiere, la interfaz USB tiene algún problema.

El valor de retorno se prueba luego de la apertura del dispositivo, y en caso de presentar un problema el programa envía una notificación de error a pantalla y luego termina.

Si la inicialización del dispositivo USB es exitosa, el siguiente paso consiste en crear el evento hEvent. Debe aclararse que este evento no es para sincronía ente threads, si no para notificar un evento de hardware USB, como es la llegada de datos al buffer de lectura. Luego de crearlo, es necesario configurar el evento de hardware al que ha de responder, a través de la función FT_SetEventNotification(), y pasándole a esta como parámetro la máscara FT_EVENT_RXCHAR. El reset del evento es automático.

Después se crea otro objeto evento adicional, hescribirtrama, que sirve para notificar la escritura de datos en el buffer intermedio entre el thread de trama básica y de multitrama (arreglo trama[]), con reset manual y estado inicial non signaled. Se crea también el mutex mtrama, que sincroniza el acceso al buffer antes mencionado.

Seguidamente se hace el envío al circuito de interfaz de la señal RESET utilizando la función FT_Write(), lo que provoca la suspensión del envío de datos hacia la computadora.

Cuando se han dejado de recibir datos, se procede a configurar los parámetros USB necesarios para asegurar el eficiencia en el flujo de datos. Los parámetros son:

- Latency timer a 255 ms, utilizando la función FT_SetLatencyTimer().
- Maximum Transfer Size en 4608 bytes, utilizando la función FT_SetUSBParameters().

Una vez realizado el cambio de parámetros, se pide al usuario que introduzca el número de canal a desplegar en pantalla, que pueden ser del 1 al 30, y la duración de la prueba en segundos. El número de canal a desplegar es notificado a los threads de canal utilizando la función Canal::Desplegar(). Los detalles de estos procedimiento se darán más adelante.

A continuación se crean los threads secundarios, utilizando la función AfxBeginThread(). En todos los casos se ha suspendido la ejecución, se da prioridad arriba de lo normal y se deja la pila (stack) con el tamaño que el thread tiene en memoria. Además, todos los objetos CWinThread son globales. Los threads creados son:

- PWinThread1: Thread de lectura. La función que contiene el código a ejecutar es tescritura.
- PWinThread2: Thread de trama básica. Función de thread: tTramaBasica.
- PWinThread3: Thread de espera de FAS. Función de thread: ttimer. Este thread se explica más adelante.
- PThreadCanal[30]: Los threads de decodificación, para estos se utiliza un arreglo de objetos CWinThread. Las funciones correspondientes son: tcanal1 para PThreadCanal[0], tcanal2 para PThreadCanal[1], y así sucesivamente.
- Pseparador: Thread de multitrama. Función: tMultiTrama.

Luego se inicia la ejecución llamando a la función ResumeThread() de cada objeto CWinThread en el siguiente orden: Thread de lectura, thread de trama básica, thread de multitrama y los de decodificación.

Una vez se tiene todos los threads corriendo, se llama a la función de espera WaitForSingleObject() sirve de temporizador del programa. Como parámetros se pasan hEvent2 y el tiempo que el usuario introdujo, convertido a milisegundos. Ya que hEvent2 nunca se pone en estado signaled, el retorno de la función depende exclusivamente del timeout.

Una vez termina el tiempo de la prueba, se envía una señal de RESET al circuito de interfaz y se cierra el handle del dispositivo USB usando FT_Close(), con lo que termina el programa. Antes de terminar, el thread principal envía automáticamente la señal de suspensión a todos los secundarios y luego los elimina de memoria.

3.3.3 Thread de lectura del buffer USB

Este proceso interactúa directamente con los drivers del controlador FTD2XX y las funciones que se proveen al usuario para interactuar con el chip controlador y el buffer de transferencia.

La estructura del thread de lectura del USB se muestra en la figura 3.4.

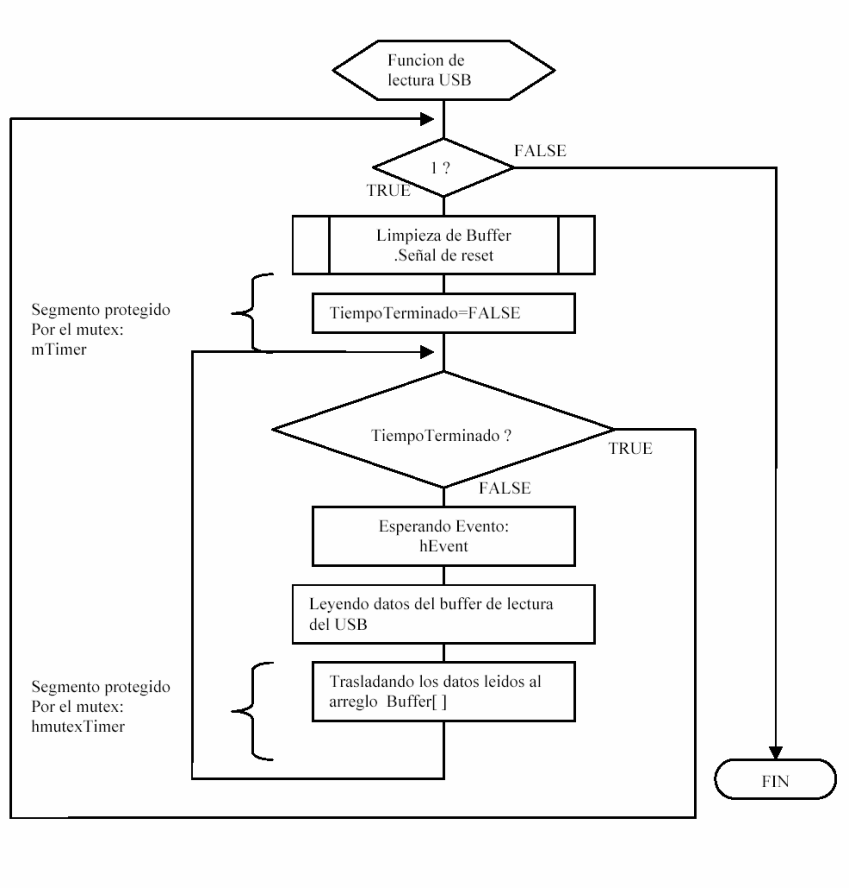


Figura 3.4. Thread de lectura del USB.

Previo a la lectura de datos en si, el circuito de interfaz es reinicializado con la función FT_Write(). La función LimpiarBuffer() limpia el buffer de transmisión y el buffer de lectura. El circuito estará listo para transmitir al limpiar el buffer de recepción con la función FT_Purge. A partir de este momento los datos que se lean provenientes de la interfaz de línea serán válidos.

Antes de iniciar la lectura es necesario modificar la bandera TiempoTerminado = FALSE, esto reiniciaría el proceso de alineación en caso de que se hubiese tenido una lectura de FAS errónea. Si esto hubiere ocurrido se habría asignado el valor TRUE a la misma bandera pero en otro thread llamado ttimer. Este segmento está protegido por el mutex

mTimer. El thread ttimer será explicado más detalladamente cuando se hable del proceso de alineación básica.

Mientras la bandera TiempoTerminado permanezca en el valor FALSE, es posible realizar la lectura de datos sin problemas, esto se hace adentro de un ciclo repetitivo del cual solo hay salida en caso de que cambie el valor de esta bandera.

Dentro de este ciclo repetitivo se utiliza la función de espera WaitForSingleObject(), ya que el driver provee una función que hace posible asignar un evento que se activa únicamente mientras exista un dato en el buffer de lectura. Una vez activado el evento se utiliza la función FT_GetStatus(), la cual retorna la causa que notificó el evento a través de la variable EventDWord, la causa se determina a través del enmascaramiento entre el valor de EventDWord y la mascara FT_EVENT_RXCHAR. Si los valores coinciden se realiza la lectura de los datos, de lo contrario simplemente se retorna a la función de espera.

Al efectuar la lectura primero se reasigna el tamaño del arreglo dynamicBuffer con un tamaño que coincide con la cantidad de bytes que se encuentran en el buffer de lectura, y es en dicho arreglo donde se escribirán los datos provenientes del buffer de lectura, los datos se leen con la función FT_Read( ).

El ultimo segmento copia los datos recibidos en el arreglo dynamicBuffer hacia el arreglo buffer[] para que puedan ser utilizados por otros threads sin retrazar la lectura de datos. Este proceso esta protegido por el mutex hmutex. La variable global tbuffer aumenta conteniendo la cantidad de datos que se han copiado en el arreglo buffer o se hace cero cuando otro thread a leído los datos de el arreglo buffer. En caso de activarse la bandera TiempoTerminado este proceso se repite desde el inicio.

3.3.4 Thread de alineación de trama básica

El proceso de alineación de trama básica está fundamentado en la implementación de la norma G.706 [8] donde se describen las secuencias que determinan la alineación de la trama, o el momento en que se declara la trama como desalineada.

La norma explica que el sistema está alineado una vez sean correctas las lecturas de la secuencia FAS-NFAS-FAS. Así mismo explica que se declara perdida la alineación al tener tres lecturas erróneas seguidas de esta secuencia. En el caso de perderse o no encontrar alineación la norma indica que se debe continuar la secuencia de inmediato hasta establecer la alineación. La figura 3.5 ilustra dicho proceso.

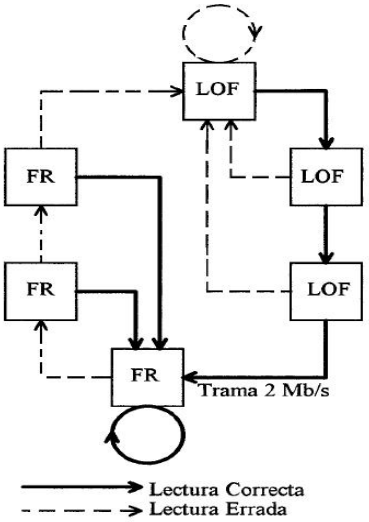


Figura 3.5. Secuencia de alineación de trama básica.

El thread de alineación de trama básica implementa esta secuencia. En el caso de perderse la alineación, la búsqueda se reinicia inmediatamente sin activar un reset en el circuito de interfaz. Esto debido a que en el buffer de lectura pueden encontrarse datos validos con los cuales se podría reestablecer nuevamente la alineación; de haber activado el reset al circuito de interfaz estos datos se hubieran eliminado innecesariamente. A parte esta el hecho que la mayoría de perdidas de alineaciones se originan por espurias y no por una sincronización errónea. Sin embargo se ha creado un thread llamado ttimer, cuya función es inicializar un conteo mientras no exista alineación o cuando se ha perdido. Si este conteo alcanza el valor en milisegundos definido por el #DEFINE ESPERA_FAS que en este caso es de 1 segundo, entonces activará la bandera TiempoTerminado =TRUE, lo cual como se ha visto en el thread descrito anteriormente inicializara todo el proceso de lectura activando un reset en el circuito de interfaz.

El proceso que ejecuta el thread de alineación básica resume en el flujograma de la figura 3.6.

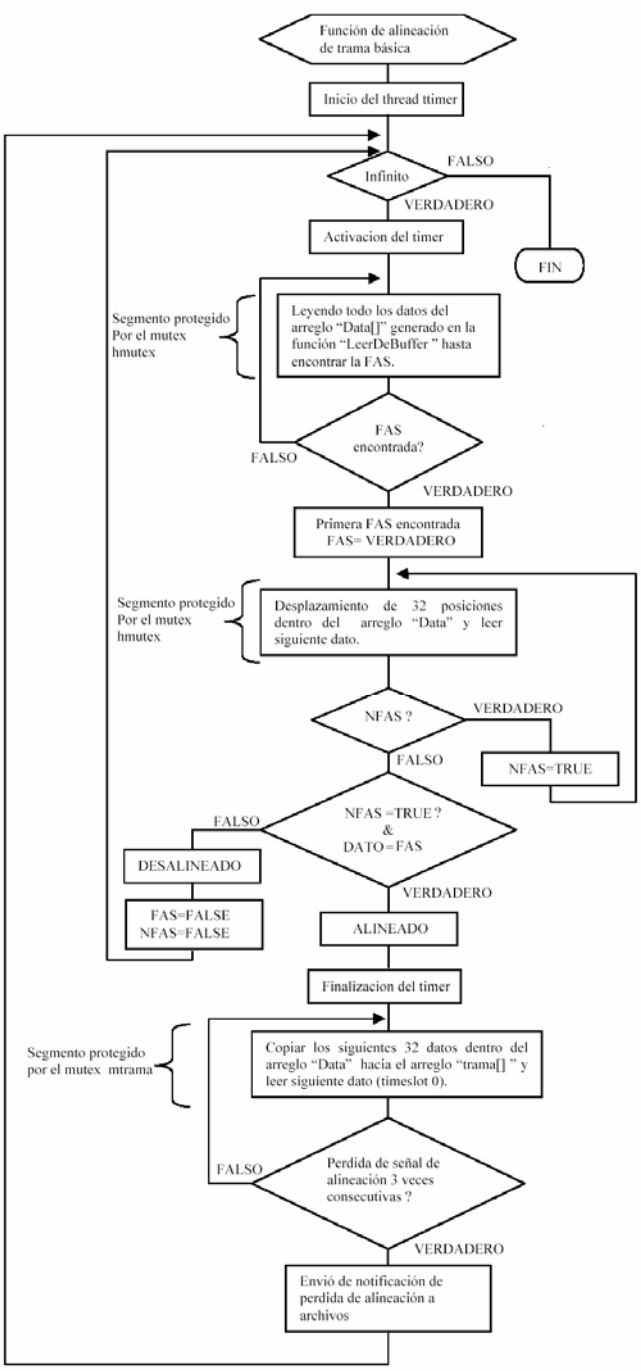


Figura 3.6. Flujograma del thread de alineación de trama básica

El thread al implementar este algoritmo como primer paso inicializa las variables locales para los procesos que se van a realizar siendo la variable local i el índice de los datos accedidos dentro del arreglo que estemos leyendo. El thread ttimer es habilitado justo antes de entrar en un lazo infinito, el conteo del ttimer es activado con la función SetEvent (hEvInicioEspera).

Seguidamente dentro de una estructura do-while se leen los datos que provienen del thread de lectura del USB; a través de la función LeerDeBuffer(), estos datos retornan en un arreglo llamado data[]. En esta misma estructura los datos accedidos en dicho arreglo son comparados con la mascara de la FAS. La estructura do-while continua ejecutándose hasta encontrar un valor que coincida con la mascara.

Al tener un valor que coincide con la FAS, se entra en una estructura while donde se efectúan lecturas también con la función LeerDeBuffer(), seguidamente se realiza una serie de comparaciones con desplazamientos 32 posiciones en el índice del arreglo data [] (esto corresponde a los 32 time slots de la trama) con el fin de encontrar la siguiente secuencia FAS-NFAS-FAS (tomando en cuenta la FAS ya encontrada). Esta secuencia solo se interrumpe al haber una lectura incorrecta que no corresponde, o finalizará al haber sido completada sin errores. El resultado se refleja en la bandera alineado, si esta es FALSE es porque o se pudo completar la alineación y regresamos al inicio del lazo infinito, de lo contrario la alineación se completo y se continua con el proceso. Si la bandera alineado=TRUE se activa el evento que detiene el conteo del ttimer con la función SetEvent(hEvFinEspera), puesto que el sistema ya está alineado.

El siguiente bloque de código nuevamente está dentro de un lazo while. Aquí los datos del arreglo data[] son enviados en tamaños correspondientes a la trama (paquetes de 32 bytes) al arreglo llamado trama[] para ser utilizados por el thread de alineación de multitrama. La variable global ctrama contiene la cantidad de datos que han sido transferidos en este proceso, este segmento a su vez está protegido por el mutex mtrama. Una vez transferidos los 32 datos hacia el arreglo trama[] se realizan comparaciones correspondientes a lo que seria el time slot 0, las cuales son la MASCARA_FAS o la MASCARA_NFAS, alternándose según corresponda.

El control de las comparaciones lo maneja la bandera nfas, que esta cambiando entre el valor TRUE y FALSE por cada comparación. Este proceso de lectura/escritura y comparación se esta realizando de forma permanente. En caso de errores en la comparación de las mascaras, se incrementan las variables cnfas y cfas. Si en cualquiera de ellas hubiera tres errores consecutivos, se rompe el segmento anidado por el ultimo while y la bandera LoF=TRUE. Esta situación reinicia todo el proceso de alineación incluyendo la activación del evento del ttimer.

3.3.5 Thread de multitrama y distribución de información de señalización

Este proceso es parecido a la alineación de trama básica, aunque es necesario aclarar que las normas de la UIT no especifican una secuencia para alinearse o desalinearse tal como es el caso de la alineación de trama básica. Sin embargo la norma G.704 proporciona la estructura de multitrama, y esto incluye una palabra de alineación la cual es 0x0. Este proceso es cíclico ya que se está revisando permanentemente la alineación de multitrama y la escritura hacia cada canal. Se considera que la multitrama se ha perdido solo si tenemos 2 lecturas erróneas consecutivas. La figura 3.7 amplía esta explicación.

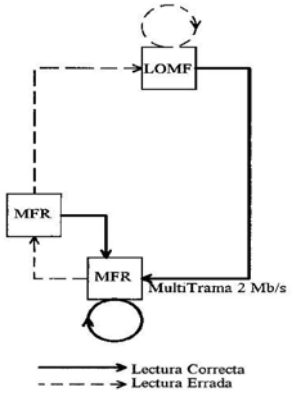


Figura 3.7. Proceso de alineación de multitrama.

El thread implementa este proceso basado en los procesos y resultaron que se llevaron a cabo en el thread de alineación básica. La figura 3.8 muestra el proceso que ejecuta el thread en su correspondiente flujograma.

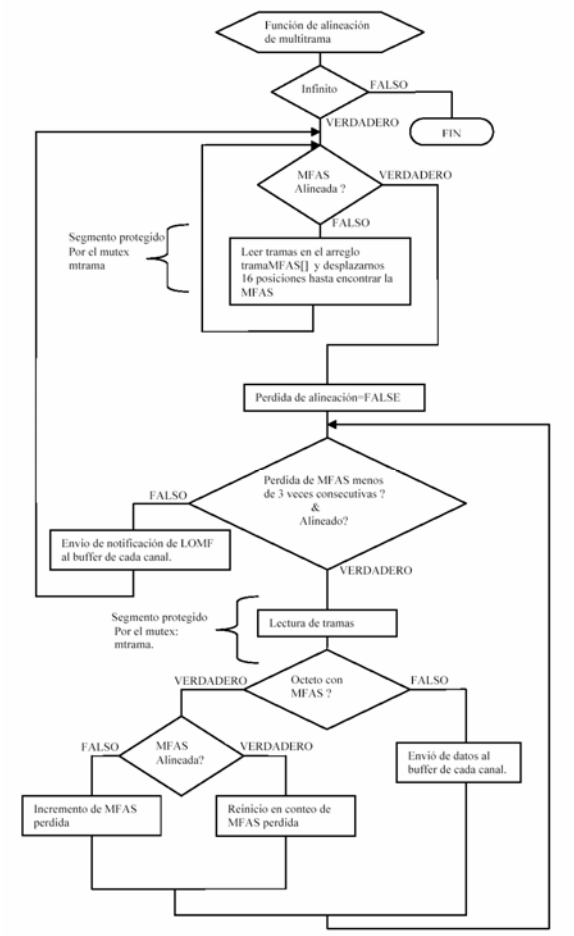


Figura 3.8. Flujograma del thread de alineación de multitrama.

Nuevamente el proceso comienza con la inicialización de las variables locales que serán utilizadas como parte de los procesos, esto previo a formar parte de un lazo anidado permanente.

Inicialmente el proceso forma parte de una estructura while. Aquí se realizan lecturas de datos por medio de la función LeerTramaMFAS() la cual retorna los datos que provienen del thread de alineación básica, dentro de otro arreglo llamado tramaMFAS[ ]. En esta estructura por medio de un lazo for se realizan lecturas en bloques de 32 bytes (en el mismo orden que se hizo en el thread de alineación básica) y dentro de estos bloques hacemos desplazamientos de 16 bytes con la variable i, lo que nos ubicaría en el canal 16. Este proceso se realiza con el fin de poder hacer comparaciones entre los datos leídos y la mascara MASCARA_MFAS. Al coincidir un dato con la mascara, la bandera mfasalineada = TRUE, eso finaliza este lazo anidado de la estructura while.

Al estar en la condición de multitrama alineada se procede a inicializar la bandera LoF=FALSE, teniendo en cuenta que esta bandera podría ser activada por otros threads en caso de que ocurriera pérdida de alineación básica.

El siguiente bloque de código también esta anidado dentro de una estructura while, aquí se realizan lecturas de datos, comprobación si hay perdida de MFAS y envió de datos a funciones miembro de los objetos de la clase canal. La lectura utiliza nuevamente la función LeerTramaMFAS(), con desplazamientos en el índice i del arreglo tramaMFAS[] de 32 posiciones. Cada vez que se realiza una lectura se incrementa la variable local indiceTrama; la cual corresponde a cada uno de los valores de la multitrama, por lo que su valor es de 0 a 15, reiniciándose cuando el incremento supera estos valores.

Si el valor de indiceTrama es 0, se hace un desplazamiento en el arreglo tramaMFAS[] de 16 posiciones desde el índice y el dato se compara con la mascara MASCARA_MFAS, si los valores no coinciden se incrementa la variable cmfas, de lo contrario cmfas es igual a 0. Si el índice de trama es diferente de 0 se hace el mismo desplazamiento dentro del arreglo, pero esta ves el dato leído es separado en su parte alta y baja para ser enviados al correspondiente arreglo del objeto de canal a través de la función Canal:: EscribiraBuffer(). Estos datos formarán parte del proceso de decodificación de línea que realiza otro thread.

El proceso saldrá de este lazo si se detecta perdida de alineación básica a través de la bandera LoF=TRUE o perdida de alineación de multitrama cuando la variable cmfas=3. Si una de estas situaciones ocurre, se envía una notificación a los objetos de Canal a través de la función Canal:: NotificarLoMF o Canal:: NotificarLoF y se reinicia el proceso dentro del thread.

3.3.6 Threads de decodificación de señalización R2 de línea y presentación

Estos threads se encargan de recibir los bits de señalización provenientes del thread de multitrama, decodificarlos y presentarlos al usuario. Para cumplir con esta función, se han creado 30 threads, uno por canal de voz, agrupados en el arreglo PThreadCanal[9].

Los procesos de decodificación se basan principalmente en las recomendaciones Q.421[] y Q.422[10] de la UIT-T, aunque poseen limitaciones, debido a que para determinar los estados de un circuito es necesario conocer ambos sentidos de transmisión. Los estados presentados al usuario utilizando el programa son a veces aproximaciones o simulaciones en aquellos estados en los que el código binario de un sentido no cambia, por ejemplo, en los estados de liberación (Released) y reposo (Idle) que poseen el mismo código (a=1, b=0).

Debido a que todos los threads de decodificación realizan exactamente los mismos procesos solo que en datos distintos, estos fueron agrupados en una clase llamada Canal, y a partir de esta se crea un arreglo de 30 objetos, llamado CanalesSen[], para procesar los datos de cada canal en cada thread.

La clase Canal posee la siguiente estructura:

```
class Canal{

private:
      HANDLE mCanal_;
      HANDLE hEvCanal_;
      OCTETO buffCanal[TCANAL];
      OCTETO buffTrabajo[TCANAL];

      char canalEnUso;
      long int tamanoBuffCanal;
      long int tamanoBuffTrabajo;
      bool desplegado;
      enum{
            SIN_SENTIDO,
            ADELANTE,
            ATRAS
      };

      static HANDLE mCuentaDesp_;
      static int cuentaDespliegue;

      CString nombreArch;
      fstream archivoCanal;

      void LeerElBuffer();
      void MandarAArchivo(OCTETO dato);


public:
```

```
        Canal(int index);
        ~Canal();
        void EscribirABuffer(OCTETO valor);
        void IniciarLectura();
        void EscribirArchLoF();
        void EscribirArchLoMF();
        void ResetDatos();
        void NotificarLoF();
        void NotificarLoMF();
        CString HexABin(OCTETO num);
        void Desplegar();

};
```

Los datos provenientes del thread de multitrama se guardan en el arreglo buffCanal[]. El arreglo es un miembro privado de la clase, por lo que otros threads pueden acceder a él únicamente a través de funciones miembros, específicamente de EscribirABuffer(), que escribe un byte (OCTETO) a la vez. Esto facilita la escritura de los otros threads, ya que solo debe hacerse un llamado a la función, y esta se encarga internamente de la sincronización (mutex mCanal_) y de la notificación de escritura (evento hEvCanal_).

Los datos recibidos son transferidos al arreglo buffTrabajo[] cuando quieren ser procesados, y esto a través de la función LeerDeBuffer(), que es miembro privada de la clase, ya que el objeto se encarga de los datos de ahí en adelante.

La clase posee entre sus miembros el objeto fstream archivoCanal, a través del cual se maneja el archivo de texto que cada uno debe generar. El nombre del archivo de texto de salida se genera en el constructor de la clase utilizando el objeto CString nombreArch, y se basa en el valor numérico de char canalEnUso, que es pasada como argumento al constructor. Se espera, por supuesto, que el valor de canalEnUso se corresponda con el canal que se está procesando.

Los archivos generados por los objetos presentan la siguiente información en cada línea: Hora actual en formato hh:mm:ss.ms, valor binario del código en el orden ABCD y el código o estado del circuito que el valor binario representa.

Por tratarse de un programa de consola, sin interfaz gráfica, se prefirió que en el despliegue a pantalla se presenta un solo canal. Debido a esta característica, la salida a pantalla se controla mediante una variable estática llamada cuentaDespliegue. Cuando el usuario escoge el canal a desplegar y lo notifica a través de la función miembro pública Desplegar(), esta incrementa el valor de cuentaDespliegue, que inicialmente está a cero, y luego cambia el valor de la variable bool desplegado del objeto a true. Si luego de esto otro objeto llama a Desplegar() y la variable cuentaDespliegue es mayor a cero, este objeto ignorará la orden de despliegue a pantalla, y mantendrá el valor de desplegado en false. Cada objeto revisa si el valor de su variable desplegado es true para mandar datos a pantalla.

Una pérdida de alineación de trama o multitrama se notifica utilizando NotificarLoF() o NotificarLoMF(), respectivamente. Estas funciones envían un byte al buffer buffCanal[],

cuyos cuatro bits más significativos poseen un valor distinto de cero para ser distinguidos de los datos normales (que solo utilizan los 4 bits menos significativos), siendo estos CODIGO_LOF (0x1) y CODIGO_MLOF (0x2).

La función que se encarga de la decodificación de los datos de señalización es IniciarLectura(). Cada objeto del arreglo CanalesSen[] llama a esta función desde el thread que le corresponde para iniciar la decodificación de los datos que le son enviados desde el thread de multitrama. La figura 3.9 muestra la operación de IniciarLectura().

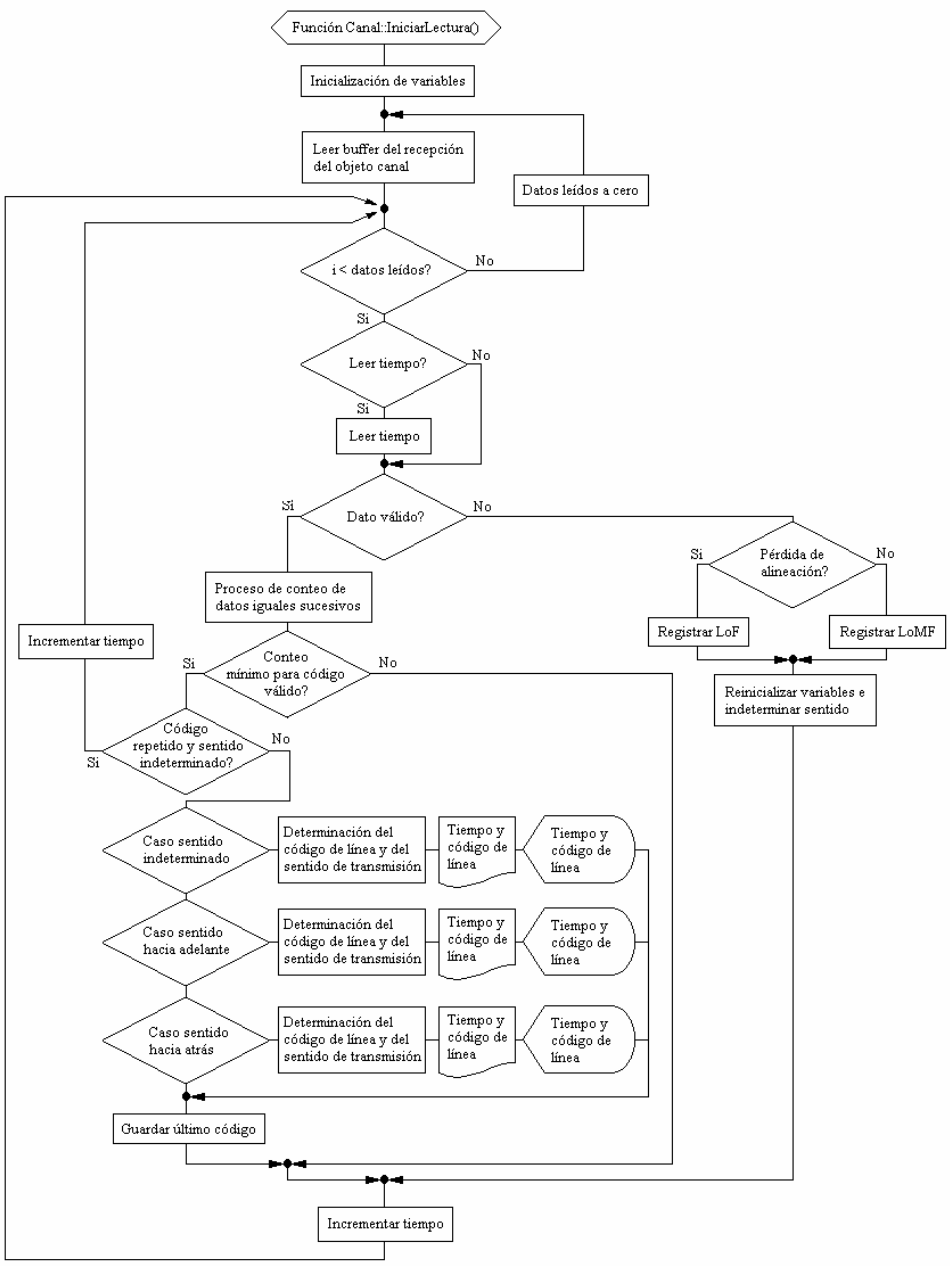


Figura 3.9 Flujograma de la función Canal::IniciarLectura().

Primero, IniciarLectura() inicializa variables locales. Entre estas variable está int sentido, con la que se representa el sentido de las señales de línea (o sea, si se trata de abonado llamado o llamante) y puede tomar los valores SIN_SENTIDO cuando no se ha determinado, ADELANTE y ATRÁS.

Otra variable que se crea es el objeto crono de la clase Tiempo, que se utiliza para guardar el tiempo actual del dato que se esta procesando que luego se agrega en los archivos de texto o en pantalla. La estructura de la clase Tiempo se muestra a continuación.

```
class Tiempo{

private:

      int mseg;
      int seg;
      int min;
      int horas;
      friend ostream& operator<<(ostream& co, const Tiempo &hora);

public:

      Tiempo();
      Tiempo operator++();
      Tiempo operator=(const struct _timeb *tFuente);
      void resetTiempo();

};
```

La clase Tiempo posee miembros en los que se almacenan la hora, minutos, segundos y milisegundos. Posee además sobrecargas a los operadores de incremento, para incrementar el tiempo en un milisegundo, y de asignación, para interactuar con las estructuras _timeb. También se ha sobrecargado el operador << para poder hacer salidas directas utilizando flujos (cout o archivos).

El objeto crono interactúa también con la estructura _timeb tInicial, que sirve luego para obtener el tiempo local.

Luego de la inicialización de las variables, la función entra en un lazo infinito. Este código se ejecutará permanentemente mientras el thread principal no mande un señal de detener la operación. La primera acción en este lazo es leer los datos de buffCanal[] y transferirlos a buffTrabajo[] usando la función Canal::LeerElBuffer(), para que puedan ser procesados.

La cantidad de datos presentes en buffTrabajo[] se guarda en tamanoBuffTrabajo. Con este valor, se inicia un lazo for para revisar dato por dato buffTrabajo[].

Primero se verifica si es necesario tomar el tiempo para el dato actual. Inicialmente debe ser tomado, y no se volverá a tomar a menos que haya una notificación de pérdida de trama o multitrama. Esto se ha hecho de esta manera debido a que, por la forma estricta en que las recomendaciones estructuran las tramas y multitramas en cuanto a duración (2.048Mbps ± 50ppm , 125μs), se garantiza que cada dato de señalización llega cada 2 ms. Solo se

necesita tomar un tiempo inicial de referencia, pues para diagnóstico importa más que las diferencias de tiempo entre códigos sean precisas, y no el tiempo absoluto. Los tiempos que se toman de la computadora no reflejan el momento en que se capturo la señal, debido a que, para llegar hasta este thread, los datos han pasado por varios buffers y por tanto han debido sufrir retrasos para ser transferidos. La lectura del tiempo se hace a través de la función ftime, miembro de la librería time.h, que retorna el valor en UCT (Universal Central Time) usando la estructura tInicial. Esta estructura es asignada a crono, que la convierte a formato hh:mm:ss.ms.

Después de esto verifica si el dato es válido, es decir, si no existe notificación de pérdidas de alineación en los 4 bits más significativos del byte de datos. Si el dato no es válido, se determina si se trata de que notificación se trata, se envían a archivo o pantalla junto el tiempo en que ocurrió y finalmente se incrementa el tiempo en 2 ms.

Si el dato es válido, comienza a contarse el número de datos repetidos. Esto se hace debido a que la recomendación Q.422 exige que para que exista un código válido debe cumplirse un timer de 20 ± 10ms. Debido a que los datos llegan cada 2 milisegundos, para cumplir el nivel mínimo de 10 ms es necesario que se reciban 5 datos seguidos del mismo valor. El valor de este timer puede modificarse con la viñeta T_RECON.

Cumplido el timer y obtenido un código válido, se revisa en primera instancia si existe una repetición de código válido mientras no se ha determinado el sentido de transmisión. Si esto es cierto, el código es ignorado y se pasa a la siguiente iteración. En caso contrario, se entra en una bloque switch cuya variable de prueba es el sentido de transmisión.

Inicialmente el sentido no se conoce, por lo que trata de determinarse. El programa utiliza como punto de referencia el código Idle, que es igual en ambos sentidos. Si después de Idle aparece un código A=0 B=0, se trata del código Seizure y el sentido en ese momento es hacia adelante. Si, en cambio, seguido al Idle aparece un A=1 B=1, el código corresponde a Seizure Acknoledge y las señales son hacia atrás.

Una vez determinado el sentido, el nombre del código se decodifica según sea el grupo de las señales, hacia delante o hacia atrás, en los dos casos extras del bloque switch. Dentro de estos casos es que se realizan las salidas a archivo y pantalla. Los datos presentados en los archivos corresponden a los cambios de código únicamente y no a todos los códigos válidos.

Dentro de los grupos existen estados que no pueden ser determinados con seguridad porque el código binario en un sentido es igual y son las señales del otro sentido quienes los distinguen. Debido a que no se poseen las señales de ambos sentidos, lo que se hace es simular los estados. Como ejemplo pueden tomarse los estados Clear Forward, Released y Idle, estos poseen el código binario 1001 (ABCD). Para lidiar con esto, lo que se hace es tomar el primer código válido como Clear Forward, el siguiente como Released y el último como Idle. Estas simulaciones son la causa de que las repeticiones de código no sean del todo eliminadas al inicio de la decodificación cuando el sentido de las señales ya se ha

determinado. Aunque estos no se corresponden exactamente a los estados verdaderos, ya se sabe que esta será la sucesión de eventos, en el caso de una llamada normal.

Luego del bloque switch, se guarda el último código válido, se incrementa el tiempo de crono en 2 ms, y se pasa a la siguiente iteración. Cuando se terminan los datos en buffTrabajo[], se termina el lazo for y se vuelve al ciclo infinito.

3.4 CONCLUSIONES Y RECOMENDACIONES AL CAPÍTULO III

- El programa desarrollado tiene la capacidad de decodificar señalización R2 digital de línea para 30 canales de voz / datos que son transportados en una línea E1. Durante la ejecución del programa se generan 30 archivos de texto que poseen, cada uno, la información de señalización de un canal. Puede escogerse también un canal para ser desplegado en pantalla.

- Algunos estados dentro de la señalización R2 pueden determinarse únicamente si se conocen las señales hacia delante y hacia atrás, debido a esto, esos estados son simulados dentro del programa. Aunque esto resta precisión al programa, es importante destacar que la estructura básica del programa es completamente funcional y exacta. Todas las señales recibidas son confiables y el único problema surge en la interpretación final.

- El programa puede ser objeto de mejoras para ampliar sus capacidades y facilitar en alguna medida su manejo, entre estas se pueden mencionar: implementación de una interfaz gráfica, uso de archivos de configuración (para evitar recompilar el programa cuando se llevan a cabo cambios en los parámetros) e incorporación de la capacidad de decodificación de señalización R2 de inter registro. Ninguno de estos cambios implica mayores alteraciones en la estructura básica del programa. En el caso de la decodificación de señales inter registro, sería necesario únicamente que el thread de multitrama enviara los canales de voz al buffer de cada thread de señalización. Para distinguir entre los datos de voz y de canal de señalización, este buffer (arreglo) podría cambiarse por uno que posea elementos de 16 bits en lugar de elementos de 8 bits, así podría utilizarse los bits extras para agregar un código identificación, de forma parecida a lo que se hace con las notificaciones de pérdida de alineación de trama básica y multitrama. Esto permitiría la sincronización precisa de ambos tipos de datos. Además debería incorporarse un módulo o función de filtrado digital para la distinción de los códigos de multifrecuencia, los cuales podrían estar implementados en otro lenguaje de programación más adecuado para el procesamiento digital de señales, que permita la creación de librerías que puedan ser llamadas desde Visual C++.

3.5 REFERENCIAS BIBLIOGRÁFICAS

[1] Cisco Systems Inc. "E1 R2 Signaling Theory". Tech notes. http://www.cisco.com/warp/customer/788/signalling/e1_r2_sig.html. 2000.

[2] Future Technology Devices Intl. (FTDI) Ltd. "Data Throughput, Latency & Handshaking". Application Note AN232B-04. 2004. http://ftdichip.com/.

[3] Future Technology Devices Intl. (FTDI) Ltd. "FTD2XX Programmer's Guide". Version 2.01. 2002. http://ftdichip.com/.

[4] Future Technology Devices Intl. (FTDI) Ltd. "Optimising D2XX Data Throughput". Application Note AN232B-03. 2004. http://ftdichip.com/.

[5] Microsoft Corp. "MSDN Library for Visual Studio 6.0". 1991-1998.

[6] Peacock, Craig. "USB in a Nutshell. Making Sense of the USB Standard". Third Release. Noviembre 2002. http://www.beyondlogic.org/.

[7] Sunrise Telecom Inc. "Technology Series. Introduction to MFC-R2 Signaling". Publication Number TEC-GEN-00 2 Rev.B. 2001.

[8] UIT-T. "Procedimientos de alineación de trama y de verificación por redundancia cíclica (VRC) relativos a las estructuras de trama básica definidas en la recomendación G.704". Rec. G.706. 1991.

[9] UIT-T. "Especificaciones del sistema de señalización R2. Señalización de línea, versión digital. Código digital de señalización de línea". Rec. Q.421. 1993.

[10] UIT-T. "Especificaciones del sistema de señalización R2. Señalización de línea, versión digital. Cláusulas relativas al equipo de señalización de línea de las centrales". Rec. Q.422. 1993.

[11] Young, Michael J. "Mastering Visual C++ 6". Sybex, Inc. ISBN: 0782122736. Agosto, 1998.

CAPÍTULO IV

RESULTADOS OBTENIDOS

INTRODUCCIÓN

El presente capítulo muestra los resultados obtenidos al implementar el instrumento virtual, cuyo diseño fue mostrada en los capítulos anteriores.

Previo a los resultados finales se hace una descripción de las diferentes pruebas que fueron realizadas para observar el comportamiento de la interfaz bajo ciertas condiciones con el objetivo de constatar un funcionamiento adecuado o detectar posibles fallas y errores en forma aislada, para determinar una posible solución.

Las pruebas de laboratorio se desarrollan de una manera bastante básica, debido a la falta de equipo adecuado. Las pruebas de campo, que involucraron el uso de equipo de comunicaciones en condiciones de operación, se realizaron en COCESNA (Consorcio Centroamericano de Servicios de Navegación Aérea) y Digicel de El Salvador. Estas pruebas requerían un grado más de precaución y responsabilidad, ya que se trataba de equipo con tráfico, que no podía ser interrumpido.

Finalmente se presentan las características técnicas las cuales es necesario tomar en cuenta para garantiza el funcionamiento correcto de el instrumento virtual.

4.1 FASE EXPERIMENTAL

Esta sección describe las pruebas que fueron necesarias para garantizar que el equipo era capaz de manejar en forma confiable y eficiente el flujo constante de datos a 2.048 Mbps y la correspondiente interpretación de esa información.

Las diferentes pruebas se han agrupado en dos tipos: pruebas de laboratorio, que son aquellas que no necesitaron datos reales pero mostraban el comportamiento del circuito a nivel lógico, y pruebas de campo, aquellas donde se necesitaba un equipo de comunicaciones funcionando con las características para las que fue diseñado este instrumento virtual.

4.1.1 Pruebas en laboratorio

Tal como se menciono anteriormente, a pesar de tener un diseño finalizado, se realizaron una serie de pruebas con los que era posible observar el comportamiento del circuito en cuanto a la robustez y la capacidad de manipular la información, esto a nivel circuital.

Para comprobar que los datos eran enviados sin tener pérdidas, se construyó un circuito que generaba un patrón repetitivo de datos a la misma velocidad que lo haría un equipo real. Los datos de este circuito debían ser alimentados al circuito y recibidos en la PC por un programa de prueba, que comprobaba la existencia de dicho patrón. El diseño se muestra en la figura 4.1.

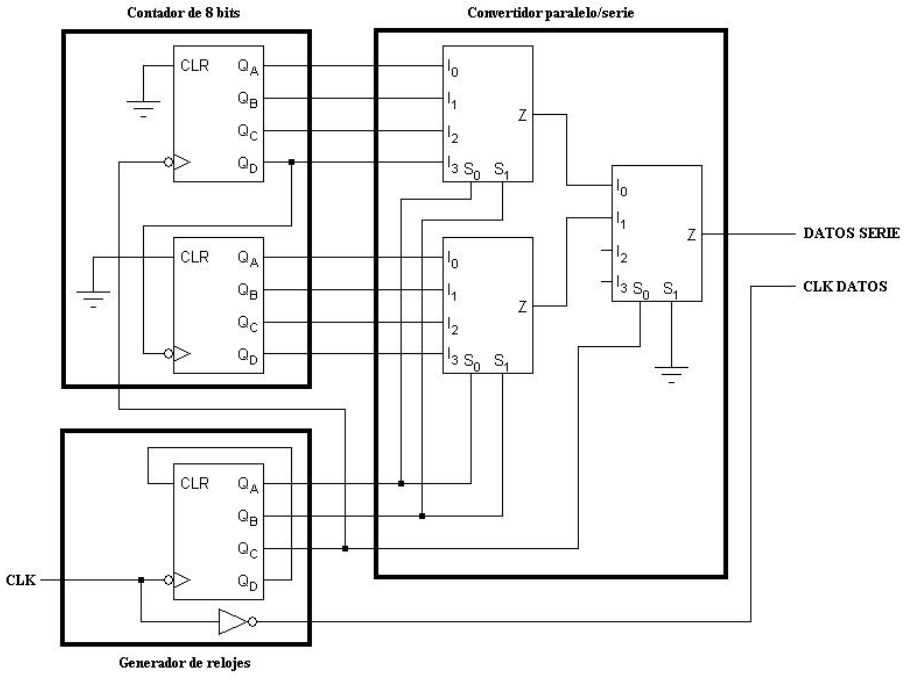


Figura 4.1 Generador de datos serie a 2.048 Mbps.

El circuito fue implementado utilizando contadores binarios de 4 bits y multiplexores de 4 a 1 bit. El patrón repetitivo usado fue un conteo binario de 8 bits (0-255). Las salidas de este circuito son los datos de conteo, convertidos en formato serie, y su correspondiente reloj de sincronismo. Para esta prueba, el XRT82D20 se configuró en modo Local Loop back, lo que significa que la señal de transmisión se retroalimenta internamente a la recepción, obteniendo el reloj y datos recuperados de esta señal.

Aunque parece redundante, este proceso permite comprobar el buen funcionamiento del XRT82D20. A su vez, esto permite que cada vez que se necesite una prueba de este tipo, no sea necesario el desmontar ningún elemento del circuito, ya que simplemente, a través de un jumper, se activa o se desactiva el modo Local Loop Back.

Para comprobar los patrones se utilizaron dos programas. Primero, los fabricantes de la interfaz USB proveen una herramienta gráfica que permite observar los datos recibidos, guardarlos en archivos e incluso enviar datos. Esta herramienta se llama dlptest y fue utilizada para constatar que se estaban recibiendo los mismos datos que generaba el contador de 8 bits.

Sin embargo, con este programa no se garantiza que todos los datos llegan a la PC, ya que en un flujo de datos como el que maneja la interfaz, una pérdida de datos podría pasar inadvertida simplemente con ver los datos recibidos en pantalla. Para eso fue necesario hacer un programa que recibía los datos y a la vez los comparaba una tras otro, con el fin de determinar que la secuencia del contador se mantenía; e informaba en un archivo si en determinado momento esta secuencia era interrumpida, producto de la pérdida de datos. Al hacer pruebas prolongadas se determinó que existía un flujo de datos firme y constante.

Como complemento a las pruebas anteriores se realizó una prueba en la que se leían patrones aleatorios, esto con el fin de observar que tan robusto es el circuito a la interferencia, ya que el flujo de datos aleatorios refleja una situación parecida a la que se experimentaría conectado a un equipo de comunicaciones en operación, donde los datos son semi aleatorios. Eléctricamente las señales aleatorias generan un grado mayor de interferencia que en este caso podría afectar el funcionamiento del circuito y alterar el flujo de datos.

Para esta prueba se utilizó un generador de patrones semi aleatorios codificados en HDB3 a una velocidad de 2.048 Mbps. Se hizo un programa que simplemente recibía los datos generados, pero incapaz de realizar comprobaciones, y esto debido a la naturaleza aleatoria de lo datos. Los efectos de la interferencia se comprobaron con la salida FULL de la FIFO conectada a un contador, cuyas salidas iban a LEDs con los que se podía observar si había o no rebosamientos. Después de una serie de pruebas en tiempos prolongados, el circuito no mostró que se interrumpiera el flujo de datos.

Al cumplirse estas condiciones durante las pruebas de laboratorio el circuito se encuentra listo para trabajar con equipo real en las pruebas de campo.

4.1.2 Pruebas de campo

A continuación se presenta una breve descripción de las pruebas realizadas en equipos de comunicación en operación en COCESNA y Digicel. Estas pueden dividirse en dos grupos: pruebas de acoplamiento y recepción de datos, y pruebas de decodificación y manejo de tráfico. Los resultados de las pruebas finales se presentan al final.

4.1.2.1 Pruebas de acoplamiento y recepción de datos

Las pruebas de acoplamiento y recepción de datos tenían como objetivo asegurar la confiabilidad de la información proveniente de la línea E1 y la capacidad de trabajo del circuito en paralelo y transparente en el sistema donde se realizarían mediciones.

Inicialmente estas pruebas consistieron en conectar el circuito a la línea E1 de transmisión de un multiplexor de primer orden OKI. Este multiplexor posee 30 canales de voz / datos de asignación fija, y es capaz de manejar señalización E&M y R2 digital.

Se verificó que no existieran alarmas en el multiplexor o pérdidas de alineación, y que el circuito mostrara la indicación de sincronización con la trama. Si estos requisitos no se cumplían, no podía confiarse de ninguna forma en el acoplamiento de la señal de entrada.

Luego se utilizó el programa dlptest para observar repeticiones o patrones de datos que mostraran, de forma básica, la llegada de datos confiables. Esto es útil en la detección de fallas en el circuito que involucran problemas con las líneas de datos. Después de comprobar los patrones y observar que no hubiera indicación de desbordamiento de la memoria FIFO del circuito, se procedió a verificar la existencia de la trama básica dentro de los datos, con la ayuda de un programa que implementaba los procedimientos de alineación de la norma G.706[]. El programa registraba parte de la información contenida en el TS0 y, sobre todo, referente a los datos que causaban pérdidas de alineación.

Los procedimientos se repitieron en el E1 de recepción del equipo, donde los datos están sometidos al ruido del medio de transmisión.

La prueba también se realizó en equipo Ericsson perteneciente a Digicel, que no estaba en operación, y que utilizaba señalización CCS7. La conexión a este equipo se hizo primero interrumpiendo el circuito y colocando un conector en "T" BNC, y luego utilizando conectores propietarios que debieron ser modificados, ya que estos poseen impedancias internas del orden de los 700Ω.

Esta prueba fue considerada la demostración definitiva de la confiabilidad de los datos y del funcionamiento del acoplamiento de señal, y en adelante la atención se centró en los programas de descodificación.

4.1.2.2 Pruebas de decodificación y manejo de tráfico

Las pruebas de decodificación y manejo de tráfico sirvieron para probar los programas del instrumento, la robustez del circuito frente a los nuevos patrones de interferencia generada por el tráfico y la estabilidad del instrumento.

Al inicio se utilizaron programas que implementaban los procedimientos de alineación de multitrama[] y que almacenaban los bits de señalización sin decodificación en archivos de texto, y las pruebas, hechas en COCESNA, consistieron en generar señalización manualmente en un canal, utilizando un aparato telefónico, mientras los demás canales manejaban muy poco tráfico. Los datos recogidos permitieron observar la secuencia en los bits y constatar que seguían el orden esperado, permitiendo así demostrar que se trataba de datos de señalización correctos.

Figura 4.2. Circuito durante pruebas en las instalaciones de Digicel.

Las pruebas de los programas con capacidad de decodificación iniciaron supervisando un canal específico, observando en pantalla los cambios y verificando los tiempos, mientras se controlaba la generación de la llamada y el sentido de transmisión. Estos procesos se llevaron a cabo en un PBX con señalización R2 en Digicel.

Finalmente el instrumento fue dejado en operación por espacios de tiempo de hasta media hora y en condiciones de alto tráfico, para comprobar su estabilidad.

4.1.2.3 Resultados de las pruebas de campo

Los resultados de las pruebas consisten en archivos de texto que contienen la información de cada canal por separado, a los que se denomina trazas.

Las trazas poseen el siguiente formato: A la izquierda se encuentra la hora en que se recibió el código, con resolución mínima en el orden de los milisegundos. Luego se presenta el estado de los bits de señalización en el siguiente orden: ABCD. Finalmente se muestra el código de señalización que el instrumento considera esta representado por los bits. En caso de que el programa no reconozca el código presente, mostrará un signo de interrogación "?".

Si existiera una pérdida de alineación de trama básica o multitrama se reporta mostrando la hora y la indicación LoF o LoMF, respectivamente.

La figura 4.2 muestra tres trazas resultado una de las pruebas finales del instrumento, de 10 minutos de duración tomadas el día 8 de noviembre de 2004.

```
***Canal: 13***

17:06:08.405 1001 Idle
17:06:16.277 1101 Seizure Acknowledge
17:06:52.725 0101 Answered
17:11:26.049 1101 Clear Back
17:11:26.173 1001 Released
17:11:26.183 1001 Idle
17:11:29.855 0001 Seizure
17:11:49.815 1001 Clear Forward
17:11:49.825 1001 Released
17:11:49.835 1001 Idle
17:11:59.039 0001 Seizure
17:12:43.401 1001 Clear Forward
17:12:43.411 1001 Released
17:12:43.421 1001 Idle
17:12:44.843 0001 Seizure
17:13:15.725 1001 Clear Forward
17:13:15.735 1001 Released
17:13:15.745 1001 Idle
17:13:39.463 0001 Seizure
17:15:31.847 1001 Clear Forward
17:15:31.857 1001 Released
17:15:31.867 1001 Idle
17:15:48.085 0001 Seizure
```

```
***Canal: 24***

17:06:08.405 0001 ?
17:07:00.049 1001 Idle
17:07:58.157 0001 Seizure
17:08:53.485 1001 Clear Forward
17:08:53.495 1001 Released
17:08:53.505 1001 Idle
17:08:59.127 1101 Seizure Acknowledge
17:09:17.199 0101 Answered
17:09:27.043 1001 Released
17:09:27.053 1001 Idle
```

```
***Canal: 26***

17:06:08.405 1001 Idle
17:06:29.013 0001 Seizure
17:06:29.327 1001 Clear Forward
17:06:29.337 1001 Released
17:06:29.347 1001 Idle
17:07:35.839 0001 Seizure
17:09:53.227 1001 Clear Forward
17:09:53.237 1001 Released
17:09:53.247 1001 Idle
17:10:27.373 0001 Seizure
17:11:13.551 1001 Clear Forward
17:11:13.561 1001 Released
17:11:13.571 1001 Idle
17:11:14.939 0001 Seizure
17:11:17.755 1001 Clear Forward
17:11:17.765 1001 Released
17:11:17.775 1001 Idle
17:11:25.961 0001 Seizure
17:12:52.271 1001 Clear Forward
17:12:52.281 1001 Released
17:12:52.291 1001 Idle
17:13:01.279 0001 Seizure
17:14:25.883 1001 Clear Forward
17:14:25.893 1001 Released
17:14:25.903 1001 Idle
17:15:01.107 0001 Seizure
```

Figura 4.3. Trazas obtenidas de pruebas de campo.

Las trazas corresponden a tres canales de voz distintos, canal 13, 24 y 26, y fueron tomadas de manera simultánea. La información mostrada corresponde a tráfico telefónico de la PBX de la empresa Digicel. Si se observa el canal 24, por ejemplo, se puede ver que inicialmente existía lo que parecería una toma, por el valor de los bits de señalización (A=0, B=0, C=0, D=1), pero como el instrumento toma como base la señal Idle para definir el sentido de transmisión, se muestra como código desconocido. Luego termina esta condición y aparece la señal de Idle. Después de la señal de Idle aparece de nuevo el código 0001, pero esta vez

es reconocido como una toma (Seizure). La llamada se completa. Luego una llamada en que aparecen señales del grupo hacia atrás, es decir, hubo un cambio en el origen de esta llamada respecto a la anterior.

4.2 CARACTERÍSTICAS TÉCNICAS

El instrumento virtual de monitoreo de señalización R2 digital está formado por dos componentes: un circuito de interfaz y un programa de decodificación.

Circuito de interfaz:

- Soporte para una línea E1 de entrada.
- Impedancia de la línea de entrada: 75Ω desbalanceada.
- Tipo de conector a la línea: BNC.
- Voltaje de alimentación: 5V.
- Cumple la Rec. G.703 de la UIT-T.
- Modo de prueba Local Loopback programable vía jumper.
- Buffer FIFO de 1024 bytes.
- Conexión a la PC vía puerto USB.
- Cable de conexión USB tipo AB.
- LED de indicación de sincronización de trama.
- Indicador de desbordamientos de memoria FIFO.

Programa de decodificación:

- Alineación de trama básica y multitrama, conforme a la Rec. G.706 de la UIT-T.
- Decodificación de señalización R2 digital de línea, según establecido en las Rec. Q.400-Q.490 UIT-T.
- Generación de archivos de texto de trazas para 30 canales simultáneos.
- Salida a pantalla de uno de los 30 canales.
- Tiempo de prueba programable.

Características de la PC para uso del programa:

- Sistema operativo Windows XP.
- Procesador equivalente Pentium III a 1GHz o superior.
- 128 MB de RAM
- Más de 10 MB de espacio en disco duro, según duración de las pruebas.
- Puerto USB 1.1

4.3 PROPUESTA DE MEJORA

A partir de la información que se presentó en el capítulo I acerca de la señalización R2 digital, se sabe que para determinar con seguridad el estado en que se encuentra un circuito telefónico es necesario conocer las señales hacia adelante y hacia atrás, tanto de línea como de registro, por lo tanto un instrumento virtual completo necesita recibir información de los canales de señalización de ambas direcciones de transmisión. Este hecho implica que se deben recibir los datos correspondientes a dos líneas E1.

El problema puede abordarse de dos formas: una es multiplexar los dos flujos de datos en circuito y que entren a la computadora utilizando una sola interfaz paralelo / USB. La otra es utilizar dos circuitos de interfaz idénticos al diseño presentado en el capítulo II, conectados al puerto USB de la misma computadora.

La primera propuesta implicaría una serie de cambios en el diseño del circuito de interfaz. Tomando de base el diagrama de bloques de la figura 2.2, el circuito para el aparato completo estaría constituido por dos circuitos de acoplamiento, dos interfaces de línea, dos etapas de sincronización y dos memorias FIFO que formarían dos flujos independientes de datos, llevando a cabo la multiplexación a la salida de las memorias para formar un flujo de 4.096 Mbps o 512 kbytes/s que sería recibido por la interfaz paralelo / USB. La multiplexación podría implementarse intercalando byte a byte los datos de ambas líneas. El programa se encargaría luego de separar los datos correspondientes a ambos E1.

En la segunda propuesta, se trabajaría con dos circuitos iguales al diseño implementado en este trabajo conectados al puerto USB de la misma PC, y el programa leería simultáneamente de ambos dispositivos. El problema en este caso está en lograr la sincronía entre ambos flujos de datos, es decir, conocer cuales datos provenientes de ambos flujos fueron recibidos en el mismo instante. Para superar este problema, sería necesario sincronizar la lectura. Este proceso se facilita si se mantiene en la estructura del programa solamente un thread de lectura, pues sería muy difícil la sincronización en caso de implementar dos threads de lectura independientes.

Para realizar la sincronización de la lectura dentro del thread, los objetos evento correspondientes a las interrupciones de llegada de datos provenientes de cada circuito deben ser relacionados a través de una función de espera WaitForMultipleObjects(), realizando la lectura de datos de los dos buffers USB (usando FT_Read) hasta que existan datos en ambos. Se esperaría que en operación normal ambos buffers posean la misma cantidad de datos al mismo tiempo, ya que están sometidos a los mismos retrasos y si el Maximum Transfer Size es igual para ambos. En todo caso, debe crearse alguna forma de manejar cantidades de datos recibidos distintas.

Debe considerarse, además, la posibilidad de cambiar el valor del Maximum Transfer Size respecto a los utilizados en el capítulo III para mejorar la fluidez de los datos. Esto deberá determinarse por medios prácticos, por ejemplo, utilizando el monitor de uso de CPU que está en la sección Rendimiento del Task Manager de Windows XP.

En ambos casos, la estructura de programa debe cambiar para ajustarse a los nuevos datos recibidos. Los cambios en el programa implicarían implementar dos threads separados de alineación de trama básica, dos threads de alineación de multitrama y 30 threads de decodificación de señalización, con las modificaciones respectivas para recibir ambos flujos de datos.

Otro consideración que hay que destacar, es que el aumento en la cantidad de datos enviados a la computadora implicará mayores retrasos, por lo que debe considerarse aumentar el tamaño de las memorias FIFO. Además, podría pensarse en el reemplazo de la interfaz paralela / USB por una similar con soporte para transferencia High Speed propia del estándar USB 2.0.

Para que el instrumento virtual sea completo, debe también implementarse la decodificación de señales inter registro, tal como se menciona con más detalle en las recomendaciones al capítulo III.

De las dos propuestas antes mencionadas se considera más viable la segunda, siendo su mayor ventaja que no implica cambios en los circuitos y así se aprovecha por completo la investigación hecha y experiencia adquirida en este trabajo.