UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

ESCUELA DE INGENIERÍA

UNIDAD ENSENADA

"SISTEMA PARA COMUNICACIÓN POR I^2C ENTRE PUERTO DE COMUNICACIÓN DE TELEVISION SONY Y PC"'

Tesis Profesional
Que para obtener el título de:

INGENIERO EN ELECTRÓNICA

Presenta:

Carlos Ruben Aguilar Benson

Director de tesis:

M.C. José Jesús Zamarripa Topete

ENSENADA, B.C. JUNIO DEL 2000

"SISTEMA PARA COMUNICACIÓN POR I^2C ENTRE PUERTO DE COMUNICACIÓN DE TELEVISION SONY Y PC"

TESIS QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN ELECTRÓNICA

PRESENTA:

Carlos Ruben Aguilar Benson

M.C. José Jesús Zamarripa Topete
Director de Tesis

M.C. Humberto Cervantes de Ávila

M.C. Juan Ivan Nieto Hipólito

Ing. Luis Talavera Balbuena

M.C. Raul Espejo Rodante

AGRADECIMIENTOS

Por su apoyo, agradezco a:

Mis padres, Abuela, Tíos y Familia Rascon Santaella

SISTEMA PARA COMUNICACIÓN POR I^2C ENTRE PUERTO DE COMUNICACIÓN DE TELEVISOR SONY Y PC

ÍNDICE **PÁGINA**

LISTA DE FIGURAS

LISTA DE TABLAS

ORGANIZACIÓN DE ESTE TRABAJO

En el capítulo I se da la introducción al uso del bus I^2C en Televisión , sin profundizar mucho en el concepto, ya que esto será ampliamente explicado en otros capítulos.

En el capítulo II se da el objetivo de este trabajo y una explicación de como se contempla el desarroyo y funcionamiento de un sistema para comunicación de I^2C , finalmente se muestra un esquema general del proyecto.

En el capítulo III se explica de manera mas amplia el concepto de I^2C, además se contemplan todas las especificaciones eléctricas y protocolo de comunicación. El fin de este capítulo es tener un conocimiento de I^2C mas profundo, necesario para poder desarroyar la parte física de este sitema.

En el capítulo IV se muestra la parte física de este sistema, que es propuesta para realizar la comunicación (interface). Ademas en este capítulo se da una explicación del uso del puerto paralelo de una PC, conocimiento necesario para el desarrollo de esta interfaz, finalmente se muestra un diagrama esquemático de esta interfaz.

La intención del capítulo V es brindar información necesaria para la creación de un archivo DLL. Se explica cómo crearlo en C++, y cómo usarlo en Visual Basic.

El capítulo VI, muestra todas la subrutinas que se crearon para comunicarse con el I^2C mediante el Protocolo de comunicación. La intención es mostrar los algoritmos usados y la manera en que respetan este protocolo.

El capítulo VII muestra posibles aplicaciones de este sistema, primero se da un Editor de memorias NVM, y finalmente un Evaluador de dispositivos.

1.- INTRODUCCIÓN

1.1.- COMUNICACIÓN DE DATOS ENTRE DISPOSITIVOS INTERNOS DE TV.

Un televisor (SONY), cuenta con un microprocesador interno que controla y hace uso de otros dispositivos, para tareas muy específicas, tales como memorias tipo NVM control de color, intensidad de brillo, control de la geometría de la imagen, ajuste de convergencia dinámica de los rayos de electrones, Nivel de volumen y una infinidad de dispositivos mas, algunos con tareas mas complicadas como uso de GEMSTAR, digitalización de la imagen, manejo de sistema Surround etc.

Para la comunicación entre el microprocesador y cada uno de estos dispositivos SONY tomó como estándar el protocolo de comunicación IIC (I^2C = Inter IC) abreviación de las iniciales en ingles de Inter Integrated Circuit.

Este protocolo de comunicación fue creado por PHILLIPS SEMICONDUCTORS y adoptado por muchas otras empresas que desarrollan semiconductores, aplicaciones de control y productos electrónicos de uso común como celulares, monitores para computadora, televisores, Direct T.V, sistemas de sonido e. t. c..

En este caso SONY Televisores ha adoptado este protocolo de comunicación por sus grandes beneficios al diseño del producto, como reducción en la cantidad de líneas para direcciones y datos, evitando así el uso de un ducto con una gran cantidad de líneas para direccionar, y otro ducto de datos también con un gran numero de líneas, puesto que el protocolo de I^2C solo hace uso de una línea **bidireccional** para datos y direcciones **SDA** y otra línea también bidireccional para reloj **SCL.**

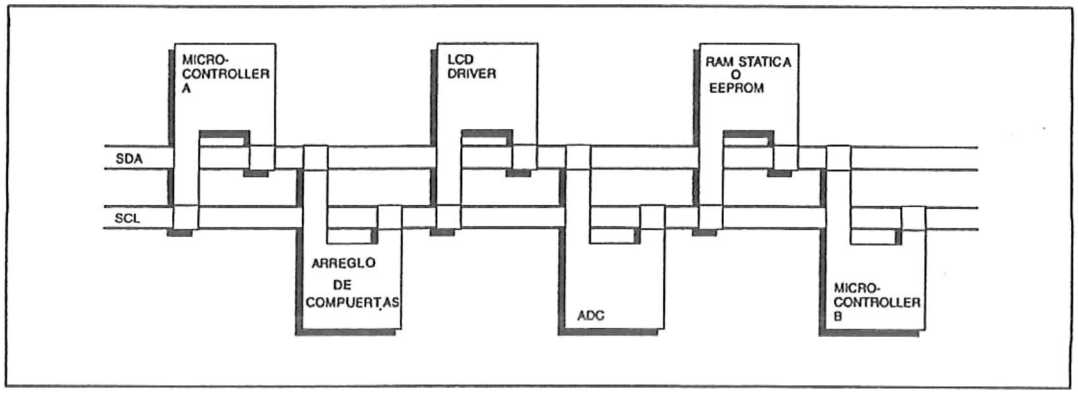


Fig. 3.b Ejemplo de una configuración usando dos microcontroladores

Este diagrama se tomó del capitulo 3, y muestra una aplicación de I^2C.

1.2.- EQUIPOS DE AJUSTE PARA EL TELEVISOR

Existen varios ajustes que se le deben hacer a un televisor, estos en su mayoría tienen un efecto directo sobre la imagen del CRT del televisor, por ejemplo: tamaño horizontal o vertical de la imagen, distorsión geométrica de la imagen, variación de los colores, control de brillo, ajuste de la convergencia y muchos más.

Todos estos ajustes en su mayoría tienen un dispositivo determinado para esa tarea, el cual puede poseer una memoria o registros y al cual se le manda el dato digital como orden para que actúe según se desea que se haga la variación de este ajuste. El cambio de este valor digital se ve reflejado en una variación analógica con efecto directo sobre la imagen vista en el CRT.

Por ejemplo: en el caso de control de Brightness , Sharpness , se cuenta con un dispositivo de control llamado Video Processor (Eje: CXA2060S) , el cual cuenta con la dirección 1000100x como su Slave Address (cuando es lectura x=1, cuando escritura x=0) .

Este circuito integrado posee un registro para el control de Brightness con 6 bits, y Sharpness con 4 bits, cuyas direcciones o SubAddress son xxx00110 para el brillo y xxx00111 como se muestra en su mapa de memoria en la tabla 1a.

Ejemplo de Dispositivo Slave

Video processor CXA2060s Slave Address: 1000100x (x=0:write, x=1:read)

SUB ADR	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
XXX00000	PON	HDW	AXIS PAL	VOL	FH HIGH	C DECOD	AGING	T1HDL
XXX00001	VIDEO SEL		S SEL		R ON	G ON	B ON	Y SEL
XXX00010	X TAL		COL SYSTEM		COL LOOP		CBPF	C TRAP OFF
XXX00011	PICTURE						NO COLOR	FSC SW
XXX00100	COLOR						C OFF	KILLER OFF
XXX00101	HUE						SHP F0	AXIS NTSC
XXX00110	BRIGHTNESS						DC TRAIN	PRE OVER
XXX00111	SHARPNESS				R CUTOFF			
XXX01000	G CUTOFF				B CUTOFF			
XXX01001	R DRIVE						ABL MODE	ABL VTH
XXX01010	G DRIVE						DY COL	RGB SEL
XXX01011	B DRIVE						GAMMA	
XXX01100	H OSC				Y DELAY			
XXX01101	FIELD FREQ		CD MODE		INTERLACE		HSS	VSS
XXX01110	V SIZE						*	H MASK
XXX01111	V POSITION						AFC GAIN	
XXX10000	S CORRECTION				V LINEARITY			
XXX10001	H SIZE						*	EW/DC
XXX10010	H POSITION						*	*
XXX10011	PIN AMP						*	*
XXX10100	CORNER PIN						*	*
XXX10101	TRAPEZIUM				EHT COMP			
XXX10110	AFC BOW				AFC ANGLE			
XXX10111	LEFT HBLK				RIGHT HBLK			
XXX11000	ASPECT						H BLK	VUNDERSCA
XXX11001	SCROLL						V ZOOM	
XXX11010	UPPER VLIN				LOWER VLIN			
XXX11011	TV/OFF	TTRG	VTM SEL		ID STOP		ID START	
XXX11100	BELL ID						ID LEVEL	

Si lo que se desea variar fuera la intensidad del Brillo se tendría que accesar a slave address 88h(10001000) y sub address 06h (00000110) y se daría como dato aquel que correspondiera con el nivel de intensidad de brillo que se deseara. Y se obtendría una tabla de equivalencias como la de la tabla 1b.

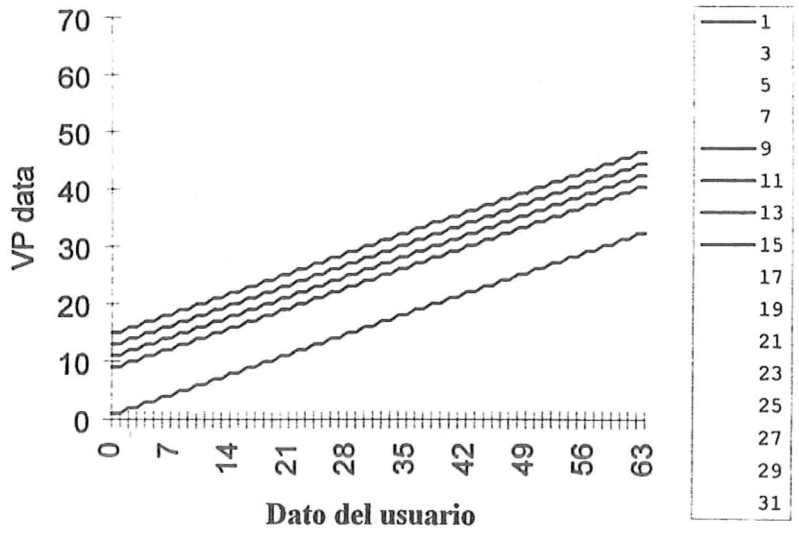


Fig 1.b

Pero para hacer posible los ajustes de un televisor en la manufactura es necesario que se cuente con equipos que hagan esto automáticamente, dicho equipo estaría cambiando el valor digital del registro correspondiente al ajuste deseado de el dispositivo correspondiente. Donde se estaría evaluando el efecto del cambio mediante el uso de una cámara u otro tipo de sensor (aunque algunas veces también podría hacerse uso de una

persona operador el cual notaria el efecto físico del cambio), una interface para comunicación I^2C con el BUS de la T.V. y una unidad de proceso para control.

Por ejemplo para el ajuste de la **geometría** de la imagen, donde se toman en cuenta las dimensiones y forma de la imagen se podría hacer uso de una cámara CCD, una computadora, **elaborar una interface para I^2C** , y desarrollar el algoritmo que llevara a cabo este ajuste.

Este sistema tendría que accesar los dispositivos tales como el Video Processor IC, y cambiar cada uno de los registros que sea necesario para que la imagen quede bien visualmente y dentro los valores especificados para una buena calidad de imagen.

Por lo que es necesario, elaborar dicha Interface de I^2C, pues sin ella nada sería posible ya que es elemental una comunicación entre Televisor y PC

Ejemplos de equipos que se usarian en la manufactura que harian uso de una interface I^2C Son:

- Equipo de Preset (Donde se dan valores promedio iniciales al Televisor)
- Equipo de Geometria (Ajuste de medidas horizontal , vertical y forma de la imagen)
- Equipo de Convergencia (donde se ajusta la convergencia de los rayos del CRT)
- Equipo de White Balance (Balanceo de los colores primarios para obtener el blanco)
- Editor de NVM. (para checar valores del mapa de memoria NVM)

2.- DESCRIPCIÓN DEL SISTEMA.

2.1.- INTRODUCCIÓN.

Para poder ajustar automáticamente el televisor se cuenta con equipos que incluyen todo en un sistema muy compacto, teniendo el sistema que monitorea los cambios en el televisor (con cámara CCD o Sensores) y una tarjeta para tener acceso al bus de I^2C , que son controlados por una unidad central de proceso CPU mediante algún algoritmo, pero estos equipos no son nada amigables para su uso mejora o mantenimiento, por razones como:

- Posee una tarjeta de comunicación I^2C de manera interna, obstaculizando así su remplazo de manera rápida en caso de que se presente alguna falla en el equipo. Lo cual es muy probable.

- El actual equipo es muy susceptible a ruidos y altos voltajes provenientes del televisor o del exterior, en ocasiones se traba al recibir altas descargas del televisor, por lo cual se requiere de constante mantenimiento.

- Por poseer un microprocesador 486 no trabajaría correctamente con sistemas de niveles más altos como WINDOWS 95, NT o 98, por lo que lo hace obsoleto para aplicaciones en windows más completas.

- El ambiente de trabajo es en DOS y no se cuenta con programas fuentes.

Por todas las razones anteriores se ve la necesidad de desarrollar una interface para comunicación I^2C propia, lo cual beneficiaría grandemente a la implementación y desarrollo de equipo mucho más amigable, ambiente WINDOWS y se contaría con los programas fuentes para poder estar sujetos a cambios o mejoras del sistema. Además de tener la posibilidad de poder crear mas aplicaciones útiles a la manufactura y diseño.

2.2.- OBJETIVO.

El objetivo de este trabajo es el diseño y construcción de un sistema para comunicación I^2C y PC, que sea sencillo para su uso y mantenimiento, usándose en ambiente WINDOWS 3.1, 95 y versiones mas altas, suficientemente robusto a altos voltajes, que presente la oportunidad de seguir desarrollando mas equipo para la manufactura, cumpliendo con el estándar de I^2C y además presentar un bajo costo en su desarrollo.

2.3.- FUNCIONAMIENTO DEL PROYECTO.

El sistema consta de los siguientes elementos:

- Una tarjeta como interface externa que se conecta al puerto paralelo (LPT1) de la PC.

- Software necesario para I/O de datos por PC, y control el sistema.

  Archivo DLL para uso de I/O de PC.

  Programas de control de I^2C lectura y escritura de dispositivos.

- Software para la aplicación del sistema.

  En este caso se propuso un editor de NVM y un evaluador de dispositivos.

  siendo estas, dos de las mas sencillas.

La tarjeta lleva a cabo la comunicacion convirtiendo líneas bidireccionales en una entrada y una salida del puerto paralelo donde la línea de entrada y la de salida poseen diferentes direcciones de acceso en el puerto paralelo.

Para la comunicacion la interface consta de dos líneas bidireccionales, una es usada para direccionar mandar y recibir datos llamada SDA, y la otra es dedicada para la marca del tiempo cómo sincronía de comunicación SCL.

Además esta tarjeta cuenta con una tercera línea usada para interrumpir al microprocesador de la TV llamada INTR la cual es unidireccional.

Aparte de la interface el sistema necesita un programa para su control. Llamado CONTROL.BAS el cual esta hecho en VISUAL BASIC 5 , este software es capaz de usar las funciones de acceso a I^2C mas comunes, como mandar leer un dato de 8 bits de un dispositivo con o sin sub-dirección (sub address), además puede leer y escribir en un dispositivo NVM.

Para que este sistema pueda hacer uso de I/O , tiene que accesar al puerto paralelo mediante funciones de INPUT ,OUTPUT las cuales se encuentran contenidas en el archivo MYIODLL.DLL.

2.4.- ESQUEMA GENERAL DEL PROYECTO.

En el proyecto el programa de aplicación hace uso de el programa de control para poder accesar al I^2C , el cual contiene todas la funciones necesarias para lectura y escritura ayudandose del archivo DLL, y la tarjeta se encarga de mostrar el BUS I^2C en el puerto paralelo como dos simples localidades, una de lectura y otra de escritura , cada vez que se hace uso de alguna de las funciones de acceso al BUS se activa la señal de INTR la cual suspende las labores del microprocesador de la TV mostrando un esquema como en la fig.2a.

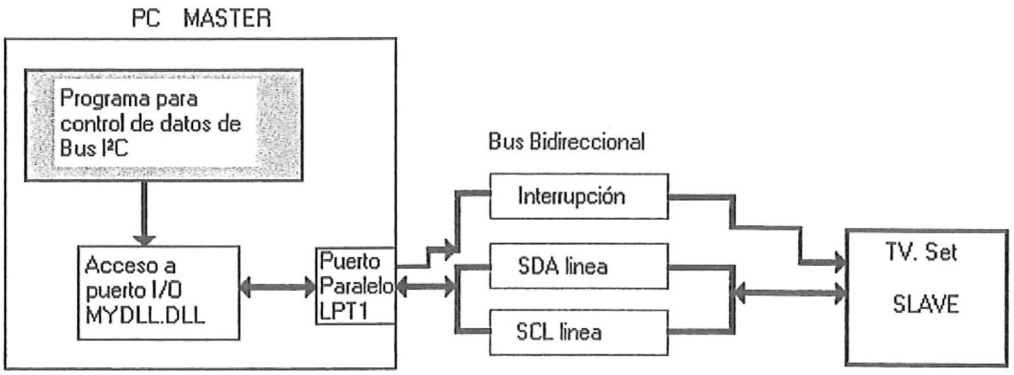


Fig. 2a.

3.- I^2C BUS ESPECIFICACIONES.

3.1.-PROTOCOLO DE COMUNICACIÓN I^2C BUS

En el desarrollo de Electrónica comercial, Telecomunicaciones y Electrónica industrial, muy seguido nos encontramos con similaridades entre diseños. Por ejemplo casi todos estos equipos poseen :

- Algún control inteligente, usualmente es un Chip MICROCONTROLADOR
- Circuitos de propósito general como Driver de LCD, Puertos de I/O remotos, RAM, EEPROM, o convertidores de datos.
- Circuitos de Aplicacion-Orientada Tales como: TUNER digital de TV, circuitos para procesamiento de señales para sistemas de radio y sistemas de video, o generadores DTMF para telefonía con marcación por tonos.

Y para explotar estas similitudes para el beneficio de diseñadores y manufactura Tanto como para maximizar la eficiencia de hardware y simplicidad del circuito, PHILLIPS desarrollo un BUS de solo 2 líneas bidireccinales para un control más eficiente ínter IC.

Este BUS es llamado Inter IC o I^2C BUS. Actualmente se cuenta con una gran cantidad de chips Bipolares o CMOS que hacen uso de este I^2C BUS en cada una de las tres categorías mencionadas previamente.

Propiedades del I^2C BUS:

- Solo usa dos líneas datos SDA y Reloj SCL.
- Cada dispositivo conectado al BUS es direccionado por software mediante una dirección única y una relación sencilla tipo MASTER/SLAVE existente todo el tiempo, donde los MASTER pueden ser Transmisor o receptor.
- Comunicacion serial de 8 bits, Comunicacion bidireccional puede hacerse hasta 100 Kbits /s en el modo standard o hasta 400 Kbits en modo rápido.
- Los chips poseen filtros internos para evitar ruidos en la linea para mantener la integridad de los datos.
- La cantidad máxima de IC s que pueden ser conectados a el mismo BUS esta limitada solamente por una capacitancia máxima de 400 pF.

3.2.- BENEFICIOS DE DISEÑO Y EN MANUFACTURA

BENEFICIOS AL DISEÑADOR.

Los integrados compatibles con el bus I^2C permiten un diseño de sistema que rapidamente se avanza desde ser un Diagrama a bloques funcional a ser un prototipo. Por otra parte desde que se van colocando estos IC's al bus sin necesidad de alguna interfase externa, se permite que el sistema sea modificado o actualizado simplemente colocando o quitando IC's directamente del bus.

A continuación se presentan algunas de las caracteristicas del Bus I^2C que son particularmente atractivas para los diseñadores.

- Los Bloques funcinales en los diagramas a bloques corresponden con los Actuales IC's; por lo que los diseños van rapidamente de un diagrama a bloques a esquemático final.

- No es necesario diseñar interfaces para Bus dado que la interface para I^2C ya se encuentra integrada dentro del chip.

- El direccionamiento integrado y el protocolo para transferencia de datos permite sistemas completamente definidos por Software.

- Los mismos tipos de IC's pueden ser utilizados en muchas aplicaciones diferentes.

- El tiempo de diseño se reduce tan rápido como los diseñadores se vayan familiarizando con los bloques funcionales usados frecuentemente, represenados por los IC's compatibles con I^2C.

- Los IC's pueden colocarse o remover de un sistema sin afectar otros circuitos del Bus.

- Diagnosticos de Fallas y Depuración son Sencillos; mal funcionamiento puede ser detectado rapidamente.

- El tiempo de desarrollo de software se reduce por biblotecas de Ensamblador o por sencillos modulos de software que son reusables .

- El consumo de corriente es extremadamente bajo.

- Alta inmunidad al ruido.

- Un gran rango de voltaje de alimentación

- Un gran rango de temperatura de operación.

BENEFICIOS PARA LA MANUFACTURA

Estos IC's no solo ayudan a diseñadores, tambien presentan características que benefician a la manufactura de equipos.

- Los IC's compatibles con I^2C son tan sencillos en su conectividad que resulta economico y fácil cualquier cambio en caso de actualización del producto.

- La capacidad multi-master del Bus I^2C permite Construcción de equipo de Prueba, el cual va directamente via externa conectado a una computadora.

- La disponibilidad de IC's compatibles con I^2C-bus son empaquetados del tipo SO (Small Outline), VSO (Very Small Outline) y DIL los cuales reducen el requerimiento de espacio.

- Dadas sus características, una familia entera de equipos puede ser desarrollada alrededor de un modelo básico, ya que para agregar o mejorar el equipo solo se agregan o se quitan IC's.

3.3.- ESPECIFICACIONES DE I^2C BUS.

Para aplicaciones de control digital de 8 bits, tales como aquellas que requieren el uso de microcontroladores, cierto criterio de diseño debe ser establecido:

- Un sistema completo usualmente consiste de al menos un microcontrolador y otros dispositivos periféricos tales como memorias y expansores de I/O.

- El costo de conectar varios dispositivos dentro del sistema debe ser minimizado.

- Un sistema que desempeña una función de control no requiere transferencia de datos a alta velocidad.

- La eficiencia completa depende de los dispositivos seleccionados y la naturaleza de la interconeccción de la estructura del Bus.

Para producir un sistema que satisfaga estos criterios, es necesario una estructura de serial bus. Ademas los serial Buses requieren de menos cableado y pocos pins de IC's. Como quiera que sea un Bus no es meramente alambre interconectado, sino que abarca todos los formatos y procedimientos para comunicación dentro del sistema.

Dispositivos comunicandose cada uno con el resto en un serial Bus deben tener alguna forma de protocolo, la cual elimina todas las posibilidades de confusión, datos perdidos y bolqueo de información. Los dispositivos rápidos deben estar dispuestos a comunciarse con dispositivos lentos. El sistema no debe ser dependiente de dispositivos conctados a el, de otra manera las modificaciones o mejoras serán imposibles. Se debe contar con un procedimiento para decidir cual dispositivo será el que tendrá el control del Bus y cuando. Y si diferentes dispositivos con diferentes velocidades de reloj están conecntados al Bus, la fuente del reloj debe ser definida.

3.4.- EL CONCEPTO I^2C BUS

El Bus I^2C soporta cualquier proceso de fabricación (NMOS,CMOS, Bipolar). Dos pistas: Datos serial (SDA) y Reloj serial (SCL), llevan la información entre los dispositivos conectados al Bus. Cada dispositivo es reconocido por una dirección única los cuales podrían ser un microcontrolador, un manejador de LCD, memoria o una interface de teclado - además puede operar como un transmisor o receptor, dependiendo del funcionamiento del dispositivo. Obviamente un manejador de LCD es solamente un receptor, pero en caso de una memoria puede ambas recibir y transmitir datos. En adición a transmisores y receptores, los dispositivos pueden ser también considerados como *master* o *slave* cuando se lleva a cabo la trasferencia de datos (Ver tabla 3a). un *master* es un dispositivo el cual inicia la transferencia de datos en el bus y genera las señales de reloj para permitir tal transferencia. Hasta este punto, cualquier dispositivo es considerado un *slave*.

TERMINO	
Transmisor	El dispositivo que manda la información al bus.
Receptor	El dispositivo que recibe la información del bus.
Master	El dispositivo que comienza una transferencia, genera señales de reloj y termina una transferencia.
Slave	El dispositivo direccionado por un master.
Multi – master	Más de un master puede tratar de controlar al bus al mismo tiempo sin interruptir el mensaje.
Arbitraje	Procedimiento para asegurar que, simultáneamente más de un master trata de controlar el bus, solamente uno tiene permitido hacerlo y el mensaje es interrumpido.
Sincronización	Procedimiento para sincronizar las señales de reloj de dos ó más dispositivos.

Tabla 3.a

El bus I^2C es un bus *multi-master*. Esto significa que más de un dispositivo capaz de controlar el bus puede ser conectado a él. Como *masters* usualmente encontramos microcontroladores, consideremos el caso de transferencia de datos entre dos microcontroladores conectados al bus I^2C (fig. 3.b). Esto resalta las relaciones *master-slave* receptor-transmisor encontradas en el bus I^2C. Debería de notarse que estas relaciones no son permanentes, pero solamente dependen de la dirección de la transferencia de datos en ese momento.

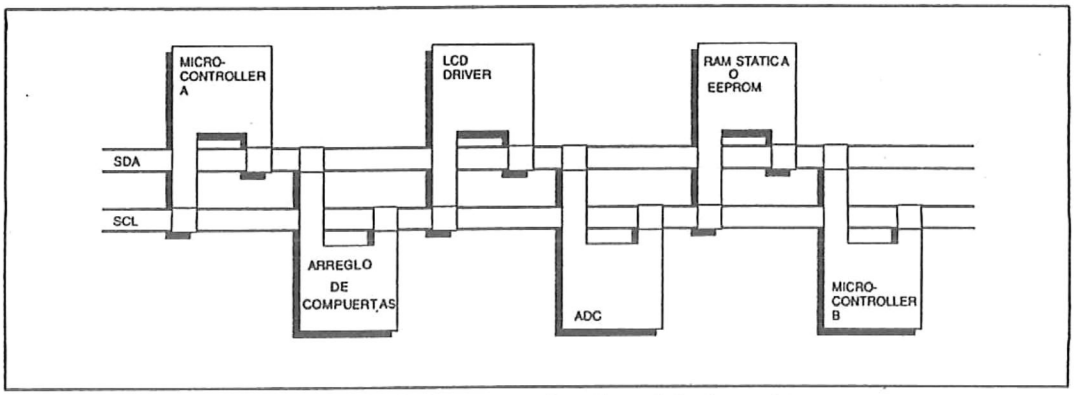


Fig. 3.b Ejemplo de una configuración usando dos microcontroladores

La transferencia de datos sería como se muestra a continuación:

1. Supongamos que el microcontrolador A quiere mandar información al microcontrolador B:
   - Microcontrolador A(master), direcciona al microcontrolador B(slave).
   - Microcontrolador A(master-transmisor), manda datos al microcontrolador B(slave-receptor).
   - Microcontrolador A termina la transferencia.
2. Si el microcontrolador A quiere recibir información del microcontrolador B:
   - Microcontrolador A(master) direcciona al microcontrolador B(slave).
   - Microcontrolador A(master-receptor) recibe datos del microcontrolador B(slave-transmisor).
   - Microcontrolador A termina la transferencia.

Aún en este caso el master (microcontrolador A) genera el reloj y termina la transferencia. La posibilidad de conectar más de un microcontrolador al bus I^2C significa que más de un master podría tratar de iniciar una transferencia de datos al mismo tiempo. Para evitar el caos que vendría por tal evento, un procedimiento de arbitraje ha sido desarrollado. Este procedimiento se basa en la conexión de AND-alambrada de todas las interfaces I^2C conectadas al bus.

Si dos ó más masters tratan de poner información en el bus, el primero en producir un "1" cuando el otro produzca un "0" el otro perderá el arbitraje. Las señales de reloj durante el arbitraje son una combinación sincronizada de los *clocks* generados por los masters utilizando la conexión AND-alambrada a la línea SCL.

La generación de las señales de reloj en el bus I^2C es siempre la responsabilidad de los dispositivos master; cada master genera sus propias señales de reloj cuando transfieren datos en el bus. Las señales de reloj del bus de un master pueden ser alteradas solamente cuando son alargadas por un dispositivo slave-lento hace hold-down en la línea de reloj, o por otro master cuando ocurre el arbitraje.

3.5- CARACTERÍSTICAS GENERALES

Ambas líneas SDA y SCL son líneas bidireccionales, conectadas a una fuente con voltaje positivo mediante una resistencia *pull-up* (ver fig. 3.c). Cuando el bus se encuentra libre, ambas líneas están en alto. Los niveles de salida de los dispositivos conectados al bus deben tener un open-drain ó open-collector, en orden para realizar una función AND-alambrada. Los datos en el bus I^2C pueden ser transferidos a una razón de hasta 100 kbits/s en el modo estándar ó hasta 400 kbits/s en el modo rápido. El número de interfaces conectadas al bus depende sólidamente en la capacitancia límite del bus que es de 400 pF.

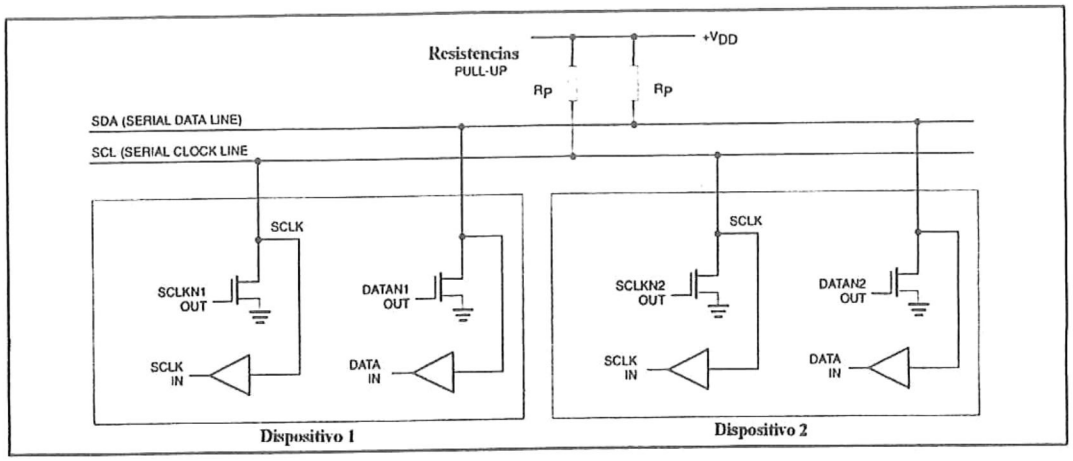


Fig 3.c

3.6.- TRANSFERENCIA DE BITS

Dada la variedad de las diferentes tecnologías de dispotivos(CMOS, NMOS, Bipolar), las cuales pueden ser conectar al bus I^2C, los niveles de "0" lógico (bajo) y "1" lógico (alto) no son fijos y dependen del nivel asociado de Vdd. Un pulso de reloj es generado por cada bit que es transferido.

3.6.1.- Validación de datos

Los datos en la línea SDA deben ser estables durante el periodo (alto) del reloj. Los estados *alto ó bajo* de la línea de datos podrían ser cambiados solamente cuando la señal de clock en la línea SCL se encuentra en *bajo* (ver fig. 3.d).

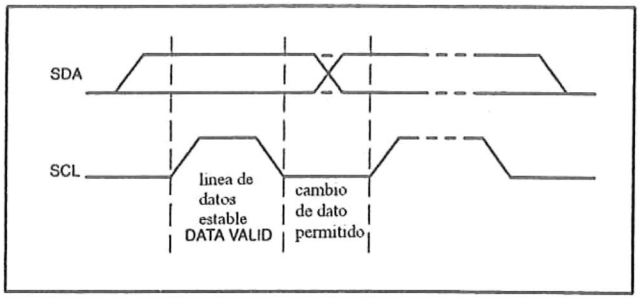


Fig. 3.d Transferencia de bit en bus

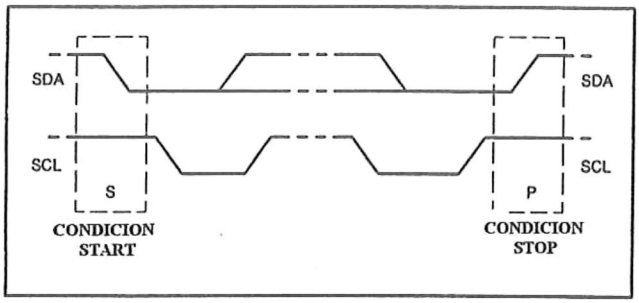


Fig 3.e Condiciones **START** y **STOP**

3.6.2.- Condiciones Start y Stop

Dentro del procedimiento del bus I^2C, situaciones únicas emergen las cuales se definen como condiciones START y STOP (ver fig. 3.e).

Una transición de alto a bajo en la línea SDA mientras la línea SCL está en alto es uno de estos casos únicos. Esta situación indica una condición START.

Una transición de alto a bajo en la línea SDA mientras la línea SCL se encuentra en alto define una condición STOP.

Las condiciones START y STOP son generadas siempre por el master. Se considera que el bus esta ocupado despues de una condición START y que está libre un cierto tiempo después del evento STOP.

La detección de las condiciones START y STOP por dispositivos conectados al bus es sencilla si ellos incorporan el hardware de interface necesario. Como quiera que sea los microcontroladores sin tal interface tienen que muestrear la línea SDA al menos dos veces por periodo de reloj para poder censar la transición.

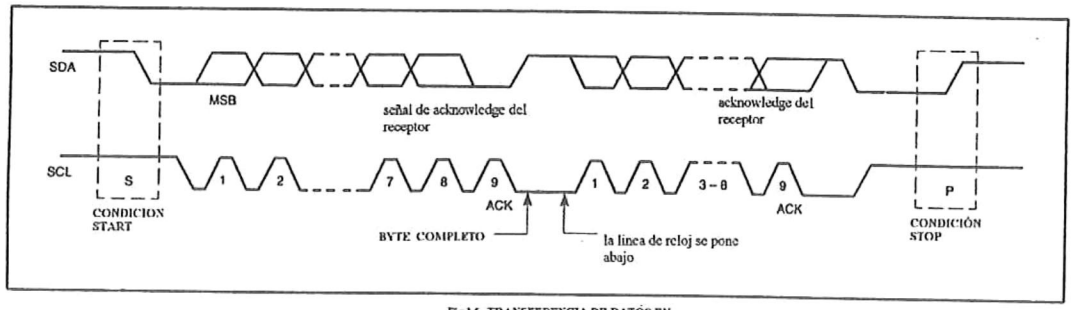


Fig.3.f TRANSFERENCIA DE DATOS EN BUS I²C

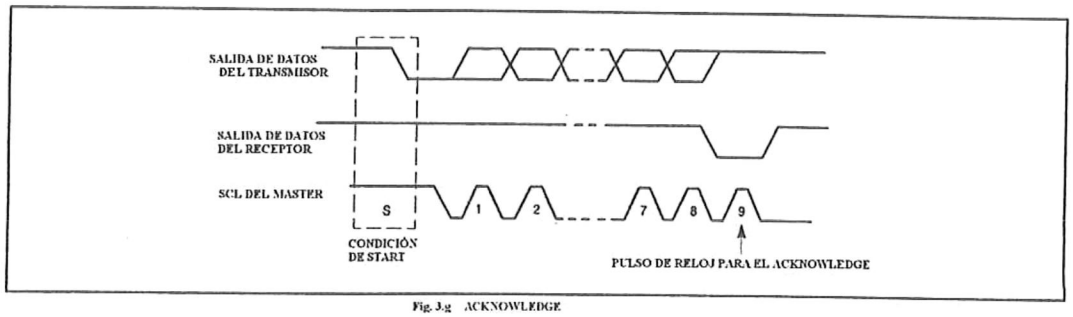


Fig. 3.g ACKNOWLEDGE

3.7.- TRANSFERENCIA DE DATOS

3.7.1.- Formato de bytes

Cada byte puesto en la línea SDA debe ser 8 bits de largo. El número de bytes que pueden ser transmitidos no tiene restricción. Cada byte tiene que ser seguido por un bit de *acknowledge*. Los datos son transferidos con el bit más significativo (MSB) al principio (ver fig 3.f). Si un receptor no puede recibir otro byte completo de datos hasta que haya hecho alguna otra función, por ejemplo sirviendo alguna interrupción interna, puede amarrar la línea de reloj SCL en *bajo* para forzar al transmisor hacia un estado de espera.

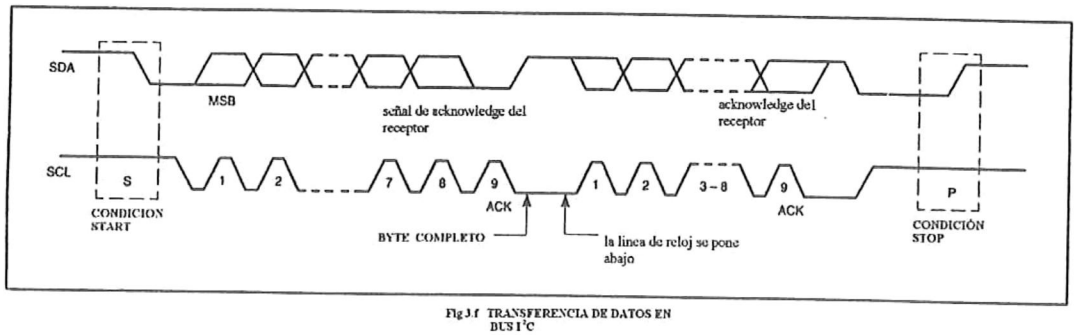


Fig 3.f TRANSFERENCIA DE DATOS EN BUS I²C

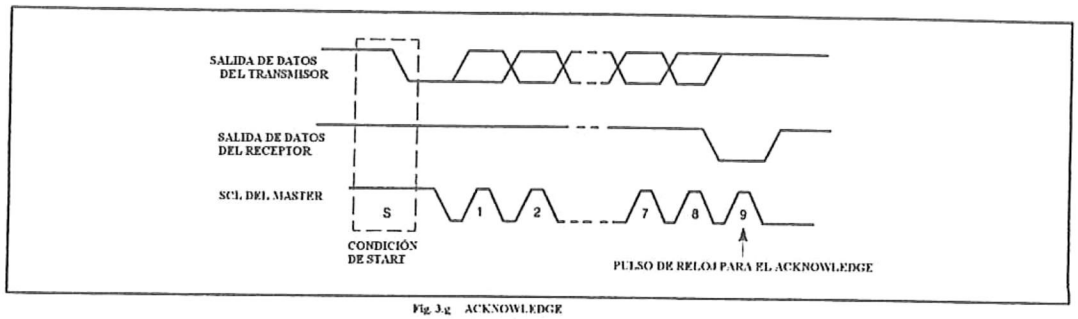


Fig 3.g ACKNOWLEDGE

La transferencia de datos entonces continua cuando el receptor está listo para otro byte de datos y suelta la línea de reloj SCL.

En algunos casos se permite usar un formato diferente que el formato I^2C bus (para dispositivos compatibles CBUS por ejemplo). Un mensaje el cual comience con tal dirección puede ser terminada por la generación de una condición STOP, aún durante la transmisión de un byte. En este caso, no es generado un *acknowledge.*

3.7.2.- Acknowledge

La transferencia de datos con acknowledge es obligatorio. El pulso de reloj de acknowledge-relacionado es generado por el master. El transmisor suelta la línea SDA en alto durante el pulso de reloj de acknowledge.

El receptor debe hacer *pull down* a la línea SDA durante el pulso del reloj de acknowledge para que se mantenga estable en bajo durante el periodo en alto de este pulso de reloj (ver fig. 3.g).

Usualmente, un receptor que ha sido direccionado es obligado a generar un acknowledge después de cada byte que ha sido recibido, excepto cuando el mensaje empieza con una dirección CBUS.

Cuando un *slave* receptor no reconoce una dirección de *slave* (por ejemplo, esta indispuesto para recibir por que está llevando a cabo alguna función en tiempo real), la línea de datos debe ser dejado en *alto* por el *slave.* El *master* puede entonces generar una condición de STOP para abortar la transferencia.

Si un *slave* receptor reconoce la dirección de *slave* pero, algún tiempo más tarde en la transferencia no se pueden recibir más bytes de datos, el *master* debe abortar nuevamente la transferencia. Esto es indicado por el *slave* no generando el acknowledge en el primer byte a seguir. El *slave* deja la línea de datos en *alto* y el *master* genera la condición STOP.

Si un *master* receptor está envuelto en una transferencia, debe señalar el fin del dato al *slave* transmisor no generando un acknowledge en el último byte que fue sacado del *slave*. El slave transmisor debe soltar la línea de datos para permitir que el master generé un STOP ó repita la condición START.

3.8.- CARACTEERÍSTICAS ELÉCTRICAS.

Los dispositivos para Bus I^2C con niveles de entrada fijos de 1.5 V y 3 V pueden cada uno tener su propia y apropiada fuente de voltaje. Unas resistencias de *Pull-Up* deben ser conectadas a una fuente de 5 V ± 10% (ver fig. 3.h).

Los dispositivos de Bus I^2C con niveles de entrada relacionados con Vdd deben tener una línea de alimentación común a la que la resistencia de *pull-up* también es conectada (ver fig.3.i).

Cuando dispositivos con niveles de entrada fijos están mezclados con dispositivos que tienen niveles de entrada relacionados a V_{DD}, los últimos dispositivos deben ser conectados a una fuente de alimentación común de 5 V ± 10% y deben tener resistencias de *pull-up* conectadas a sus pins de SDA y SCL como se muestra en la fig. 3.j.

Los niveles de entrada (input) son definidos de tal manera como se muestra a continuación:

- El margen de ruido en nivel LOW *(bajo)* es 0.1 V_{DD}
- El margen de ruido en nivel HIGH *(alto)* es 0.2 V_{DD}

- Como se muestra en la Fig.3.k. resistencias en serie (Rs)de 300 Ω pueden ser usadas para protección contra Picos de alto voltaje (High-Voltage spikes) en las lineas de SDA y SCL (como por ejemplo por el flash-over de un cañón de T.V.)

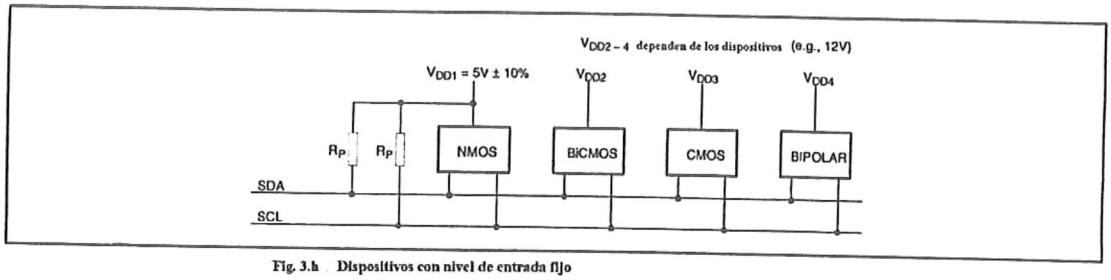


Fig. 3.h Dispositivos con nivel de entrada fijo

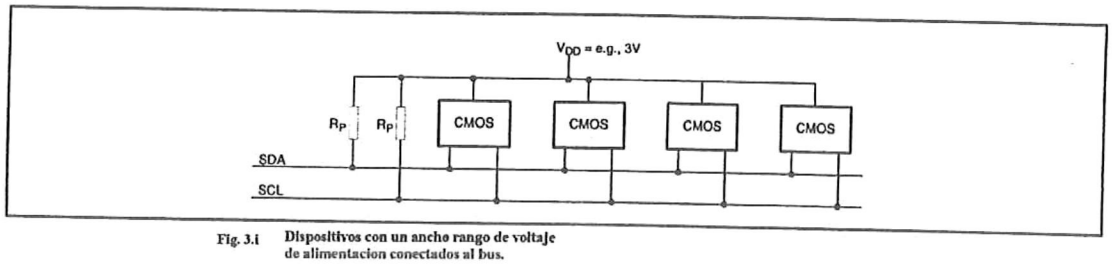


Fig. 3.i Dispositivos con un ancho rango de voltaje de alimentacion conectados al bus.

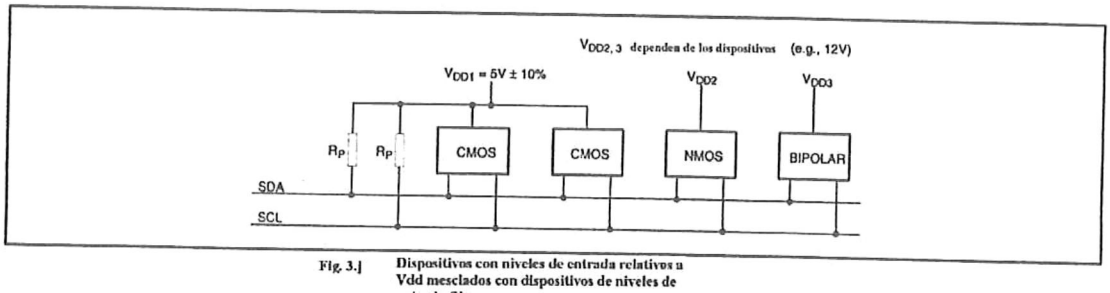


Fig. 3.j Dispositivos con niveles de entrada relativos a Vdd mesclados con dispositivos de niveles de entrada fijos.

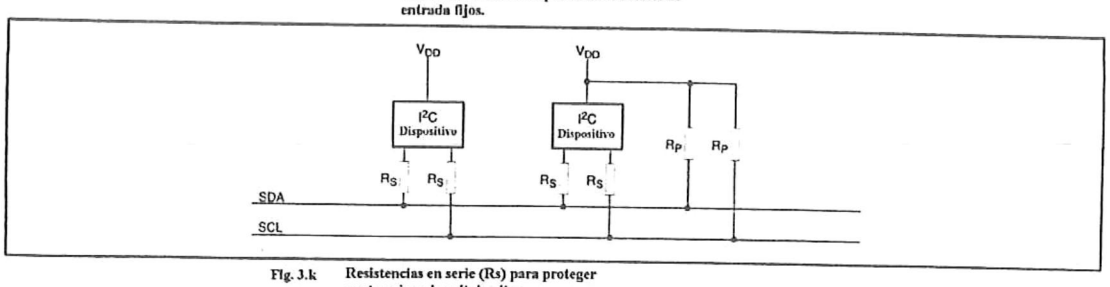


Fig. 3.k Resistencias en serie (Rs) para proteger contra picos de voltaje altos

3.8.1.- Máximo y Mínimo valor de resistencias Rs y Rp.

Para el modo standard los dispositivos del bus I^2C , los valores de las resistencias Rp y Rs en la fig. 3.l. dependen de los siguientes parámetros:

- Fuente de voltaje
- Capacitancia de el Bus.

La fuente de voltaje limita los valores máximo y mínimo de la resistencia Rp debido a la corriente mínima de 3 mA con V_{OLmax}=0.4 V para los niveles de salida. V_{DD} como una funcion de Rp es mostrada in la fig.3.m. El margen de ruido deseado de 0.1 Vpp fala el nivel de bajo (LOW), limita el valor máximo de Rs. Rsmáx. como una función de Rp es mostrada en la Fig. 3.n.

La capacitancia del Bus es la capacitancia total de cableado y pistas, conecciones y pins. Esta capacitancia limita el valor máximo de Rp debido al tiempo de respuesta especificado. En la Figura 3.n se muestra a Rp max como una función de la capacitancia del Bus.

La corriente maxima en el nivel de ALTO (HIGH) de cada conexión de Entrada/salida (Input/output) tiene un valor máximo especificado de 10μA. Debido a el margen de ruido deseado de 0.2 Vpp para el nivel de ALTO (HIGH), esta corriente de entrada limita el valor máximo de Rp. Este limite depende de V_{DD} . La corriente de entrada total de nivel ALTO (HIGH) esta mostrada como función de R_{pmax} en la fig.3.p.

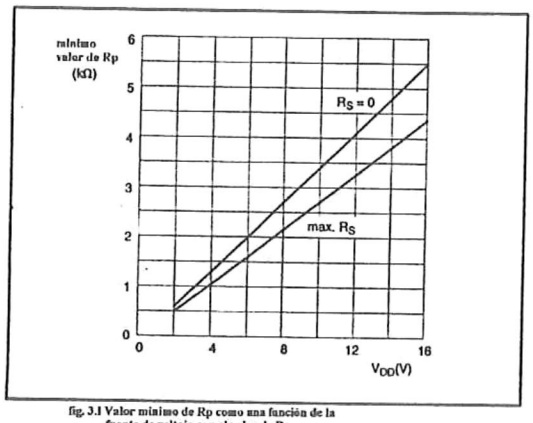


fig. 3.l Valor mínimo de Rp como una función de la fuente de voltaje con el valor de Rs como un parametro

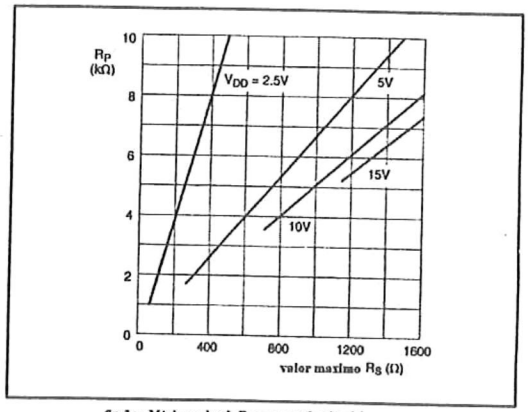


fig 3.m Máximo valor de Rs como una función del Rp con la fuente de voltaje como un parametro

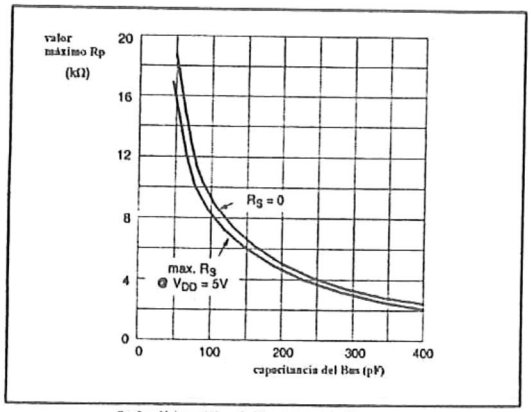


fig. 3.n Valor máximo de Rp como una función de la capacitancia del bus .

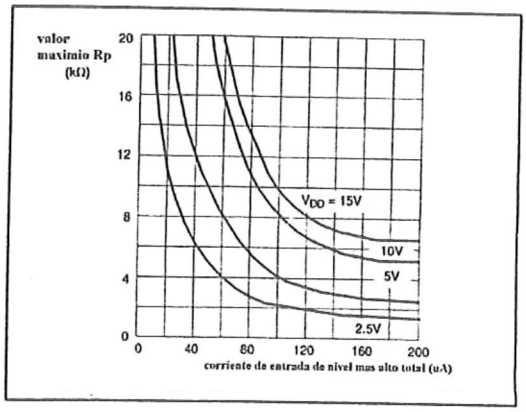


fig. 3.o Corriente de entrada de nivel mas alto total como una función del valor máximo de Rp con la fuente de voltaje como parametro.

4.-MÓDULO DE INTERFACE.

4.1.- SISTEMA GENERAL.

El hardware consiste en dos líneas de comunicación para el bus que muestra la información del bus a la PC , mostrando salidas de *open collector* hacia el bus, y además cuenta con la línea de interrupción *INT* que permite tomar como esclavo al microcontrolador de la T.V.

Cada una de estas líneas protege a ambas partes (PC y TV) de posibles descargas de voltaje, esto lo hace con ayuda de *opto-acopladores* , en este caso el integrado ECG3094 el cual consta de dos compuertas NAND con *open collector* de alta velocidad TTL.

El acceso a este circuito es mediante el puerto paralelo con las siguientes direcciones:

0: &h378

1: &h278

2: &h3bc

Cada una de estas direcciones cuenta con tres registros *data, status* y *control.* En los cuales se depositarán los datos que se enviarán ó se leerán al Bus de la TV.

La manera en que el sistema obtendrá los datos será haciendo uso del registro de *status* con la dirección base seleccionada para accesar al puerto, y para enviar datos se hará uso de los registros *data* y *control*

.

- Para la línea SDA de salida será el registro *data* por el pin 9.
- Para la línea SCL de salida será el registro *control* por el pin 17.
- Para la línea SDA de entrada será el registro *status* por el pin 11.
- Para la línea SCL de entrada será el registro *status* por el pin 15.
- Para la línea INT será el registro *data* por el pin 8.

Al poseer estos registros se puede separar la señal proveniente del Bus en dos líneas. De esta manera se puede accesar a estos registros usando un *programa de control* para el sistema, el cual se encargará de interpretar y acomodar el flujo de la información.

4.2.- CONDICIONES DE USO.

- Este circuito operará de manera externa a la PC lo cual nos indica que se utiliza una fuente de alimentación regulada con un voltaje de +5 volts.

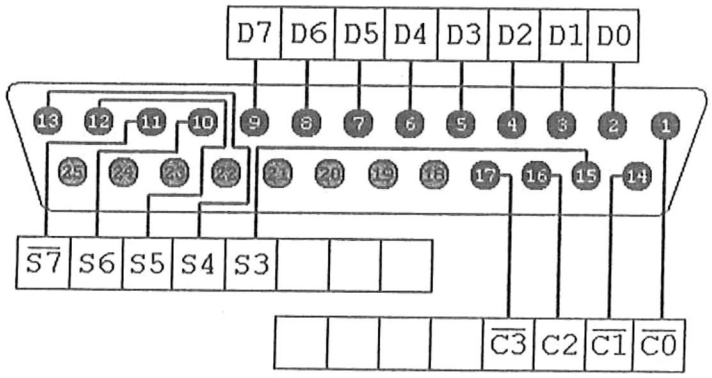
- La fuente siempre debe estar encendida siempre que se desee tener acceso al Bus.

- El sistema debe contar con un programa de control que pueda interpretar la información del Bus, tomando en cuenta que se han descompuesto en dos las líneas SCL y SDA.

- No se debe utilizar cableado muy largo entre el conector Molex4p de la tarjeta y el conector del Bus de la TV, para asegurar los datos es mejor un cable corto. Y deben de evitarse capacitancias parásitas que afecten a nuestro bus.

4.3.- PUERTO PARALELO DE PC.

La vía de acceso a la tarjeta del Bus es usar el area del BIOS en 40:8 que es el primer puerto paralelo LPT1 a partir de la cual inician los puertos paralelos disponibles los cuales tambien pueden referenciarce normalmente como puertos de PC 278h, 378h, 3bch, y cada uno de estos puertos posee 3 registros: *data, status* y *control*. La siguiente figura muestra la distribución de los pins en conector hembra DB-25.



Conector Db 25 Hembra

Fig. 4.a

- 8 pins de salida accesados por el registro de *data.*
- 5 pins de entrada (uno invertido) accesados por el registro *status.*
- 4 pins de salida (tres invertidos) accesados por el registro *control.*

Las direcciones de estos registros son obtenidas a partir de la base que se ha seleccionado para trabajar:

Registros -NO DISPONIBLE

	7	6	5	4	3	2	1	0	I/O Port
DATA	x	x	x	x	x	x	x	x	Base = 278/378/3BC Hex
STATUS	$\overline{x}$	x	x	x	x	-	-	-	Base+1
CONTROL	-	-	-	-	$\overline{x}$	x	$\overline{x}$	$\overline{x}$	Base+2

Nota: S7, C0, C1 & C3 estan invertidos

Fig 4.b

Para poder dividir cada una de las líneas se tomo lo siguiente:

- Para la línea SDA de salida será el registro *data* por el pin 9.
- Para la línea SCL de salida será el registro *control* por el pin 17.
- Para la línea SDA de entrada será el registro *status* por el pin 11.
- Para la línea SCL de entrada será el registro *status* por el pin 15.
- Para la línea INT será el registro *data* por el pin 8.

Por ejemplo si el programa de control desea poner en alto la línea SDA se estaría refiriendo al registro de *data* y al pin 9:

```
MyOutPort &h378,127 'SDA en alto
```

Y en caso que se quisiera leer la línea SDA sería el registro *status* y el pin 11:

```
Dato%= MyInPort (&h379)   'lee línea SDA
```

4.4.- ANALISIS DE UNA LINEA DEL BUS.

La tarjeta en sí consta de un circuito básico el cual es aplicado para ambos sentidos, para cada línea se obtienen dos sentidos (In, Out). A continuación se muestra la línea SDA del Bus, mostrando su división en Entrada (SDA In) y Salida (SDA Out).

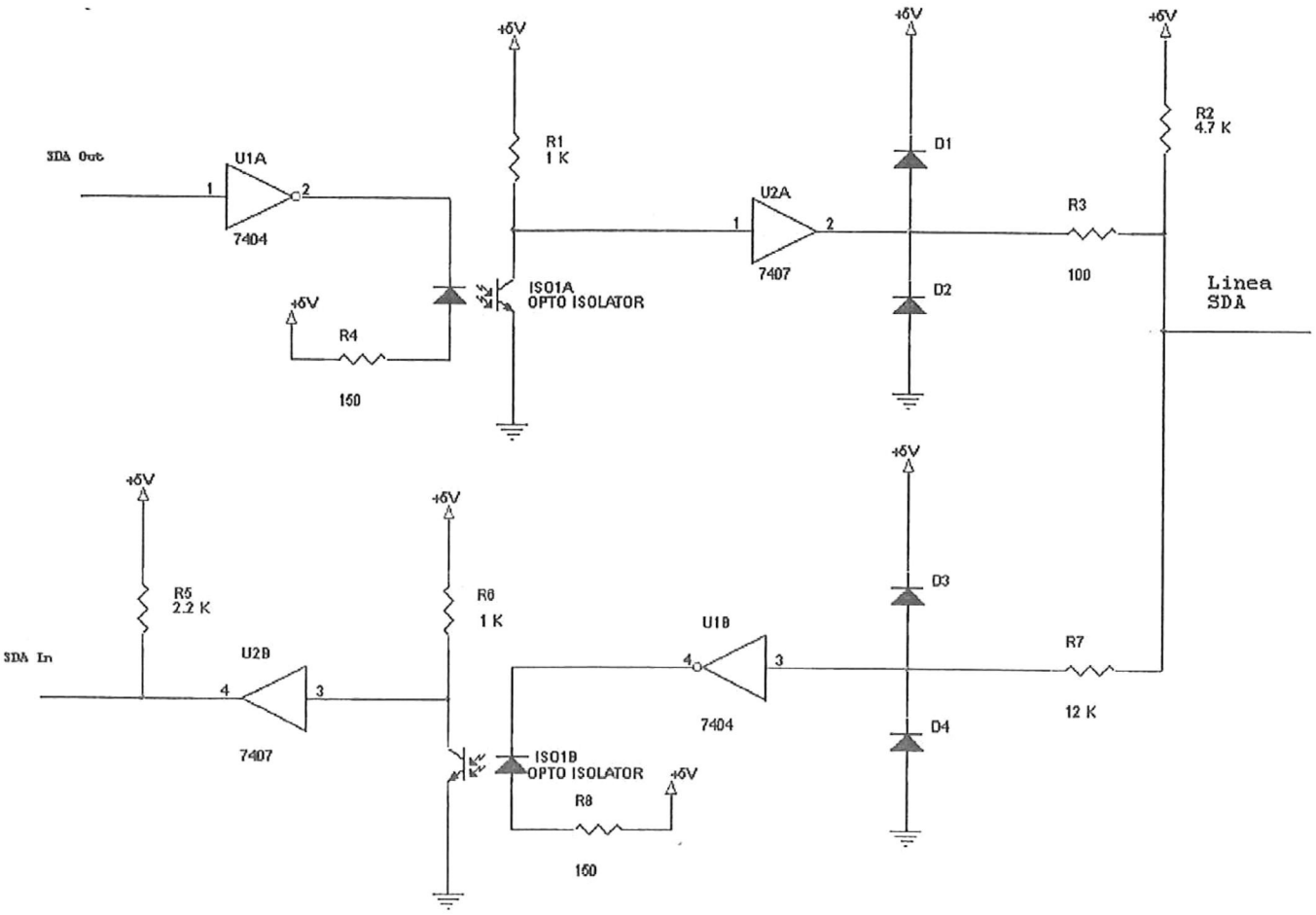


Fig 4.c

En la figura 4.c se puede observar como la línea SDA es una terminal común que va al bus directamente y después es descompuesta en dos líneas SDA in y SDA out las cuales van directamente conectadas al puerto Paralelo de PC.

Ambas líneas poseen diodos en el extremo que va conectado al punto común, esto para evitar que picos de corriente provenientes de la TV, afecten gravemente la circuitería , además poseen las resistencias Rp de pull-up por sus característica de colector-abierto ,también tiene la resistencia en serie Rs que sirve para proteger picos de alto voltaje.

Cada una de las líneas posee un *optoacoplador* el cual tiene la función de aislar nuestro circuito de grandes voltajes, para el caso de este proyecto se escogió como optoacoplador el integrado ECG 3094 , que es dispositivo opto-aislador con salida TTL.

Características de ECG 3094

- Es dual (contiene dos compuertas NAND)
- Posee la característica de Salida a Colector abierto.
- Posee alta velocidad de respuesta (75ns).
- Puede aislar hasta 3000 volts.
- Máxima corriente de salida 16 mA por canal

La *línea SDA* Out va conectada al pin #9 del puerto paralelo, por lo que al recibir los datos estos serán invertidos inmediatamente por la compuerta 7404 , después pasarán a través del optoacoplador pero este no invertirá el dato ya que el led esta polarizado directamente y solo activará la salida de colector cuando este led este apagado que es cuando se tenga un '0' en la línea SDA out , por lo que se puede decir que la línea SDA out invierte el dato enviado a esta línea. Después pasarán a través de la compuerta 7407 que dará la característica de colector abierto, aquí el flujo de datos es de la PC a la TV.

La línea SDA in, va conectada al pin # 11 que es un pin negado , por lo que será necesario que el programador contemple todo lo de programación necesario para que la aplicación se lleve a cabo, posee las mismas características de flujo de datos que la línea anterior.

4.5.- DIAGRAMA COMPLETO.-

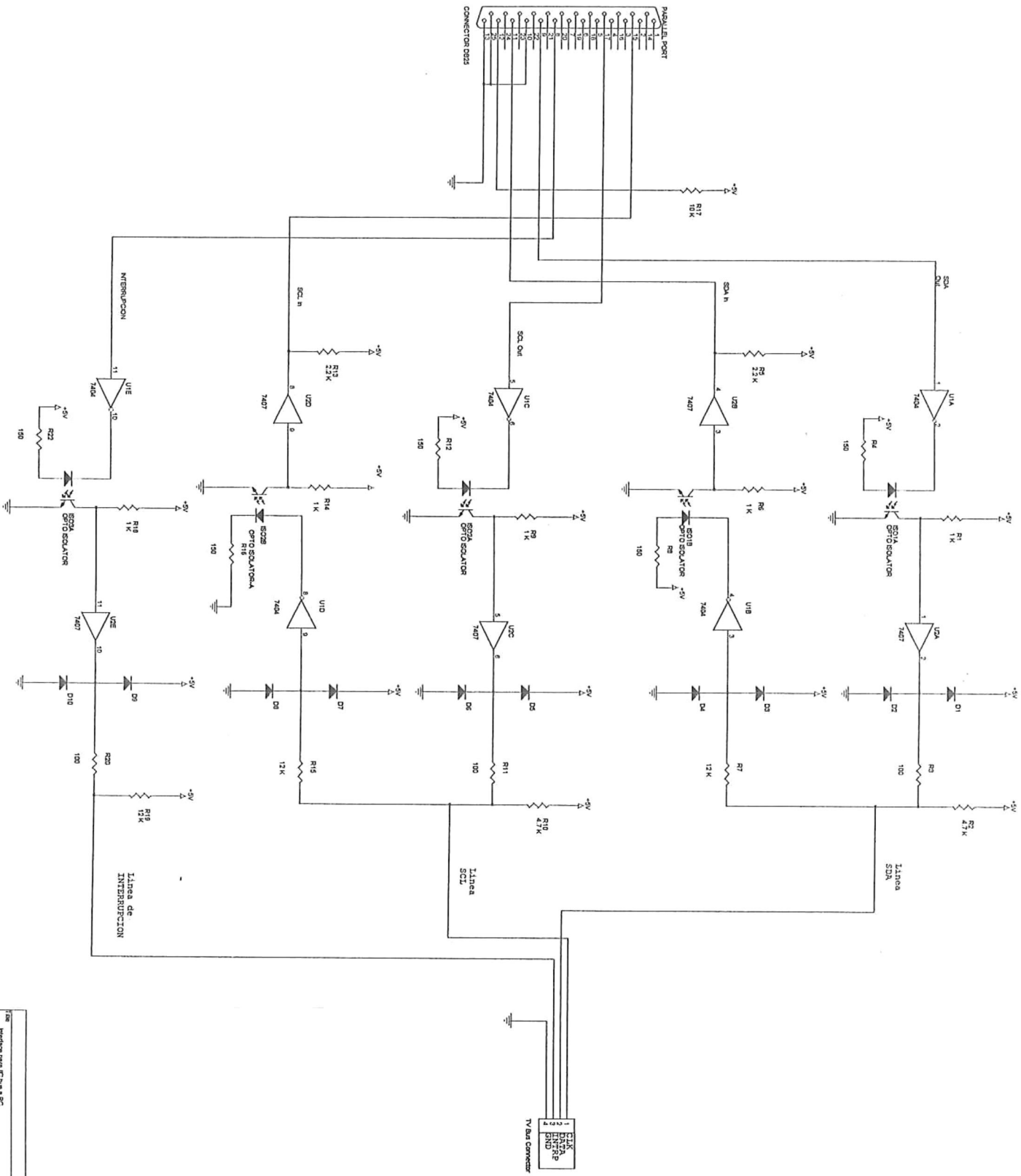
Pues al unir las dos líneas SDA y SCL obtenemos el diagrama completo que se puede observar en la siguiente figura. Pero debe recordarse de que algunas líneas se encuentran invertidas, pero por programación es corregido este problema.

Las líneas invertidas son:

La línea SDA in.

La línea SCL out

Observe que en línea SCL in el LED se encuentra conectado al revés que los demás, y esto se debe a que la línea esta invertida y para corregir esto se tuvo que invertir la respuesta de este optoacoplador 2B.



Title: Interface para IIC bus a PC			
Size: C	Document Number: Carlos Ruben Aguilar Barroso		Rev: 1
Date: Friday, September 17, 1999	Sheet 1 of 1		

5.- CREACIÓN DE DLL PARA USO DE I/O EN PC

5.1.- NECESIDAD DE DLL PARA ENTRADAS Y SALIDAS DE LA PC.

Cuando se desea tener un control de hardware en el Bus, implica la necesidad de una forma de acceso a este hardware de la PC, es entonces cuando se requiere de un driver.

Como es sabido, VISUAL BASIC no tiene la capacidad de tener acceso directo al I/O del hardware. Las instrucciones estándar de BASIC INP y OUT no están disponibles en Visual Basic, por lo que nos vemos en la necesidad de crear un driver con funciones que reemplacen esas dos instrucciones dejando que Visual Basic tenga acceso a bajo nivel del hardware de I/O.

La solución para este problema es la creación de un DLL (Dinamic Link Library, Bibliotecas de Vínculos Dinámicos), que son librerías que desde un principio se propusieron para sistemas operativos Windows, para soportar funciones utilizadas comúnmente. Los descendientes de Windows, incluyendo Windows NT, Windows 95 y 98, así como OS/2 también dependen de DLL's para proporcionar una gran parte de su funcionalidad.

En general, los DLL's son solo colecciones de funciones en una biblioteca. Sin embargo, al contrario de sus primas estáticas (archivos **.lib**), los DLL's no se vinculan directamente a los archivos ejecutables mediante el editor de enlaces. En lugar de esto solo se incluye información de referencia en el archivo ejecutable. Después, el código de la biblioteca se carga en tiempo de ejecución. Esto permite que varios procesos compartan bibliotecas en memoria, reduciendo así la cantidad de memoria requerida para ejecutar diferentes aplicaciones que comparten muchas de las mismas bibliotecas y ayudando a mantener manejable el tamaño de los archivos .EXE.

Para fines de este proyecto, y como el lenguaje que se escogió es Visual Basic se necesita un driver que proporcione las funciones para acceso al I/O, además de que este driver tiene que brindar libertad en programación y a su vez permitir la generación de un programa .EXE más compacto por razones de mejor manejo de memoria RAM y que aparte deje abierta la posibilidad de que otras personas tengan fácil comprensión para mejorar el archivo fuente de la aplicación.

5.2.- CREACIÓN DE UN DLL.

Para crear un DLL, se tomó como base el lenguaje de **Borland** C++ para las versiones de la 3.1 en adelante, crear un DLL resulta bastante sencillo, solo necesita de algunas variaciones sobre la generación de un código ordinario en C++, con algunas otras variantes para su compilación en el *setup* del compilador, como seleccionar plataforma de trabajo y modificación de archivos .def.

Básicamente los DLL's se dividen por plataformas de trabajo en dos grupos, los que son para aplicaciones de 16 bits y los que son para 32 bits. En este caso se harán accesos al hardware de 8 bits por el I/O del puerto paralelo, y se brindará la opción para futuras aplicaciones con 16 bits en dependencia de modificaciones al hardware.

Una vez que se seleccionó la plataforma de trabajo, se puede iniciar el desarrollo del código el cual se basa en exportar funciones que pueden ser utilizadas en aplicaciones pensadas para esa plataforma de trabajo.

Para que las aplicaciones sean capaces de utilizar las funciones de su DLL, cada función debe tener una entrada en la tabla de exportación del DLL.

En aplicaciones de 32 bits, para hacer que el compilador agregue una entrada a la tabla de exportaciones para una función, tiene dos opciones:

- Puede exportar funciones utilizando el modificador *_declspec* (*dllexport*) delante de todas las declaraciones de sus funciones.

- Utilizando archivos de definición de módulos (**.def**) lo cual da mayor control sobre el proceso de exportación de funciones, lo cual se utiliza mas a menudo que el método *_declspec*.

Archivos de definición de módulos

La sintaxis de los archivos .def en C++ es muy directa, particularmente porque la mayoría de las opciones mas complicadas que se usaban en versiones anteriores no son aplicables en Win32.

En estos archivos .def solo basta con proporcionar el nombre y la descripción de la biblioteca, seguida por una lista de funciones que se exportan, por ejemplo, para este proyecto tendría una estructura básica como:

```
LIBRARY    "MYDLL"
DESCRIPTION     'MY DLL ejemplo de librería DLL'
EXPORTS
     MyInitialize
     MyOpenPort      @3   NONAME
     MyInpPort       @4
     MyOutPort       @5
     MyClosePort
```

Puede especificar un número ordinal para una función añadiéndolo a la línea de la función en la sección `EXPORTS` con `@`. Este ordinal después puede utilizarse en llamadas a `GetProcAddress()`. En realidad el compilador asignará ordinales a todas las funciones exportadas, pero la forma en que esto se hace es algo impredecible si no se especifican los ordinales explícitamente.

Además, la opción `NONAME` indica al compilador que no incluya el nombre de la función en la tabla de exportación del DLL. En algunos casos, esto puede ahorrar mucho espacio en el archivo DLL

En aplicaciones de 16 bits, también puede exportar las funciones de dos maneras de tener una entrada en la tabla de exportación del DLL:

- Haciendo uso del modificador *_export* en el prototipo de la función.

En este proyecto sería algo como:

```
extern "C" int FAR PASCAL _export MyInpPort(int puerto)
```

- Utilizando los archivos .def que se explicaron con anterioridad.

`LibMain` es un buen lugar para alojar y liberar memoria global para un DLL. `LibMain` es además bueno para inicializar variables globales en un DLL. En este proyecto se utilizó la manera tradicional de hacer uso del `LibMain`, para inicializar variables y alojar memoria global haciendo:

```
     Int FAR PASCAL LibMain (HANDLE hInstance, WORD wDataSeg,
WORD wHea pSize, LPSTR lpszCmdLine)
     {
          if (wHeapSize >0 ) UnlockData (0);
          return 1;
     }
```

Manejo de memoria en DLL's (Uso de #pragma)

A diferencia de las bibliotecas estáticas que se convierten en parte efectiva del código de una aplicación, las bibliotecas de vínculos dinámicos en las versiones de Windows anteriores a Win32 manejaban la memoria de manera diferente. Bajo Win16, la memoria DLL se mantenía fuera del espacio de direcciones de una tarea, proporcionando la posibilidad de compartir memoria entre tareas mediante la memoria global de un DLL compartido.

En Win32 la memoria del DLL se asigna dentro del espacio de memoria del proceso que la carga. Cada proceso obtiene su propia copia de la memoria "global" del DLL, que se reinicializa cuando un nuevo proceso carga el DLL. Esto significa que el DLL no puede utilizarse para compartir memoria entre procesos de la misma forma en que se hacía bajo Win16.

Sin embargo, puede sacarse de la manga algunos trucos con el segmento de datos del DLL que le permitirán crear una sola sección de memoria que sea compartida por todos los procesos que utilizan el DLL.

En este proyecto la variable `puerto` es de tipo `Int` y se quiere que sea utilizado por todos los procesos que carguen el DLL. Esto se hace de la siguiente manera:

```
#pragma data(".myseg")
   int puerto;
   //mas variables compartidas
   //mas variables compartidas
#pragma data()
#pragma comment(lib, "msvcrt" ".section:.myseg,rws");
```

Todas las variables que se declaren entre los `pragma data_seg` serán ubicadas en el segmento llamado `.myseg`. El `#pragma comment` no es solo un comentario en el sentido tradicional; en realidad indica a la biblioteca de tiempo de ejecución de C que marque a su nueva sección con permisos de lectura, de escritura y de ser compartida.

`#pragma argsused`

Aparte se cuenta con **`#pragma argsused`** la cual solo afecta a la siguiente función y su efecto es desabilitar la aparición de el mensaje de precaución **`"Parameter 'parameter' is never used"`** , " `argsused`" pragma es permitida solamente entre definición de funciones.

PROPUESTA DE DRIVER PARA I/O

Con toda la teoría explicada anteriormente podemos construir un driver bastante funcional y muy básico que se encargará de dar el acceso al hardware de PC ayudándonos de algunas librerías ya existentes en C++, y así sustituir a las tradicionales funciones INP y OUT y tendríamos lo siguiente:

```
//MYDLL.CPP -->MYDLL.DLL para I/O de Hardware de PC
// por Carlos Ruben Aguilar Benson
//Este ejemplo correrá para Borland C++ 3.1 y versiones
//mas altas.
// NOTA:   MyInpPort=INP      MyOutPort=OUT
//         no es necesario alterar el archivo .def
#include <bios.h>
#include <stdio.h>
#include <stdio.h>
#include <windows.h>
#include <dos.h>
#pragma argsused

Int FAR PASCAL LibMain (HANDLE hInstance, WORD
wDataSeg,WORD wHeapSize, LPSTR lpszCmdLine)
{
     if (wHeapSize >0 ) UnlockData (0);
     return 1;
}

extern "C" int FAR PASCAL _export MyInpPort(int Puerto)
{
  int lectura;

   lectura=inportb(puerto);
   return(lectura);
}
extern "C" int FAR PASCAL _export MyOutPort(int puerto,
    int datout)
{
   outportb(puerto,datout);
   return 0;
}
```

El driver propuesto anteriormente hace uso de las funciones Inportb() y Outportb() ya existentes en C++ dentro de sus librerías .lib pero se podría ahorrar el uso de estas librerías utilizando un poco de subrutinas en lenguaje ensamblador que seria bastante mejor ya que el programa Ocuparía menos memoria al crear el .EXE

5.3.-USO DEL DLL EN VISUAL BASIC.

Una vez creado el driver para I/O (MYDLL.DLL) entonces se pueden empezar a desarrollar aplicaciones que manejen el Hardware, en este caso se hará desde Visual Basic.

Para hacer uso de las funciones contenidas en la librería se tiene que crear un módulo que contenga la declaración de las funciones que se desea usar, la declaración de las funciones `MyInpPort` y `MyOutPort` en este proyecto fueron:

```
Declare function MyInpPort lib "A:\MYDLL.DLL"
                (byval puerto as integer) as integer

Declare sub MyOutPort lib "A:\MYDLL.DLL"
           (byval puerto as integer,datout as integer)
```

Así Visual Basic buscará por MyInpPort y MyOutPort en A:\MYDLL.DLL
A partir de este momento ya es posible hacer uso de estas funciones y tener acceso de I/O a nuestro hardware.

6.- PROGRAMA PARA CONTROL DEL SISTEMA

6.1.- REQUERIMIENTOS DEL PROGRAMA

Una vez construida nuestra tarjeta de interface y teniendo acceso a este hardware mediante el driver que se desarrolló en el capítulo 5 (MYDLL.DLL), podemos dar paso al desarrollo de un software, el cual nos ayudará para llevar a cabo las tareas de comunicación I^2C.

Y los requerimientos que debe cumplir este software son:

- Que adecúe la transferencia de datos para el protocolo de comunicación I^2C Bus y brinde asi la posibilidad de hacer lecturas y escrituras a los dispositivos internos de la T.V.

- El software de control debe dar a la PC la imagen de un *master*, es decir, los algoritmos de las funciones de este software deben simular la lógica y características que posee un *master* en el protocolo de comunicación I^2C Bus.

- Debe poseer las funciones básicas de comunciación I^2C como: lectura con o sin *sub address*, escritura con o sin *sub address*.

- Debe ser capaz de tolerar diferentes dispositivos, como por ejemplo, CPU y NVM's y otros dispositivos con diferentes características internas que podrían alterar un poco el protocolo de comunicación.

- Debe en todo momento tener un buen control del reloj, ya que de esto depende, la rapidez y la eficiencia en la transferencia de datos.

- Debe trabajar con un protocolo de comunciación a 8 bits.

- Debe brindar la opción para modificar un factor de incremento en el tiempo de reloj, para que así el programador pueda hacer ajustes al tiempo y la aplicación se adecúe a las necesidades.

6.2- ESQUEMA GENERAL

Este programa posee las funciones básicas para cumplir con el protocolo de I^2C, como generar la condicion START , STOP , Dar ACKNOWLEDGE, espera de ACKNOWLEDGE , Transmitir un Byte al Bus, Recibir un Byte del Bus.

Aparte de estas funciones básicas también posee otras mas estructuradas que se apoyan de las otras funciones básicas para llevar a cabo tareas como: Leer un byte de un dispositivo Slave, escribir un byte a un dispositivo Slave, Escribir o leer a un dispositivo con un Sub address, generar un Multi-Read que lea un bloque de datos de un dispositivo Slave, etc.

También posee funciones que controlan el uso de el puerto paralelo y Bus, para tareas como Selección de puerto, inicializar puerto y Cerrar puerto.

6.3 FUNCIONES DEL PROGRAMA

Rutinas para manejo del Bus

BusOpen: Abre un canal de comunicación al Bus I^2C. Este comando se usa antes de mandar cualquier otro comando al Bus.

Sintaxis: `BusOpen x,y`

`x: 0,1 o 2`
Esto selecciona uno de los 3 puertos paralelos
0: &h378
1: &h278
2: &h3bc

y: 0 o 1
Cambia el estado del monitor del Bus
0: desactivar el monitor del Bus
1: activar el monitor del Bus

BusClose: Cierra el canal de comunicación. El Bus será liberado y el contenido de los registros del puerto será almacenado a su contenido original antes de que el *BusOpen* fuera usado.

Sintaxis: `BusClose`

BusInit: Este comando inicializa una nueva comunicación con el Bus. Este comando ademas limpia el Bus. Todos los dispositivos en el Bus serán reiniciados.

Sintaxis: `BusInit`

BusSetTimeOut: Pone un valor para el Time Out .

Sintaxis: `BusSetTimeOut x`

Esto hace que la rutina BusWaitForAck escanee 22 veces por la señal ACK. Si no se ha recibido ACK la rutina será abortada.

Rutinas Básicas para el protocolo

BusStart: Genera la condición *Start* en el Bus

Sintaxis: `BusStart`

BusStop: Genera la condición *Stop* en el Bus.

Sintaxis: `BusStop`

BusGiveAck: Manda la señal de *ACK* al *slave*

Sintaxis: `BusGiveAck`

BusWaitForAck: Espera por la señal de *ACK* del *slave*. Si el esclavo no responde despues de un tiempo (TimeOut) entonces un mensaje es depositado en el monitor del Bus.

Sintaxis: `BusWaitForAck`

BusTransmit: Transmite un byte al Bus. Esto puede ser dirección o dato.

Sintaxis: `BusTransmit dato`
dato no puede ser mas alto que 255 (&hFF) ya que el televisor sólo maneja hasta este rango.

BusReceive: Lee un byte del Bus.

Sintaxis: `BusReceive`
El resultado de la lectura es almacenado en *BusDato*

Rutinas complejas

Estas rutinas están construidas a partir de las rutinas básicas para el protocolo.

BusRead: Lee un byte de un dispositivo *slave*

Sintaxis:
```
BusRead address%
Print BusDato
```
Donde `address%` es una dirección válida de lectura entre 0 y 255. El dispositivo `address%` será accesado en modo de lectura y un byte será proporcionado. El resultado queda almacenado en `BusDato`.

BusWrite: Escribe un byte al dispositivo esclavo

Sintaxis: `BusWrite address%,data%`
Donde `address%` y `data%` son valores de un byte (0..255). El dispositivo en la dirección `address%` será accesado en modo de escritura y `data%` será transmitido.

Buswwread: Bus I^2C con escritura lectura. Este comando accesa un dispositivo con un subaddress para tomar datos de el.

Sintaxis: `Buswwread address%,subaddr%`

Buswwsend: Bus I^2C con escritura escritura. Escribe a un dispositivo con un subaddress.

Sintaxis: `Buswwsend address%,subaddr%,data%`

Ejemplo: `Buswwsend 64,2,44`
Esto escribirá el dato 44 al registro 2 del dispositivo 64

6.4.- ALGORITMOS DE LECTURA Y ESCRITURA

Algoritmos para Rutinas Básicas

- Generar condición *start*

```
inicio
   poner línea SDA en alto   'registro base+0 con 127
   retardo
   poner línea SCL en alto   'registro base+2 con 8
   retardo
   poner línea SDA en bajo   'registro base+0 con 255
   retardo
   poner línea SDA en bajo   'registro base+0 con 255
   retardo
   poner línea SCL en bajo   'registro base+2 con 0
   retardo
fin
```

- Generar la condición *stop*

```
inicio
   poner línea SDA en bajo   'registro base+0 con 255
   retardo
   poner línea SDA en bajo   'registro base+0 con 255
   retardo
   poner línea SCL en alto   'registro base+2 con 8
   retardo
   poner línea SCL en alto   'registro base+2 con 8
   retardo
   poner línea SDA en alto   'registro base+0 con 127
   retardo
fin
```

- Generación de *acknowledge* para el dispositivo *slave*

```
inicio
   poner línea SCL en bajo   'registro base+2 con 0
   retardo
   poner línea SDA en bajo   'registro base+0 con 255
   retardo
   poner línea SCL en alto   'registro base+2 con 8
   retardo
   poner línea SCL en bajo   'registro base+2 con 0
   retardo
   poner línea SDA en alto   'registro base+0 con 127
   retardo
fin
```

- Abortar transmisión y limpiar Bus

```
inicio
   for Busa%=0 to 5
        poner línea SDA en bajo 'registro base+0 con
                                '255
        retardo
        poner línea SCL en alto 'registro base+2 con 8
        retardo
        poner línea SDA en alto 'registro base+0 con
                                '127
        retardo
        poner línea SCL en bajo 'registro base+2 con 0
        retardo
fin
```

- Recibir un byte del *slave*

```
inicio
   BusDato%=0
   poner línea SDA en alto   'registro base+0 con 127
   retardo
   for Busa%=8 to 0 step -1
        Busb%=2^Busa%
        poner línea SCL en alto 'registro base+2 con 8
        retardo
        lee línea SDA (reg. base+1) y guarda en BusIn
```

```
        hacer BusIn=BusIn AND 128
        poner línea SCL en bajo  'reg. Base+2 con 0
        retardo
        si BusIn=128 entonces
             BusDato%=BusDato+Busb%
        next Busa%
  fin
```

- Transmitir un byte al Bus

```
Inicio
     Busa%=0
     For Busa%=7 to 0 step-1
          Busb%=2^ Busa%
          Si (dato% AND Busb%)= Busb% entonces
               poner línea SDA en alto
               Retardo
          Si no poner línea SDA en bajo
               Retardo
          Poner línea SCL en alto
          Retardo
          Poner línea SCL en bajo
          Retardo
     Next Busa%
     Poner línea SDA en alto  'liberar línea SDA
     Retardo
```

- Esperar por acknowledge del *slave*

```
Inicio

  Poner línea SDA en alto
  Poner línea SDA en alto
       Retardo

  Poner línea SCL en alto
  Poner línea SCL en alto
       Retardo

  Acknowledge=0
  Acktimer=0
  Mientras acknowledge=0
       Acktimer=acktimer+1
```

```
        Busmon=(Dato leído en la línea SDA)AND 128
        Si Busmon < > 128 entonces
             Acnowledge=1
        Si acktimer=BusTimeOut entonces
             Acknowledge=1
        Poner línea SDA en alto
        Poner línea SDA en alto
             Retardo
        Poner línea SCL en bajo
        Poner línea SCL en bajo
    Fin
```

- Escribir un byte al *slave*

```
Inicio

        Adr%=adr% AND 254 'asegurar que la
                          dirección es par
        BusDevadr%=adr%
        Generar condición start
        Transmitir byte por el bus(adr%)
        Esperar por acknowledge
        Busdevadr%=0
        Transmitir byte por el bus (dato)
        Esperar por acnowledge
        Generar condición stop
Fin
```

- Leer con *subaddress*

```
Inicio

        Adr%=adr% or 1 'asegurar que adr es impar
        Busdevadr%=adr%
        Generar condición start
        Transmitir byte por el bus (adr%)
        Esperar por acknowledge
        Busdevadr%=0
        Transmitir byte por el bus (subadr%)
        Esperar por acknowledge
        Generar condición start
        Adr%=adr% + 1
        Transmitir byte por el bus (adr%)
        Recibir byte por el bus
        Generar condición stop
```

```
        Cargar byte leído=busdato%
        En Buswwread
    Fin
```

- Escribir con *subaddress*

```
Inicio

        Adr%=adr% AND 254 'asegurar que es par
        Busdevadr=adr%
        Generar condición start
        Transmitir un byte por el bus (adr%)
        Esperar por acknowledge
        Busdevadr=0
        Transmitir un byte por el bus (subadr%)
        Esperar por acknowledge
        Transmitir un byte por el bus (dato%)
        Esperar por acknowledge
        generar condición stop
Fin
```

7.-APLICACIONES DEL SISTEMA.

A continuación se muestran algunas aplicaciones, las cuales son herramientas de uso común en el departamento de diseño de software.

7.1- EDITOR DE NVMs (editor de Memoria no volatil NVM).

Con este sistema se puede obtener un Editor de Memorias tipo NVM (Non-Volatile-Memory). Esta memorias son ya muy comunes en aplicaciones digitales para almacenar informacion ya que la conserban incluso sin necesidad de mantener presente un voltage Vcc de alimentacion, y es por esta razon que una de sus aplicaciones mas comunes es dentro del diseño de un televisor , ya que un televisor tiene que guardar toda su programación de colores tamaños convergencia volumen, efectos de sonido ajustes y muchas cosas mas que son programadas por el diseñador o datos dados por el usuario, esto aún siendo que el televisor se encuentre desconectado.

Exixten muchas marcas en el mercado las cuales poseen algunos tipos de memorias NVM , ademas existen varios tipos de acceso a estos tipos de memoria, pero la mas común es por **I^2C Bus** , siendo La marca Phillips la primera en desarrollar Memoria NVM con este tipo de acceso.

Como se mencionó anteriormente con este sistema (funciones de comunicación y tarjeta para la interfaz), se puede desarrollar un editor de memorias tipo NVM, y este software contaría con las siguientes características:

- Capaz de escribir y leer a la memoria NVM
- Brindar la opción de seleccionar páginas de memoria NVM
- Poder almacenar el buffer de datos obtenidos de todas la páginas de la NVM en un archivo.
- Poder transferir los datos del editor de memorias hacia una hoja de Microsoft EXCEL.
- Reportar errores de comunicación.

Este editor proporciona los siguientes beneficios :

- Se pueden evaluar datos en los diferentes dispositivos internos de una TV ya que cada uno de estos guarda sus datos en NVM y hace un refresh de estos datos cada que se inicia un nuevo cuadro de la imagen.
- Se pueden hacer evaluaciones para los diferentes datos iniciales de la TV.
- Se pueden editar o generar parches (PATCH) de software para corregir BUGS de programación.
- Se puede utilizar como herramienta de desarrollo de software para monitorear los datos en las direcciones del mapa de memoria NVM, usándose a la par con el emulador del microprocesador.

La siguiente fig.7a muestra el editor de memorias de NVM Cuando ha leído la pagina Slave address A0:

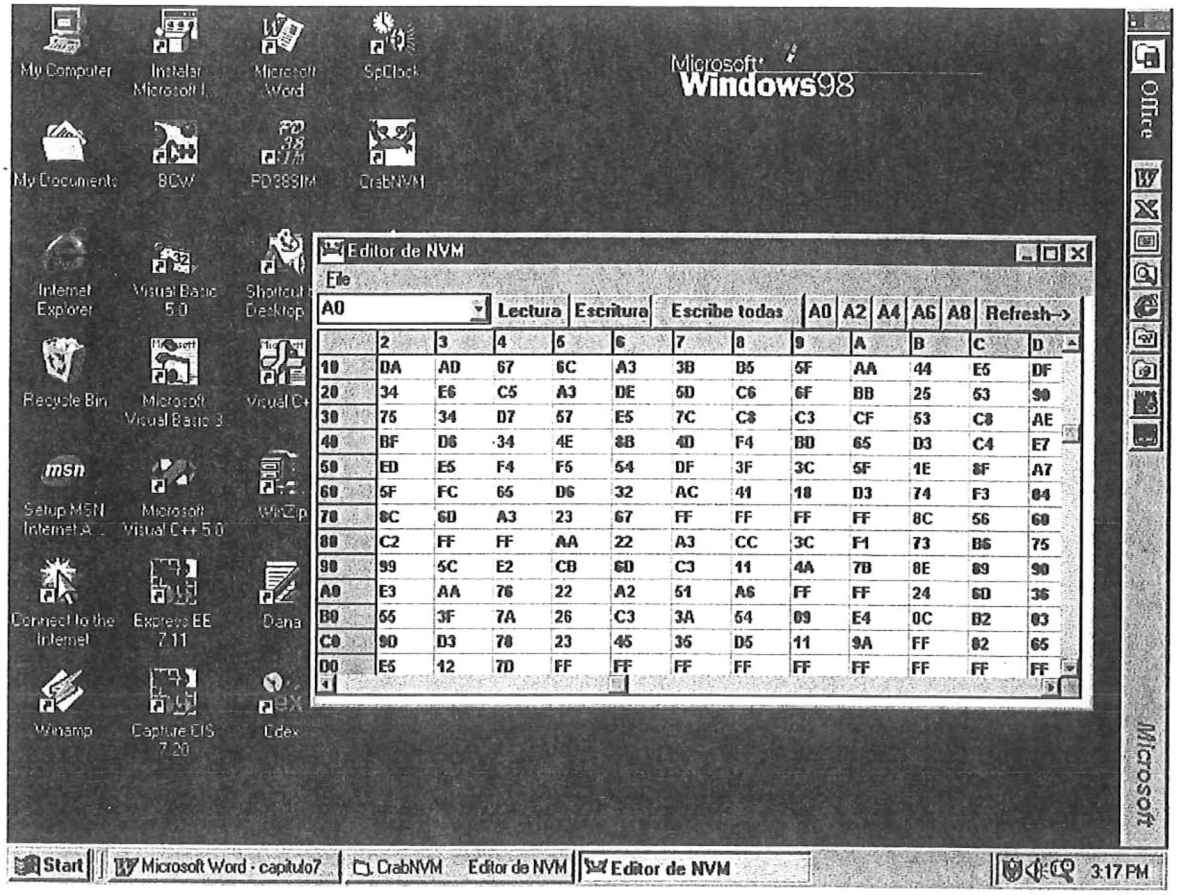


Fig 7a

La sig. Fig. 7b muestra las opciones validas para este editor de memorias NVM.

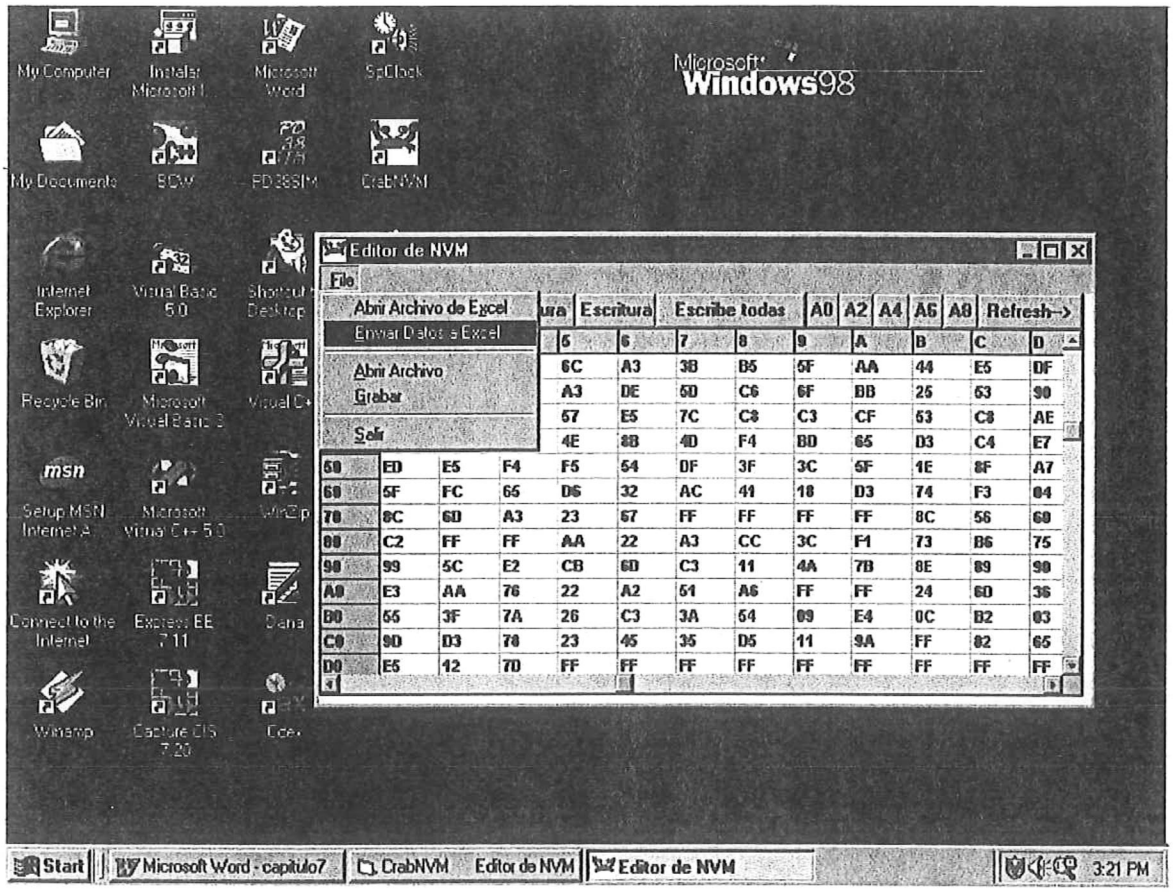


Fig. 7b

En la opción ***Grabar***, pasamos los datos actuales del buffer de todas las paginas de la NVM a un archivo con extensión *.NVM el cual se encuentra almacenado en un formato propio de este editor y no es compatible con otras aplicaciones.

La opción de ***Enviar datos a EXCEL*** , manda la pagina de Actual selección a una hoja de EXCEL para la manipulación de los datos a manera de reporte.

7.2 EVALUADOR DE DISPOSITIVOS

Esta aplicación es una muestra mas como herramienta de diseño ya que al poseerla es muy sencillo poder evaluar el comportamiento de los diversos dispositivos involucrados en el proyecto a desarrollar.

En el caso del diseño de un televisor , se pueden evaluar gran variedad de dispositivos directamente con esta herramienta de diseño tales dispositivos pueden ser:

(por mencionar solo algunos)

- PIP Processor (Picture in Picture Processor).

  Mitsubishi M65669 Processor Slave address:24h
- Audio Processor.

  JRC NJM1130G Processor , slave address:82h
- Video Processor.

  Sony CXA2155S YC jungle (trinorma), slave address:88h
- Comb Filter.

  NEC MPD64082 3D comb filter device.slave address:B8h
- Tunner.

  SONY WA412 tunner device,slave address:C6h
- A/V Switches digitales.

Dicho evaluador de dispositivos debe contar con la petición de:

-Slave address.

-Sub Address.

-Dato a Evaluar.

La sig. Fig.7c muestra el Evaluador de dispositivos y mostrando el uso del dispositivo Video Processor (88 Slave address, escribiendo un dato de E3 al registro 19 sub address). Siendo que en esta dirección de sub address se encuentra la medida horizontal de la TV.

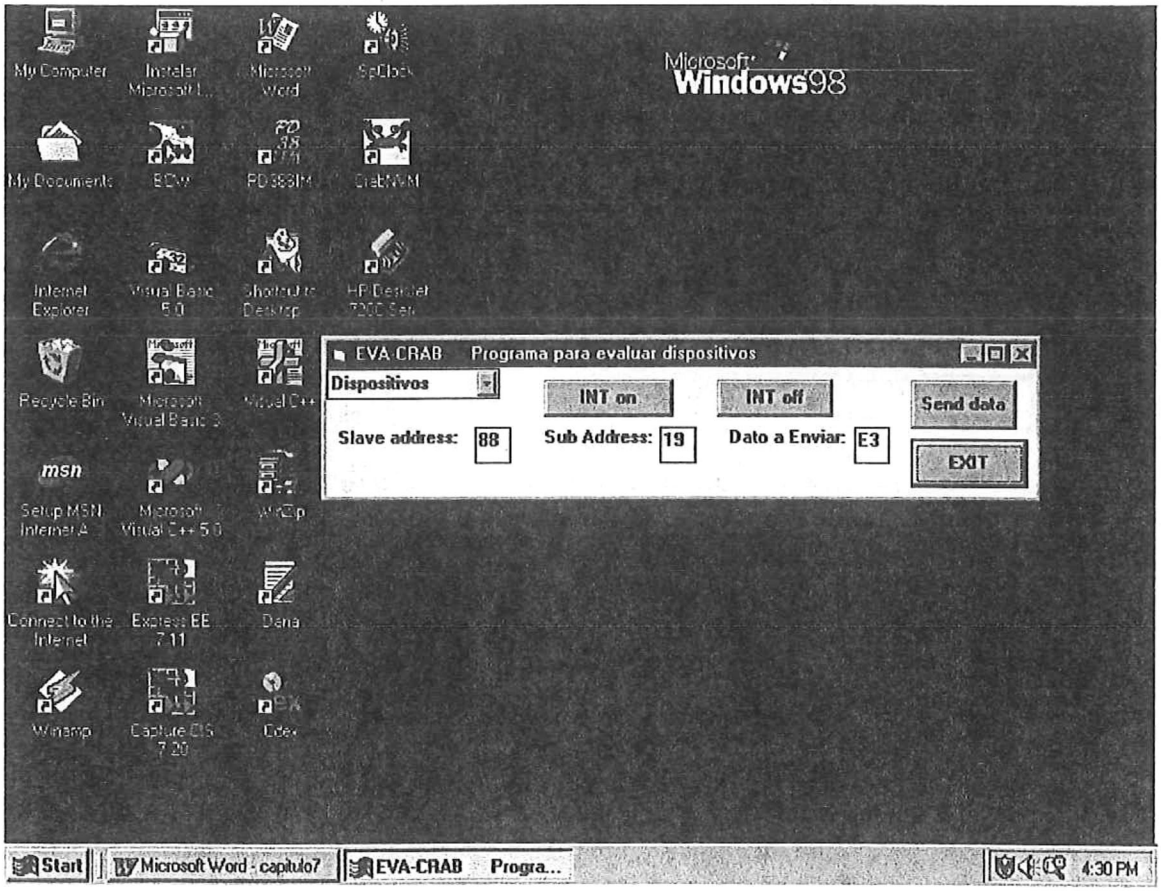


Fig 7c

8.-CONCLUSIONES.

Dada la facilidad de programación que aporta El lenguaje Visual Basic utilizando estas biblotecas y la libertad que brinda el utilizar las funciones INP OUT del DLL, se pueden obtener gran variedad de aplicaciones a las áreas de Diseño de sistemas de prueba.

A su vez se Pueden seguir desarrollando herramientas de Diseño para el área de desarrollo de software, con aplicaciones muy especificas para cada dispositivo involucrado en el desarrollo del diseño según sea el caso.

Las Biblotecas prestan gran facilidad de modificación si se presentara la necesidad de tener que modificarlas para adaptarlas según sea el caso.

Por lo que esta investigación y desarrollo resultó en una rica fuente de información para el desarrollo de futuros proyectos que hagan uso del I^2C bus, ya que actualmente no se contaba con suficiente conocimiento sobre este tipo de protocolos de comunicación entre dispositivos pues Puede direccionar mandar y recibir datos.

9. BIBLIOGRAFÍA.

I^2C Bus Notes

Phillips Semiconductors Document

1998 Update

C++ Manual de referencia

Herbert Schildt

Mc Graw Hill, 1994

Visual C++ 5 para desarrolladores

David Bennett

Sams,Prentice Hall , 1998

Visual Basic 5

Nathan Gurewish

Sams,Prentice Hall

10.- APÉNDICE

Este estudio proporciona la realización de un sistema de comunicación entre el bus de la TV SONY (o cualquier otra que haga uso de **I^2C bus**) y una PC, con fines de desarrollo de software en El area de Diseño o aplicación en sistemas de prueba al producto en las líneas de producción.

Haciendo uso de una sencilla programación en Visual Basic, utilizando un DLL para I/O desarrollado en C, se puede lograr la construcción de la aplicación deseada, brindando mucha libertad en el acceso al **I^2C Bus** mediante una Sencilla interfaz.

GLOSARIO

DLL: (Dinamic Link Library) Biblotecas de funciones que originalmente se propusieron para sistemas operativos Windows, para soportar funciones utilizadas comúnmente.Los descendientes de Windows, incluyendo Windows NT, Windows 95 y 98, asi como OS/2 también dependen de DLL's para proporcionar una gran parte de su funcionalidad.

I^2C: De sus siglas en inglés (Inter IC) , Es un protocolo de comunicación bidireccional que sólo hace uso de dos líneas de transmisión: una línea de datos y otra para reloj.

NVM: (Non volatile memory) Memoria que puede conservar datos aún cuando no cuente con la fuente de voltaje.

SCL: Línea bidireccional usada para la transmisión de pulsos de reloj en el bus I^2C.

SDL: Línea bidireccional usada para la transmisión de datos en el bus I^2C.

Sub-address: Sub Dirección o sub localidad de un dispositivo, esta puede ser una localidad de memoria o un registro de un dispositivo.

Slave-address: Dirección del dispositivo esclavo en la comunicación.

Video processor: (Microprocesador de video) Dispositivo encargado de todas las tareas en el control del Video en una TV.