UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DISEÑO E IMPLEMENTACIÓN DE UN OSCILOSCOPIO DIGITAL DIDÁCTICO, UTILIZANDO DISPOSITIVOS LÓGICOS PROGRAMABLES COMPLEJOS (CPLD) CON INTERFAZ VGA HACIA UN MONITOR DE COMPUTADORA

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO MECÁNICO ELECTRICISTA
ÁREA ELÉCTRICA Y ELECTRÓNICA

P R E S E N T A N :

AÑORVE GARDUÑO ALEJANDRO
JUÁREZ MARTÍNEZ GUMERSINDO
OTERO CARRETO ALFONSO SALVADOR
VARGAS AMADOR CARLOS ROBERTO
VELÁZQUEZ HERNÁNDEZ FRANCISCO

DIRECTOR DE TESIS

M. en I. JORGE VALERIANO ASSEM

MÉXICO, D.F. 2002

DISEÑO E IMPLEMENTACIÓN DE UN OSCILOSCOPIO DIGITAL DIDÁCTICO, UTILIZANDO DISPOSITIVOS LÓGICOS PROGRAMABLES COMPLEJOS (CPLD) CON INTERFAZ VGA HACIA UN MONITOR DE COMPUTADORA

Alejandro Añorve Garduño
Alfonso Salvador Otero Carreto
Carlos Roberto Vargas Amador
Francisco Velázquez Hernández
Gumersindo Juárez Martínez

Director de Tesis:

M. en I. Jorge Valeriano Assem

A Dios, por todos los dones que he recibido de Él.

A Víctor y a Carmen, quienes desde el día en que nací me han brindado todo su cariño y su apoyo, y nunca han dejado de estar orgullosos de mis logros. Gracias por ser los padres perfectos para mí.

A Lupita: Por tener que soportar todo este tiempo pasado –y los años que tenemos por delante– a su eterna rival y mi dulce compañera: mi pasión por aprender.

A todos y cada uno de los maestros que han compartido su tiempo conmigo. Gracias por su ejemplo, por enseñarme lo que debo y lo que no debo hacer.

A todos mis familiares y amigos que jamás dejaron de creer que algún día llegaría al final de este sueño. Gracias a todos aquellos que supieron conservar el trozo de corazón que algún día les compartí.

Y por último a Jorge, nuestro asesor de tesis, y a mis compañeros de equipo en la realización de este trabajo. Gracias por soportar mis ratos de enfado y por todo el esfuerzo que le dedicaron al proyecto.

A Dios: Por darme la luz de la vida
y llenarme de bendiciones.

A mis padres porque en la unidad de su amor y su esfuerzo colosal, han posiblitado todas mis ilusiones. Gracias por brindarme ese hermoso rincón de bugambilias y eucaliptos.

A mis hermanos Rubén y Noé, por su ejemplo de coraje y dedicación, por sus cuidados, por los momentos que me brindaron para que yo aprendiera de ellos.

A mis familiares, amigos y maestros, porque a lo largo de mi trayectoria sus consejos me han ayudado a lograr esta meta.

Y por último a mis hijos que aunque aún no los conozco, son inspiración de todo lo que aprendo.

¡Que Dios los bendiga!

Cuán a menudo pensamos
Que somos nosotros los que
Cortamos el hielo y paleamos la nieve
Para abrirnos paso hasta **Dios**...
Cuando todo el tiempo
Él estaba
Derritiendo hielo en nuestros corazones,
Y soplando la arena de nuestros ojos.
¡ Él estaba abriendo paso hacia nosotros!

Con Amor y Veneración
Para **Margarita y Porfirio**, que con su amor
Y apoyo incondicional han hecho posible la
realización de este sueño.

A **Lorena**, mi hermana, por serlo en todo momento.

A mi querida sobrina **Yuritzi**,
con el deseo de un futuro prominente.

Con cariño y admiración
Para **Ma. Del Carmen y Servando**, quienes creyeron en mi
Y con sus consejos me mantuvieron en el camino.

Para **Alma, Ángela y Jorge**, como un acto de fe.

Al M. I. **Jorge Valeriano Assem** por su valiosa
Ayuda y dedicación en el presente trabajo.

Con infinito Amor
Para **Claudia Verónica**, que llego a mi vida
Colmándola de amor y ternura, ayudándome
en todo momento y siendo fuente de inspiración
para mí.

A todos Gracias.

Francisco Velázquez Hernández

Doy gracias al Todopoderoso por permitirme llegar a uno de los momentos más importantes de mi vida y agradezco en especial para este logro a mis padres: Jesús Juárez Romero y Juana Martínez, así como a mi amada esposa María Socorro.

Gumersindo Juárez Martínez

CONTENIDO

I. INTRODUCCIÓN

El osciloscopio (instrumento que permite la visualización de una señal eléctrica para su observación y medición de sus características) ha sido, es y será un instrumento ampliamente usado, además, de ser herramienta indispensable para el técnico, ingeniero eléctrico o electrónico.
En la actualidad las escuelas técnicas y Universidades de clase media, tienen cantidad limitada de estos instrumentos en sus laboratorios y se requeriría de una inversión muy grande para cubrir la gran demanda de alumnos.
El presente trabajo propone una posible solución a este problema partiendo de la premisa de construir un osciloscopio digital de bajo costo, que pueda ser utilizado de manera didáctica y práctica por los alumnos de la Facultad de Ingeniería, que pueda ser transportado con facilidad, utilizando un monitor VGA, basado en la tecnología de los dispositivos lógicos programables complejos (CPLD), que son circuitos integrados, formados por una matriz de compuertas lógicas y flip-flops, que proporcionan una solución al diseño y que pueden ser de apoyo para el desarrollo de hardware digital, que al interactuar con el monitor de computadora funcione como un osciloscopio didáctico. Los bloques principales que conforman el osciloscopio y que se desarrollan en el presente trabajo, se muestran en la figura 1.1.

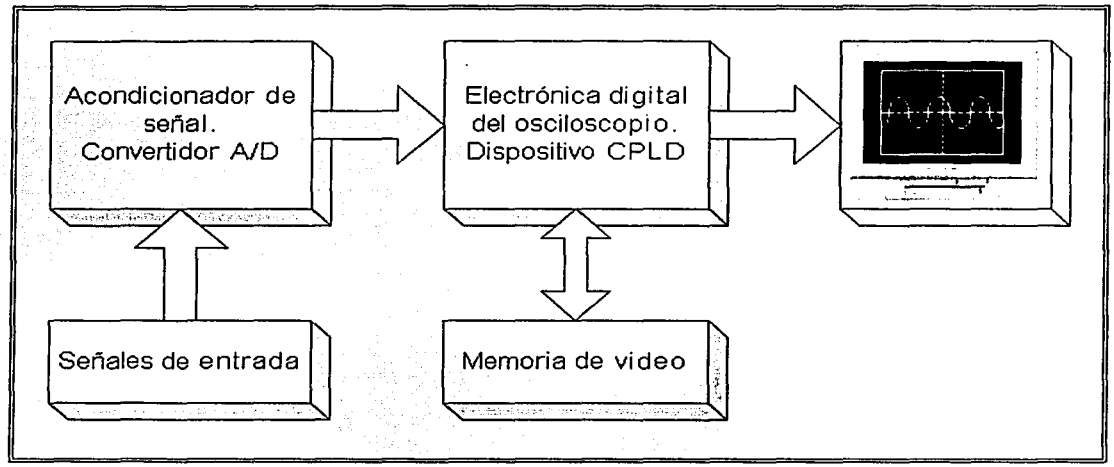


Figura 1.1 Diagrama a bloques del proyecto.

En el desarrollo de este proyecto, se utilizan CPLD de la familia ALTERA, basados en la tecnología SRAM y EEPROM, los cuales pueden ser configurados o programados en una misma tarjeta llamada UP1 (también de uso didáctico) que contiene integrados los CPLD, donde se desarrolla la electrónica digital, que es uno de los bloques del osciloscopio y se subdivide en tres módulos principales que se programaron en el dispositivo CPLD y que se denominan *Controlador de Video*, *Controlador de VRAM* y *Base de Tiempos*. Este bloque recibe la información del convertidor analógico digital. ALTERA poseé varias familias de CPLD, con una extensa gama de densidades, desde 300 hasta 250 000 compuertas. La tarjeta UP1 mencionada anteriormente y usada en el prototipo, contiene un dispositivo FLEX10K (EPF10K20) y un dispositivo MAX7000 (EMP7128S), que serán descritos detalladamente en la parte que describe el entorno de diseño Max + Plus II, que es un software que trabaja en ambiente Windows y suministra una plataforma múltiple e independiente de una arquitectura en particular, que se adapta fácilmente a las necesidades de diseño de circuitos electrónicos. Y por ello ofrece una forma fácil de diseñar proyectos y hacer su programación directa en los dispositivos.

Gracias a las necesidades tecnológicas actuales, se han desarrollado los llamados HDL que son lenguajes de descripción de hardware, los cuales nos permiten describir hardware utilizando especificaciones de alto nivel, es decir, son lenguajes de alto nivel que nos sirven para crear hardware. Los HDL disponibles actualmente nos permiten diferentes formas de especificar un diseño, estas formas también conocidas como "estilos de escritura", las cuales van desde especificaciones solo funcionales hasta especificaciones estructurales. VHDL es uno de estos lenguajes de descripción de hardware que es originado por el departamento de Defensa de los EEUU a mediados de 1980. El algoritmo de programación que se creó para el funcionamiento de nuestro prototipo, está desarrollado en un Lenguaje de Descripción de Hardware (VHDL), que es compatible con el entorno de diseño MAX+PLUS II y los dispositivos de ALTERA.

Para poder digitalizar la señal, se lleva acabo un acondicionamiento y una conversión (analógico-digital), en otro de los bloques del osciloscopio. El acondicionador de señal adecua las señales de entrada al osciloscopio para que resulten compatibles con el convertidor analógico digital. Este último también forma parte de este mismo bloque y es necesario como ya se mencionó anteriormente, para proporcionar información al bloque de electrónica digital en un formato que éste pueda manipular.

Finalmente, la señal que fue procesada en el bloque de la electrónica digital se visualiza en una pantalla de monitor VGA.

Estamos seguros que este osciloscopio despertará interés en varias escuelas para que sus alumnos dispongan de un instrumento básico para su desarrollo profesional y laboral.

II. MARCO TEÓRICO

II.1 CONCEPTOS BÁSICOS DEL OSCILOSCOPIO

El osciloscopio es un instrumento electrónico que permite la visualización de una señal eléctrica para la observación y medición de sus características. A continuación se listan las funciones más básicas:

- Determinar directamente el período y el voltaje de una señal.
- Determinar indirectamente la frecuencia de una señal.
- Determinar que parte de la señal es DC y cual AC.
- Localizar averías en un circuito.
- Medir la fase entre dos señales.
- Determinar que parte de la señal es ruido y como varia este en el tiempo.

Este instrumento puede medir un gran número de fenómenos, provisto del transductor adecuado (un elemento que convierte una magnitud física en señal eléctrica) es capaz de darnos el valor de una presión, ritmo cardiaco, potencia de sonido, nivel de vibraciones en un coche, temperatura, etc.

Los osciloscopios pueden ser analógicos ó digitales. Los primeros trabajan directamente con la señal aplicada, ésta una vez amplificada desvía un haz de electrones en sentido vertical proporcionalmente a su valor. En contraste los osciloscopios digitales utilizan previamente un convertidor analógico-digital (A/D) para almacenar digitalmente la señal de entrada, reconstruyendo posteriormente esta información en la pantalla.

Ambos tipos tienen sus ventajas e inconvenientes. Los analógicos son preferibles cuando es prioritario visualizar variaciones rápidas de la señal de entrada en tiempo real. Los osciloscopios digitales se utilizan cuando se desea visualizar y estudiar eventos no repetitivos (picos de tensión que se producen aleatoriamente).

Para entender el funcionamiento de un osciloscopio es necesario detenerse un poco en los procesos internos llevados a cabo por este aparato. Empezaremos por el tipo analógico ya que es el más sencillo.

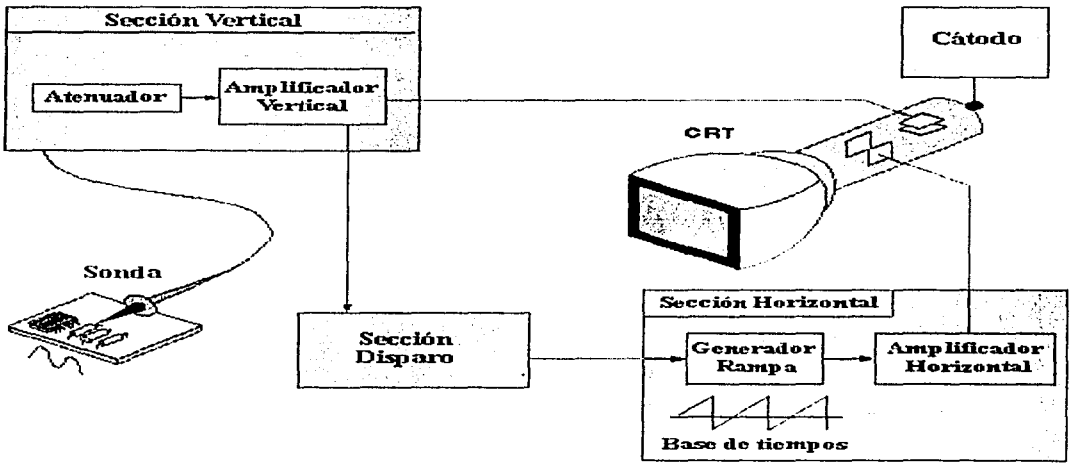


Figura 2.1 Diagrama de bloques del osciloscopio analógico.

II.1.1 Osciloscopio Analógico

Consta básicamente de circuitos atenuadores y amplificadores verticales, circuitos de disparo y de barrido, circuito amplificador horizontal y un tubo de rayos catódicos (CRT) con su respectivo circuito de control (Fig. 2.1).
El elemento central del osciloscopio es el CRT (Fig. 2.2), el cual es un tubo de vidrio en el que se ha hecho un alto vacío. El CRT poseé tres partes importantes: cañón de electrones, placas deflectoras y pantalla fosforescente.
Los rayos catódicos, electrones emitidos por un filamento caliente llamado cátodo (-), son acelerados hacia un ánodo (+) que se mantiene a un alto potencial eléctrico V, de varios miles de volts. En el centro del ánodo hay un agujero por el que atraviesa sólo un pequeño haz de electrones, el cual continúa su trayectoria hasta chocar con la pantalla, originando un punto luminoso en el lugar del choque, punto "P" en la Fig. 2.2, debido a que la pantalla está recubierta internamente con una sustancia fosforescente.

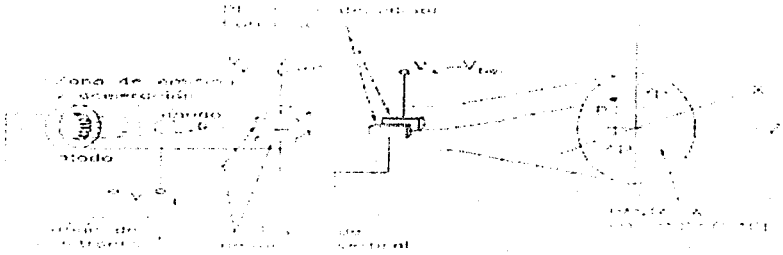


Figura 2.2 Tubo de rayos catódicos.

El haz de electrones puede ser desviado, tanto en la dirección vertical como en la horizontal por campos eléctricos producidos entre los pares de placas de desviación al aplicárseles voltajes (Vvert y Vhor). Estas desviaciones (Xp e Yp en la Fig.2.2) serán proporcionales a los voltajes aplicados, es decir:

$$Xp \mu Vhor = Vx \quad Yp \mu Vver = Vy \quad (Ec1)$$

El haz de electrones pasará sin desviarse cuando estos dos voltajes (Vy y Vx) sean nulos. La Fig. 2.3a muestra la posición del punto "P" para diferentes casos, todos con voltajes constantes.

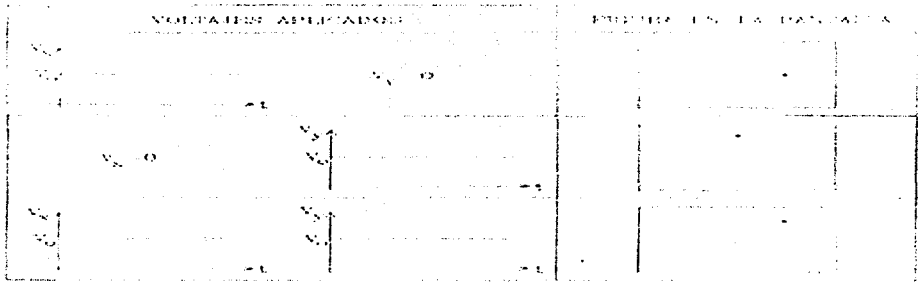


Figura 2.3a Voltajes constantes aplicados al tubo de rayos catódicos

Movimiento del punto P. Cuando a las placas de deflexión se les conectan voltajes variables, el punto "P" de la pantalla "camina" sobre ésta. El tipo específico de movimiento de P depende de los voltajes aplicados sobre las placas. A continuación se describe el proceso de despliegue de una señal capturada por un osciloscopio:

Cuando se conecta la punta de medición a un circuito, la señal se dirige a la sección vertical. Dependiendo de donde situemos el mando del amplificador vertical atenuaremos la señal ó la amplificaremos. En la salida de este bloque ya se dispone de la suficiente señal para atacar las placas de deflexión verticales (que naturalmente están en posición horizontal) y que son las encargadas de desviar el haz de electrones que surge del cátodo e impacta en la capa fluorescente del interior de la pantalla, en sentido vertical. Hacia arriba si la tensión es positiva con respecto al punto de referencia (GND) ó hacia abajo si es negativa.
La señal también atraviesa la sección de disparo para de esta forma iniciar el barrido horizontal (este es el encargado de mover el haz de electrones desde la parte izquierda de la pantalla a la parte derecha en un determinado tiempo). El trazado se consigue aplicando la parte ascendente de un diente de sierra a las placas de deflexión horizontal (las que están en posición vertical), y puede ser regulable en tiempo actuando sobre el mando TIME-BASE. El retrazado (recorrido de derecha a izquierda) se realiza de forma mucho más rápida con la parte descendente del mismo diente de sierra. De esta forma la acción combinada del trazado horizontal y de la deflexión vertical traza la gráfica de la señal en la pantalla. La sección de disparo es necesaria para estabilizar las señales repetitivas (se asegura que el trazado comience en el mismo punto de la señal repetitiva).

II.1.2 Osciloscopio Digital

Los osciloscopios digitales poseen además de las secciones explicadas anteriormente un sistema adicional de proceso de datos que permite almacenar y visualizar la señal.

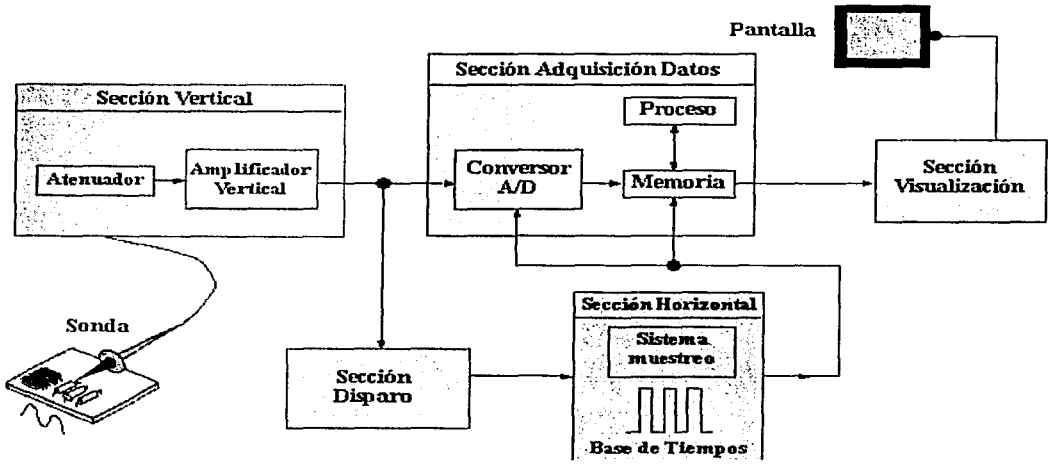


Figura 2.3b Diagrama de bloques del osciloscopio digital.

Cuando se conecta la punta de medición de un osciloscopio digital a un circuito, la sección vertical ajusta la amplitud de la señal de la misma forma que lo hacia el osciloscopio analógico.

El convertidor analógico-digital del sistema de adquisición de datos hace un muestreo de la señal a intervalos de tiempo determinados y convierte la señal de voltaje continua en una serie de valores digitales llamados *muestras*. En la sección horizontal una señal de reloj determina cuando el

convertidor A/D toma una muestra. La velocidad de este reloj se denomina velocidad de muestreo y se mide en muestras por segundo.

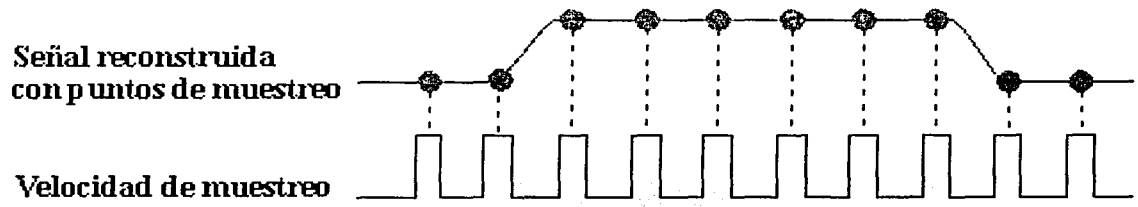


Figura 2.4 Señal muestreada

Los valores digitales muestreados se almacenan en una memoria como puntos de señal. El número de los puntos de señal utilizados para reconstruir la señal en pantalla se denomina registro. La sección de disparo determina el comienzo y el final de los puntos de señal en el registro. La sección de visualización recibe estos puntos del registro, una vez almacenados en la memoria, para presentar en pantalla la señal.

Dependiendo de las capacidades del osciloscopio se pueden tener procesos adicionales sobre los puntos muestreados, incluso se puede disponer de un predisparo, para observar procesos que tengan lugar antes del disparo.

Fundamentalmente, un osciloscopio digital se maneja de una forma similar a uno analógico, para poder tomar las medidas se necesita ajustar el mando AMPL., el mando TIMEBASE así como los mandos que intervienen en el disparo.

Métodos de muestreo. Tratan de explicar como se las arreglan los osciloscopios digitales para reunir los puntos de muestreo. Para señales de lenta variación, los osciloscopios digitales pueden perfectamente reunir más puntos de los necesarios para reconstruir posteriormente la señal en la pantalla. No obstante, para señales rápidas el osciloscopio no puede recoger muestras suficientes y debe recurrir a una de estas dos técnicas:

- Interpolación. Es decir, estimar un punto intermedio de la señal basándose en el punto anterior y posterior.
- Muestreo en tiempo equivalente. Si la señal es repetitiva es posible hacer un muestreo durante unos cuantos ciclos en diferentes partes de la señal para después reconstruir la señal completa.

Muestreo en tiempo real con Interpolación. El método estándar de muestreo en los osciloscopios digitales es el muestreo en tiempo real: el osciloscopio reúne los suficientes puntos como para reconstruir la señal. Para señales no repetitivas ó la parte transitoria de una señal es el único método válido de muestreo.

Los osciloscopios utilizan la interpolación para poder visualizar señales que son más rápidas que su velocidad de muestreo. Existen básicamente dos tipos de interpolación:

Lineal: Simplemente conecta los puntos muestreados con líneas.

Senoidal: Conecta los puntos muestreados con curvas según un proceso matemático, de esta forma los puntos intermedios se calculan para rellenar los espacios entre puntos reales de muestreo.

Usando este proceso es posible visualizar señales con gran precisión disponiendo de relativamente pocos puntos de muestreo.

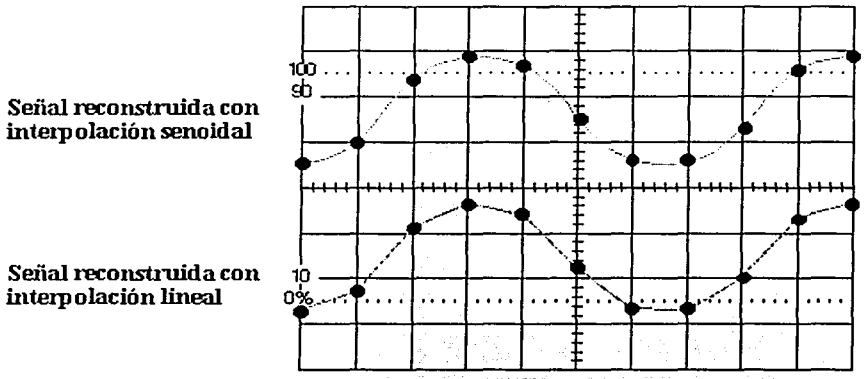


Figura 2.5 Tipos de interpolación para el muestreo de una señal en tiempo real.

Muestreo en tiempo equivalente. Algunos osciloscopios digitales utilizan este tipo de muestreo. Se trata de reconstruir una señal repetitiva capturando una pequeña parte de la señal en cada ciclo. Existen dos tipos básicos:

Muestreo secuencial- Los puntos aparecen de izquierda a derecha en secuencia para conformar la señal.

Muestreo aleatorio- Los puntos aparecen aleatoriamente para formar la señal.

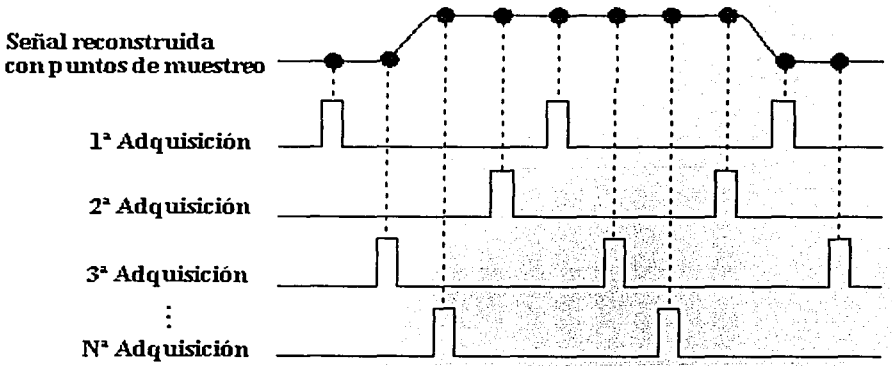


Figura 2.6a Muestreo en tiempo equivalente.

II.1.3 Parámetros qué Influyen en la Calidad de un Osciloscopio.

Los términos definidos en esta sección nos permitirán comparar diferentes modelos de osciloscopio disponibles en el mercado.

- Ancho de Banda. Especifica el rango de frecuencias en las que el osciloscopio puede medir con precisión. Por convenio el ancho de banda se calcula desde 0Hz (continua) hasta la frecuencia a la cual una señal de tipo senoidal se visualiza a un 70.7% del valor aplicado a la entrada (lo que corresponde a una atenuación de 3dB).

- Tiempo de subida. Es otro de los parámetros que nos dará, junto con el anterior, la máxima frecuencia de utilización del osciloscopio. Es un parámetro muy importante si se desea medir con fiabilidad pulsos y flancos (recordar que este tipo de señales poseen transiciones entre niveles de tensión muy rápidas). Un osciloscopio no puede visualizar pulsos con tiempos de subida más rápidos que el suyo propio.

- Sensibilidad vertical. Indica la facilidad del osciloscopio para amplificar señales débiles. Se suele proporcionar en mV por división vertical, normalmente es del orden de 5 mV/div (llegando hasta 2 mV/div).

- Velocidad. Para osciloscopios analógicos esta especificación indica la velocidad máxima del barrido horizontal, lo que nos permitirá observar sucesos más rápidos. Suele ser del orden de nanosegundos por división horizontal.

- Exactitud en la ganancia. Indica la precisión con la cual el sistema vertical del osciloscopio amplifica ó atenúa la señal. Se proporciona normalmente en porcentaje máximo de error.

- Exactitud de la base de tiempos. Indica la precisión en la base de tiempos del sistema horizontal del osciloscopio para visualizar el tiempo. También se suele dar en porcentaje de error máximo.

- Velocidad de muestreo. En los osciloscopios digitales indica cuantas muestras por segundo es capaz de tomar el sistema de adquisición de datos (especificamente el convertidor A/D). En los osciloscopios de calidad se llega a velocidades de muestreo de Mega Muestras por Segundo (MMPS). Una velocidad de muestreo grande es importante para poder visualizar pequeños periodos de tiempo. En el otro extremo de la escala, también se necesita velocidades de muestreo bajas para poder observar señales de variación lenta. Generalmente la velocidad de muestreo cambia al actuar sobre el mando TIMEBASE para mantener constante el número de puntos que se almacenaran para representar la forma de onda.

- Resolución vertical. Se mide en bits y es un parámetro que nos da la resolución del convertidor A/D del osciloscopio digital. Nos indica con que precisión se convierten las señales de entrada en valores digitales almacenados en la memoria. Técnicas de cálculo pueden aumentar la resolución efectiva del osciloscopio.

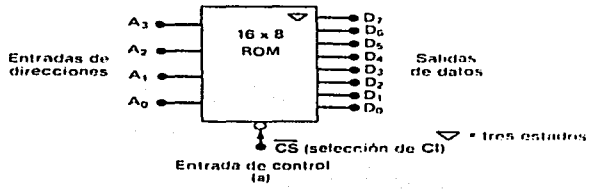
- Longitud del registro. Indica cuantos puntos se memorizan en un registro para la reconstrucción de la forma de onda. Algunos osciloscopios permiten variar, dentro de ciertos límites, este parámetro. La máxima longitud del registro depende del tamaño de la memoria de que disponga el osciloscopio. Una longitud del registro grande permite realizar zooms sobre detalles en la forma de onda de forma muy rápida (los datos ya han sido almacenados), sin embargo esta ventaja es a costa de consumir más tiempo en hacer un muestreo de la señal completa.

II.2 MEMORIAS

II.2.1 Memorias Solo de Lectura (ROM)

Las memorias de solo lectura son un tipo de memoria de semiconductor que están diseñadas para retener datos que son permanentes o que no cambian con mucha frecuencia. Durante la operación normal, no pueden escribirse nuevos datos en una ROM pero sí puede leerse información de ella. Para algunas ROM los datos que están almacenados tienen que grabarse durante el proceso de fabricación; para otras ROM esto se puede hacer en forma eléctrica. El proceso de grabar datos se conoce como **programación** de la ROM. Algunas ROM no pueden alterar sus datos una vez que se hayan programado; otras pueden **borrarse** y reprogramarse con la frecuencia que se desee. Un uso importante de las ROM se encuentra en el almacenamiento de programas en microcomputadoras. Ya que todas los ROM son *no volátiles*, estos programas no se pierden cuando la microcomputadora es desconectada. Cuando se enciende la máquina, puede empezar de inmediato a ejecutar el programa almacenado en ROM.

Diagrama de bloques en ROM. Un diagrama de bloques común para una ROM se muestra en la figura 2.6b. Tiene tres conjuntos de señales: entradas de dirección, entrada(s) de control y salidas de datos. Podemos determinar que la ROM de la figura 2.6b, almacena 16 palabras, ya que tiene $2^4 = 16$ posibles direcciones y cada palabra contiene 8 bits, puesto que hay ocho salidas de datos. Por lo tanto, ésta es una ROM de 16 X 8. Otra manera de describir la capacidad de la ROM consiste en decir que almacena 16 bytes de datos.



Palabra	A_3	A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	1	1	0	1	1	1	1	0
1	0	0	0	1	0	0	1	1	1	0	1	0
2	0	0	1	0	1	0	0	0	0	1	0	1
3	0	0	1	1	1	0	1	0	1	1	1	1
4	0	1	0	0	0	0	0	1	1	0	1	1
5	0	1	0	1	0	1	1	1	1	0	1	1
6	0	1	1	0	0	0	0	0	0	0	0	0
7	0	1	1	1	1	1	1	0	1	1	0	1
8	1	0	0	0	0	0	1	1	1	1	0	0
9	1	0	0	1	1	1	1	1	1	1	1	1
10	1	0	1	0	1	0	1	1	1	0	0	0
11	1	0	1	1	1	1	0	0	0	1	1	1
12	1	1	0	0	0	0	1	0	0	1	1	1
13	1	1	0	1	0	1	1	0	1	0	1	0
14	1	1	1	0	1	1	0	1	0	0	1	0
15	1	1	1	1	0	1	0	1	1	0	1	1

Palabra	A_3 A_2 A_1 A_0	D_7-D_0
0	0	DE
1	1	3A
2	2	85
3	3	AF
4	4	19
5	5	7B
6	6	00
7	7	ED
8	8	3C
9	9	FF
10	A	88
11	B	C7
12	C	27
13	D	6A
14	E	D2
15	F	58

Figura 2.6b ROM de 16 X 8

La entrada de control CS significa selección de CI. Esta es esencialmente una entrada de habilitación que habilita o deshabilita las salidas ROM. Muchas ROM tienen dos o más entradas de control que deben estar activas para habilitar las salidas de datos y con esto poder leer datos de la dirección

seleccionada. En algunos CI ROM una de las entradas de control se emplea para colocar la ROM en un modo de espera de bajo consumo de corriente de la fuente de alimentación del sistema.

La entrada que se muestra en la figura 2.6b, se encuentra activa en BAJO. Por lo tanto debe estar en el estado BAJO para habilitar la ROM y que los datos aparezcan en las salidas de datos. En esta ROM no existe la entrada R/W (lectura/escritura), debido a que la ROM no puede grabarse en condiciones normales de operación.

Arquitectura de la ROM

En la arquitectura interna de un CI ROM (Fig. 2.7), existen cuatro partes básicas: decodificador de renglones, decodificador de columnas, disposición de registros y buffers de salida

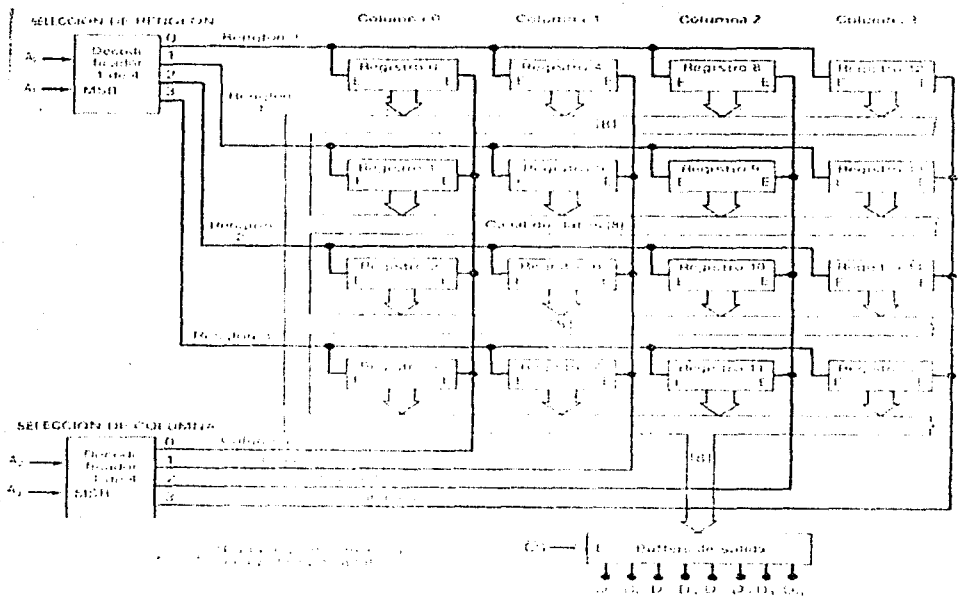


Figura 2.7 Arquitectura de la ROM de 16 X 8.

Arreglo de registros. El arreglo de registros almacena los datos que han sido programados en la ROM. Cada registro contiene un número de celdas de memoria que es igual al tamaño de la palabra. En el caso de nuestra ROM de 16 X 8, cada registro almacena una palabra de 8 bits. Los registros se disponen en un arreglo de matriz cuadrada que es común a muchos circuitos de memoria de semiconductor. Podemos especificar la posición de cada registro como ubicada en un renglón y una columna específicos. Por ejemplo, el registro 0 se encuentra en el renglón 0/columna 0 y el registro 9 está en el renglón 1/columna 2.

Las ocho salidas de datos de cada registro se conectan a un canal de datos interno que corre a través de todo el circuito. Cada registro tiene 2 entradas de habilitación (E); ambas tienen que ser ALTAS a fin de que los datos del registro sean colocados en el canal.

Decodificadores de direcciones. El código de dirección aplicado $A_3A_2A_1A_0$ determina que registro del arreglo será habilitado para colocar una palabra de datos de 8 bits en el canal. Los bits de dirección A_1A_0 se alimentan a un decodificador 1 de 4 que activa una línea de selección de renglón, y los bits de selección A_3A_2 se alimentan a un segundo decodificador 1 de 4 que activa una línea de selección de columna. Solamente un registro estará en el renglón y la columna seleccionados por las entradas de dirección, y estará habilitado.

Buffers de salida. El registro habilitado por las entradas de dirección coloca la palabra que contiene sobre el canal de datos. Estos datos entran en los buffers de salida, los cuales se encargan de transmitirlos hacia las salidas externas siempre y cuando CS esté en bajo. Si CS está en alto los buffers de salida se encuentran en el estado de alta impedancia, con lo que de D_7 hasta D_0 se encuentran flotando.

Temporización de la ROM

Habrá un retardo en la propagación entre la aplicación de entradas de una ROM y la aparición de salidas de datos durante una operación de lectura. Este retardo, denominado tiempo de acceso, t_{ACC}, es una medida de la velocidad de operación de la ROM.. El tiempo de acceso se describe gráficamente por medio de las formas de onda de la figura 2.8.

La forma de onda de más arriba representa las entradas de dirección, la de en medio es una selección de CI en BAJO, $\overline{CS}$, y la demás abajo representa las salidas de datos. Al tiempo t_0 las entradas de dirección están en algún nivel específico, algunas en ALTO y algunas en BAJO. $\overline{CS}$ es ALTA, de manera que las salidas de datos de la ROM se encuentran en su estado Alta-Z (representado por la línea sombreada).
Antes de t_1, las entradas de dirección cambian a una nueva dirección para realizar una nueva operación de lectura. En t_1 la nueva dirección es válida; es decir, cada entrada de dirección está en un nivel lógico válido. En este punto la circuitería interna de la ROM empieza a decodificar las nuevas entradas de dirección para seleccionar el registro que enviará sus datos a los buffers de salida. En t_2 la entrada $\overline{CS}$ es activada para habilitar los buffers de salida. Finalmente, en t_3, las salidas cambian del estado Alta-Z a los datos válidos que representan los almacenados en la dirección especificada.

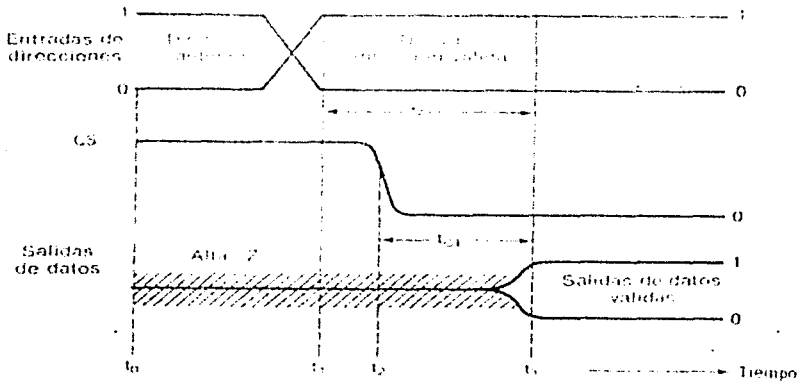


Figura 2.8 Temporización común de una operación de lectura en ROM .

El retardo entre t1, y t3, cuando la nueva dirección y las salidas de datos se vuelven válidas, es el tiempo de acceso t_{ACC}. En general, las ROM bipolares tienen tiempos de acceso que van desde 30 hasta 90 ns, para los dispositivos NMOS este tiempo varía desde 35 hasta 500 ns.

Otro importante, parámetro de temporización es el tiempo de habilitación de salida, t_{OE}, que es el retardo entre la entrada CS y la salida de datos válida. Valores comunes de t_{OE} son de 10 a 20 ns para ROM bipolares y de 25 a 100 ns para ROM MOS. Este parámetro de temporizado es importante en situaciones donde las entradas de dirección están ya en sus nuevos valores, pero las salidas de la ROM no han sido habilitadas aún. Cuando CS pasa a BAJA para habilitar las salidas, el retardo será t_{OE}.

Tipos de ROM

Los diversos tipos de ROM difieren en la forma en que son programados y en su capacidad para ser borrados y reprogramados.

ROM programada por mascarilla (MROM). Este tipo de ROM tiene sus localidades de almacenamiento escritas (programadas) por el fabricante según las especificaciones del cliente. Se utiliza un negativo fotográfico llamado *mascarilla* para controlar las conexiones eléctricas en el circuito. Se requiere una mascarilla especial por cada conjunto diferente de información para ser almacenada en la MROM. Ya que las mascarillas con costosas, este tipo de MROM es económico sólo si se necesita una cantidad considerable de la misma MROM. Algunas MROM de este tipo se encuentran disponibles como dispositivos preprogramados tomados de una tabla o manual con información que comúnmente se utiliza, como fórmulas matemáticas y códigos generadores de caracteres para exhibiciones en tubo de rayos catódicos (CRT). Una desventaja importante de este tipo de MROM es que no puede reprogramarse en el caso de un cambio de diseño que requiera una modificación del programa almacenado. Las ROM programadas por mascarilla todavía presentan el enfoque más económico cuando se necesita una cantidad considerable de ROM programadas idénticas.

ROM programables (PROM). Para las aplicaciones de bajo volumen, los fabricantes han creado PROM con conexión fusible, que no se programan durante el proceso de fabricación sino que son programadas por el usuario. Sin embargo, una vez programada una PROM se parece a una MROM en que no puede borrarse ni reprogramarse.

La estructura de la PROM con conexión fusible es muy semejante a la MROM en cuanto que ciertas conexiones quedan intactas o bien son abiertas a fin de programar una celda de la memoria como un 1 o un 0. En la MROM estas conexiones se hacen de las líneas de habilitación a las bases de los transistores. En un PROM cada una de estas conexiones se hace con una pequeña conexión fusible que viene intacta del fabricante. El usuario puede fundir selectivamente cualquiera de estas conexiones fusibles para producir en la memoria los datos almacenados que se desean. Comúnmente esto se lleva a cabo aplicando con mucha precaución un voltaje controlado al dispositivo para producir un flujo de corriente que ocasionará que la conexión fusible se abra en forma semejante a cuando se funde el fusible. Una vez que se funde una conexión fusible, ya no puede volver a conectarse.
El proceso de programación de una PROM y luego la verificación de los datos programados puede consumir mucho tiempo y ser muy tedioso cuando se efectúa manualmente. Se encuentran disponibles varios programadores comerciales de PROM por varios cientos de dólares, que permiten que se introduzca un programa desde el teclado en memoria lectura/escritura (RWM) y luego realizar la fundición del fusible y la verificación automática sin intervención del usuario.

ROM programable y borrable (EPROM). Una EPROM puede ser programada por el usuario y también puede *borrarse* y reprogramarse tantas veces como se desee. Una vez programada, la EPROM es una memoria *no volátil* que contendrá sus datos almacenados indefinidamente. El proceso para programar una EPROM implica la aplicación de niveles de voltaje especiales (comúnmente en el orden de 10 a 25 V) a las entradas adecuadas del circuito en una cantidad de tiempo especificada (por lo general 50 ms por localidad de dirección). El proceso de programación usualmente es afectado por

un circuito especial de programación que está separado del circuito en el cual la EPROM trabajará por último. El proceso de programación completo puede llevar varios minutos para una EPROM.
En una EPROM las celdas de almacenamiento son transistores MOSFET que tienen una compuerta de silicio sin ninguna conexión eléctrica (es decir, una compuerta flotante). En su estado normal, cada transistor está apagado y cada celda guarda un 1 lógico. El transistor puede encenderse mediante la aplicación de un pulso de programación de alto voltaje, el cual inyecta electrones de alta energía en la región formada por la compuerta flotante. Estos electrones permanecen en esta región una vez que ha finalizado el pulso ya que no existe ninguna trayectoria de descarga. Esto mantiene al transistor encendido de manera permanente, aun cuando se retire la potencia de alimentación del dispositivo; con esto la celda guarda ahora un 0 lógico. Durante el proceso de programación se emplean las direcciones y terminales de la EPROM para seleccionar las celdas de memoria que serán programadas con ceros así como las que se dejarán como unos.
Una vez que se ha programado una celda de la EPROM, se puede borrar su contenido exponiendo la EPROM a la luz ultravioleta (UV), la cual se aplica a través de la ventana que se encuentra sobre el encapsulado del circuito. La luz UV produce una fotocorriente que va desde la compuerta flotante hacia el sustrato de silicio; con esto se apaga el transistor y se lleva de nuevo a la celda hacia el estado 1 lógico. El proceso de borrado requiere entre 15 y 30 minutos de exposición a los rayos UV. Desafortunadamente no existe ninguna forma de borrar sólo algunas celdas; la luz UV borra todas las celdas al mismo tiempo, por lo que una EPROM borrada almacena sólo unos lógicos. Una vez borrada, la EPROM puede volverse a programar.

PROM eléctricamente borrable (EEPROM). Como se observó antes, las EPROM tienen dos desventajas importantes. Primero, tienen que ser retiradas de sus bases a fin de ser borradas y reprogramadas. Segundo, el borrado retira todo el contenido de la memoria; esto requiere una reprogramación completa aún cuando sólo tenga que alterarse una palabra de la memoria. La PROM eléctricamente borrable (EEPROM) se inventó alrededor del año 1980 como una mejora a la EPROM.
La EPROM aprovecha la misma estructura de la compuerta flotante de la EPROM. Agrega la característica de borrado eléctrico a través de la adición de una delgada región de óxido arriba del drenaje de la celda de memoria MOSFET. Aplicando un voltaje ALTO (21 V) entre la compuerta y el consumo del MOSFET, puede inducirse una carga en la compuerta flotante, donde permanecerá aun cuando se suspenda el suministro de energía. La inversión del mismo voltaje produce una eliminación de las cargas capturadas de la compuerta flotante y borra la celda. Ya que este mecanismo de transporte de cargas requiere cargas muy bajas, la programación y el borrado de una EPROM puede hacerse por lo general en el circuito (es decir, sin una fuente de luz UV y unidad programadora de PROM).
Una ventaja importante ofrecida por las EEPROM sobre las EPROM es la capacidad de borrar y reprogramar eléctricamente palabras individuales en el arreglo de la memoria. Otra ventaja es que una EEPROM completa puede borrarse en cerca de 10 ms (en circuito) versus cerca de 30 minutos de una EPROM en luz UV externa. Una EEPROM también puede ser programada con mayor rapidez, requiere sólo un pulso de programación de 10 ms por cada palabra de datos, en comparación con 50 ms de una EPROM.

Debido a que la EEPROM puede borrarse y reprogramarse aplicando voltajes adecuados, no necesitamos retirarla del circuito del cual forma parte, siempre que los componentes de soporte adicionales también sean parte de la circuitería. La circuitería de soporte incluye el voltaje de programación de 21 V (Vpp), que generalmente se genera a partir de la fuente de + 5 V a través de un convertidor DC a DC, y circuitería para controlar la temporización y secuenciación de 10 ms de las operaciones de borrado y programación.

II.2.2 MEMORIA DE ACCESO ALEATORIO DE SEMICONDUCTOR (RAM)

RAM significa memoria con acceso aleatorio, lo cual quiere decir que se puede tener acceso fácilmente a cualquier localidad de dirección de memoria. Muchos tipos de memoria se pueden clasificar como de acceso aleatorio, pero cuando el término RAM se utiliza con memorias de

semiconductor, generalmente se considera que significa memoria de lectura y escritura (RWM) en contraste con la ROM. Comúnmente en la práctica se usa el término RAM para referirnos a la RWM de semiconductor, lo utilizaremos de aquí en adelante.

La RAM se emplean en las computadoras como medios de almacenamiento temporal para programas y datos. El contenido de muchas de las localidades de dirección de la RAM será leído y escrito a medida que la computadora ejecuta un programa. Esto requiere que la RAM tenga ciclos de escritura y lectura rápidos para que no reduzca la velocidad de operación de la computadora.

Una gran desventaja de la RAM es que son volátiles o pierden toda la información contenida en ellas si se interrumpe el suministro de potencia. Sin embargo, algunas RAM CMOS emplean una cantidad tan pequeña de potencia en el modo de espera (alta impedancia), que se pueden alimentar con baterías cada vez que se interrumpe la fuente de alimentación principal. Por supuesto, la ventaja principal de la RAM es que se puede escribir en ella y también se puede leer de ella muy rápidamente con la misma facilidad.

Arquitectura de la RAM

La RAM consta de varios registros, cada uno de los cuales almacena una sola palabra de datos y con una dirección única. Las RAM comúnmente vienen con capacidades de palabras de 1K, 4K, 8K, 16K, 64K, 128K, 256K, y tamaños de palabra de 1, cuatro u ocho bits. La capacidad de palabras y el tamaño de éstas pueden extenderse combinando circuitos integrados de memoria.

Figura 2.9 Memoria RAM 64 X 4.

La figura 2.9, muestra la arquitectura simplificada de una RAM que almacena 64 palabras de 4 bits cada una(es decir, una memoria de 64 X 4). Estas palabras tienen direcciones que van de 0 a 63_{10}. A fin de seleccionar una de las 64 localidades de dirección para leer o escribir, se aplica un código de dirección binario a un circuito decodificador. Ya que $64 = 2^6$, el decodificador requiere un código de entrada de seis bits. Cada código de dirección activa una determinada salida del decodificador la cual, a su vez, habilita su registro correspondiente. Por ejemplo, supongamos un código de dirección aplicado de

$$A_5A_4A_3A_2A_1A_0 = 011010$$

Como $011010_2 = 26_{10}$, la salida del decodificador 26 pasará a estado alto, seleccionan el registro 26 para una operación de lectura o bien de escritura.

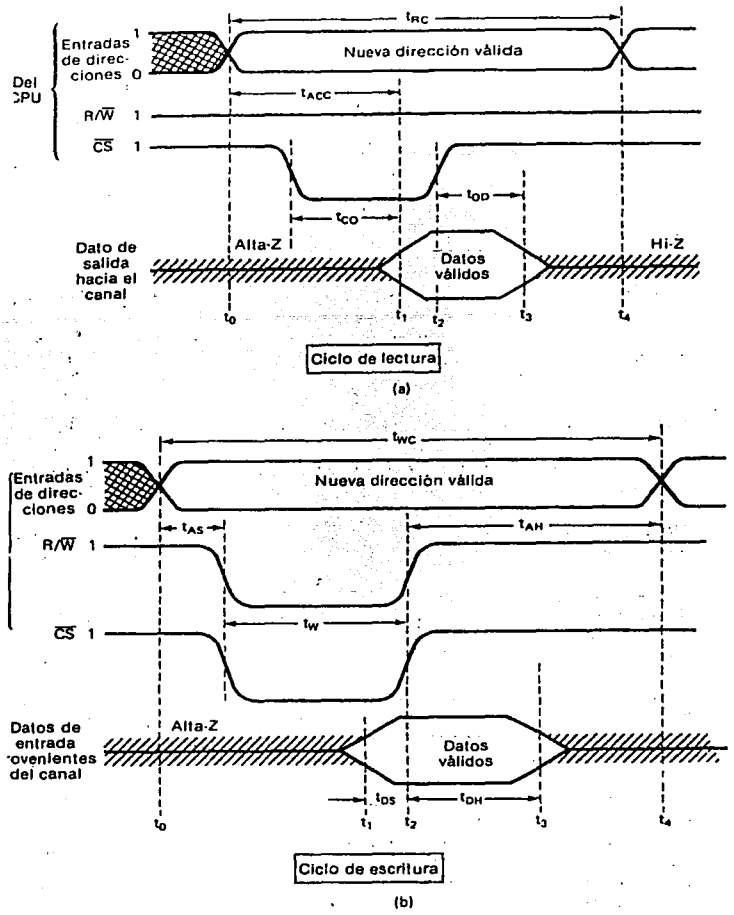


Figura 2.10 Operación de lectura y escritura de la RAM.

Operación de lectura. El código de dirección selecciona un registro del circuito de memoria para leer o escribir. A fin de leer el contenido del registro seleccionado, la entrada LECTURA/ESCRITURA (R/W) debe ser un 1. Además, la entrada CS (SELECCIÓN DE CI) debe ser activada (un 1 en este caso). La combinación de R/W = 1 y CS = 1 habilita los buffers de salida de manera que el contenido del registro seleccionado aparecerá en las cuatro salidas de datos. R/W = 1 también deshabilita los buffers de entrada de manera que las entradas de datos no afecten la memoria durante la operación de lectura.

Operación de escritura. Para escribir una nueva palabra de cuatro bits en el registro seleccionado se requiere que R/W = 0 y CS = 1. Esta combinación habilita los buffers de entrada de manera que la palabra de cuatro bits aplicada a las entradas de datos se cargará en el registro seleccionado. R/W = 0 también deshabilita los buffers de salida que son de tres estados, de manera que las salidas de datos se encuentren en estado de alta-Z durante una operación de escritura. La operación de escritura, desde luego, destruye la palabra que estaba almacenada antes en la dirección.

Selección CI. Muchos circuitos de memoria tienen una o más entradas CS que se usan para habilitar o deshabilitar al circuito en su totalidad. En el modo deshabilitado todas las entradas y salidas de datos se deshabilitan (Alta-Z) de manera cuando se combinen CI de memoria para obtener mayores memorias. Cuando las entradas CS se encuentran en su estado activo, se dice que el CI de memoria ha sido seleccionado; de otro modo se dice que no está seleccionado.

Terminales comunes de entrada/salida (E/S). A fin de conservar terminales en un encapsulado de CI, los fabricantes a menudo combinan las funciones de entrada y salida de datos utilizando terminales comunes de E/S. La entrada R/W controla la función de estas terminales E/S. Durante una operación de lectura, las terminales E/S actúan como salidas de datos que reproducen el contenido de la localidad de dirección seleccionada. Durante una operación de escritura, las terminales E/S actúan como entradas de datos.

II.3 TECNOLOGÍA DE LOS DISPOSITIVOS LÓGICOS PROGRAMABLES (PLD)

Para reducir el número de CI utilizados en un diseño, es necesario colocar cada vez más y más funciones lógicas sobre un CI. Esto, claro está, se hace con tecnología LSI y VLSI para funciones estándares tales como las memorias, microprocesadores, sintetizadores de voz, calculadoras y otros más. Estos dispositivos contienen cientos y miles de compuertas lógicas interconectadas dentro del CI de forma tal que éste funcione de una manera definida de antemano. Sin embargo, existen muchas situaciones de diseño para las que no existe ninguna solución LSI o VLSI. Por lo general, estas situaciones requieren que los diseñadores interconecten varios CI SSI o MSI para obtener las funciones lógicas que necesitan. El desarrollo reciente de los *dispositivos lógicos programables* (PLD) ofrece a los diseñadores lógicos una manera de reemplazar varios CI estándares con un solo CI.
La figura 2.11, ilustra la idea básica utilizada por todos los CI programables. Esta muestra un arreglo de compuertas AND y otro de compuertas OR que se pueden conectar entre sí para generar cuatro salidas, cada una de las cuales pueden ser cualquier función lógica de dos variables de entrada A y B.
Cada entrada está conectada a dos buffer, uno no inversor y otro inversor que producen las formas verdadera y negada de cada variable. También existen *líneas de entrada* hacia el arreglo de compuertas AND. Cada compuerta AND está conectada a dos líneas de entrada diferentes, lo que permite generar el producto único de las variables de entrada. Las salidas de las compuertas AND reciben el nombre de *líneas producto*.
Cada una de líneas producto está conectada mediante una conexión fusible a una de las cuatro entradas que tiene cada compuerta OR. Al inicio, cuando todas las conexiones fusibles están intactas, la salida de cada compuerta OR es 1.

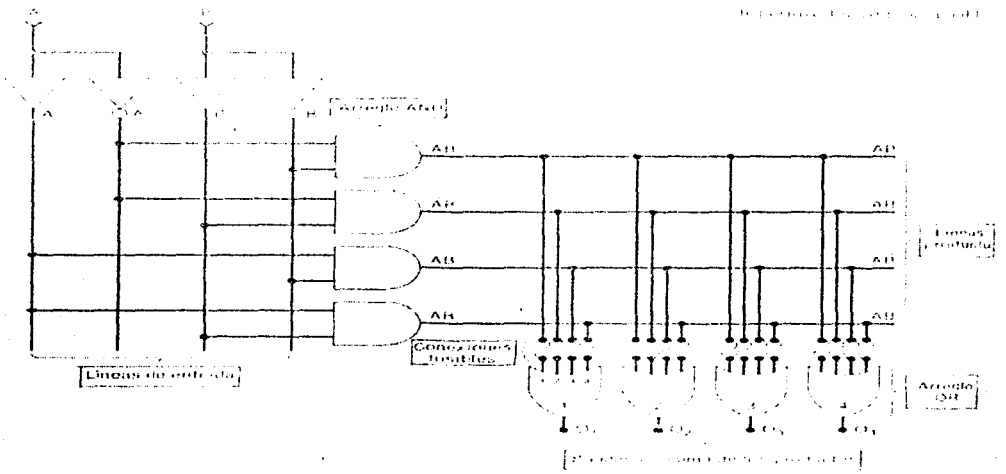


Figura 2.11 Ejemplo de dispositivo lógico programable.

Simbología PLD. El ejemplo de la figura 2.11, tiene sólo dos variables de entrada y con ello el diagrama de circuito ya esta bastante denso. Para simplificar, los fabricantes de este tipo de dispositivos han adoptado una simbología simplificada para representar los circuitos internos de estos dispositivos.

La figura 2.12, muestra un ejemplo de la simbología empleada para una compuerta AND con cuatro entradas. Primero, note que los buffer de entrada están representados por un solo buffer que tiene dos salidas. Después, note que sólo se muestra una línea que va hacia la compuerta AND para representar las cuatro entradas. Las conexiones que vienen de las líneas de entrada para variables y que van hacia la compuerta AND están señaladas con una X o con un punto. Una X representa una conexión fusible intacta. El punto representa una conexión alambrada (es decir, que no puede cambiarse). La ausencia de cualquiera de estos símbolos significa que no existe ninguna conexión. En este ejemplo, las entradas A y B están conectadas en la compuerta AND con la finalidad de generar el producto $A\overline{B}$.

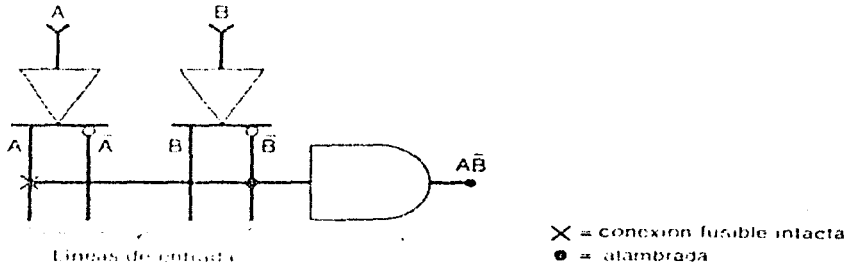


Figura 2.12 Simbología simplificada para PLD

Arquitecturas PLD

II.3.1 PROM.

Existen varias arquitecturas comunes utilizadas en los PLD. La primera que se examinará es la PROM. La figura 2.13, muestra una PROM de 16 X 4 que también pueden funcionar como PLD. La PROM tiene cuatro entradas que están totalmente decodificadas por el arreglo de compuertas AND; esto es, cada compuerta genera uno de los 16 posibles productos AND. Las conexiones de las líneas de entrada hacia el arreglo AND son alambradas, mientras que las conexiones de las líneas producto AND hacia las entradas de las compuertas OR son programables.
La PROM puede generar cualquier función lógica posible de las variables de entrada debido que genera todos los términos AND posibles. En general, la aplicación que requiera que se encuentre disponible cualquier combinación de las entradas, es un buen candidato para una PROM. Sin embargo, las PROM se vuelven poco prácticas cuando se tiene que dar cabida a un número grande de entradas, debido a que se duplica el número de conexiones fusibles por cada variable que se añada.

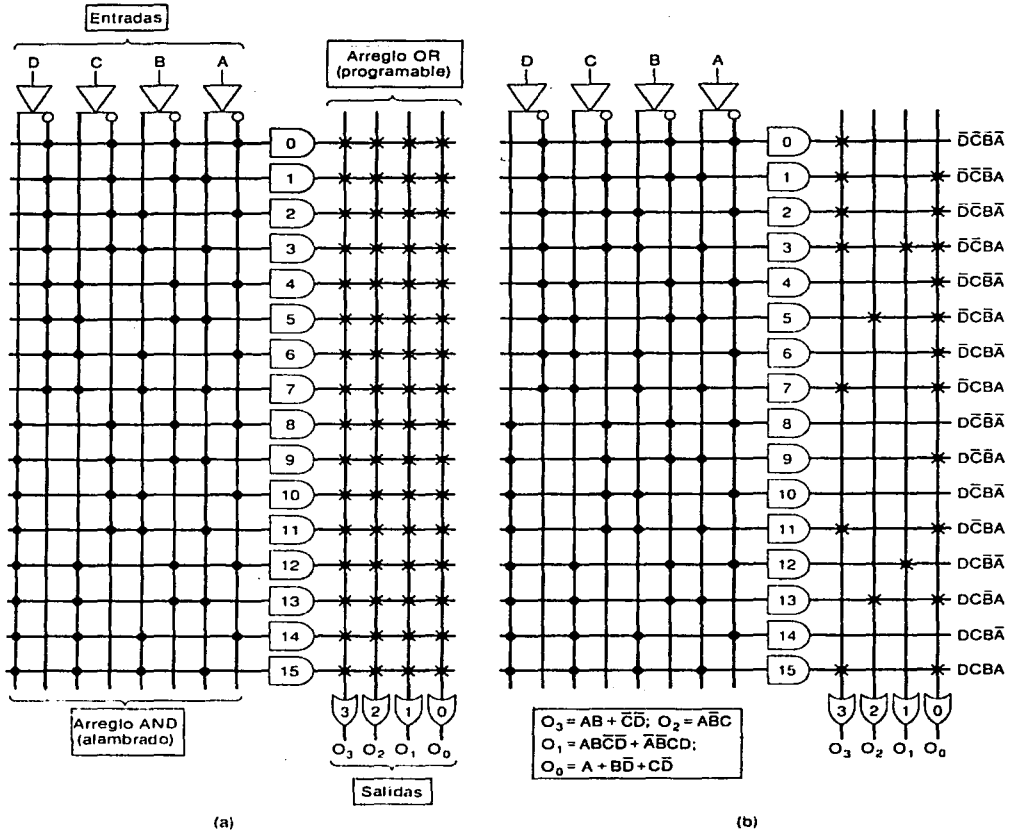


Figura 2.13 (a) Arquitectura de una PROM que funciona como un PLD; (b) las conexiones se funden para programar las salidas de las funciones lógicas dadas.

II.3.2 PAL

Lógica en un Arreglo Programable. Existen muchas aplicaciones que no requieren que todas las combinaciones de las entradas sean programables. Esto conduce al desarrollo de una clase de PLD denominada lógica en un arreglo programable (PAL). La arquitectura en un PAL es un poco diferente ala de una PROM como lo muestra la figura 2.14.
Las PAL tienen los mismos arreglos AND y OR que la PROM, pero en la PAL son programables las entradas de las compuertas AND mientras que las de las compuertas OR son alambradas. Esto significa que se puede programar cualquier compuerta AND para generar cualquier producto que se desee de las cuatro variables de entrada junto con sus respectivos complementos. Cada compuerta OR está alambrada solo con cuatro salidas AND. Esto limita cada función de salida a cuatro términos del tipo producto. Si se requiere de una función que contenga más de cuatro términos de esta clase, no es posible implementarla con esta PAL; se tiene que utilizar para ello una que tenga más entradas OR. Si se requieren menos de cuatro términos del tipo producto, se puede hacer que los no necesarios sean cero.

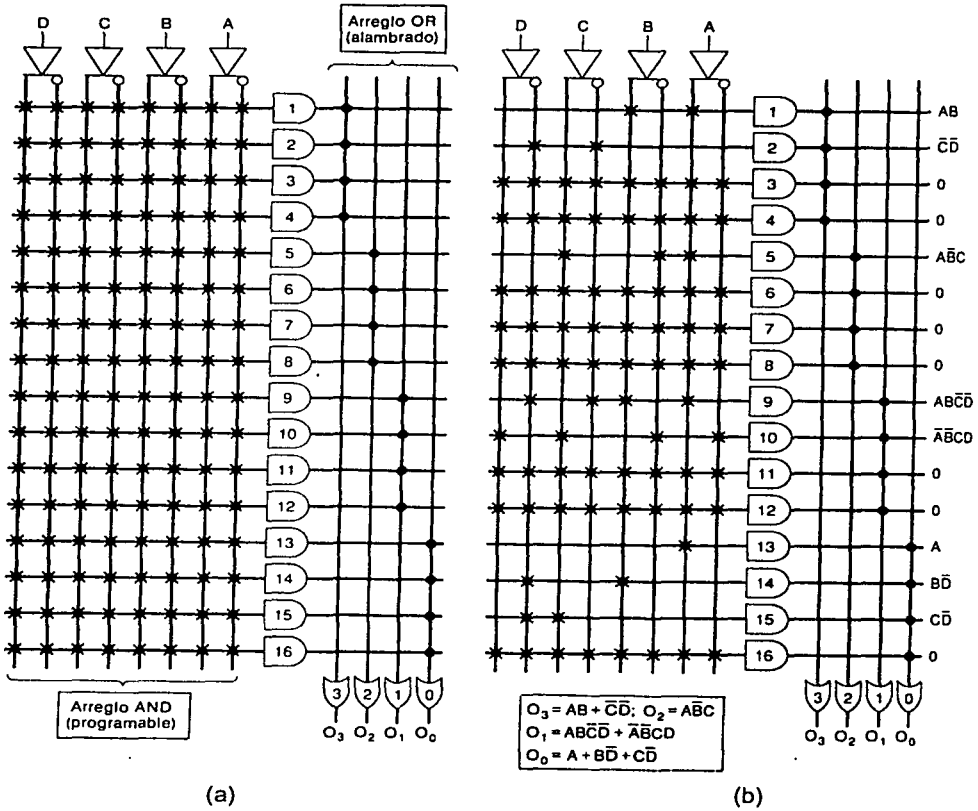


Figura 2.14 (a) Arquitectura PAL típica; (b) La misma PAL programada para obtener las funciones deseadas.

II.3.3 PLA

Arreglos lógicos programables. Un PLA combina las características de la PROM y el PAL proporcionando tanto un arreglo OR programable como un arreglo AND también programable. Esta característica, ilustrada en la figura 2.15, lo convierte en el más versátil de los tres tipos de PLD. Sin embargo, presenta algunas desventajas. Dado que tiene dos conjuntos de conexiones fusibles, es más difícil de fabricar, programar y probar que una PROM o un PAL. Los PLA también son conocidos como *arreglos programables en campo (FPLA).*

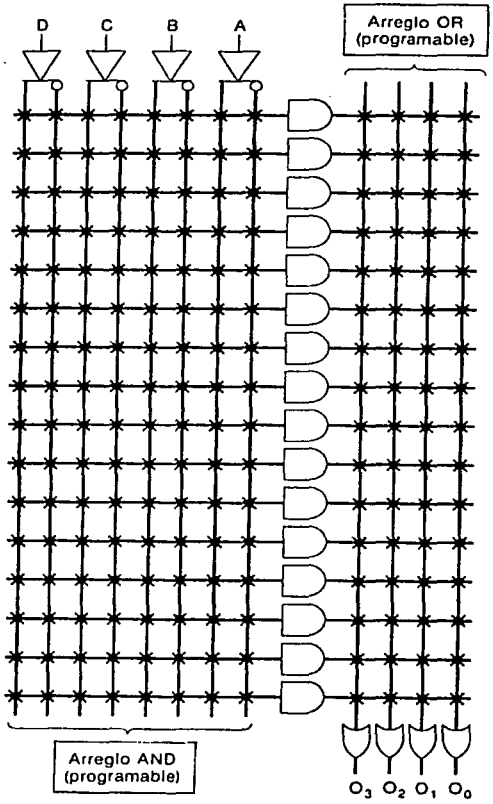


Figura 2.15 Arquitectura PLA típica.

II.3.4 GAL

Las GAL son dispositivos de matriz lógica genérica. Están diseñados para emular muchas PAL pensadas para el uso de macroceldas. Si un usuario tiene un diseño que se implementa usando varias PAL comunes, puede configurar varias de las mismas GAL para emular cada de uno de los otros dispositivos. Esto reducirá el número de dispositivos diferentes en existencia y aumenta la cantidad comprada. Comúnmente, una cantidad grande del mismo dispositivo debería rebajar el costo

individual del dispositivo. Estos dispositivos también son eléctricamente borrables, lo que los hace muy útiles para los ingenieros de diseño.

Programación. Cuando los PLD fueron introducidos por primera vez, el diseñador lógico tenía que desarrollar un mapa de conexiones fusibles que mostraba las conexiones que debían quemarse y lo enviaba al fabricante de la PROM, PAL o PLA. El fabricante entonces programaba el dispositivo de acuerdo con el mapa de conexiones, probaba su funcionamiento y lo enviaba de vuelta al diseñador. En años recientes, la disponibilidad de equipo de programación de relativo bajo costo ha vuelto conveniente que los usuarios programen sus propios PLD.
En el mercado existen programadores universales que pueden programar PROM, PAL y PLA más comunes. El dispositivo que va a programarse se coloca en la base del programador; éste programa y prueba el dispositivo de acuerdo con los datos que fueron proporcionados por el usuario.
La programación junto con los datos de prueba se desarrollan utilizando para ello programación disponible en el mercado y que se pueda ejecutar sobre computadoras personales estándar. Al utilizar esta programación, el usuario introduce en la computadora los datos que describen las funciones lógicas que desea que queden programadas en el PLD además de la información sobre cómo probar el dispositivo. Entonces el programa genera el mapa de conexiones fusibles y los datos de prueba en una forma que pueda enviarse sobre un cable hacia la memoria del programador de PLD. Una vez que el programador tiene los datos, procede a programar y probar el dispositivo. Cuando termina de hacerlo, el programador indica si el dispositivo ha pasado o no el procedimiento de prueba. Si lo pasa, el dispositivo se quita de la base del programador y se coloca en el prototipo del circuito para realizar más pruebas con él.

En los dispositivos actuales se utilizan fundamentalmente dos tipos de estructuras programables.

- Matrices Lógicas Programables (PAL).
- Memorias RAM (Look-up Tables).

Ventajas de los PLDs

- *Velocidad.* Mayor que SSI; menor de ASICs.
- *Densidad de Integración.* Menor que ASICs.
- *Costo de Desarrollo.* Mucho menor que ASICs.
- *Desarrollo de prototipos, Depuración y Verificación.* Más sencillos que en ASICs.
- *Modificación de diseños.* Sencilla.
- *Costo.* Depende del volumen.

Diferentes siglas para los PLDs

- PLD. *Programmable Logic Device.*
- PLA. *Programmable Logic Array.*
- PAL. *Programmable Array Logic.*
- GAL. *Generic Array Logic.*
- CPLD. *Complex PLD.*
- EPLD. *Erasable PLD.*
- HCPLD. *High Complexity PLD.*
- FPGA. *Field Programmable Gate Array.*

Clasificación de los Dispositivos Lógicos Programables.

El esquema de la Fig. 2.16, es un cuadro sinóptico de una clasificación típica de los dispositivos lógicos programables de acuerdo a la densidad de integración.

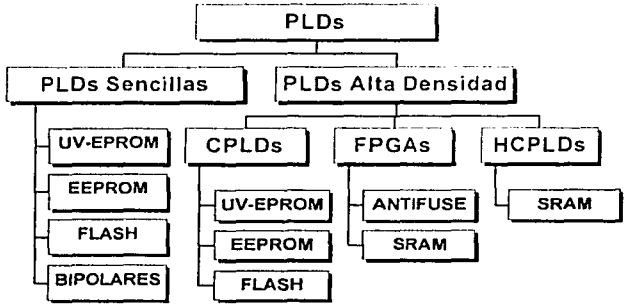


Fig. 2.16. Clasificación de los PLDs

Modos de Programación.

- Estructuras PAL. No volátil.
  - EPROM, EEPROM (Tecnología de programación por compuerta flotante).
  - PROM (Tecnología de programación por antifusible).
  - Programación en sistema (ISP: In System Programmability).
  - Programación autónoma.
- Memorias RAM. Volátil
- Distintos modos de Configuración.
  - Pasivos, Activos.
  - Serie, paralelo.

Configuración en sistema (In System Configurability ISC).

II.4 HERRAMIENTA DE DISEÑO MAX+PLUS II

El software MAX+PLUS II (Múltiple Array Matrix Programmable Logic User System II) suministra una plataforma múltiple e independiente de una arquitectura en particular, que se adapta fácilmente a las necesidades de cualquier diseño. Y por ello ofrece una forma fácil de diseñar proyectos y hacer su programación directa en los dispositivos.

Existen herramientas para crear los diseños de una manera jerárquica, poderosa síntesis lógica, compilación, partición, simulación funcional y en tiempo, simulación enlazada con varios dispositivos, análisis de tiempo, etc. MAX+PLUS II incluye once programas de aplicación y el ambiente integrado.

Se pueden realizar varios diseños en aplicaciones diferentes al mismo tiempo, también es posible ejecutar simultáneamente el compilador, simulador, analizador en el tiempo y el programador.

Este sistema de desarrollo soporta MULTIPLATAFORMAS, incluyendo Windows 95, NT, Windows 3.1,X-Windows (para Sun SPARCstation), HP9000 e IBM RISC System .

Con MAX+PLUS II es posible programar desde un PC y ahorrar costos y tiempo que se emplean implementando un diseño en discreto, automatiza muchas tareas de diseño que pueden ser reutilizadas para diseños más complejos, por ejemplo, si se desea implementar una ALU (unidad

aritmético-lógica) es posible reutilizar el diseño dentro de las múltiples funciones de un Microprocesador; cuenta con un sintetizador lógico que optimiza el uso de recursos y elimina lógicas redundantes , a su vez el programa particiona automáticamente los componentes que se necesitan dentro del diseño.

MAX+PLUS II soporta la Librería de Módulos Paramétricos LPM (Library of Parameterized Modules) que simplifica notoriamente diseños de alta densidad, con LPM se crean arquitecturas independientemente de los símbolos que se deban usar en MAX+PLUS II o en cualquier otra herramienta. Cuenta con un editor gráfico que permite usar diferentes especificaciones, creando automáticamente los símbolos de LPM .

Las once aplicaciones incluidas en el software de MAX+PLUS II son las siguientes:

- **Hierarchy Display (Jerarquía del Proyecto).** Muestra la distribución actual de la jerarquía del proyecto. Constituye un árbol con ramas que representan los subdiseños. Se puede identificar el tipo de archivo por medio de su extensión.

- **Graphic Editor (Editor Gráfico).** Te permite crear un diseño visual en un ambiente (WYSIWYG) " lo que ves es lo que tienes". Puedes usar los bloques predeterminados por ALTERA, o crear los que se necesiten.

- **Symbol Editor(Editor de Símbolos).** Puede convertir diseños previos en símbolos nuevos para ser utilizados en la jerarquía.

- **Text Editor (Editor de Texto).** Puede crear y editar diseños de texto basados en los lenguajes AHDL, VHDL y Verilog HDL. También permite crear, ver o editar archivos ASCII.

- **Waveform Editor (Editor de vectores).** Tiene dos funciones básicas (i) Como herramienta de diseño, (ii) Como herramienta para crear vectores de prueba y observar los vectores resultantes de la simulación.
- **Floorplan Editor.** Es para hacer una asignación manual de pines en el dispositivo con relación al diseño lógico.

- **Compiler.** Procesa los proyectos creados, encontrando posibles errores y generando los archivos de programación necesarios, así como la información necesaria para la simulación.
- **Simulator.** Permite revisar la operación lógica y funcional dando los diagramas de tiempo de las entradas del circuito lógico para encontrar el diagrama de tiempo de las salidas.

- **Timing Analyzer.** Analiza el rendimiento (performance) del circuito lógico, después de haber pasado por el compilador.

- **Programmer.** Sirve para programar, verificar, examinar y probar los dispositivos de ALTERA.

- **Message Processor.** Muestra mensajes de error, advertencias, e información sobre el estado actual del proyecto.

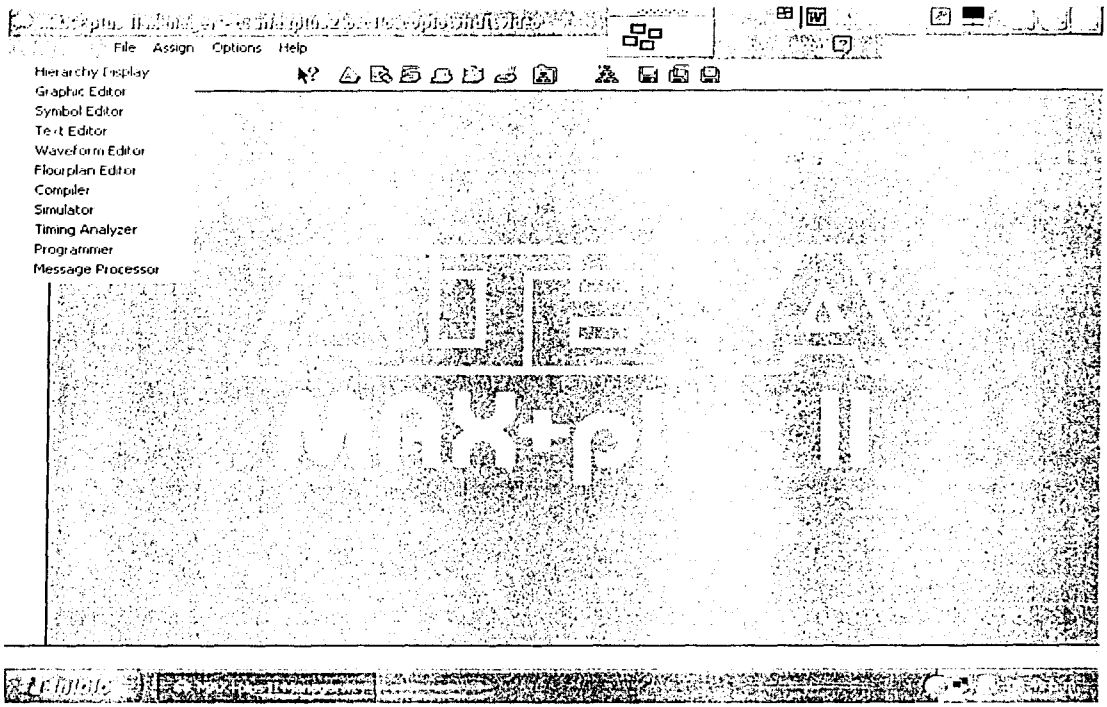


Figura 2.16 Las once aplicaciones de Max-Plus II. Cortesía de ALTERA.

COMO DISEÑAR EN MAX-PLUS II

Introducir un diseño significa el proceso de describir la arquitectura del diseño utilizando el método que sea soportado por Max-Plus II. Existen muchos métodos para introducir un diseño, tres editores a seleccionar (Text Editor, Graphic Editor, Waveform Editor), y dos editores secundarios para ayudar a introducir un diseño (Floorplan Editor, Symbol Editor).
La captura de cualquier esquemático, se efectúa en el editor gráfico.
La captura de diseños hechos en lenguaje de descripción VHDL, se efectúa en el editor de texto.

Esta sección suministra información para aprender a diseñar con Max-Plus II, y de esta forma, saber que los diseños trabajan apropiadamente al ser compilados en Max-Plus II.

Al introducir un diseño, tanto el dispositivo a usar, como la asignación de parámetros puede ser hecha por nosotros o permitir que el proyecto al ser compilado haga esta asignación automáticamente.

Entenderemos por "recurso" una porción de un dispositivo de ALTERA como un pin o celdas lógicas que realizan un trabajo específico. También se puede asignar manualmente los recursos, para asegurarnos que el compilador acomode exactamente el diseño en la forma que lo deseamos.

Un diseño jerárquico se logra utilizando las metodologías "top-down" (arriba-abajo) o "bottom-up" (abajo-arriba). Se pueden crear diseños grandes y complejos, fáciles de manejar o utilizar una librería de módulos (LPM) para crear subdiseños.

Diseñando con el Editor de Texto (Text Editor)

El editor de texto (Text editor) permite utilizar un lenguaje de alto nivel para ejecutar el diseño, actualmente el más utilizado es VHDL, se pueden utilizar librerías externas para el uso de funciones especiales, es el caso de la librería IEEE .

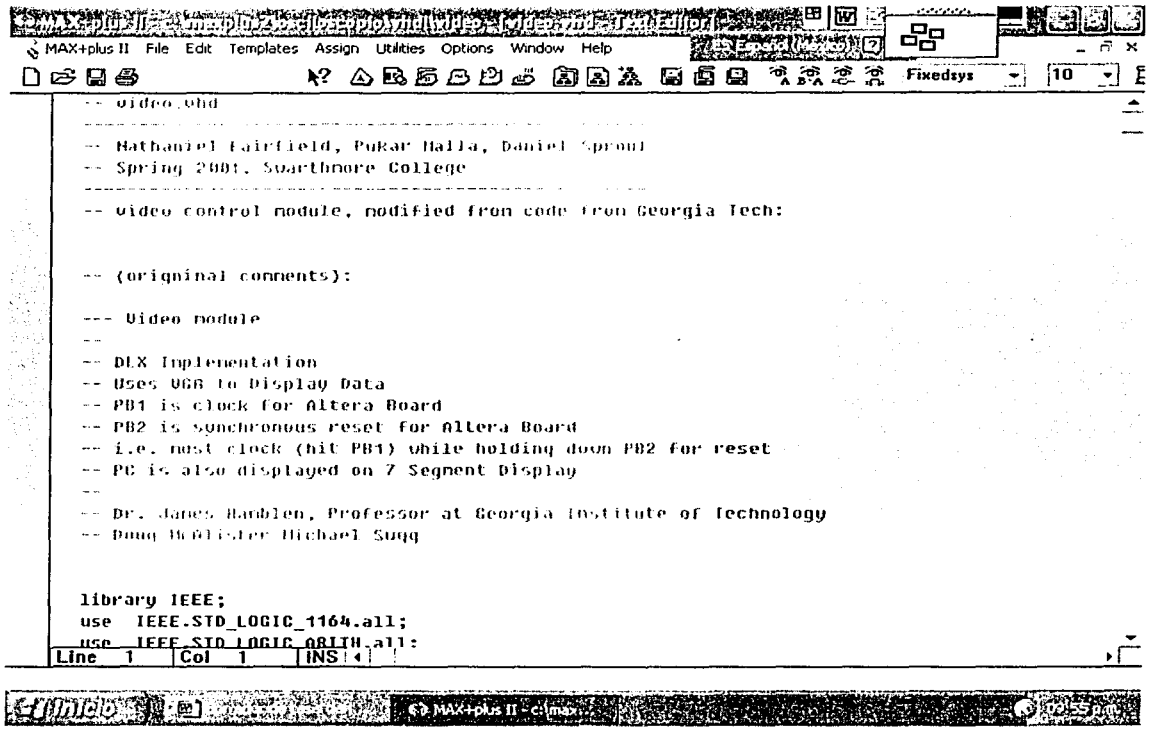


Figura 2.17 Editor de Texto. Cortesía de ALTERA.

Existen diversas opciones a cambio de utilizar lenguajes de programación, MAX+PLUS II ofrece un editor gráfico permite ejecutar enlaces entre componentes que ya están predefinidos. Esto facilita el desarrollo del plan, ya que lo único que se necesita es enlazar componentes.

Diseñando con el Editor Gráfico (Graphic Editor)

Este editor nos permite introducir diseños en un ambiente visual, permite crear archivos (gdf) que pueden incluir una combinación de megafunciones, macrofunciones y símbolos primitivos.

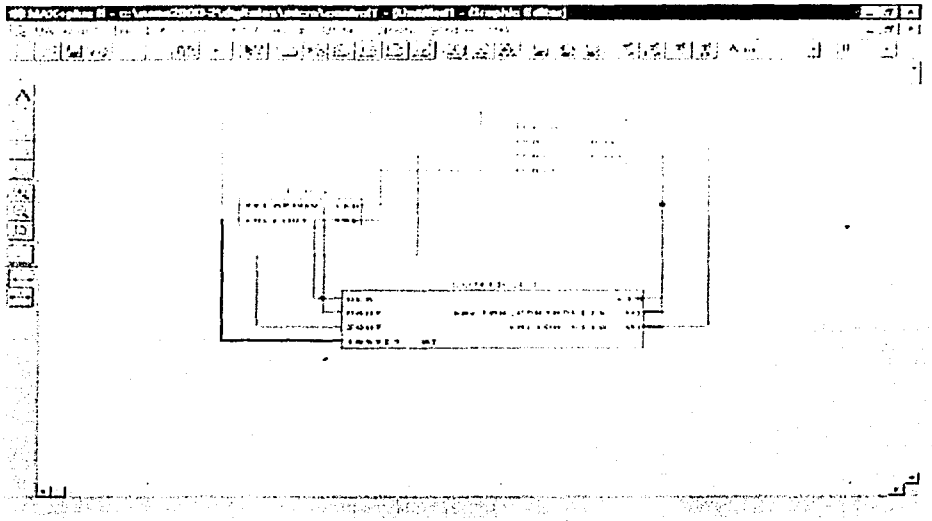


Figura 2.18 Editor Gráfico. Cortesía de ALTERA.

Compilador (Compiler)

El compilador (compiler) interpreta e implementa la lógica en LPM para optimizar la utilización del dispositivo, revisa de manera superficial los posibles errores que se pueden presentar en todo el proceso. Parte de ésta revisión se basa en detectar los errores de sintaxis ó lógicos que existan al editar el programa. Cuando usted compila se ejecutan una serie de archivos de programa que permiten verificar si el programa posee una sintaxis correcta y a la vez permite ver las inconsistencias que tenga el programa a pesar de haber compilado correctamente (Warnings) .

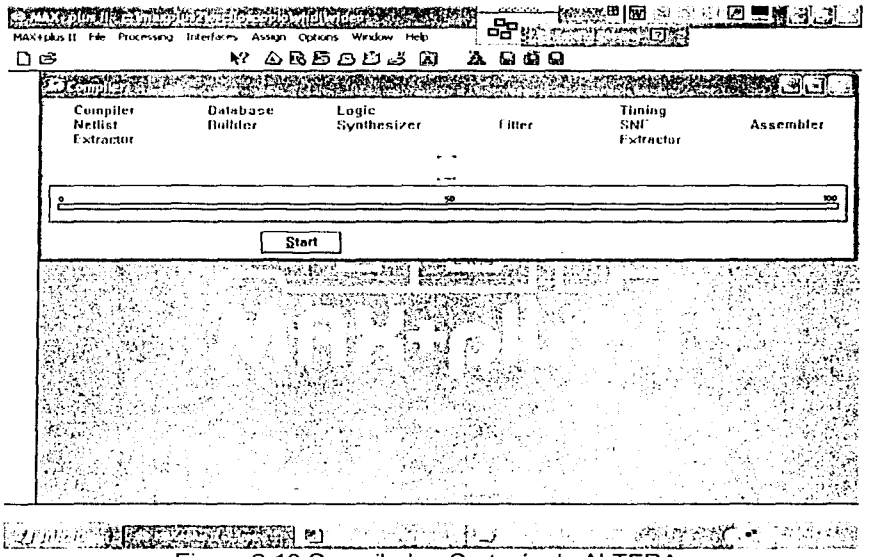


Figura 2.19 Compilador. Cortesía de ALTERA.

Simulador (Simulator)

El MAX+PLUS II Simulador permite verificar el funcionamiento del programa , el hecho de que compile no implica que simule correctamente, se pueden presentar problemas de tipo síncrono. Dentro del proceso de simulación existen dos pasos: El primero es crear un archivo con extensión scf el cual pertenece al editor de forma de onda, es decir se debe ingresar al editor de forma de onda, salvarlo con el mismo nombre del archivo que se edita en VHDL o gráficamente y con extensión SCF. El segundo paso es salvar y simular .

Editor de Onda (Wave Form Editor)

El Editor de Forma de onda (Waveform editor) permite crear una lista con la forma de onda de las entradas y analiza la forma de onda resultante en las salidas que se contemplen dentro del diseño.

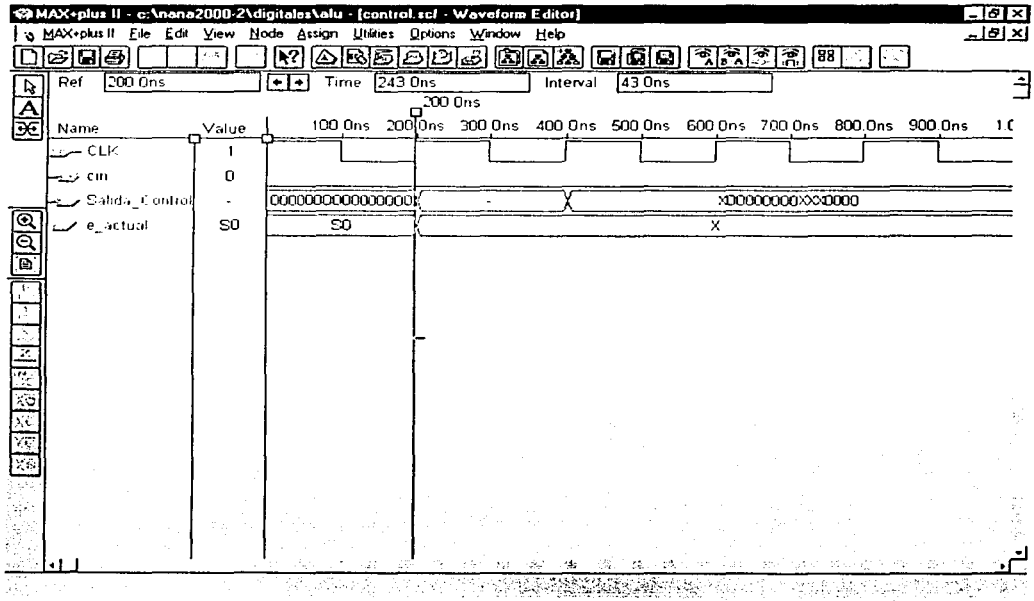


Figura 2.20 Simulador-Editor de Onda. Cortesía de ALTERA.

Analizador de Tiempo (Timing Analyzer)

MAX+PLUS II cuenta el analizador de tiempo (Timing Analyzer) que analiza el tiempo total de ejecución en tiempo real del diseño, ya que como se expuso con anterioridad, los dispositivos también cuentan con tiempos de retardo.

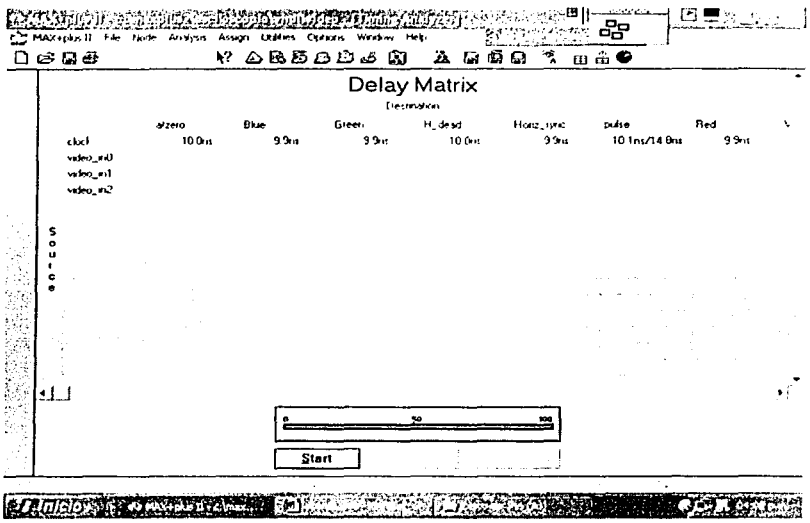


Figura 2.21 Analizador de tiempo. Cortesía de ALTERA.

Programador (Programmer)

Para la programación del dispositivo de debe contar con una tarjeta de aplicación y un programa auxiliar adicional a MAX+PLUS II denominado UNIVERSE, esta tarjeta cuenta con una interfase que va conectada al PC en el puerto paralelo. Antes de programar, cabe aclarar que el programa ubica automáticamente las entradas y salidas en los pines del dispositivo, sin embargo existe una manera de ubicar manualmente los pines, esto se hace por medio del FLOORPLAN EDITOR.
El editor de plan (Floorplan editor), permite apreciar una implementación física del diseño lógico, en este se pueden asignar las entradas y salidas a los pines del dispositivo aleatoriamente.

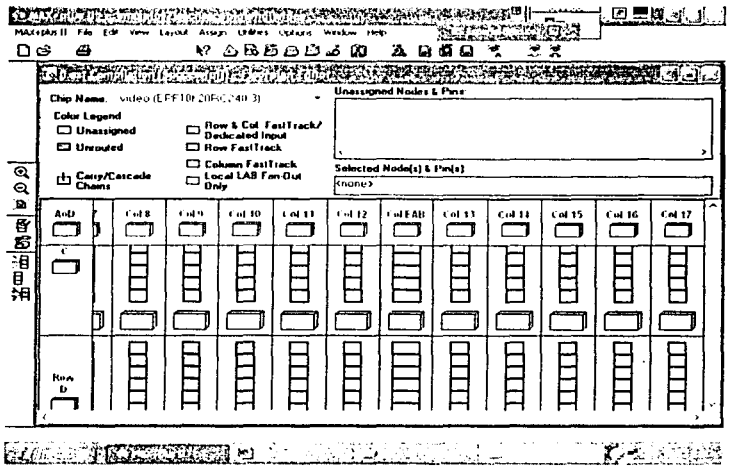


Figura 2.22 Floorplan Editor. Cortesía de ALTERA.

El último paso es programar, MAX+PLUS II cuenta con varios pasos para el proceso final.

- "programmer"
- "examine" realiza una revisión física de los dispositivos externos
- "program" meta de todo el proceso de diseño.
- "Verify" Examina que la programación se haya ejecutado con éxito.

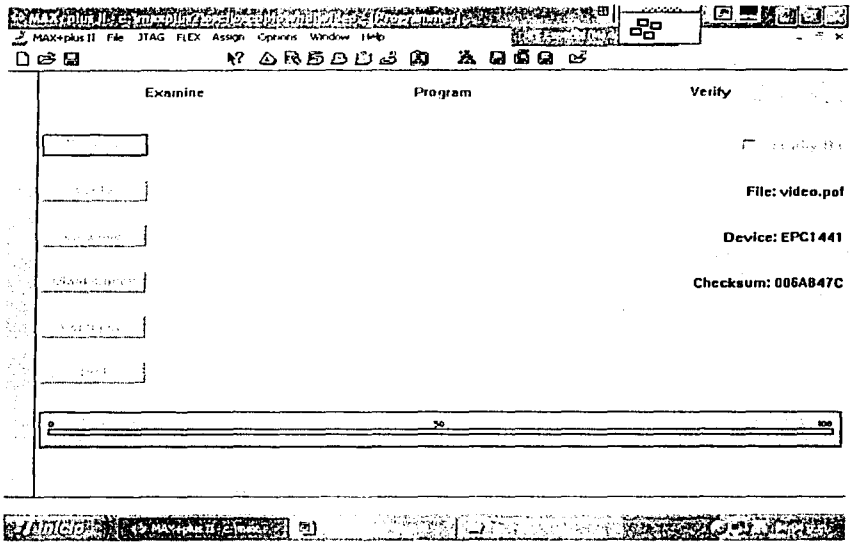


Figura 2.23 Programador. Cortesía de ALTERA.

II.5 ARQUITECTURA DE LOS DISPOSITIVOS DE ALTERA.

- Celdas lógicas (logic cell). Es un bloque básico en cualquier dispositivo.
- Macroceldas (macrocell). Bloque básico basado en suma de productos (SOP) en cualquier dispositivo Max.
- Elementos lógicos (logic elements). Bloque básico basado en " look-up table" en cualquier dispositivo Flex.
- Arreglos lógicos (logic array). Grupo de celdas lógicas.

Altera maneja siete familias lógicas programables, cuatro que utilizan (SOP)

- Clásica
- Max 5000
- Max 7000 y Max 7000S
- Max 9000

Tres familias basadas en tablas de (LOOK-UP)

- Flash logic
- Flex 8000
- Flex 10k

TARJETA DE DESARROLLO UP1

El paquete de diseño de laboratorio de programa universitario de ALTERA, fue diseñado como una herramienta para el desarrollo de prácticas de laboratorio utilizando la tecnología de dispositivos lógicos programables complejos (CPLD).
La tarjeta de desarrollo UP1 ver figura 2.24, tiene dos dispositivos CPLD, el primero es el EPM7128S de 84 pines en encapsulado PLCC, el segundo es el EPF10K20 de 240 pines en encapsulado RQPF. La tarjeta UP1 puede ser utilizada desde la versión 7-21 del software MAX+PLUS II.

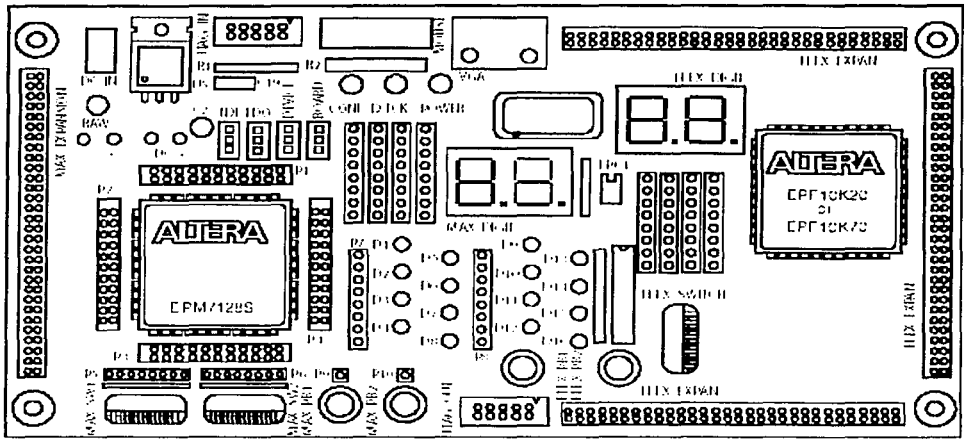


Figura 2.24 Diagrama de bloques de la tarjeta de desarrollo UP1. Cortesía de ALTERA.

DC_IN & RAW Power Input (Entrada para conector de DC)
El DC_IN & RAW Power Input es una entrada que acepta un conector hembra de 2.5 mm X 5.55 mm. La alimentación de entrada aceptable de DC es de 7 hasta 9 volts con un mínimo de 350 mA. El RAW Power Input consiste en dos orificios para conectar una fuente no regulada. El orificio marcado con un signo más (+) es la entrada positiva; el orificio marcado con un signo menos (-) es el común de la tarjeta.

Regulador de voltaje
La UP1 trae un regulador interno (LM340T) para asegurar la alimentación de 5 V a los dispositivos Altera. Un LED verde identificado con la etiqueta POWER se ilumina cuando la corriente esta fluyendo a través de la alimentación regulada de 5 V DC.

Oscilador
La UP1 contiene un oscilador de cristal de 25 175 MHz. La salida del oscilador maneja una entrada de reloj global en el dispositivo EPM7128S (pin 83) es una señal TTL que maneja una entrada de reloj global del EPM7128S (pin 83) y en el Flex 10K (pin 91).

JTAG_IN Header

Es la entrada donde se conecta el cable Byte-BlasterMV que se utiliza para transferir la información para la grabación (programación) del dispositivo desde el puerto paralelo hasta la UP1. Los datos son transferidos hacia los dispositivos vía pin TDI y desde los dispositivos vía pin TDO. La tabla 2.1, identifica la nomenclatura de este conector

Pin	Señal JTAG
1	TCK
2	GND
3	TDO
4	VCC
5	TMS
6	Sin conexión
7	Sin conexión
8	Sin conexión
9	TDI
10	GND

Tabla 2.1 Nomenclatura del conector JTAG_IN

Jumpers

La UP1 tiene cuatro jumpers de tres pines cada uno (TDI, TDO, DEVICE y BOARD) que en conjunto conforman el JTAG. El JTAG conjunta una variedad de modos a configurar (por ejemplo, para programar solo el EPM7128S o el EPF10K20, programar ambos dispositivos o conectar múltiples tarjetas UP juntas). La Tabla 2.3, muestra la configuración de los jumpers para una acción deseada.

TD1	TDO	DEVICE	BOARD
C1	C1	C1	C1
C2	C2	C2	C2
C3	C3	C3	C3

Tabla 2.2 Posiciones de los conectores C1, C2 y C3.

Acción deseada	TD1	TD0	DEVICE	BOARD
Solo programa el dispositivo EPM7128S	C1 & C2	C1 & C2	C1 & C2	C1 & C2
Solo configura el dispositivo EPF 10K20	C2 & C3	C2 & C3	C1 & C2	C1 & C2
Programa/configura ambos dispositivos	C2 & C3	C1 & C2	C2 & C3	C1 & C2
Conecta varias tarjetas juntas	C2 & C3	ABIERTO	C2 & C3	C2 & C3

Tabla 2.3 Define las posiciones de los jumpers para cada configuración.

Durante la configuración, el LED verde CONF_D se apagará y el LED verde TCK modulará para indicar que el dato es transferido. Después de que el dispositivo ha logrado la configuración, el LED CONF_D se iluminará.

II.5.1 Dispositivo EMP7128S (figura 2.25)

Es un dispositivo de mediana densidad dentro de la familia Max 7000S, se basa en tecnología EEPROM. Tiene capacidad para 128 macroceldas sumando en total una capacidad aproximada de 2500 compuertas.

Este chip contiene 84 patas con 128 macroceldas. Tiene un arreglo de "AND's" programables, "OR" fijas y un flip-flop que se puede configurar independientemente. Este chip tiene una capacidad aproximada de 2500 compuertas y es ideal para diseños combinacionales y funciones secuenciales. En la siguiente figura se muestra el diagrama a bloques de este dispositivo.

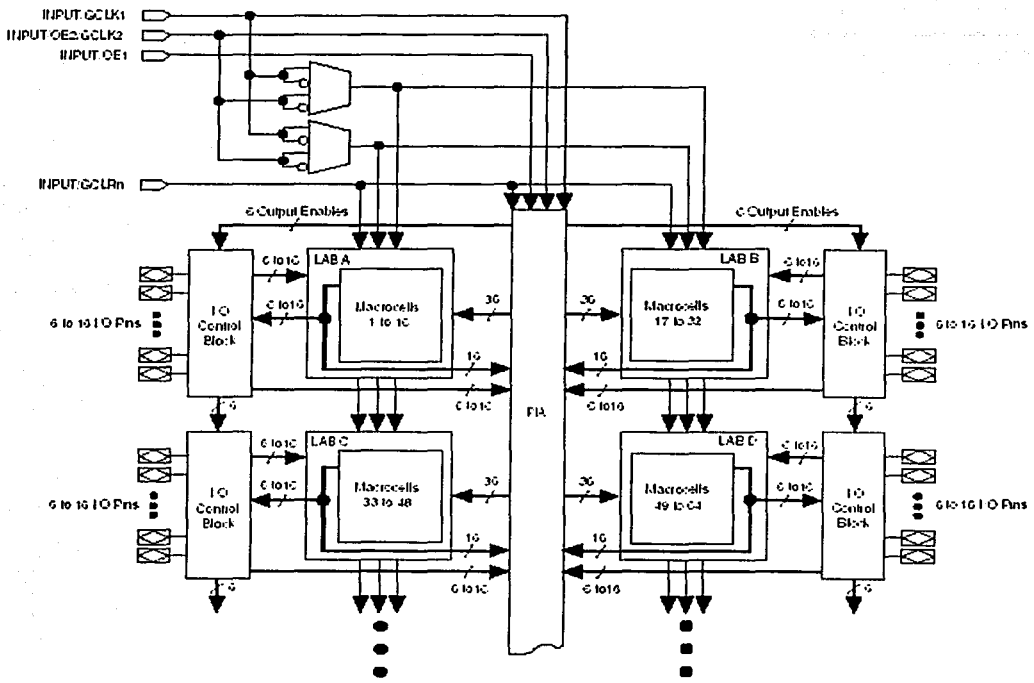


Figura 2.25 Dispositivo Max7000S

Uso del Dispositivo EPM7128S

La UP1 provee los siguientes recursos para el dispositivo EPM7128S

- Socket para montar un dispositivo PLCC de 84 pines
- Pines de señales que son accesibles vía conectores hembras
- Conexión JTAG para el Byte Blaster
- Dos push-button
- Dos dipswitches de 8 bits cada uno
- 16 leds
- Dos displays de 7 segmentos
- Un oscilador de cristal de 25.175 MHz

- Un puerto de expansión con 42 pines de E/S y los pines dedicados globales CLR, OE1, OE2 y GCLK2

Conectores alrededor del EPM7128S. Estos conectores rodean al dispositivo EPM7128S y proveen acceso a todos los pines del chip. La distribución de estos pines se muestra en la tabla 2.4.

P1		P2		P3		P4	
Outside	**Inside**	**Outside**	**Inside**	**Outside**	**Inside**	**Outside**	**Inside**
75	76	12	13	33	34	54	55
77	78	14	15	35	36	56	57
79	80	16	17	37	38	58	59
81	82	18	19	39	40	60	61
83	84	20	21	41	42	62	63
1	2	22	23	43	44	64	65
3	4	24	25	45	46	66	67
5	6	26	27	47	48	68	69
7	8	28	29	49	50	70	71
9	10	30	31	51	52	72	73
11	X	32	X	53	X	74	X

Tabla 2.4 Conectores alrededor del dispositivo EPM7128S

Push-Buttons MAX_PB1 & MAX_PB2
Son dos push-buttons que proveen señales activas en baja, y son puestos arriba a través de resistores de 10 K Ω. Conexiones a estas señales son fácilmente hechas insertando cables en el conector hembra de los push-button. La punta del otro cable debe ser insertada en la entrada hembra apropiada asignada al pin E/S del dispositivo EPM7128S.

MAX_SW1 & MAX_SW2 SWITCHES
Cada uno contiene 8 switches que proveen señales lógicas. Estos switches son puestos arriba a través de resistores de 10 KΩ. Conexiones a estas señales son hechas insertando una punta de cable en el conector hembra alineado con el interruptor apropiado. Luego debe insertarse la punta del otro cable en la entrada hembra apropiada asignada al pin E/S del EPM7128S. La salida del interruptor es un 1 lógico cuando el interruptor está abierto y un 0 lógico cuando el interruptor está cerrado.

Leds D1 hasta D6
Son 16 LEDs que están puesta arriba a través de resistencias de 330 Ω. Un LED es iluminado cuando un "0" lógico es aplicado al conector hembra asociado con el LED. Los LEDs D1 hasta el D8 son conectados en la misma secuencia a los conectores hembra (por ejemplo D1 es conectado a la posición 1, etc.) Los LEDs D9 hasta el D16 son conectados en la misma secuencia a los conectores hembra (ejemplo, D9 es conectado en la posición 1,etc.). Las posiciones de la numeración de los LEDs se ilustra en la figura 2.26.

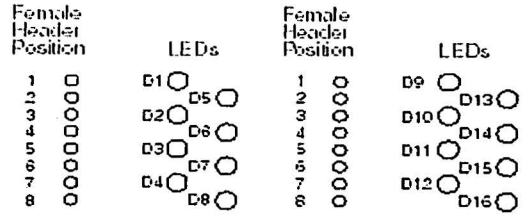
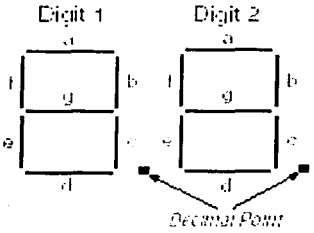


Figura 2.26 Posición de los LEDs.

MAX_DIGIT Display

Es un display dual de siete segmentos conectado directamente al dispositivo EPM7128S. Cada LED segmento del display puede ser iluminado manipulando el pin E/S respectivo al que está conectado en el dispositivo EPM7128S con un 0 lógico. La figura 2.27, muestra el nombre de cada segmento.



2.27 Nombre de los segmentos en los displays.

Segmento del display	Pin para dígito 1	Pin para dígito 2
a	58	69
b	60	70
c	61	73
d	63	74
e	64	76
f	65	75
g	67	77
Punto decimal (.)	68	79

Tabla 2.5 Conexiones de los pines del dispositivo hacia los displays.

MAX_EXPANSION

Es un renglón dual de agujeros de 0.1 pulgadas para acceder a las señales de E/S y señales globales en el dispositivo EPM7128S , Vcc y Gnd. La figura 2.28 muestra la convención de números para los agujeros. La Tabla 2.6 lista los nombres de las señales contra los pines conectados a cada agujero.

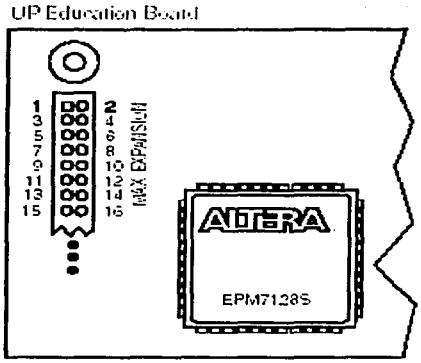


Figura 2.28 Convención de números para los agujeros.

Número de agujero	Señal/Pin	Número de agujero	Señal/Pin
1	RAW	2	GND
3	Vcc	4	GND
5	Vcc	6	GND
7	Sin conexión	8	Sin conexión
9	Sin conexión	10	Sin conexión
11	Sin conexión	12	GLCRn/1
13	OE1/84	14	OE2/GCLK2/2
15	4	16	5
17	6	18	8
19	9	20	10
21	11	22	12
23	15	24	16
25	17	26	18
27	20	28	21
29	22	30	24
31	25	32	27
33	28	34	29
35	30	36	31
37	33	38	34
39	35	40	36
41	37	42	39
43	40	44	41
45	44	46	45
47	46	48	48
49	49	50	50
51	51	52	52
53	54	54	55
55	56	56	57
57	Vcc	58	GND
59	Vcc	60	GND

Tabla 2.6 Nombres de las señales contra los pines conectados en cada agujero.

II.5.2 Dispositivo FLEX 10K (figura 2.29)

La Up1 proporciona los siguientes recursos para el dispositivo EPF10K20. Los pines de este dispositivo están preasignados a los switches y LEDs sobre la tarjeta. Estos recursos son:

- Conexión JTAG para el Byte Blaster
- Socket para una EPROM de configuración ECP1
- Dos switches push-buttons
- Un dipswitch octal
- Un display dual de siete segmentos
- Un oscilador de 25.175 MHz
- Puerto VGA
- Puerto para Mouse
- Tres puertos de expansión, cada uno con 42 pines de E/S y siete pines globales

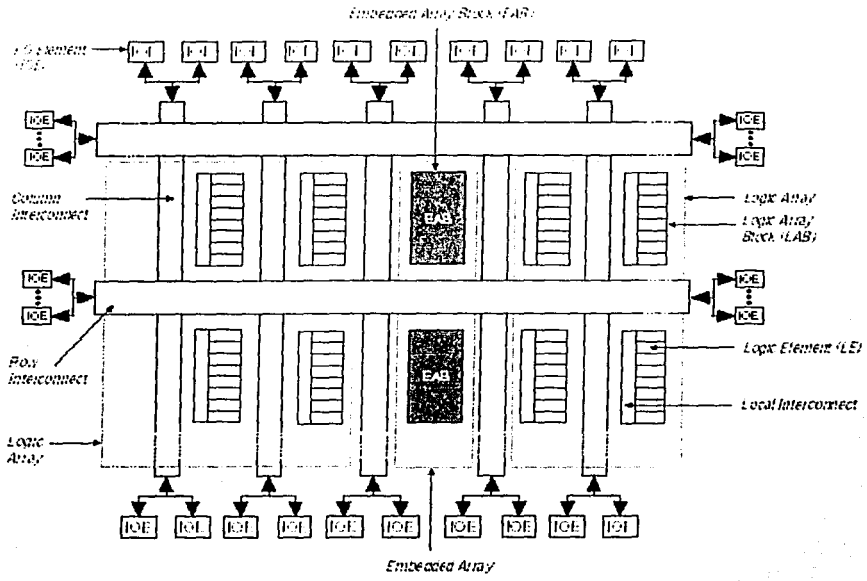


Figura 2.29 El dispositivo Flex 10K

Push buttons FLEX_PB1 y FLEX_PB2
Estos son dos push buttons que proveen señales activas bajas a dos pines de E/S de propósito general en el FLEX 10K. FLEX_PB1 se conecta al pin 28, y el FLEX_PB2 se conecta al pin 29. Cada push button esta puesto arriba a través de resistores de 10 K Ω.

FLEX_SW1 Switches
Contiene ocho switches que proporcionan señales lógicas en "1" cuando el switch está abierto y "0" cuando el switch está cerrado. La tabla 2.7, ilustra la asignación de pines a cada switch.

Switch	EPF10K20 Pin
FLEX_SWITCH-1	41
FLEX_SWITCH-2	40
FLEX_SWITCH-3	39
FLEX_SWITCH-4	38
FLEX_SWITCH-5	36
FLEX_SWITCH-6	35
FLEX_SWITCH-7	34
FLEX_SWITCH-8	33

Tabla 2.7 Asignación de pines para los switches FLEX_SW1.

Display FLEX-DIGIT

Es un display dual de siete segmentos conectado directamente al dispositivo EPF10K20. Cada segmento puede ser iluminado poniendo un "0" lógico en el pin correspondiente del dispositivo. La tabla 2.8, ilustra la asignación de pines para cada display.

Segmento del display	Pin para digito 1	Pin para digito 2
a	6	17
b	7	18
c	8	19
d	9	20
e	11	21
f	12	23
g	13	24
Punto decimal	14	25

Tabla 2.8 Asignación de pines para los displays de siete segmentos.

Interfaz VGA

La interfaz VGA permite al FLEX 10K controlar un monitor de video externo. Esta interfaz está integrada de un simple diodo resistor y un conector D-sub de 15 pines (identificado VGA), donde la conexión del monitor puede conectarse a la UP1. El diodo resistor y el conector D-sub están diseñados para generar voltajes que conforman el estándar VGA.
La información de color, renglón y columna indicados en la pantalla es enviada desde el FLEX 10K al monitor por medio de 5 señales. Tres señales VGA son la información de color: rojo, verde y azul, mientras las otras dos señales son para la sincronización vertical y horizontal. Manipulando estas señales representamos imágenes en la pantalla del monitor. La tabla 2.9, lista las conexiones del conector D-sub y el FLEX 10K.

Señal	Conector D-sub pines	Pines del FLEX 10K
Rojo	1	236
Verde	2	237
Azul	3	238
GND	6,7,8,10,11	-
Horiz_Sync	13	240
Vert_Sync	14	239
Sin conexión	4,5,9,15	-

Tabla 2.9 Conexiones del conector D-sub.

FLEX_EXPAN_A, FLEX_EXPAN_B, FLEX_EXPAN_C
Son renglones duales de agujeros de 0.1 " para acceso a las señales de E/S, señales globales, de Vcc y Gnd de los pines del dispositivo EPF10K20. En la figura 2.30, se muestra la convención de números para estos agujeros. En las tablas 2.10, 2.11 y 2.12, se ilustran los pines conectados a cada agujero.

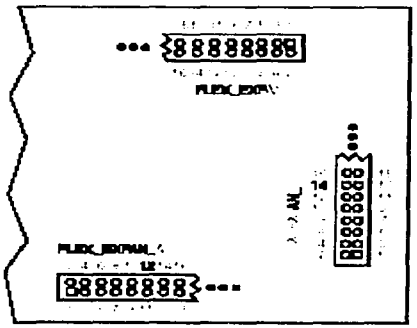


Figura 2.30 Convención de números para los agujeros de los conectores.

Número de agujero	Señal/Pin	Número de agujero	Señal/Pin
1	RAW	2	Gnd
3	Vcc	4	Gnd
5	Vcc	6	Gnd
7	Sin conexión	8	DI1/99
9	DI2/92	10	DI3/210
11	DI4/212	12	DEV_CLR/209
13	DEV_OE/213	14	DEV_CLK2/211
15	45	16	46
17	48	18	49
19	50	20	51
21	53	22	54
23	55	24	56
25	61	26	62
27	63	28	64
29	65	30	66
31	67	32	68
33	70	34	71
35	72	36	73
37	74	38	75
39	76	40	78
41	79	42	80
43	81	44	82
45	83	46	84
47	86	48	87
49	88	50	94
51	95	52	97
53	98	54	99
55	100	56	101
57	Vcc	58	GND
59	Vcc	60	GND

Tabla 2.10 Nombre de las señales contra los pines conectados en cada agujero del FLEX_EXPAN_A

Número de agujero	Señal/Pin	Número de agujero	Señal/Pin
1	RAW	2	Gnd
3	Vcc	4	Gnd
5	Vcc	6	Gnd
7	Sin conexión	8	DI1/99
9	DI2/92	10	DI3/210
11	DI4/212	12	DEV_CLR/209
13	DEV_OE/213	14	DEV_CLK2/211
15	109	16	110
17	111	18	113
19	114	20	115
21	116	22	117
23	118	24	119
25	120	26	126
27	127	28	128
29	129	30	131
31	132	32	133
33	134	34	136

35	137	36	138
37	139	38	141
39	142	40	143
41	144	42	146
43	147	44	148
45	149	46	151
47	152	48	153
49	154	50	156
51	157	52	158
53	159	54	161
55	162	56	163
57	Vcc	58	GND
59	Vcc	60	GND

Tabla 2.11Nombres de las señales contra los pines conectados en cada agujero del FLEX_EXPAN_B

Número de agujero	Señal/Pin	Número de agujero	Señal/Pin
1	RAW	2	Gnd
3	Vcc	4	Gnd
5	Vcc	6	Gnd
7	Sin conexión	8	DI1/99
9	DI2/92	10	DI3/210
11	DI4/212	12	DEV_CLR/209
13	DEV_OE/213	14	DEV_CLK2/211
15	175	16	181
17	182	18	183
19	184	20	185
21	186	22	187
23	188	24	190
25	191	26	192
27	193	28	194
29	195	30	196
31	198	32	198
33	200	34	201
35	202	36	203
37	204	38	206
39	207	40	208
41	214	42	215
43	217	44	218
45	219	46	220
47	221	48	222
49	223	50	225
51	226	52	227
53	228	54	229
55	230	56	231
57	Vcc	58	GND
59	Vcc	60	GND

Tabla 2.12 Nombres de señales contra los pines conectados en cada agujero del FLEX_EXPAN_C.

II.6 LENGUAJES DE DESCRIPCIÓN DE HARDWARE (HDL)

Gracias a las necesidades tecnológicas actuales, se han desarrollado los llamados HDL que son lenguajes de descripción de hardware, los cuales nos permiten describir hardware utilizando especificaciones de alto nivel, es decir, son lenguajes de alto nivel que nos sirven para crear hardware.

Los HDL disponibles actualmente nos permiten diferentes formas de especificar un diseño, estas formas también conocidas como "estilos de escritura", las cuales van desde especificaciones solo funcionales hasta especificaciones estructurales VHDL (Very High Speed Integrated Circuit Hardware Description Languages) es uno de estos lenguajes de descripción de hardware que es originado por el departamento de Defensa de los EEUU a mediados de 1980.

VHDL fue establecido como un estándar IEEE 1076 en 1987 y en 1993 fue actualizado al estándar IEEE 1164, así VHDL se ha convertido en un estándar comercial, para la descripción modelado y síntesis de circuitos y sistemas digitales.

II.6.1 VHDL

VHDL tiene las siguientes características:

- Los diseños se pueden descomponer en forma jerárquica.
- Cada elemento tiene una interface bien definida y una especificación precisa del comportamiento.
- Las especificaciones de comportamiento pueden usar un algoritmo o una estructura actual de hardware para definir la operación que realiza un elemento.
- Se puede modelar la concurrencia, inherente al hardware VHDL puede manejar estructuras de circuitos secuenciales asíncronos y síncronos.
- La operación lógica y el comportamiento en el tiempo de un diseño se puede simular.

Otra de las ventajas del VHDL el que con este se pueden escribir y sintetizar rápidamente circuitos 5, 10, 20 mil compuertas además se pueden diseñar y simular en un chip cualquier diseño desde un circuito combinacional simple hasta un sistema de microprocesador completo.

VHDL tiene además poderosos constructores con los cuales es muy sencillo escribir un código que describa lógica de control compleja, soporta librerías de diseño y la creación de componentes reutilizables.

VHDL permite crear un diseño sin tener que escoger a priori un dispositivo en el cual implementarlo, además permite simular la misma descripción del diseño que se sintetiza, lo que permite localizar errores en el funcionamiento del sistema y corregir los antes de que se implemente físicamente el diseño.

Existen varios pasos a seguir en el proceso de diseño de VHDL, a esto se le conoce comúnmente como el flujo del diseño. Este comienza con la descripción del sistema a un nivel de bloques. Los diseños lógicos de grandes dimensiones son generalmente jerárquicos y VHDL nos permite trabajar con facilidad para definir módulos y sus interfaces.

El siguiente paso es escribir el código VHDL, el cual se puede escribir en cualquier editor de texto, sin embargo la mayoría de los ambientes de diseño incluyen un editor de texto especializado para VHDL.

El siguiente paso en la compilación del código. Un compilador de VHDL analiza en el código los errores de sintaxis y checa la compatibilidad con otros módulos. Así también crea la información interna necesaria para que un simulador procese posteriormente el diseño.

La siguiente etapa es la simulación, un simulador VHDL permite definir y aplicar entradas al diseño y observar sus salidas, sin tener que construir físicamente el sistema.

A continuación siguen las etapas de síntesis. Esta etapa de síntesis convierte la descripción VHDL en un conjunto de primitivas o componentes que pueden ser ensamblados en la tecnología del dispositivo elegido.
El último paso es la verificación de tiempos del circuito implementado. En esta etapa se pueden calcular con acertada precisión los retrasos en el circuito debido a diversos factores.

Entidades y Arquitectura
Los bloques constitutivos básicos de cualquier diseño VHDL se llaman declaración de entidad y cuerpo de la arquitectura.

La entidad es la unidad primaria del lenguaje que identifica a los dispositivos mediante la definición de su interfaz. En estos términos podemos decir que una entidad es una abstracción de un diseño que representa un sistema completo o una tarjeta o un chip incluso una pequeña función. Así se afirma que una entidad y cuerpo de la arquitectura.

La declaración de una entidad describe las entradas y salidas del diseño de una entidad. Esta descripción de entradas y salidas es útil en la utilización de dicha entidad dentro de un diseño jerárquico.

Puertos
Cada señal E/S en la declaración de una entidad se refiere con el nombre de port, que es la analogía con un pin en el símbolo esquemático. Un puerto es un objeto de datos que se le puede asignar valores y usar en expresiones. El conjunto de puertos definidos para una entidad se refiere como port declaration, y cada puerto que se declare debe tener un nombre, una dirección y un tipo de dato asociado.

Modo
El modo describe la dirección en la cual los datos son transferidos hacia el puerto, este puede ser: in, out, inout, y buffer, y si este no es especificado será asignado in.
El uso de estos modos es el siguiente:

IN: Los datos fluyen únicamente hacia el interior de la entidad.
OUT: Los datos fluyen únicamente desde la fuente en el interior de la entidad hacia el puerto de salida.
BUFFER: Se utiliza para retroalimentación interna, es decir, el puerto también se usa como un operador dentro de la misma arquitectura.
INOUT: Es utilizado para el manejo de señales bidireccionales.

Además de los identificadores de los puertos y sus modos también se deben declarar los tipos de los datos de los puertos. Estos son provistos por el estándar IEEE 1076193, los cuales son boolean, bit, bit vector, integer. Los más útiles y bien soportados son los del paquete IEEE std_logic_1164 que son std_ulogic y std_logic. Para que estos tipos sean reconocidos es necesario hacerlos visibles a la entidad mediante la forma de una librería y cláusulas como en los lenguajes de programación.

Cuerpo de la Arquitectura
La arquitectura es una unidad secundaria del lenguaje. Esta se encarga de describir el funcionamiento del dispositivo identificado por ella. Para lograr lo anterior se sirve de los procesos en las descripciones funcionales y de los componentes en las estructurales. Esto indica que la entidad es vista como la caja negra, pero en el cuerpo de la arquitectura se describe el contenido de ella.

VHDL cuenta con 3 estilos de arquitecturas diferentes:

Descripción de flujo de datos (data flow)
Descripción por comportamiento (behavioral)
Descripción estructural (estructural)

Descripción de Flujo de Datos
En este, las siguientes dos características son utilizadas en el diseño:

- Se especifica la manera en que los datos son transferidos de los puertos de entrada a los puertos de salida.
- Se utilizan únicamente asignaciones mediante expresiones en las que se indica como cambian los puertos de salida en función de los puertos de entrada, ya sean asignaciones condicionales mediante instrucciones concurrentes o simples ecuaciones.

En VHDL todos los bloques son concurrentes, es decir que se ejecutan en todo momento, esto indica que no existe un orden especifico de la ejecución de las asignaciones, por lo cual el orden en que las instrucciones son ejecutadas depende de los eventos ocurridos en las señales, como sucede en un circuito.

Se puede decir que en este estilo no se utilizan sentencias secuenciales sino solo asignaciones concurrentes.

Estructuras de Ejecución Concurrente
En este estilo existen estructuras que se ejecutan concurrentemente e indican la asignación a puertos de salida en función de los puertos de entrada. Las estructuras más comunes son

- Asignación Condicional WHEN... ELSE
- Asignación WITH...SELECT...WHEN
- Ecuaciones Booleanas

Descripción Por Comportamiento
Estas descripciones son parecidas a las descripciones hechas en un lenguaje de programación de alto nivel. En esta no se especifica la estructura o la forma en que se deben conectar los componentes sino únicamente a describir su comportamiento.

Esto último consiste en una serie de instrucciones, que ejecutadas secuencialmente, modelan el comportamiento del circuito.

En VHDL las descripciones por comportamiento implican el uso de por lo menos un bloque Process, en el cual se escriben las instrucciones de VHDL que son ejecutadas secuencialmente.

Los procesos se pueden utilizar en el interior de cualquier arquitectura definiendo para sí mismo una región de declaraciones y para la codificación secuencial. Esta región de codificación puede contener únicamente instrucciones secuenciales. En la zona de declaraciones se permite designar constantes, señales, tipos de datos o algún alias.

La sintaxis de un proceso es la siguiente.

```
Process (lista sensitiva)
        - declaraciones
begin
        - instrucciones secuenciales
end process;
```

La lista sensitiva es opcional y define que señales provocan que las instrucciones dentro del bloque comiencen a ser ejecutadas.

Así cualquier cambio en alguna de las señales de esta lista, provoca que el proceso sea llamado. La especificación de dicha lista nos puede servidor para especificar si estamos modelando lógica combinacional o secuencial.

El funcionamiento del proceso es de la siguiente manera:

Las señales dentro de la lista sensitiva funcionan como entradas de interrupción y las instrucciones secuenciales se encuentran dentro de la rutina única de servicio de interrupción.

Cuando alguna de las señales de la lista sensible cambia, provoca que el proceso comience a funcionar y a ejecutar toda esta rutina de ejecución secuencial con la particularidad de que lo que resulte de este procesamiento se asigne únicamente al final de la estructura.

Estructuras de Ejecución Secuencial

IF THEN ELSE
Esta instrucción permite probar una condición y elegir 2 caminos de acción diferente, dependiendo si la condición se cumple o no.

FOR LOOP
Esta instrucción es un ciclo iterativo que permite repetir un bloque secuencial un determinado número de veces conocido previamente mediante una variable conocida que sirve para contar las iteraciones.

WHILE LOOP
Esta instrucción secuencial es un ciclo iterativo que permite repetir un bloque secuencial un número de veces no determinado. El número de iteraciones depende de la condición que se prueba al inicio del ciclo.

WAIT
Esta instrucción es utilizada en procesos que no tienen lista sensitiva, ya que esta define implícitamente la lista sensible del proceso.

Existen 3 variantes que son:

WAIT ON	Que espera los cambios de las señales especificadas
WAIT FOR	Detiene la simulación durante el tiempo especificado
WAIT UNTIL	Espera a que se cumpla la condición especificado.

Descripción Estructural

En una descripción estructural se declaran los componentes que se utilizan, se crean las instancias de los mismos y después mediante los nombres de los modos, se realizan las conexiones entre las instancias de los componentes. Estas son útiles cuando se trata de diseños jerárquicos bien modularizados.

Un componente representa a una entidad declarada en un diseño o librería.

La instanciación del componente puede extenderse como una instrucción concurrente que especifica la interconexión de las señales del componente dentro del diseño en el que esta siendo utilizado. Existen 2 formas de hacer la instanciación de los componentes.

Por Asociación de Identificadores
En este tipo es necesario utilizar el operador de asociación "=>" para indicar como se conectan los puertos del componente con los puertos o señales de la arquitectura en la que esta siendo utilizado dicho componente.

Por Asociación de Posición.
Aquí no es necesario nombrar los puertos del componente. Solamente se colocan las señales, variables o expresiones en el lugar donde deseamos que sean conectadas. El orden con que se declararon los puertos del componente es el orden con que se deben utilizar en la instalación del componente.

Identificadores.
Los identificadores son formas a partir de caracteres alfabéticos y numéricos que siguen las reglas:

- El primer carácter debe ser una letra
- El último carácter no puede ser un guión bajo
- Dos guiones bajos seguidos no son permitidos.

Objetos de Datos.
Estos retienen valores de tipo especifico, estos se clasifican en cuatro clases y se deben de declarar antes de utilizarse. Estas clases son:

- Constantes.
- Señales.
- Variables.
- Archivos.

Las Constantes.
Una constante mantiene el valor asignado, y que no puede ser cambiado en la descripción del diseño, este valor de la constante generalmente es asignado durante la declaración.

Señales.
Las señales pueden representar alambres y por tanto pueden interconectar componentes , de hecho los puertos pueden ser declarados como señales.

Variables
Las variables son utilizadas en procesos y subprogramas y deben ser declaradas en zona declarativa . Las variables no representan señales ni elementos de memoria, son solamente utilizadas con propósitos computacionales.

Las asignaciones a las variables son inmediatas, y no tienen una forma de onda, es decir, mantienen un solo valor en el tiempo.

El uso mas general de las variables es para el almacenamiento temporal de datos. El alcance de la variable únicamente es en el proceso en el cual a sido declarada.

Archivos.
Se utilizan para leer datos que sirven de estimulo y escribir datos a la salida de los procesos de simulación

Alias
Es un sobrenombre con el que se conoce también a cualquier identificador de algún objeto existente.

Tipos de Datos.
Los objetos que podemos representar en VHDL deben pertenecer a un conjunto bien definido, el cual es conocido como tipo de datos y se encarga de agrupar objetos con características comunes.

VHDL es un lenguaje fuertemente tipificado, lo cual significa que objetos de diferentes tipos no pueden ser asignados entre sí sin la previa conversión del tipo.

A continuación daremos los diferentes tipos.

Tipo Escalar.
Estos tienen un orden, lo que permite que los operadores de relación sean utilizados. Existen cuatro categorías :

- Enumeración
- Enteros
- Flotantes
- Físicos.

Enumeración.
Es una lista de valores que un objeto de este tipo puede tener. Son muy útiles cuando se trabaja con máquinas de estado.

El orden con el que son listados en la declaración del tipo define su relación. El valor de mas a la izquierda es menor que los otros valores.

El estándar IEEE 1164 define un tipo adicional llamado std_ulogic de la siguiente forma:
Std_ulogic is ( ),

U NO INICIALIZADO
X DESCONOCIDO
0 CERO.
1 UNO
Z ALTA IMPEDANCIA
W DESCONOCIDO DÉBIL.
L CERO DÉBIL
H UNO DÉBIL
"-" NO IMPORTA

Tipo Entero.
Los enteros y sus operadores aritméticos y relacionales son predefinidos en VHDL. Un a señal o variable del tipo entero y que va a ser sintetizada debe tener asociado un intervalo.

Tipo Flotante.
El tipo flotante es para aproximar números reales. También como en los enteros se debe especificar un intervalo.

Tipo Físico.
Los valores del tipo físico son utilizados como unidades de medición, el único tipo predefinido es el de tipo time y su unidad primaria es el femtosegundo.

Tipos Compuestos.
Los objetos del tipo escalar pueden mantener solo un valor en un tiempo de simulación. Los objetos de datos del tipo compuesto pueden mantener múltiples valores en un tiempo dado. Estos consisten en el tipo arreglo y en el tipo registro.

Tipo Arreglo.
Este consiste en múltiples elementos del mismo tipo, los más comunes se enumeran a continuación.

Type bit_vector is array (natural range <>)of bit;
Type std-ulogic_vector is array (natural range <>)of std_ulogic,
Type std-logic_vector is array (natural range <>)of std_ulogic,

La cláusula <> significa que el número de bits no es especificado.

Tipo Registro.
Un elemento del tipo registro tiene varios elementos de diferentes tipos, los elementos individuales del registro pueden ser referenciados por el nombre del elemento.

LOGICA COMBINACIONAL

Utilizando Sentencias Concurrentes
Las sentencias concurrentes siempre se localizan fuera de cualquier proceso conceptualmente estas sentencias se ejecutan concurrentemente, por lo tanto el orden en que aparezcan las diferentes sentencias no es importante.

Ecuaciones Booleanas
Las ecuaciones booleanas pueden ser utilizadas en sentencias tanto concurrentes como secuenciales de asignación.
Los operadores lógicos son la clave fundamental para el uso de ecuaciones booleanas los operadores lógicos "and, or , nand, xor, xnor" se encuentran predefinidos para los tipos bit y boolean así como también para arreglos de una dimensión de esos mismos tipos cabe hacer notar que los dos operandos deben ser de la misma longitud en numero de bits. Los operadores lógicos no tienen un orden de precedencia por tanto es necesario el uso de paréntesis para dar precedencia a los operadores.

With-Select-When
Esta sentencia permite una asignación selectiva de señales, esto significa que un valor es asignado a una señal con base en el valor de otra señal de selección.

When-Else
Esta sentencia esta disponible para realizar asignación condicional de señales, lo cual significa que un valor es asignado a una señal con base en una condición.

Operadores Relacionales
Los operadores relacionales son utilizados para probar igualdad, desigualdad y ordenamiento. Los operadores de igualdad y desigualdad son "=" y "/=" los cuales son definidos para todos los tipos discutidos en este manual. Los operados de magnitud son "<, <=, >, >=" los cuales son definidos para tipos escalares o tipos arreglo con un rango discreto. Los resultados de los operadores relacionales es un valor de verdad "verdadero" o "falso".

Sobrecarga de Operadores
La sobrecarga de operadores permite utilizar los mismos operadores con múltiples tipos de datos, en especial para aquellos que no son predefinidos en el estándar IEEE 1076. Los operadores pueden ser sobrecargados utilizando funciones definidas por el usuario, pero muchos de los operadores sobrecargados son ya definidos en los estándares IEEE 1164 y 1076 3.

Instalación de Componentes
La instanciación de componentes son sentencias concurrentes que especifican la interconexión de señales en diseño.

Usando Sentencias Secuenciales.
Las sentencias secuenciales son todas aquellas contenidas en un proceso, una función o un procedimiento. Estos dos últimos no son tratados en este manual. La colección de sentencias que constituyen el proceso, a su vez constituye una sola sentencia concurrente entre sí misma. Sin embargo dentro del proceso, todas las sentencias son secuenciales y por tanto es importante el orden en que aparezcan. En esta sección vamos a describir como utilizar procesos y sentencias secuenciales para describir lógica combinacional.

If-Then-Else
Este constructor es utilizando para elegir un conjunto de sentencias a ser ejecutadas con base en la prueba de una condición que puede ser "verdadera" o "falsa".

Si la condición se evalúa a verdadero se ejecuta el primer bloque de sentencias "hacer algo", de lo contrario se evalúa el segundo bloque.

Cuando se utiliza este constructor para especificar lógica combinacional es muy importante especificar completamente la sentencia if then else o especificar el valor por default de la asignación esto para evitar generar un lazo de memoria no deseado.

Case-When
Esta sentencia es utilizada para especificar un conjunto de sentencias a ejecutarse con base en el valor de una señal de selección.

LOGICA SINCRONA.
En esta sección mostraremos como escribir código VHDL para modelar lógica sincronía, la cual esencialmente implica bloques de lógica combinacional conectada a elementos de memoria como flip-flops controlados por señales de reloj.

Sentencia Wait Until.
El comportamiento de un flip-flop o de un registro, también es posible descubrirlo utilizando la sentencia wait until en lugar de la sentencia if.

La interpretación es que el proceso esta suspendido hasta que la condición establecida por la sentencia wait until se hace "verdadera". Cuando esto ocurre entonces las sentencias que aparecen después del wait until son ejecutadas, al terminar las sentencias se vuelve a esperar hasta que ocurra otro flanco de subida en la señal clock. De esta manera existirá una sincronía.

Funciones Rising_Edge y Falling_Edge.
El paquete std_logic_1164 define dos funciones llamadas rising_edge y falling_edge que son útiles para detectar flancos de subida y de bajada, esta se pueden utilizar par sustituir las expresiones if (clk'event and clk = '1') y if (clk'event and clk = '1') respectivamente.

Empleo de Señales de Reset en Lógica Síncrona.
En el mundo del hardware no siempre los circuitos se inicializan en un valor determinado. Entonces si uno desea tener señales de reset o preset globales es necesario incluirlas explícitamente en el código para poder manipularlas.

Operadores Aritméticos
Los operadores aritméticos mas comúnmente utilizados para síntesis son la suma y la resta, los cuales son muy útiles para describir sumadores, restadores, incrementadores, decrementadores, contadores etc. Todos los operadores aritméticos son predefinidos para los tipos entero y flotante.

Buffers Tres Estados y Señales Bidireccionales
La mayoría de los dispositivos de lógica programable tienen salidas tres estados o bidireccionales. Los buffers de salida son puestos en alta impedancia cuando no manejan el bus en un tiempo determinado. Los puertos bidireccionales permite compartir un conjunto de puertos para funcionar tanto como entradas como salidas cabe hacer notar que no todos los dispositivos programables permiten la utilización de buses internos tres estados, lo cual significa que en ellos no pueden ser implementados como buses internos, son solo como buses externos. Los diseños aquí presentados que utilizan estas características están descritos utilizando el estilo de descripción por comportamiento.

Tres Estados
Los valores que una señal tres estados pueden tener son '0', '1' y 'z', todos ellos son soportados por el tipo std logic.

Bidireccional
El valor del contador es cargado con el valor actual en los pines asociados con las salidas del contador (funcionan como entradas), o estas salidas muestran el valor de la cuenta actual (funcionan como entradas), o estas salidas muestran el valor de la cuenta actual (funcionan como salidas) este funcionamiento bidireccional depende del estado del habilitador de salida "oe".

Iteraciones Condicionales
La sentencia next es utilizada para saltar alguna operación con base en condiciones especificas.

Salida Forzada de una Iteración
La sentencia exit es utilizada para salir de un ciclo, y puede ser utilizada para verificar alguna condición ilegal.

Loops
Las sentencias de loops (ciclos) son utilizadas para implementar operaciones repetitivas y consisten de ciclos "for" y while". El ciclo for es ejecutado un numero especifico de iteraciones con base en un valor de control sin embargo el ciclo "while" se ejecuta continuamente hasta que una condición lógica lo termina al hacerse verdadera. El ciclo "for" es sintetizable por casi todas las herramientas que el ciclo "while" no lo es.

Maquina de Estados Utilizando dos Procesos
En este método uno de los procesos indica que la siguiente asignación de estado se hará con base en el presente estado y en las entradas presentes, pero no indica cuando el siguiente estado llega a ser el presente estado. Esto sucede en forma síncrona con la señal de reloj de la forma en que se describe en un segundo proceso debido a estos dos procesos es que recibe el nombre de maquina de estados utilizando dos procesos. El código que describe esta maquina de estados es el siguiente:

```
library ieee,
use ieee std_logic_1164 all;

entity fsm2p is port(
read_write, ready, clk in bit;
oe, we
end fsm2p,
architecture state_machine of fsm2p is
type statetype is(idle, decision, read, write),
signal present_state, next_state statetype,
begin
process (present_state, read_write, ready)begin
case present_state is
when idle=>
oe<='0',we <='0',
if ready ='1' then
next_state <=decision,
```

```
else
next_state<=idle,
end if;
when decision =>
oe<='0',we <= '0',
if read_write='1' then
next_state<=read,
else
next_state<=write;
end if,
when read=>
oe<='1',we<='0',
if ready='1'then
next_state<=idle,
else
next_state<=read;
end if;
when write=>
oe<='0'we<='1',
if ready = '1'then
next_state<=write,
end if,
end case;
end process,
process (clk) begin
if (clk'event and clk='1') then
present_state<=next_state,
end if,
end process,
end architecture state_ machine,
```

Al inicio de la arquitectura definimos un tipo llamado "statetype" que es un enumerado de cuatro valores "idle, decisión, read, write" los cuales son nombres que nos representan cada uno de los cuatro estados disponibles en la maquina de estado del 41. Después definimos dos señales de uso local a la arquitectura llamadas "present_state y next_state" que son del tipo que acabamos de crear y que nos van a servir para representar el estado actual y el estado siguiente.

El primer proceso es sensible al estado presente (present_state) así como a las señales de lectura escritura (read_write) y listo (ready). Este proceso lo que hace es probar, utilizando la sentencia case, el valor del "present_state" si el número de casos corresponde con el número de estados.

En cada caso contemplado por el case dos son las acciones a realizar. La primera consiste en establecer el valor de las salidas "oe" y "we" para ese estado, debido a que se trata de una maquina de tipo moore, estos valores son directamente tomados del diagrama de estados. La segunda acción para cada caso consiste en definir las transiciones del estado actual hacia el estado siguiente, en otras palabras calculamos cual es el siguiente estado en función de las entradas que sean importantes en el estado actual (ready o read_ write), esta prueba se hace utilizando el constructor if-else y anidándolos en caso de tener que probar varias entradas de manera que el resultado de esta prueba es asignar a "next_state" el estado siguiente que puede ser aluno de los cuatro estados posibles definidos por "statetype" según corresponda.

El segundo proceso es sensible a la señal de reloj "clk" y sincronizado con respecto al flanco de subida de la misma señal de reloj. El objetivo de este estado es actualizar el valor de la señal de estado presente "present_state" con el valor de la señal del estado siguiente. "next_state", que fue calculado en el primer proceso.

Esto significa que la transición del estado presente al estado siguiente esta sincronizada con respecto a la señal de reloj del sistema y aunque se calcula en el proceso uno, no se asigna sino sincronizadamente con la señal de reloj en el proceso una en función del estado presente solamente debido a que se trata de una maquina Moore.

El funcionamiento de la máquina de estados puede verificarse rápidamente utilizando cualquier simulador que acepte entrada VHDL El código para esta máquina fue simulado utilizando el simulador de la herramienta MAX+PLUS II de Altera. La transición al estado siguiente es función del estado actual así como de las entradas. El funcionamiento esta sincronizado con respeto al flanco de subida de la señal de reloj.

II.7 CONVERTIDORES A/D y D/A

En el mundo que nos rodea, podemos apreciar que la gran mayoría de las actividades llevadas a cabo por los humanos, está presente un sistema informático, un CPU, o para ser más precisos, un sistema electrónico controlado por microprocesador. Estos dispositivos realizan todas sus tareas y funciones apoyándose en la lógica digital ya que el tratamiento de información mediante el sistema binario ofrece múltiples ventajas y posibilidades casi ilimitadas. Sin embargo, las señales procedentes del exterior, que precisan ser analizadas por nosotros, son generalmente analógicas (temperaturas, presiones, humos, etc.) y en raras ocasiones disponemos directamente de datos en formato digital.

Del mismo modo cuando deseamos actuar sobre algún elemento físico sobre el que deseamos ejercer una influencia para alterar su estado funcional, necesitaremos una señal analógica en forma de tensión, frecuencia, etc. Para llevar a cabo esta unión entre los bloques analógico y digital, de forma que podamos intercambiar informaciones en ambos sentidos, se diseñaron los sistemas convertidores analógicos/digitales (ADC) y los digitales/analógicos (DAC).

El proceso completo de intercambio de información puede representarse según el diagrama de bloques de la figura 2.31, en el cual puede apreciarse cómo mediante un sensor se toma una señal muestra de la magnitud física que deseamos medir o analizar; Esta señal generalmente eléctrica es amplificada entre los márgenes oportunos y posteriormente filtrada para ser inyectada en un convertidor A/D que después de un proceso de conversión proporcionará a su salida una información en forma de palabra digital (n bits) que puede ser analizada y convenientemente tratada por la CPU de control o microprocesador de control.

Efectuado este análisis por el microprocesador que gobierna el sistema procederá a proporcionar una serie de datos en formato digital para actuar sobre las partes oportunas del elemento a controlar (motores, etc.). Esta información es transformada por un convertidor D/A en una señal eléctrica analógica que después será amplificada e inyectada en un circuito de potencia capaz de manejar los accionamientos exteriores necesarios.

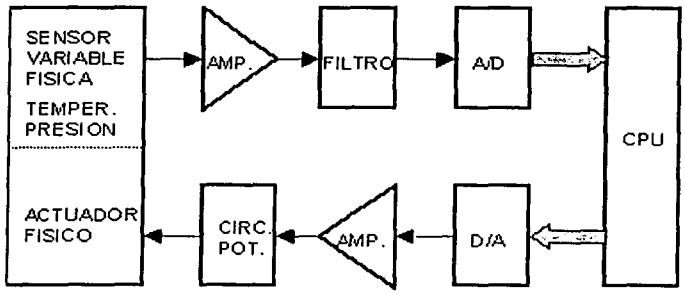


Fig. 2.31 Proceso de intercambio de información

II.7.1 Convertidores analógicos-digitales.

Los convertidores A/D son dispositivos electrónicos que establecen una relación biunívoca entre el valor de la señal en su entrada y la palabra digital obtenida en su salida. La relación se establece en la mayoría de los casos, con la ayuda de una tensión de referencia.

La conversión analógica a digital tiene su fundamento teórico en el teorema de muestreo y en los conceptos de cuantificación y codificación.

Teorema de muestreo

Si una señal continúa, ***S(t)***, tiene una banda de frecuencia tal que ***fm*** sea la mayor frecuencia comprendida dentro de dicha banda, dicha señal podrá reconstruirse sin distorsión a partir de muestras de la señal tomadas a una frecuencia ***fs*** siendo ***fs > 2 fm.***

En la figura 2.32, se muestra un esquema simplificado del proceso de muestreo.

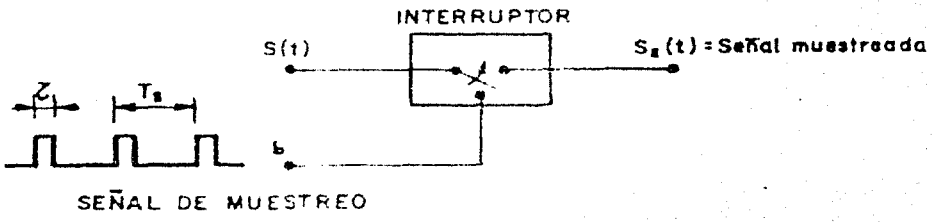


Fig. 2.32 Proceso de muestreo

El interruptor no es del tipo mecánico, puesto que por lo general ***fs*** es de bastante valor. Suelen emplearse transistores de efecto campo como interruptores, para cumplir los requerimientos que se le exigen entre los que se encuentran:

- Una elevada resistencia de aislamiento cuando los interruptores (transistores) están desconectados.
- Una baja resistencia si los interruptores están conectados o cerrados.
- Una elevada velocidad de conmutación entre los dos estados de los interruptores.

En la siguiente figura 2.33, se ofrece las formas de las tres señales principales:

S(t)	señal a muestrear
G	señal muestreadora
***Sa*(t)**	señal muestreada

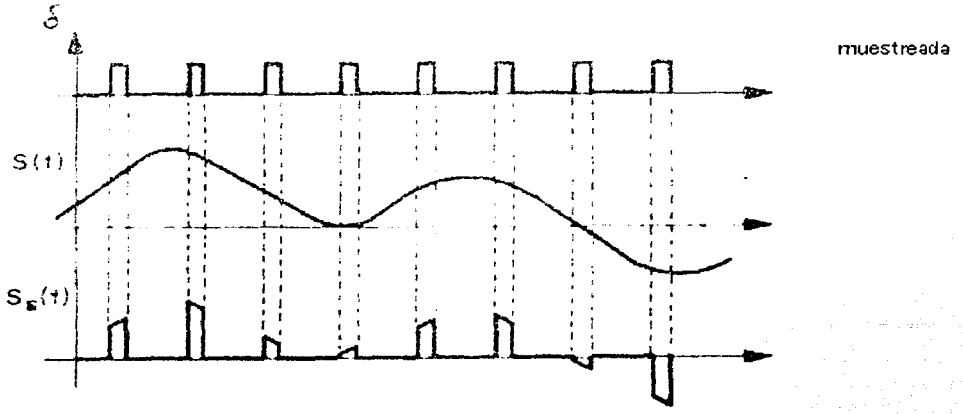


Fig. 2.33 Señales Muestrear, Muestreadora y Muestreada

Desde el punto de vista de la cuantificación de la señal muestreada, lo ideal sería que el tiempo en que el interruptor está cerrado, fuese prácticamente cero, ya que de otro modo, la señal muestreada puede variar en dicho tiempo y hacer imprecisa su cuantificación.

Debe tenerse en cuenta que para la reconstrucción de la señal original, a partir de la muestreada, se emplea un filtro de paso bajo, el cual deberá tener una función de transferencia como se indica en la figura 2.34:

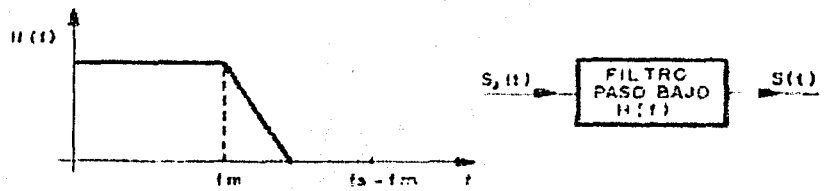


Fig.2.34 Función de transferencia

Obsérvese que la respuesta del filtro, debe ser plana hasta una frecuencia, como mínimo, igual a ***fm***, para caer posteriormente de forma brusca a cero, antes de que la frecuencia alcance el valor de ***fs-fm***.

> Mediante la aplicación del Teorema del Muestreo, se pueden transmitir varias señales, por un mismo canal de comunicación. Para ello se muestrea sucesivamente varias señales S_1, S_2, S_3, ... y las señales muestreadas se mandan por el canal de comunicación. A este sistema se le denomina *"multiplexado en el tiempo"*

Al otro extremo del canal habrá que separar las distintas señales muestreadas para hacerlas pasar después por el filtro paso bajo que las reconstruya.

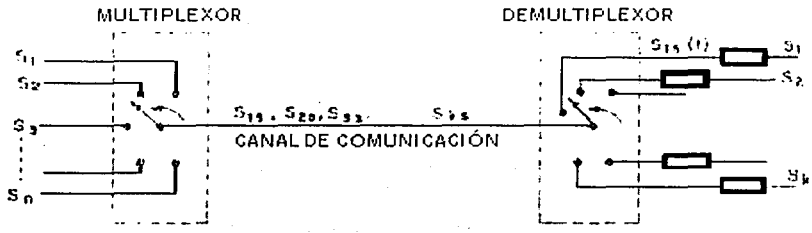
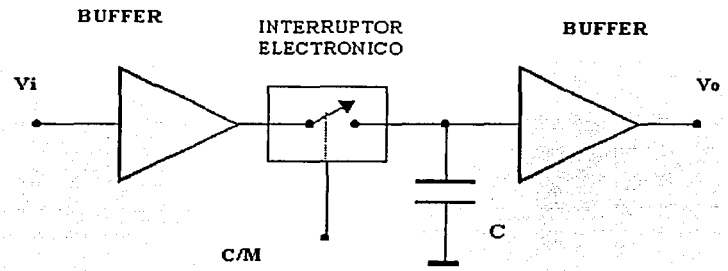


Fig. 2.35 Representación de multiplexor y demultiplexor

En la figura 2.35, el multiplexor y el demultiplexor se han representado mediante conmutadores rotativos sincronizados, los cuales, evidentemente no son adecuados, dada la gran frecuencia de giro ***fs,*** necesaria en este sistema. Para ello se emplean multiplexores y demultiplexores ***electrónicos***. En este sistema de transmisión de señales es imprescindible, el perfecto sincronismo entre los dos extremos del canal.

Circuitos de captura y mantenimiento (S/H: Sample and Hold)

Los circuitos de captura y mantenimiento se emplean para el muestreo de la señal analógica (durante un intervalo de tiempo) y el posterior mantenimiento de dicho valor, generalmente en un condensador, durante el tiempo que dura la transformación A/D, propiamente dicha. El esquema básico de un circuito de captura y mantenimiento, así como su representación simplificada, se ofrece en la figura 2.36.



(a)

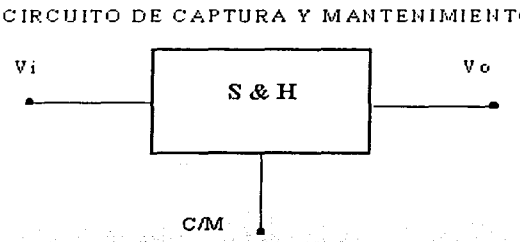


Figura 2.36 (a) Circuito de captura; (b) Circuito de mantenimiento.

El funcionamiento del circuito de la figura 2.36, es el siguiente: El convertidor A/D manda un impulso de anchura tw por la línea C/M, que activa el interruptor electrónico, cargándose el condensador C, durante el tiempo tw. En el caso ideal, la tensión en el condensador sigue la tensión de entrada. Posteriormente el condensador mantiene la tensión adquirida cuando se abre el interruptor.

En la siguiente figura 2.37 se muestran las formas de las señales de entrada, salida y gobierno del interruptor.

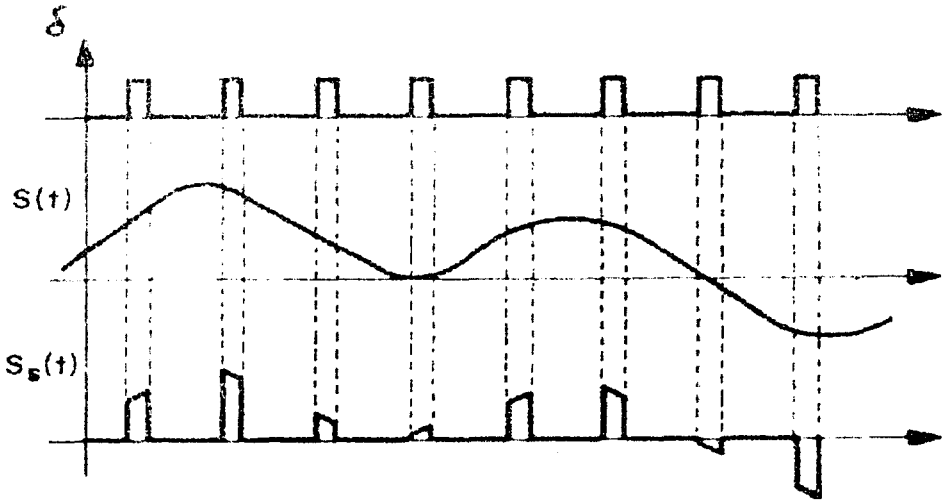


Figura 2.37 Señales de entrada y salida.

El gráfico tiene un carácter ideal, puesto que tanto la carga como la descarga del condensador están relacionadas estrechamente con su valor y con el de las resistencias y capacidades parásitas asociadas al circuito. Se recalca el hecho de que el control de la señal C/M procede del convertidor A/D, que es el único que conoce el momento en que finaliza la conversión de la señal.

Este valor una vez amplificado está listo para ser introducido en la etapa comparadora, la cual es la encargada de verificar su igualdad con la señal analógica a ser convertida. Los grandes fabricantes de semiconductores ofrecen gran variedad de convertidores con muy diversos niveles de precisión empleando distintas tecnologías de fabricación y apoyándose en diferentes métodos o técnicas de conversión, entre las que podemos destacar:

Método Secuencial: Este método es el más sencillo pero lento a la vez. Consiste en ir incrementando el valor digital (a la entrada del convertidor D/A) comenzando de 0 y terminando una vez que el comparador delata que la salida del amplificador es igual a la entrada analógica. Este valor digital será el resultado de la conversión.
El problema es que la velocidad de conversión se alarga a medida que la entrada analógica es mas elevada. Es decir, para convertir una señal equivalente a 10 (0A hexadecimal) se necesitarán 10 pasos de testeos, en cambio con una señal equivalente a 127 (7F hexadecimal) se necesitarán 127 comparaciones antes de detectar el valor.

Método de Aproximaciones Sucesivas: Este método es el más usado y veloz, aunque requiere una programación algo más compleja. Consiste en ir poniendo a "1" cada bit comenzando por el más significativo. Por lo tanto, considerando que trabajamos con 8 bits de resolución, el primer paso es colocar a 1 el bit 7 (sería 1000000=128 en decimal) y realizar la comparación.
Si por ejemplo, el comparador indicase que la entrada analógica es menor que la salida del amplificador, significará que el valor es inferior a 128 con lo cual ya sabemos que estamos en la mitad inferior del byte (0 a 127) y por lo tanto el bit 7 debe ser apagado (poner a 0), esto ya evita tener que comprar la mitad superior (128 a 255). El paso siguiente será poner a 1 el bit 6 y de esta forma

revisaremos si el valor está entre 0 y 63 o entre 64 y 127. Dependiendo de la salida del comparador se dejará en 1 o en 0 el bit 6. Y así sucesivamente se irán verificando los restantes bits (0 a 5).
El resultado será el valor digital correspondiente a la conversión.
Este método solo requiere de 8 comparaciones (trabajando con resoluciones de 8 bits) para detectar cualquier valor de entrada a diferencia del método secuencial que requería de 256 comparaciones en el caso extremo.

Convertidor A/D con comparadores.

Es el único caso en que los procesos de cuantificación y codificación están claramente separados. El primer paso se lleva a cabo mediante comparadores que discriminan entre un número finito de niveles de tensión. Estos comparadores reciben en sus entradas la señal analógica de entrada junto con una tensión de referencia, distinta para cada uno de ellos. Al estar las tensiones de referencia escalonadas, es posible conocer si la señal de entrada está por encima o por debajo de cada una de ellas, lo cual permitirá conocer el estado que le corresponde como resultado de la cuantificación. A continuación será necesario un codificador que nos entregue la salida digital como lo muestra la figura 2.38.

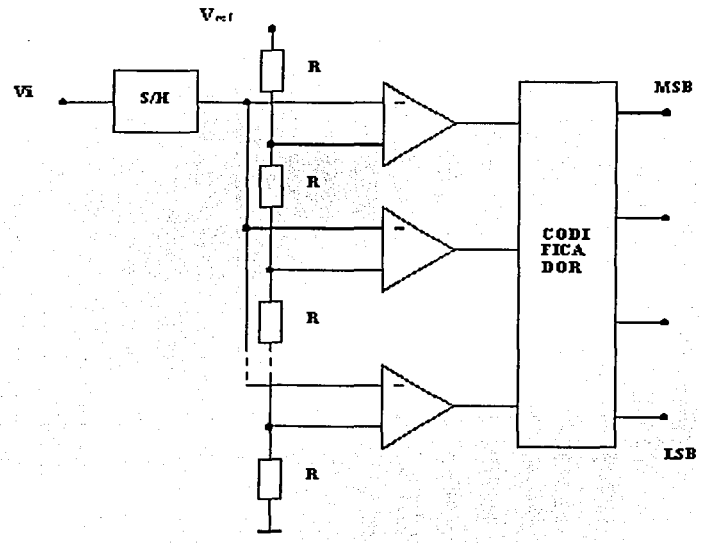


Fig.2.38 Comparador con codificador

Este convertidor es de alta velocidad, ya que el proceso de conversión es directo en lugar de secuencial, reduciéndose el tiempo de conversión necesario a la suma de los de propagación en el comparador y el codificador. Sin embargo, su utilidad queda reducida a los casos de baja resolución, dado que para obtener una salida de N bits son necesarios 2N-1 comparadores, lo que lleva a una complejidad y encarecimiento excesivos en cuanto se desee obtener una resolución alta.

Convertidor A/D con contadores.

Llamado también con rampa en escala. Usa el circuito más sencillo de los convertidores A/D y consta básicamente de los elementos reflejados en la figura 2.39.

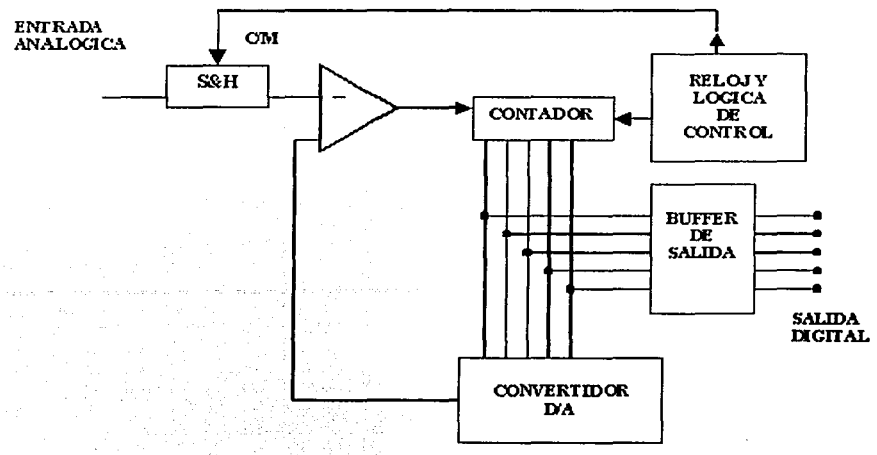


Fig.2.39 Un comparador, reloj, circuito de captura y mantenimiento (S&H), contador, convertidor D/A y buffers de salida.

Una vez que el circuito de captura y mantenimiento (S/H), ha muestreado la señal analógica, el contador comienza a funcionar contando los impulsos procedentes del reloj. El resultado de este valor se transforma en una señal analógica mediante un convertidor D/A, proporcional al número de impulsos de reloj recibidos hasta ese instante. La señal analógica obtenida se introduce al comparador en el que se efectúa una comparación entre la señal de entrada y la señal digital convertida en analógica. En el momento en que esta última alcanza el mismo valor (en realidad algo mayor) que la señal de entrada, el comparador bascula su salida y se produce el paro del contador. El valor del contador pasa a los buffers y se convierte en la salida digital correspondiente a la señal de entrada. Este convertidor tiene dos inconvenientes: escasa velocidad y tiempo de conversión variable. El segundo inconveniente puede comprenderse fácilmente con la ayuda de la figura 2.40, en la que se aprecia que el número de impulsos de reloj (tiempo), precisos para alcanzar el valor V_{i2} en el convertidor D/A depende del valor de V_{i1}.

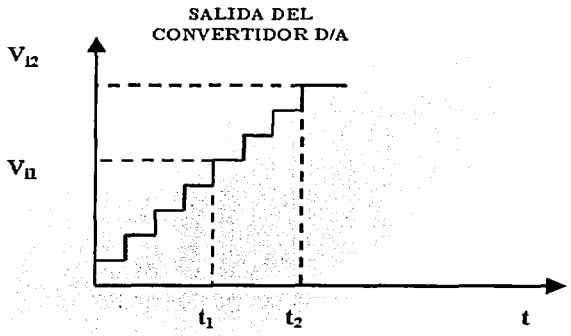


Fig.2.40 Salida del convertidor

Dicho tiempo de conversión viene dado por la expresión:

$$t = \frac{Vi \times 2^n}{f \times V_{fondo\ escala}}$$

Convertidor A/D con integrador.

Este tipo de convertidores son más sencillos que los anteriores ya que no utilizan convertidores D/A. Se emplean en aquellos casos en los que no se requiere una gran velocidad, pero en los que es importante conseguir una buena linealidad. Son muy usados en los voltímetros, osciloscopios digitales y equipo de medición .

Utilización Práctica del Convertidor A/D:

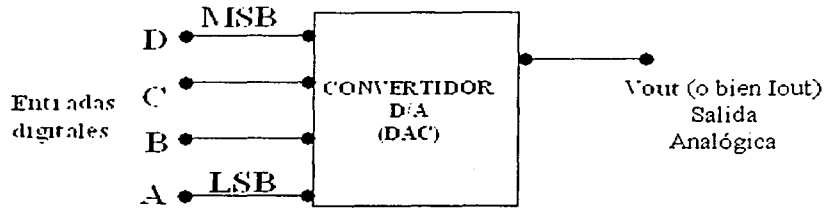
Los convertidores A/D son muy útiles en robótica para la interpretación de todo tipo de sensores del tipo analógicos. Un caso práctico de uso puede ser para la construcción de un brazo mecánico en el cual se desee utilizar motores DC en lugar de los motores Paso a Paso. En nuestro caso será para nuestro osciloscopio.

II.7.2 Convertidores Digitales - Analógicos

Por lo dicho en tema anterior, observamos que los ADC y los DAC funcionan como interfase entre un sistema o dispositivo completamente digital (como una computadora) y el mundo analógico externo. Esta función se vuelve crecientemente más importante a medida que las microcomputadoras poco costosas entran en áreas de control de procesos donde el control de la computadora no era viable con anterioridad.

Las dos operaciones E/S relativas al proceso de mayor importancia son la conversión de digital a analógico (D/A) y la conversión de analógico a digital (A/D). Ya que muchos métodos de conversión A/D utilizan el proceso de conversión D/A, examinaremos primero la conversión D/A.

Básicamente, la conversión D/A es el proceso de tomar un valor representado en código digital (como binario directo o BCD) y convertirlo en un voltaje o corriente que sea proporcional al valor digital. Este voltaje o corriente es una cantidad analógica, ya que puede tomar diferentes valores de cierto intervalo. La figura 2.41(a), muestra el diagrama a bloques de un convertidor D/A común de 4 bits. No nos interesan los circuitos internos hasta más adelante. Por ahora, examinaremos las diversas relaciones de entrada y salida. Las entradas digitales D, C, B y A se derivan generalmente del registro de salida de un sistema digital. Los $2^4 = 16$ diferentes números binarios representados por estos 4 bits se enlistan en la figura 2.41(b). Por cada número de entrada, el voltaje de salida del convertidor D/A es un valor distinto. De hecho, el voltaje de salida analógico Vout es igual en volts al número binario. También podría tener dos veces el número binario o algún otro factor de proporcionalidad. La misma idea sería aplicable si la salida de D/A fuese la corriente Iout.



(a)

D	C	B	A	Vout
0	0	0	0	0 Volts
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15 Volts

(b)

Fig. 2.41 Convertidor D/A de 4 bits con salida de voltaje.

Valores de entrada Para el DAC de la figura 2.41, debe observarse que cada entrada digital contribuye con una cantidad diferente de salida analógica. Esto se puede apreciar fácilmente si se examinan los casos donde sólo una entrada es ALTA:

D C B A	V_{OUT} (V)
0 0 0 1 ⟶	1
0 0 1 0 ⟶	2
0 1 0 0 ⟶	4
1 0 0 0 ⟶	8

Fig. 2.42.

A las contribuciones de cada entrada digital se les asignan valores según su posición en el número binario. Por lo tanto, A, que es el LSB, tiene un valor de 1 V, B tiene un valor de 2 V, C de 4 V y D, el MSB, tiene el mayor, 8 V. Los valores se duplican sucesivamente por cada bit, comenzando con el LSB. Por consiguiente, podemos considerar a Vout como la suma con valor de las entradas digitales.

Por ejemplo, para hallar Vout para la entrada digital 0111 podemos señalar dos valores a los bits C, B y A a fin de obtener 4 V + 2 V + 1 V = 7 V.

Resolución (tamaño de etapa) La resolución de un convertidor D/A se define como la menor variación que puede ocurrir en la salida analógica como resultado un cambio en la entrada digital. Haciendo referencia de la tabla de la figura 2.42, podemos apreciar que la resolución es 1V, puesto que Vout puede variar en no menos que 1 V cuando cambie el código de entrada. La resolución siempre es igual al valor del binario LSB y también se conoce como tamaño de etapa, ya que es la cantidad Vout que variará cuando el código de entrada pase de una etapa a la siguiente. Esto se ilustra de manera más gráfica en la figura 2.43, donde las entradas digitales están derivando de las salidas de un contador binario de 4 bits. El contador es llevado continuamente a través de 16 estados por la entrada del cronómetro. La forma de onda en la salida D/A es una escalinata repetitiva que llega hasta 1 V por etapa cuando el contador avanza de 0000 a 1111. Cuando el contador regresa a 0000, la salida D/A retorna a 0 V. La resolución o tamaño de etapa es la dimensión de los saltos en la forma de onda de escalinata. En este ejemplo cada etapa es 1 V.
Aunque la resolución puede expresarse como la cantidad de voltaje o corriente por etapa, resulta más útil expresarla como un porcentaje de la salida de escala completa. Para ilustrar lo antes dicho, el convertidor D/A de la figura 2.43 tiene una salida de escala completa máxima de 15 V (cuando la entrada digital es 1111). El tamaño de etapa es 1 V, lo cual da una resolución porcentual de

$$\text{Resolución porcentual} = \frac{\text{tamaño de etapa}}{\text{escala total (F.S.)}} \times 100\%$$

$$= \frac{1\ \text{V}}{15\text{V}} \times 100\% = 6.67\%$$

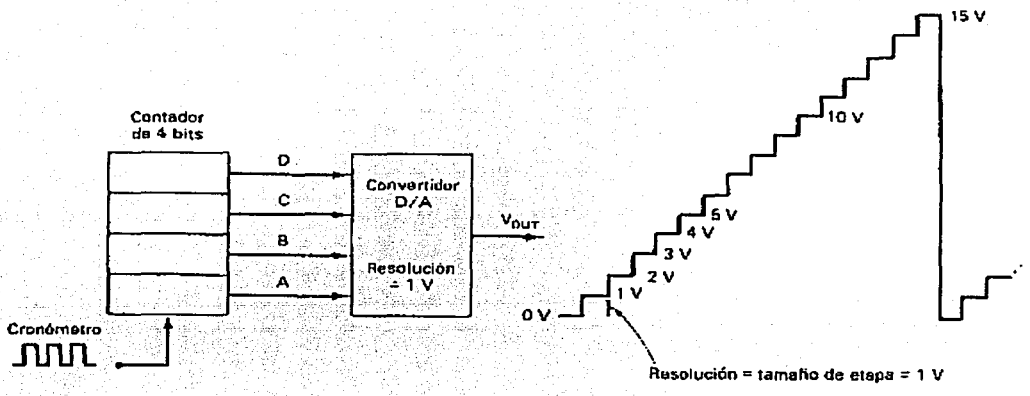


Fig.2.43 Formas de onda de salida del convertidor D/A cuando las entradas son obtenidas de un contador binario.

Esto significa que es sólo el número de bits el que determina la resolución porcentual. Si se incrementa el número de bits, aumenta el número de etapas hasta llegar a la escala completa, de manera que cada etapa sea una parte menor del voltaje de escala completa. Muchos fabricantes de DAC especifican la resolución como el número de bits.

Código de entrada BCD Los convertidores D/A que hemos considerado hasta ahora han hecho uso de un código de entrada binario. Muchos convertidores D/A utilizan un código de entrada BCD donde se emplean grupos de código de 4 bits por cada dígito decimal. La figura 2.44, muestra el diagrama de un convertidor de 8 bits (dos dígitos) de este tipo. Cada grupo de código de 4 bits puede variar de 0000 a 1001, de manera que las entradas BCD representan cualquier número decimal de 00 a 99. Dentro de cada grupo de código los valores de los diferentes bits se proporcionan igual que el código binario, pero los valores del grupo son diferentes por un factor de 10. Por ejemplo, A0, el LSB del dígito menos significativo (LSD), podría tener un valor de 0.1 V. Por lo tanto, B0, C0 y D0 serían 0.2, 0.4 y 0.8 V, respectivamente. El valor de A 1,el LSB del MSD, seria 1 V (10 veces A0). Análogamente, B1, C1 y D1 serían 2, 4 y 8 V, respectivamente.

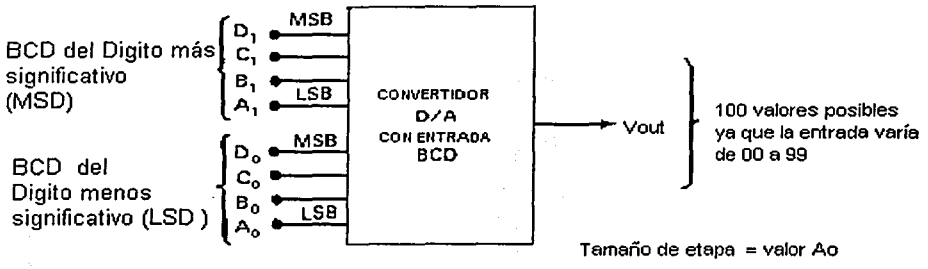


Fig. 2.44 Convertidor D/A que utiliza código de entrada BCD.

CIRCUITOS CONVERTIDORES D/A

Existen varios métodos y circuitos para producir la operación D/A que se ha descrito. Examinaremos sólo uno de los esquemas básicos, para dar al lector una introducción a la teoría que se utiliza. No es importante conocer los diversos esquemas de circuitos ya que los convertidores D/A se tienen a disposición como circuitos impresos o bien como paquetes encapsulados que no requieren ningún conocimiento de circuitos. En su lugar, es importante conocer las características significativas de realización de los convertidores D/A, en términos generales, de manera que se puedan utilizar en forma inteligente.

La figura 2.45, (a) muestra el circuito básico de un tipo de convertidor D/A de 4 bits. Las entradas A, B, C y D son entradas binarias que se supone tienen valores de 0V o bien 5V. El amplificador operacional sirve como amplificador sumatorio, el cual produce la suma con valor asignado de estos voltajes de entrada. Puede rememorarse que el amplificador sumatorio multiplica cada voltaje de entrada por la proporción de la resistencia de retroalimentación Rf a la resistencia de entrada correspondiente Rin y las resistencias de entrada varían de 1 a 8 k ohms. La entrada D tiene Rin = 1 k ohms, de manera que el amplificador de la sumatoria pase el voltaje en D sin atenuación. La entrada C tiene Rin 2 k ohms, de manera que será atenuada en 0.5. En forma análoga, la entrada B será atenuada en 0.25 y la entrada A, en 0.125. Por consiguiente, la salida del amplificador se puede expresar como

$$V_{out} = -(V_D + 0.5V_C + 0.25V_B + 0.125V_A) \qquad (1)$$

El signo negativo está presente debido a que el amplificador sumatorio es un amplificador de inversión de polaridad, pero no lo estudiaremos aquí.

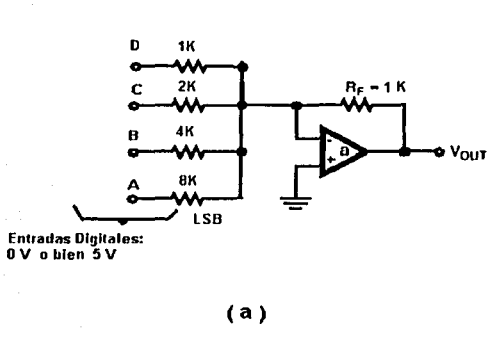
La salida del amplificador sumatorio evidentemente es un voltaje analógico que representa una suma con valor asignado de las entradas digitales, como lo muestra la tabla de la figura 2.45(b). Esta tabla

enlista todas las posibles condiciones de entrada y el voltaje de salida del amplificador resultante. La salida es evaluada con cualquier condición de entrada poniendo las entradas indicadas en 0 V o bien en 5 V. Por ejemplo, si la entrada digital es 1010, entonces $V_D = V_B = 5$ V y $V_C = V_A = 0$ V. En consecuencia, al utilizar (1) se tiene

$$Vout = -(5\ V + 0\ V + 0.25 \times 5\ V + 0\ V)$$

$$= -6.25\ V$$

La resolución de este convertidor D/A es igual a la asignación de un valor del LSB, que es 0.125 X 5 V = 0.625 V. Como se muestra en la tabla, la salida analógica aumenta en 0.625 V cuando el número de entrada binario avanza una etapa.



D	C	B	A	Vout	
0	0	0	0	0 Volts	
0	0	0	1	-0.625	LSB
0	0	1	0	-1.25	
0	0	1	1	-1.875	
0	1	0	0	-2.5	
0	1	0	1	-3.125	
0	1	1	0	-3.75	
0	1	1	1	-4.375	
1	0	0	0	-5	
1	0	0	1	-5.625	
1	0	1	0	-6.25	
1	0	1	1	-6.875	
1	1	0	0	-7.5	
1	1	0	1	-8.125	
1	1	1	0	-8.75	
1	1	1	1	-9.375 Volts	Escala Completa

(b)

Fig.2.45 Convertidor D/A simple que utiliza amplificador sumatorio amp-ab.

Exactitud de la conversión La tabla de la figura 2.45(b) da los valores ideales de V0~ en los diversos casos de entrada. La aproximación que logre este circuito al producir estos valores depende principalmente de dos factores: (1) la precisión de las resistencias de entrada y retroalimentación y (2) la precisión de los niveles de voltaje d entrada. Las resistencias pueden hacerse muy exactas (dentro de 0.01% de los valores deseados) por compensación, pero los niveles de voltaje de entrada deben manejarse en forma distinta. Las entradas digitales no pueden tomarse directamente de las salidas de los FF o compuertas lógicas, puesto que los niveles lógicos de salida de estos dispositivos no son valores exactos como 0 V y 5 V sino que varían en un intervalo determinado. Por esta razón, se necesita insertar un amplificador del nivel de precisión entre cada entrada lógica y su resistencia de entrada al amplificador de la sumatoria. Esto se muestra en la figura 2.46.

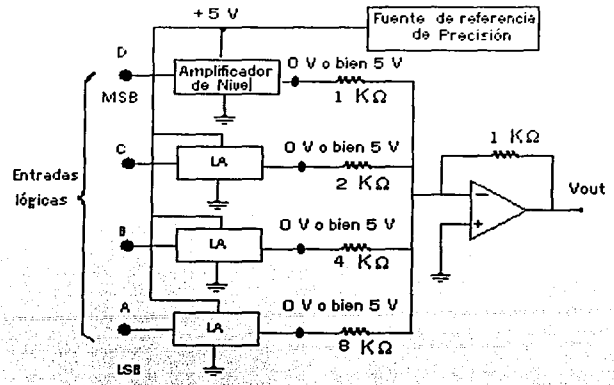


Fig.2.46 Convertidor D/A completo de 4 bits incluyendo amplificadores con nivel de precisión.

Los amplificadores de nivel producen niveles de salida exactos de 5 V y 0 V, según si las entradas digitales son ALTAS o bien BAJAS. Se requiere una fuente de referencia precisa y muy estable de 5 V para producir el nivel exacto de 5 V.

DAC con salida de corriente Muchos otros circuitos se pueden utilizar para producir la conversión D/A y no es nuestro objetivo presentarlos todos aquí. Sin embargo, analizaremos otro método DAC que hace uso de una salida de corriente analógica en vez de salida de voltaje. El esquema básico se puede apreciar en la figura 2.47 (a) en relación con un DAC de 4 bits.

El circuito utiliza cuatro trayectorias de corriente en paralelo, cada una controlada por un interruptor semiconductor. El estado de cada interruptor es controlado por los niveles de voltaje que se aplican a las entradas binarias. La corriente que fluye por cada trayectoria paralela es determinada por Vref y la resistencia en la trayectoria.

A las resistencias se asignan valores binarios y la corriente total, Iout dependerá de que interruptores se abran o se cierren, de acuerdo con las entradas binarias. Esta corriente de salida puede forzarse a fluir a través de una carga, R_L, pero R_L debe hacerse r menor que R de manera que no afecte el valor de la corriente. En teoría, R_L debe corto a tierra.

A fin de que Iout sea exacta, R_L debe ser un corto a tierra. Una manera con lograr esto consiste en hacer uso de un amp-ab (amplificador abierto) como convertidor de corriente en voltaje, como se muestra en la figura 2.47 (b). Aquí, Iout del DAC se conecta a la entrada "—" de los amp-ab, que está virtualmente a tierra. La retroalimentación negativa del amp-ab forza a una corriente igual a Iout a fluir a través de Rf para producir Vout = - Iout X Rf. Por lo tanto, Vout será un voltaje analógico que es proporcional a la entrada binaria en el DAC. Esta salida analógica puede impulsar una amplia gama de cargas sin ser afectada.

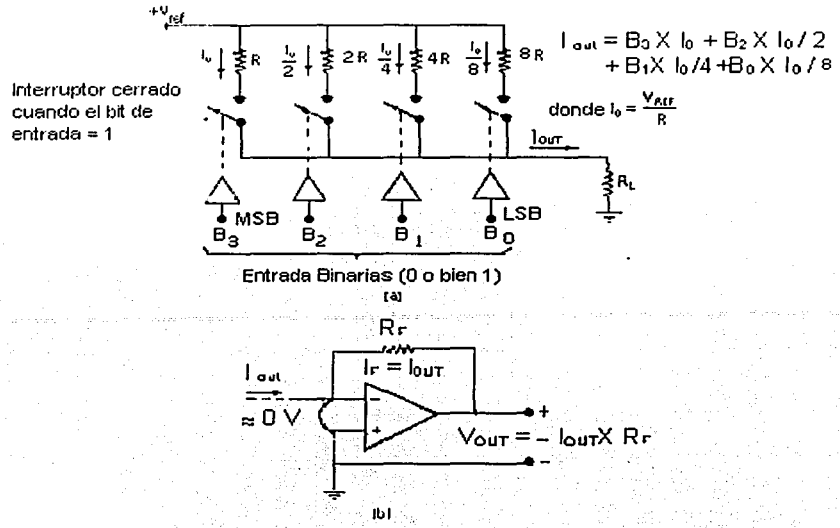


Fig.2.47 (a) DAC básico con salida de corriente; (b) conectado a un convertidor de corriente en voltaje con amp-ab.

DAC multiplicativos Muchos DAC requieren una fuente de voltaje de referencia que se usa internamente para ayudar en la generación de la salida analógica. En algunos DAC este voltaje de referencia tiene que ser un valor fijo y exacto a fin de producir una salida exacta. En otros tipos el voltaje de referencia puede en realidad ser variable y aún cambiar polaridades. Este último tipo de DAC se llama DA C multiplicativo ya que la salida analógica es el producto de la entrada binaria y el voltaje de referencia.

Por ejemplo, se observó que la corriente de salida del DAC de la figura 2.47 (a) es proporcional a Vref. Es decir,

$$Iout = k(Vref \times B)$$

Donde B representa la entrada binaria y k es la constante de proporción Podemos determinar el valor de k a partir de los resultados del ejemplo 10.6, donde se halló que Iout.= 1.875 mA para Vref = 10V y $B=1111_2 = 15_{10}$ Por lo tanto, se tiene

$$1.875\ mA = k(10\ V \times 15)$$

o bien

$$k = 1.875\ mA/150\ V = 0.0125\ mA / V$$

Una vez que se conoce k, Iout puede obtenerse para cualquier valor de Vref y entrada binaria.

Un DAC multiplicativo se puede usar para multiplicar un voltaje analógico (Vref) por un valor binario para producir una salida analógica. Algunos DAC multiplicativos permiten que Vref sea de cualquier polaridad de manera que el producto final pueda tener cualquier polaridad. A éstos se los llama DAC multiplicativos de dos cuadrantes. Algunos DAC permiten asimismo que la entrada binaria sea de cualquier polaridad utilizando el MSB como un bit del signo. A éstos se los llama DAC multiplicativos de cuatro cuadrantes.

Especificaciones DAC.

Se dispone de una amplia variedad de DAC como circuitos integrados o bien como paquetes encapsulados autocontenidos. Uno debe estar familiarizado con las especificaciones más importantes de los fabricantes a fin de evaluar un DAC en una determinada aplicación.

Resolución Como se mencionó antes, la resolución porcentual de un DAC depende únicamente del número de bits. Por esta razón, los fabricantes por lo general especifican una resolución de DAC como el número de bits. Un DAC de 10 bits tiene una resolución más sensible (mayor exactitud) que uno de 8 bits.

Precisión Los fabricantes de DAC tienen varias maneras de especificar la precisión o exactitud. Las dos más comunes se las llama error de escala completa y error de linealidad, que normalmente se expresan como un porcentaje de la salida de escala completa del convertidor (%F.S.) El error de escala completa es la máxima desviación de la salida del DAC de su valor estimado (teórico). Por ejemplo, supóngase que al DAC de la figura 2.48 tiene una exactitud de ±0.01%. Como este convertidor tiene una salida de escala completa de 9.375 V, este porcentaje se convierte en

$$\pm 0.01\% \times 9.375\ V = \pm 0\ 9375\ mV$$

Esto significa que la salida de este DAC puede, en cualquier instante, variar 0.9375 mV de su valor esperado.

El error de linealidad es la desviación máxima en el tamaño de etapa del teórico. Por ejemplo el DAC de la figura 2.47, tiene un tamaño de etapa estimado de 0.625V. Si este convertidor tiene un error de linealidad de ±0.01% F.S., esto significaría que el tamaño de etapa real podría apartarse del estimado hasta 0.9375 mV. Algunos de los DAC más costosos tienen errores de escala completa y de linealidad en el intervalo de 0.01-0.1%.

Es importante entender que la exactitud o precisión y resolución de un convertidor D/A deben ser compatibles. Es lógico tener una resolución de, por ejemplo, 1 por ciento y una salida a F.S. de 10 V, y poder producir un voltaje analógico de salida dentro de 0.1 V de cualquier valor deseado, suponiendo que haya exactitud perfecta.

No tiene sentido tener una precisión costosa de 0.01% de F.S. (o bien, 1 mV) si la resolución ya limita la proximidad al valor deseado a 0.1 V. Esto mismo puede decirse al tener una resolución muy pequeña (de muchos bits) en tanto que la exactitud es insuficiente; se trata de un desperdicio de bits de entrada.

Tiempo de respuesta La velocidad de operación de un DAC se especifica como tiempo de respuesta, que es el tiempo que se requiere para que la salida pase de cero a escala completa cuando la entrada binaria cambia de todos los ceros a todos los unos. Los valores comunes del tiempo de respuesta variarán de 50 ns a l0us. En general, los DAC con salida de corriente tendrán tiempos de respuesta más breves que aquellos con una salida de voltaje. Por ejemplo, el DAC 1280 puede operar como salida de corriente o bien de voltaje. Su tiempo de respuesta a su salida es 300 ns cuando se utiliza salida de corriente y 2.5us cuando se emplea salida de voltaje. La razón principal de esta diferencia es el tiempo de respuesta del amp-ab que se utiliza como convertidor de corriente en voltaje.

Voltaje de balance En teoría, la salida de un DAC será cero Vout cuando la entrada binaria es todos los ceros. En la práctica, habrá un voltaje de salida pequeño producido por el error de desbalance del amp-ab. Este desplazamiento es comúnmente 0.05% F.S Casi todos los DAC con voltaje tendrán una capacidad de ajuste de balance externo que nos permitirá eliminar el error de desbalance.

APLICACIONES DAC

Los DAC se utilizan siempre que la salida de un circuito digital tiene que ofrecer un voltaje o corriente analógico para impulsar o activar un dispositivo analógico. Algunas de las aplicaciones más comunes se describen a continuación.

Control La salida digital de una computadora puede convertirse en una señal de control analógica para ajustar la velocidad de un motor, la temperatura de un horno o bien para controlar casi cualquier variable física.

Análisis automático Las computadoras pueden ser programadas para generar las señales analógicas (a través de un DAC) que se necesitan para analizar circuitos analógicos. La respuesta de salida analógica del circuito de prueba normalmente se convertirá en un valor digital por un ADC y se alimentará a la computadora para ser almacenada, exhibida y algunas veces analizada.

Control de amplitud digital Un DAC multiplicativo se puede utilizar para ajustar digitalmente la amplitud de una señal analógica. Recordemos que un DAC multiplicativo produce una salida que es el producto de un voltaje de referencia y la entrada binaria. Si el voltaje de referencia es una señal que varia con el tiempo, la salida del DAC seguirá esta señal, pero con una amplitud determinada por el código de entrada binario. Una aplicación normal de esto es el "control de volumen" digital, donde la salida de un circuito o computadora digital puede ajustar la amplitud de una señal de audio. .

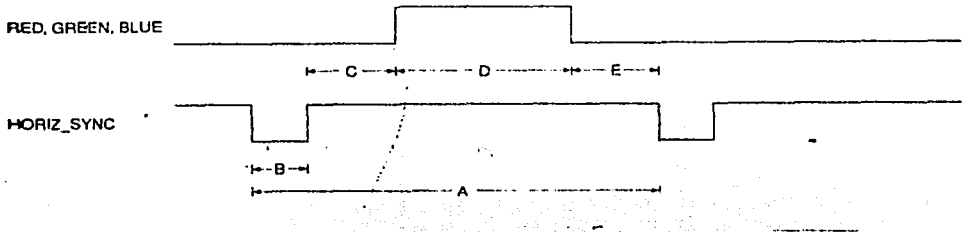
Convertidores A/D Varios tipos de convertidores A/D utilizan DAC que son parte de sus circuitos. Observaremos esto en la siguiente sección.

II.8 CONTROLADOR VGA.

Este controlador consiste en una película de pixeles que se puede dividir en renglones y columnas. Un monitor típico contiene 480 renglones por 640 columnas. Cada pixel puede desplegar varios colores dependiendo de la mezcla de las señales roja, verde y azul.
Este controlador cuenta con un reloj interno que determina cuándo se prendera cada píxel. Este reloj opera a una frecuencia especifica para VGA de 25.175 MHz. El controlador refresca la pantalla en forma parcial utilizando los pulsos de sincronía horizontal y vertical.

El controlador inicia cada ciclo de refresco prendiendo un pixel en la esquina superior izquierda de la pantalla, la cual puede ser considerada como el origen del plano X – Y. Una vez refrescado el primer píxel el controlador continúa con los restantes del renglón, y al finalizar el último pixel de este renglón enviará un pulso de sincronía horizontal al monitor con lo cual se empezara a refrescar el siguiente renglón, este proceso se repite hasta que se barre toda la pantalla, en este momento un pulso de sincronía vertical en el monitor ocasionara el inicio del barrido en el origen del plano X –Y, es decir la parte superior izquierda de la pantalla.

Para que el monitor trabaje apropiadamente, debe recibir datos en tiempos específicos con pulsos específicos. Los pulsos de sincronía horizontal y vertical deben ocurrir en tiempos determinados para sincronizar al monitor mientras este recibe la información de color. Las figuras 2.48 (a) y 2.48 (b) muestran el diagrama de tiempos para la información de color con respecto a los pulsos de sincronía horizontal y vertical.



Parameters	A	B	C	D	E
Time	31.77 µs	3.77 µs	1.89 µs	25.17 µs	0.94 µs

Figura 2.48 (a) Ciclo de Refresco Horizontal.

La frecuencia de operación y el número de pixeles que el controlador debe refrescar en el monitor determinan el tiempo requerido para refrescar cada píxel y la totalidad de la pantalla. Las siguientes ecuaciones permiten calcular los tiempos requeridos para que el controlador desempeñe correctamente sus funciones.

Tpixel = I / fCLK = 40 ns

TROW = A = B + C + D + E = (Tpixel X 640 pixels) + row + guard bands = 31.77 J.1s

Tscreen = O = P + Q + R + S = (T ROW X 480 rgws) + guardbands = 16.6 ms

donde:

Tpixel = Tiempo requerido para actualizar un pixel

fCLK = 25.175 MHz

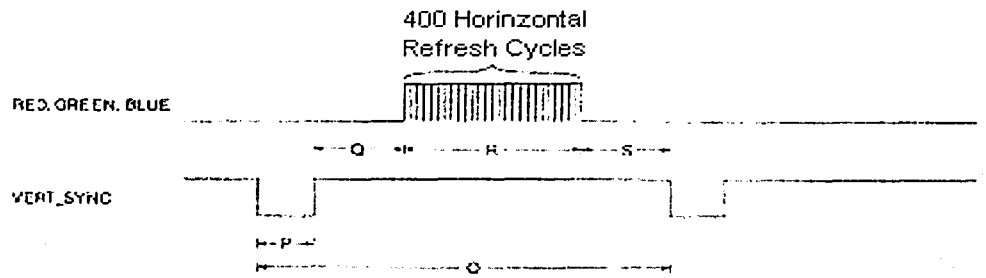
T ROW = Tiempo requerido para actualizar un renglón

T screen = Tiempo requerido para actualizar la pantalla

B, C, E = Guard bands

P, Q, S = Guard bands

El controlador escribe en la pantalla enviando las señales de sincronía horizontal y vertical, así como las señales de color rojo, verde y azul cuando este se encuentra en estado de espera. Una vez que el tiempo de los pulsos de sincronía horizontal y vertical es exacto el controlador solo necesita mantener la localización de los pixeles y mandar la información de color correcta para cada píxel.



Parameters	O	P	Q	R	S
Time	16.6 ms	64 µs	1.02 ms	15.25 ms	0.35 ms

Figura 2.48 (b) Ciclo de refresco vertical .

II.9 ACONDICIONAMIENTO DE LA SEÑAL

Introducción

El acondicionador de señal se encarga de preparar las señales para su correcta codificación una vez que ésta ha sido tomada. Para poder lograr esto, es necesario utilizar dos etapas las que comprenden el atenuador y los amplificadores operacionales.

II.9.1 Atenuadores

Circuitos básicos de entrada

Frecuentemente la salida de un transductor eléctrico es de tal naturaleza que debe modificarse para producir una señal más utilizable. Por ahora, deseamos discutir los circuitos que se pueden utilizar como modificadores de entrada. El transductor puede ser de tipo activo o pasivo; un elemento activo proporciona una energía a la entrada del circuito, mientras que un elemento pasivo solamente modifica las propiedades eléctricas del circuito por medio de un cambio en alguna característica, tal como resistencia, capacitancia o inductancia. Un transductor de presión cuya resistencia eléctrica cambie con la presión es un elemento pasivo, mientras que los cristales piezoeléctricos o las celdas fotoeléctricas son ejemplos de elementos activos. El transductor particular debe conectarse al circuito apropiado, el cual modificará la señal para que, finalmente, pueda obtenerse en un indicador a un registrador. Obviamente, se pueden usar muchos tipos de circuitos, pero solamente podremos mencionar algunos casos representativos.
Un tipo sencillo de circuito de entrada podría usar el flujo de corriente a través de un transductor pasivo de resistencia para dar una indicación del valor de la resistencia.

Ley de voltaje de Kirchhoff y circuitos serie

La ley de voltaje de Kirchhoff (LVK), tiene tres expresiones equivalentes: en cualquier instante alrededor de un lazo, en direcciones del movimiento de las manecillas de reloj o en dirección opuesta del movimiento de las manecillas del reloj.

1. La suma algebraica de las caídas de voltaje es igual a cero.
2. La suma algebraica de la elevación de voltaje es cero.
3. La suma algebraica de las caídas de voltaje es igual a la suma algebraica de la elevación de voltajes.

En las anteriores expresiones, la palabra "algebraica " significa que el signo de la caída y elevación de voltaje está incluida en todas las sumas. Recuérdese que una elevación de voltaje es una caída de voltaje negativa, y que una caída de voltaje es una elevación de voltaje negativa. Para lazos que no contengan fuente de corriente, la expresión más conveniente de la LVK es la tercera, con la restricción de que las caídas de voltaje se producen sólo a través de resistencia y que las elevaciones de voltaje se producen sólo a través de fuentes de voltajes.

En la aplicación de la LVK, la dirección del movimiento de las manecillas del reloj frecuentemente se toma como referencia de un lazo de corriente, como se ilustra en el circuito serie de la figura 2.49, en donde la LVK es aplicada en la dirección de la corriente. (Este es un circuito serie porque la misma corriente I fluye a través de todos los componentes). La suma de las caídas de voltaje a través de los resistores, $V_1 + V_2 + ,, V_n$, es igual a la elevación de voltaje V_S a través de la fuente de voltaje : $V_1 + V_2 + ,, V_n = V_S$, Por lo tanto en la relación IR de la ley de Ohm se sustituye n para determinar los voltajes de los resistores:

$$V_S = V_1 + V_2 + ,,V_n = IR_1 + IR_2 + ,, IR_n = I (R_1 + R_2 + ,, R_n) = IR_T$$

De donde $I = Vs / R_T$ y $R_T = R_1 + R_2 + ,, R_n$. R_T es la resistencia total de los resistores conectados en serie.
También se emplea el término de resistencia equivalente, cuyo símbolo es Req.
De lo anterior se evidencia que, en general, la resistencia total es el resistor conectado en serie (resistores en serie), es igual a la suma de las resistencias individuales:

$$R_T = R_1 + R_2 + ,,, R_n$$

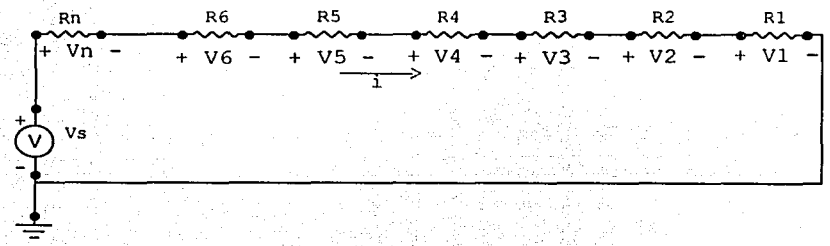


Fig. 2.49 Circuito serie .

Además, si las resistencias son del mismo valor (R), y si existen N de ellas, entonces $R_T = NR$. Resulta más sencillo calcular la corriente de un circuito serie utilizando el concepto de resistencia total que la aplicación directa de la LVK.

Si un circuito serie tiene más de una fuente de voltaje, se plantea de la misma manera

$$I (R_1 + R_2 + R_3 + ,,,) = V_{S1} + V_{S2} + V_{S3} + ,,,$$

En donde cada término V_S se considera positivo para una elevación de voltaje y negativo para cada caída de voltaje en la dirección de I.

La LVK se aplica con poca frecuencia a lazos que contienen fuentes de corriente, debido a que el voltaje a través de la fuente de corriente es desconocido y no existe una fórmula para calcularlo.

División de Voltaje

La división de voltaje o regla del divisor de voltaje se aplica a configuraciones de resistores en serie. Esta regla proporciona el voltaje a través de cualquier resistor en términos de resistencias y de voltaje a través de la combinación en serie . La fórmula de la división de voltaje para el circuito mostrado en la figura 2.49 es fácil de determinar. Supóngase que se trata de calcular el voltaje V_2. Por medio de la ley de Ohm, $V_2 = IR_2$, Asimismo, $I = V_S / (R_1 + R_2 +,,R_n)$ Al eliminarse I resulta que :

$$V_2 = R_2 / (R_1 + R_2 + R_n)V_S$$

En general, para cualquier número de resistor en serie con una resistencia total R_T y con un voltaje V_S a través dela combinación en serie, el voltaje V_X a través d un resistor R_X es:

$$V_X = R_X V_S/R_T$$

Esta es la fórmula para la división de voltaje o regla de división de voltaje. En esta fórmula, V_S y V_X deben tener polaridades opuestas: es decir, alrededor de una trayectoria cerrada un voltaje debe ser una caída de voltaje y el otro voltaje una elevación de voltaje. Si ambos voltajes son elevaciones o caídas. La fórmula necesita la introducción de un signo negativo. El voltaje V_S no necesita ser el de la fuente. Éste es precisamente el voltaje total a través de los resistores en serie.

Medición de Resistencia como transductor

En la figura 2.50 se presenta un diagrama esquemático para este tipo de medición. En este caso, un cambio en la resistencia R representa un cambio en la variable física medida. Esto se indica por el contacto móvil en la figura. De esta manera, la corriente está dada por

$$i = \frac{Vs}{R + Ri}$$

Supongamos que la resistencia máxima del transductor es Rm. Entonces, se puede escribir la corriente en forma adimensional como

$$\frac{i}{Vs/R1} = \frac{1}{(R/Rm)(Rm/Ri) + 1}$$

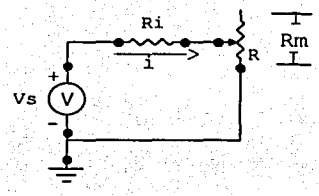


Fig. 2.50 Circuito de entrada sensible a la corriente. El circuito detecta el cambio en la resistencia R del transductor.

Sería deseable que la corriente de salida variara de una manera lineal con la resistencia del transductor.

El circuito discutido se puede modificar ligeramente usando un voltímetro, tal como lo muestra la figura 2.51. Supongamos que la impedancia interna del voltímetro es muy grande comparada con la resistencia en el circuito; por lo cual podemos despreciar la corriente que pasa a través del medidor. La corriente sigue estando dada por

$$i = \frac{Vs}{R + Ri}$$

Supóngase que V sea el voltaje en el transductor, como está indicado en la figura 2.50 Entonces

$$\frac{V}{Vs} = \frac{iR}{i(R + Ri)} = \frac{(R/Rm)(Rm/Ri)}{1+(R/Rm)(Rm/Ri)}$$

Ahora hemos obtenido una indicación de voltaje como una medición de la resistencia R, pero se sigue obteniendo una salida no lineal. La ventaja que presenta el circuito de la figura 2.51 sobre el de la figura 2.50 es que frecuentemente es más fácil realizar la medición de voltaje que de corriente. A este circuito se le llama circuito sensible al voltaje.

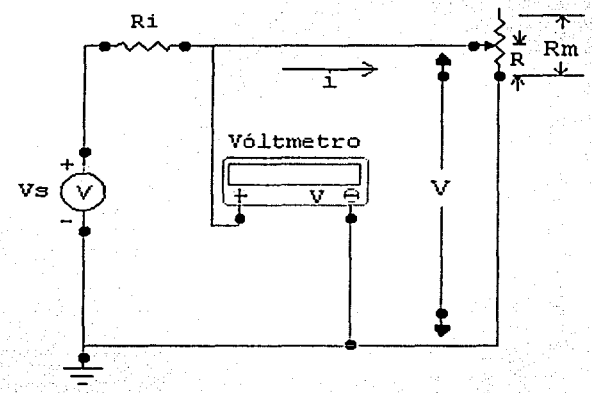


Fig. 2.51 Circuito de entrada sensible al voltaje. El cambio en la resistencia R se indica a través del cambio en la lectura del voltaje.

En los dos circuitos analizados se ha utilizado una medición de la corriente como indicación del valor de la resistencia variable del transductor. En algunos casos es más conveniente utilizar un circuito divisor de voltaje, como el que se ilustra en la figura 2.52. En este circuito, el voltaje fijo Vo se conecta a la resistencia total del transductor Rm, mientras que el cursor se conecta al voltímetro cuya resistencia interna es Ri. Si la impedancia del medidor es suficientemente grande, el voltaje indicado V será directamente proporcional a la resistencia variable R; esto es

$$\frac{V}{Vo} = \frac{R}{Rm} \quad \text{para} \quad Ri >> R$$

Si tenemos un medidor de resistencia finita, circulará a través de él una corriente que afecta la medición del voltaje. Si consideramos la resistencia interna del medidor, la corriente que proporciona la fuente de voltaje es

$$i = \frac{Vo}{Rm - R + Ri\,R/(R+Ri)}$$

El voltaje indicado es entonces

$$V = Vo - i(Rm - R)$$

o sea

$$\frac{V}{Vo} = \frac{R/Rm}{(R/Rm)(1-R/Rm)(Rm/Ri)+1}$$

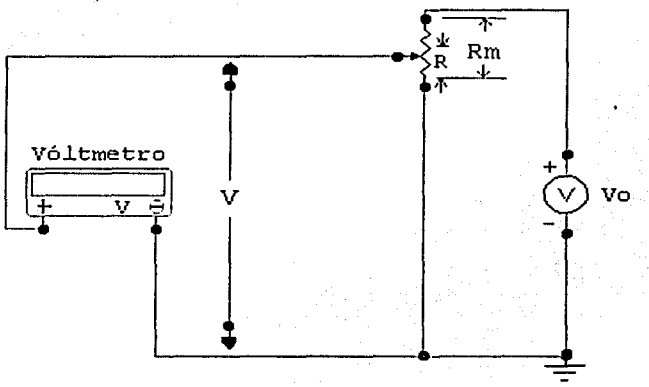


Fig. 2.52 Circuito divisor de voltaje.

El circuito divisor de voltaje que se ilustra en la figura 2.52 tiene la desventaja de que el efecto de carga del medidor afecta al voltaje leído. Esta dificultad se puede salvar utilizando un circuito potenciómetro para equilibrar voltajes, como el indicado en la figura 2.53. En este circuito un voltaje conocido Vo se aplica a la resistencia Rm, mientras que el voltaje desconocido se aplica al cursor de la misma resistencia Rm a través de un galvanómetro con resistencia interna Ri. En alguna posición del cursor el galvanómetro indicará cero corriente, y el voltaje desconocido se puede calcular a partir de

$$\frac{V}{Vo} = \frac{R}{Rm}$$

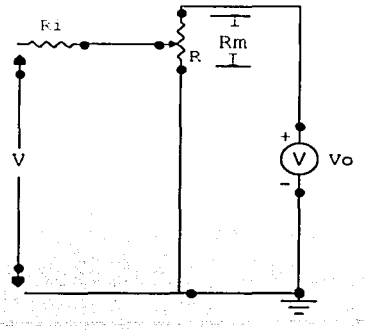


Fig. 2.53 Circuito potenciómetro para equilibrar voltajes.

Note que la resistencia interna del galvanómetro no afecta la lectura; sin embargo, afecta la sensibilidad del circuito. El circuito potenciómetro se utiliza ampliamente para efectuar mediciones precisas de voltajes pequeños, particularmente aquellos 'generadores por termopares. Para determinar exactamente el valor del voltaje desconocido V, el valor del voltaje de la fuente Vo debe conocerse exactamente. Generalmente se utiliza una fuente de voltaje, aunque ésta representa una fuente no muy digna de confianza debido a que sus características varían con el tiempo. Este circuito se esquematiza en la figura 2.54. Cuando el interruptor S_2 está en la posición A y el interruptor S_1 está cerrado, el circuito es el mismo que el de la figura 2.53. De esta manera, cuando se abre el interruptor S_1, y el interruptor S_2 se coloca en la posición B, queda conectada la celda Es al circuito. La resistencia variable de compensación Rc se ajusta hasta que el galvanómetro indica las condiciones de equilibrio. La resistencia de protección R_1 se puede puentear cerrando el interruptor S_1, posteriormente se efectúa un ajuste fino en la resistencia de compensación. La resistencia de protección R_1 se coloca en el circuito para evitar que circule excesiva corriente en la malla del circuito resistor y también para proteger el circuito analógico que se empleara más adelante. Observe que el voltaje de la celda estándar se conecta a la porción fija de la resistencia R. Una vez que Vo se ha estandarizado, el interruptor S_1 se puede colocar en la posición A y, consecuentemente, se puede medir el voltaje desconocido Vo.

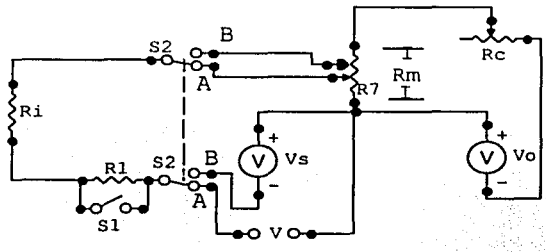


Fig. 2.54 Circuito potenciómetro que incorpora características para la estandarización del voltaje de la batería.

2.9.2 Amplificadores Operacionales.

Introducción.

El concepto original del AO (amplificador operacional) procede del campo de los computadores analógicos, en los que comenzaron a usarse técnicas operacionales en una época tan temprana como en los años 40. El nombre de amplificador operacional deriva del concepto de un amplificador dc

(amplificador acoplado en continua) con una entrada diferencial y ganancia extremadamente alta, cuyas características de operación estaban determinadas por los elementos de realimentación utilizados. Los primeros amplificadores operacionales usaban el componente básico de su tiempo: la válvula de vacio. El uso generalizado de los AOs no comenzó realmente hasta los años 60, cuando empezaron a aplicarse las técnicas de estado sólido al diseño de circuitos amplificadores operacionales, fabricándose módulos que realizaban la circuitería interna del amplificador operacional mediante diseño discreto de estado sólido. Entonces, a mediados de los 60, se introdujeron los primeros amplificadores operacionales de circuito integrado. En unos pocos años los amplificadores operacionales integrados se convirtieron en una herramienta estándar de diseño, abarcando aplicaciones mucho más allá del ámbito original de los computadores analógicos. Con la posibilidad de producción en masa que las técnicas de fabricación de circuitos integrados proporcionan, los amplificadores operacionales integrados estuvieron disponibles en grandes cantidades, lo que, a su vez contribuyó a rebajar su coste. Hoy en día el precio de un amplificador operacional integrado de propósito general, con una ganancia de 100 dB, una tensión offset de entrada de 1 mV, una corriente de entrada de 100 nA. Y un ancho de banda de 1 MHz es de unos cuantos centavos de dólar. El amplificador, que era un sistema formado antiguamente por muchos componentes discretos, ha evolucionado para convertirse en un componente discreto él mismo, una realidad que ha cambiado por completo el panorama del diseño de circuitos lineales.

Principios Básicos de los Amplificadores Operacionales

El amplificador operacional ideal.

Los fundamentos básicos del amplificador operacional ideal son relativamente fáciles. Quizás, lo mejor para entender el amplificador operacional ideal es olvidar todos los pensamientos convencionales sobre los componentes de los amplificadores, transistores, tubos u otros cualesquiera. En lugar de pensar en ellos, piensa en términos generales y considere el amplificador como una caja con sus terminales de entrada y salida. Trataremos, entonces, el amplificador en ese sentido ideal, e ignoraremos qué hay dentro de la caja.

$$Vo = a\ Vd$$

$$a = \text{infinito}$$

$$Ri = \text{infinito}$$

$$Ro = 0 \quad BW \text{ (ancho de banda)} = \text{infinito}$$

$$Vo = 0 \text{ si } Vd = 0$$

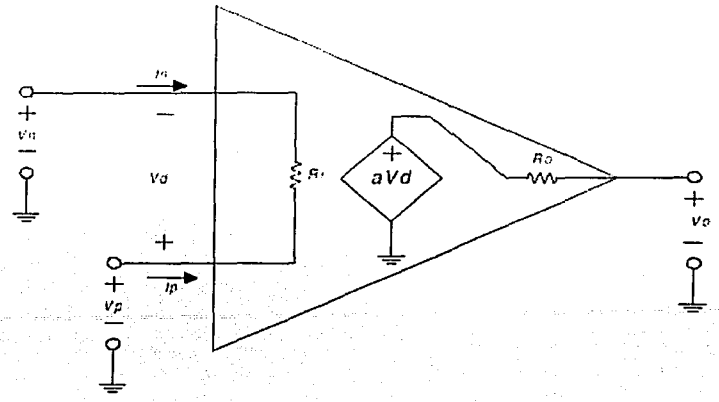


Fig. 2.55 Amplificador ideal.

En la figura 2.55 se muestra un amplificador idealizado. Es un dispositivo de acoplo directo con entrada diferencial, y un único terminal de salida. El amplificador sólo responde a la diferencia de tensión entre los dos terminales de entrada, no a su potencial común. Una señal positiva en la entrada inversora (–), produce una señal negativa a la salida, mientras que la misma señal en la entrada no inversora (+) produce una señal positiva en la salida. Con una tensión de entrada diferencial, Vd, la tensión de salida, Vo, será a Vd, donde a es la ganancia del amplificador. Ambos terminales de entrada del amplificador se utilizarán siempre independientemente de la aplicación. La señal de salida es de un sólo terminal y está referida a tierra, por consiguiente, se utilizan tensiones de alimentación bipolares (±)

Teniendo en mente estas funciones de la entrada y salida, podemos definir ahora las propiedades del amplificador ideal que son las siguientes:

- La ganancia de tensión es infinita
- La resistencia de entrada es infinita
- La resistencia de salida es cero: Ro = 0
- El ancho de banda es infinito
- La tensión offset de entrada es cero: $V_0 = 0$ si Vd = 0.

A partir de estas características del AO, podemos deducir otras dos importantes propiedades adicionales. Puesto que, la ganancia en tensión es infinita, cualquier señal de salida que se desarrolle será el resultado de una señal de entrada infinitesimalmente pequeña. La tensión de entrada diferencial es nula. También, si la resistencia de entrada es infinita. No existe flujo de corriente en ninguno de los terminales de entrada. Estas dos propiedades pueden considerarse como axiomas, y se emplearán repetidamente en el análisis y diseño del circuito del AO. Una vez entendidas estas propiedades, se pude, lógicamente, deducir el funcionamiento de casi todos los circuitos amplificadores operacionales.

Configuraciones básicas del amplificador operacional

Los amplificadores operacionales se pueden conectar según dos circuitos amplificadores básicos: las configuraciones (1) inversora y (2) no inversora. Casi todos los demás circuitos con amplificadores operacionales están basados, de alguna forma, en estas dos configuraciones básicas. Además, existen variaciones estrechamente relacionadas de estos dos circuitos, más otro circuito básico que es una combinación de los dos primeros: el amplificador diferencial.

El amplificador inversor

La figura 2.56 ilustra la primera configuración básica del AO. El amplificador inversor. En este circuito, la entrada (+) está a tierra, y la señal se aplica a la entrada (-) a través de R1, con realimentación desde la salida a través de R2.

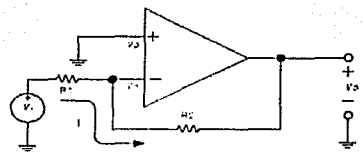


Fig. 2.56 Amplificador inversor.

Aplicando las propiedades anteriormente establecidas del AO ideal, las características distintivas de este circuito se pueden analizar como sigue. Puesto que el amplificador tiene ganancia infinita, desarrollará su tensión de salida, V_0, con tensión de entrada nula. Ya que, la entrada diferencial de A es:

$$Vd = Vp - Vn, \Longrightarrow Vd = 0.- \text{ Y si } Vd = 0,$$

entonces toda la tensión de entrada Vi, deberá aparecer en R_1, obteniendo una corriente en R_1

$$I = \frac{V_i}{R_1}$$

Vn está a un potencial cero, es un punto de tierra virtual Toda la corriente I que circula por R_1 pasará por R_2, puesto que no se derivará ninguna corriente hacia la entrada del operacional (Impedancia infinita), así pues el producto de I por R_2 será igual a - V_0

$$I = -\frac{V_0}{R_2}$$

$$\frac{V_i}{R_1} = -\frac{V_0}{R_2}$$

Por lo que:

$$V_0 = -\frac{R_2}{R_1} \cdot V_i$$

Luego la ganancia del amplificador inversor:

$$\frac{V_0}{V_i} = -\frac{R_2}{R_1}$$

Deben observarse otras propiedades adicionales del amplificador inversor ideal. La ganancia se puede variar ajustando bien R_1, o bien R_2. Si R_2 varía desde cero hasta infinito, la ganancia variará también desde cero hasta infinito, puesto que es directamente proporcional a R_2. La impedancia de entrada es igual a R_1, y Vi y R_1 únicamente determinan la corriente I, por lo que la corriente que circula por R_2 es siempre I, para cualquier valor de dicha R_2.

La entra del amplificador, o el punto de conexión de la entrada y las señales de realimentación, es un nudo de tensión nula, independientemente de la corriente I. Luego, esta conexión es un punto de tierra virtual, un punto en el que siempre habrá el mismo potencial que en la entrada (+). Por tanto, este punto en el que se suman las señales de salida y entrada, se conoce también como nudo suma. Esta última característica conduce al tercer axioma básico de los amplificadores operacionales, el cual se aplica a la operación en bucle cerrado: En bucle cerrado, la entrada (–) será regulada al potencial de entrada (+) o de referencia.

Esta propiedad puede aún ser o no ser obvia, a partir de la teoria de tensión de entrada de diferencial nula. Es, sin embargo, muy útil para entender el circuito del AO, ver la entrada (+) como un terminal de referencia, el cual controlará el nivel que ambas entradas asumen. Luego esta tensión puede ser tierra o cualquier potencial que se desee.

El amplificador no inversor

La segunda configuración básica del AO ideal es el amplificador no inversor, mostrado en la figura 2.57. Este circuito ilustra claramente la validez del axioma 3.

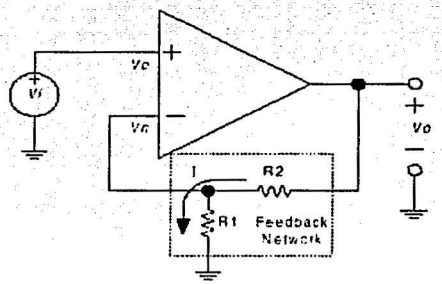


Fig. 2.57 Amplificador no inversor.

En este circuito, la tensión Vi se aplica a la entrada (+), y una fracción de la señal de salida, Vo, se aplica a la entrada (-) a través del divisor de tensión R_1 - R_2. Puesto que, no fluye corriente de entrada en ningún terminal de entrada, y ya que Vd = 0, la tensión en R_1 será igual a Vi.

Así pues

$$V_i = I \cdot R_1$$

y como

$$V_0 = I \cdot (R_1 + R_2)$$

tendremos pues que: si lo expresamos en términos de ganancia:

$$\frac{V_0}{V_i} = \frac{R_1 + R_2}{R_1}$$

que es la ecuación característica de ganancia para el amplificador no inversor ideal. También se pueden deducir propiedades adicionales para esta configuración. El límite inferior de ganancia se produce cuando R_2 = 0, lo que da lugar a una ganancia unidad. En el amplificador inversor, la corriente a través de R_1 siempre determina la corriente a través de R_2, independientemente del valor de R_2, esto también es cierto en el amplificador no inversor. Luego R_2 puede utilizarse como un control de ganancia lineal, capaz de incrementar la ganancia desde el mínimo unidad hasta un máximo de infinito. La impedancia de entrada es infinita, puesto que se trata de un amplificador ideal.

Configuraciones basadas en los circuitos inversor y no inversor

El amplificador diferencial

Una tercera configuración del AO conocida como el amplificador diferencial, es una combinación de las dos configuraciones anteriores. Aunque está basado en los otros dos circuitos, el amplificador diferencial tiene características únicas. Este circuito, mostrado en la figura 2.58, tiene aplicadas señales en ambos terminales de entrada, y utiliza la amplificación diferencial natural del amplificador operacional.

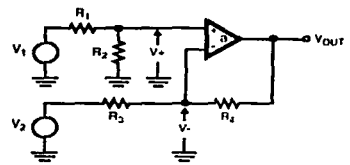


Fig. 2.58 Amplificador diferencial.

Para comprender el circuito, primero se estudiarán las dos señales de entrada por separado, y después combinadas. Como siempre Vd = 0 y la corriente de entrada en los terminales es cero. Recordemos que Vd = V(+) – V(–) ==> V(–) = V(+). La tensión a la salida debida a V_1 la llamaremos V_{01}

$$V(+) = \frac{V_1}{R_1 + R_2} \cdot R_2$$

y como V(–) = V(+)

La tensión de salida debida a V_1 (suponiendo $V_2 = 0$) valdrá:

$$V_{01} = \frac{V_1 \cdot R_2}{R_1 + R_2} \cdot \frac{R_3 + R_4}{R_3}$$

Y la salida debida a V_2 (suponiendo $V_1 = 0$) será, usando la ecuación de la ganancia para el circuito inversor, V_{02}

$$V_{02} = -V_2 \frac{R_4}{R_3}$$

Y dado que, aplicando el teorema de la superposición la tensión de salida $V_0 = V_{01} + V_{02}$ y haciendo que R_3 sea igual a R_1 y R_4 igual a R_2 tendremos que:

$$V_{01} = \frac{V_1 \cdot R_2}{R_1} \; V_{02} = -V_2 \frac{R_2}{R_1}$$

por lo que concluiremos

$$V_0 = (V_1 - V_2) \cdot \frac{R_2}{R_1}$$

que expresando en términos de ganancia:

$$\frac{V_0}{V_1 - V_2} = \frac{R_2}{R_1}$$

que es la ganancia de la etapa para señales en modo diferencial. Esta configuración es única porque puede rechazar una señal común a ambas entradas. Esto se debe a la propiedad de tensión de entrada diferencial nula, que se explica a continuación. En el caso de que las señales V_1 y V_2 sean idénticas, el análisis es sencillo. V_1 se dividirá entre R_1 y R_2, apareciendo una menor tensión V(+) en R_2. Debido a la ganancia infinita del amplificador, y a la tensión de entrada diferencial cero, una tensión igual V(–) debe aparecer en el nudo suma (–). Puesto que la red de resistencias R_3 y R_4 es igual a la red R_1 y R_2, y se aplica la misma tensión a ambos terminales de entrada, se concluye que Vo debe estar a potencial nulo para que V(–) se mantenga igual a V(+); Vo estará al mismo potencial que R_2, el cual, de hecho está a tierra. Esta muy útil propiedad del amplificador diferencial, puede utilizarse para discriminar componentes de ruido en modo común no deseables, mientras que se amplifican las señales que aparecen de forma diferencial. Si se cumple la relación

$$\frac{R_4}{R_3} = \frac{R_2}{R_1}$$

La ganancia para señales en modo común es cero, puesto que, por definición, el amplificador no tiene ganancia cuando se aplican señales iguales a ambas entradas. Las dos impedancias de entrada de la etapa son distintas. Para la entrada (+), la impedancia de entrada es $R_1 + R_2$. La impedancia para la entrada (–) es R_3. La impedancia de entrada diferencial (para una fuente flotante) es la impedancia entre las entradas, es decir, $R_1 + R_3$. El sumador inversor Utilizando la característica de tierra virtual en el nudo suma (–) del amplificador inversor, se obtiene una útil modificación, el sumador inversor, figura 2.59.

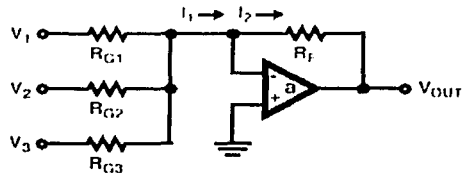


Fig. 2.59 Sumador Inversor.

En este circuito, como en el amplificador inversor, la tensión V(+) está conectada a tierra, por lo que la tensión V(–) estará a una tierra virtual, y como la impedancia de entrada es infinita toda la corriente I_1 circulará a través de R_F y la llamaremos I_2. Lo que ocurre en este caso es que la corriente I_1 es la suma algebraica de las corrientes proporcionadas por V_1, V_2 y V_3, es decir:

$$I_1 = \frac{V_1}{R_{g1}} + \frac{V_2}{R_{G2}} + \frac{V_3}{R_{g3}}$$

y también

$$I_2 = -\frac{V_{OUT}}{R_F}$$

Como $I_1 = I_2$ concluiremos que:

$$V_{OUT} = -\left(V_1 \cdot \frac{R_F}{R_{G1}} + V_2 \cdot \frac{R_F}{R_{G2}} + V_3 \cdot \frac{R_F}{R_{G3}}\right)$$

que establece que la tensión de salida es la suma algebraica invertida de las tensiones de entrada multiplicadas por un factor corrector, que el alumno puede observar que en el caso en que $R_F = R_{G1} = R_{G2} = R_{G3}$ ==> $V_{OUT} = -(V_1 + V_2 + V_3)$ La ganancia global del circuito la establece R_F, la cual, en este sentido, se comporta como en el amplificador inversor básico. A las ganancias de los canales individuales se les aplica independientemente los factores de escala R_{G1}, R_{G2}, R_{G3},... etc. Del mismo modo, R_{G1}, R_{G2} y R_{G3} son las impedancias de entrada de los respectivos canales. Otra característica interesante de esta configuración es el hecho de que la mezcla de señales lineales, en el nodo suma, no produce interacción entre las entradas, puesto que todas las fuentes de señal alimentan el punto de tierra virtual. El circuito puede acomodar cualquier número de entradas añadiendo resistencias de entrada adicionales en el nodo suma. Aunque los circuitos precedentes se han descrito en términos de entrada y de resistencias de realimentación, las resistencias se pueden reemplazar por elementos complejos, y los axiomas de los amplificadores operacionales se mantendrán como verdaderos. Dos circuitos que demuestran esto, son dos nuevas modificaciones del amplificador inversor.

El integrador

Se ha visto que ambas configuraciones básicas del AO actúan para mantener constantemente la corriente de realimentación, I_F igual a I_{IN}.

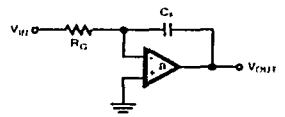


Fig. 2.60 Integrador.

Una modificación del amplificador inversor, el integrador, mostrado en la figura 2.60, se aprovecha de esta característica. Se aplica una tensión de entrada V_{IN}, a R_G, lo que da lugar a una corriente I_{IN}. Como ocurría en el amplificador inversor, V(-) = 0, puesto que V(+) = 0, y por tener impedancia infinita toda la corriente de entrada Iin pasa hacia el capacitor C_F, llamaremos a esta corriente I_F. El elemento realimentador en el integrador es el capacitor C_F. Por consiguiente, la corriente constante I_F, en C_F da lugar a una rampa lineal de tensión. La tensión de salida es, por tanto, la integral de la corriente de entrada, que es forzada a cargar C_F por el lazo de realimentación. Debido a que no se utilizará este circuito en el proyecto omitiremos su estudio, solo lo hemos mencionado como referencia de la teoría del AO.

El diferenciador

Una segunda modificación del amplificador inversor, que también aprovecha la corriente en un capacitor es el diferenciador mostrado en la figura 2.61.

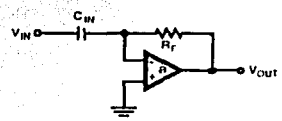


Fig. 2.61. Diferenciador

En este circuito, la posición de R y C están al revés que en el integrador, estando el elemento capacitativo en la red de entrada. Luego la corriente de entrada obtenida es proporcional a la tasa de variación de la tensión de entrada. Debido a que no se utilizará este circuito en el proyecto omitiremos su estudio, solo lo hemos mencionado como referencia de la teoría del AO.

El seguidor de tensión

Una modificación especial del amplificador no inversor es la etapa de ganancia unidad mostrada en la figura 2.62.

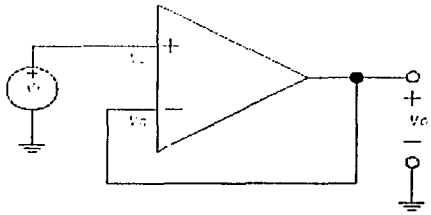


Fig. 2.62 Seguidor de tensión

En este circuito, la resistencia de entrada se ha incrementado hasta infinito, y R_F es cero, y la realimentación es del 100%. V_0 es entonces exactamente igual a Vi, dado que Es = 0. El circuito se conoce como "seguidor de emisor" puesto que la salida es una réplica en fase con ganancia unidad de la tensión de entrada. La impedancia de entrada de esta etapa es también infinita.

III. DISEÑO E IMPLEMENTACIÓN.

En este capítulo se describen detalladamente los pasos que seguimos para diseñar e implementar cada uno de los bloques que conforman al osciloscopio digital (Fig. 3.1):

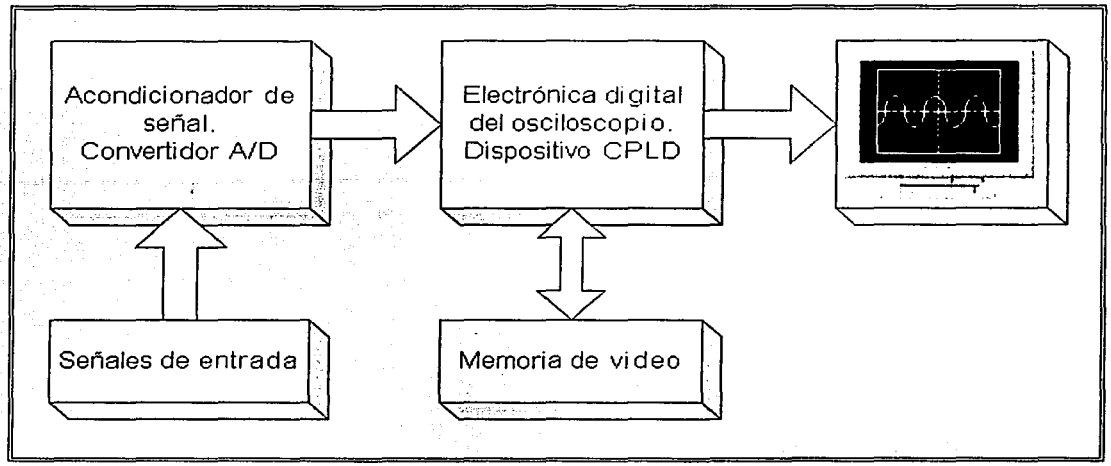


Fig. 3.1 Diagrama a bloques del osciloscopio digital.

Como se puede apreciar en la Fig. 3.1, el diseño del proyecto se divide en dos bloques principales:

- Acondicionador de señal. Adecua las señales de entrada al osciloscopio para que resulten compatibles con el convertidor analógico digital. Este último también forma parte de este mismo bloque y es necesario para proporcionar información al bloque de electrónica digital en un formato que éste pueda manipular. Es en éste bloque donde se genera la señal de deflexión vertical del osciloscopio.

- Electrónica digital del osciloscopio. Recibe la información digitalizada de la señal a medir y la procesa con el fin de almacenarla y poderla desplegar en un monitor VGA sobre un sistema coordenado de voltaje con respecto al tiempo, mediante la implementación de un controlador de video (driver). Este bloque también genera la señal de barrido horizontal.

III.1 ACONDICIONAMIENTO DE SEÑAL

En esta etapa hemos considerado varias fases. Estas consisten en 4, como se muestra en la figura 3.2:

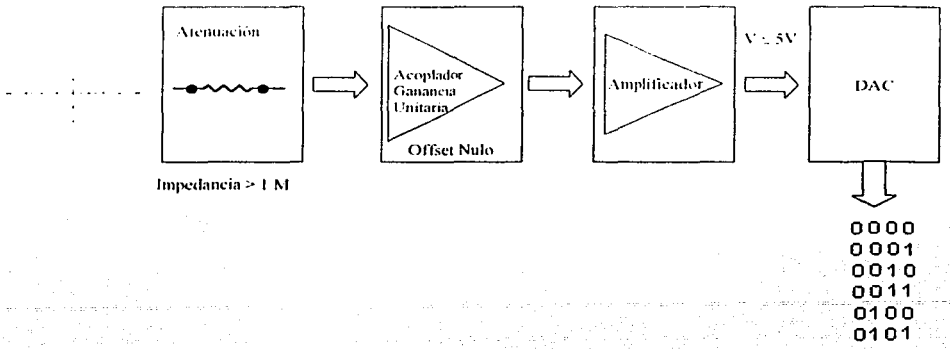


Fig. 3.2 Etapas de acondicionador de señal.

Acondicionador de Entrada

- Atenuación
- Acoplador
- Amplificador

Convertidor Analógico Digital

Como en capítulos anteriores dentro del marco teórico se ha descrito el funcionamiento de cada uno de estos, nos evocaremos más a la parte de implementación.

Acondicionador de Entrada:

El acondicionador de entrada es representado por la figura 3.2, muestra en bloques como actúan cada uno de los elementos que interviene en la adquisición de las señales externas para convertirlas en señales mesurables en la siguiente etapa de conversión digital.

Atenuación

En esta etapa consideraremos un grupo de resistencias que nos atenúan las señales de entrada como lo muestra la figura 3.2, de tal forma que sean más manejables las señales para los dispositivos analógicos que al final de cuentas nos permitan digitalizar dichas señales.

Para la implementación de esta etapa utilizaremos la división de voltaje o regla de división de voltaje que se aplica a configuraciones de resistores en serie, esta regla proporciona el voltaje a través de cualquier resistor en términos de resistencia y de voltaje a través de la combinación de serie. La fórmula de la división de voltaje para el circuito mostrado en la figura 3.3 es fácil de determinar.

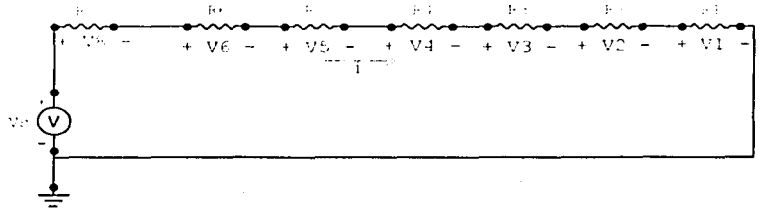


Fig.3.3 Circuito resistivo con división de voltajes.

Supóngase que se trata de calcular el voltaje V_2 por medio de la ley de ohm, $V_2 = IR_2$, asimismo, $I = V_S/(R_1 + R_2 + R_3 + ,,,R_8)$, al eliminarse I resulta que:

$$V_2 = R_2 V_S /(R_1 + R_2 + R_3 + ,,,R_8)$$

En general, para cualquier número de resistores en serie con una resistencia total R_T y con un voltaje V_S a través de combinación en serie, el voltaje V_X a través de un resistor R_X es

$$V_X = R_X V_S/R_T$$

Por consiguiente consideramos la fórmula anterior para calcular los resistores considerando las escalas de voltajes de V_S (entrada) a medir que serán de V = 10 mV / división, V = 100 mV / división, V = 300 mV / división, V = 1 V / división, V = 3 V / división, V = 10 V / división, V = 30 V / división, para cada uno de ellos esta asociada una salida a 10 mV / división, La salida de nuestro atenuador será de 80mV pico a pico a una escala total (10 mv / división) dicho en otras palabras tomemos la figura 3.3 y hagamos las respectivas asociaciones.

V_S = 30 V	V_1 = 10 mV	con R_1
V_S = 10 V	V_2 = 10 mV	con R_1
V_S = 3 V	V_3 = 10 mV	con R_1
V_S = 1 V	V_4 = 10 mV	con R_1
V_S = .30 V	V_5 = 10 mV	con R_1
V_S = .10 V	V_6 = 10 mV	con R_1
V_S = .03 V	V_7 = 10 mV	con R_1
V_S = .01 V	V_8 = 10 mV	con R_1

Considerando que la entrada es de 1 M entonces podríamos considerar encontrar el valor de cada una de las resistencias como lo muestra la siguiente tabla:

VS	RT	V1 a 8	I = VS / RT	R	
30V	1M	10 mV	3X10-5	$R_1 = V_1 / I$	= 333.33Ω
10V	1M	10 mV	1X10-5	$R_2 = V_2 / I - R_1$	= 666.67Ω
3 V	1M	10 mV	3X10-6	$R_3 = V_3 / I-(R_1+R_2)$	= 2333.33Ω
1 V	1M	10 mV	1X10-6	$R_4 = V_4 / I-(R_1+,,R_3)$	= 6666.67Ω
.3 V	1M	10 mV	3X10-7	$R_5 = V_5 / I-(R_1+,,R_4)$	= 23333.33Ω
.1 V	1M	10 mV	1X10-7	$R_6 = V_6 / I-(R1+,,R_5)$	= 66666.67Ω
.03 V	1M	10 mV	3X10-8	$R_7 = V_7 / I-(R1+,,R_6)$	= 233333.33Ω
.01 V	1M	10 mV	1X10-8	$R_8 = V_8 / 1-(R1+,,R_7)$	= 666666.67Ω

Los valores comerciales que introducimos y con los que probamos dieron los siguientes:

$R_1 = 330\Omega$ $R_2 = 2\,(390)\Omega$ $R_3 = 2.7K\Omega$ $R_4 = 2(3.9)K\Omega$ $R_5 = 27K\Omega$ $R_6 = 2(39)K\Omega$
$R_7 = 270K\Omega$ $R_8 = 2\,(390)K\Omega$

De una forma similar obtuvimos la siguiente tabla en la que se aprecia los valores muy próximos a $V_{1\,a\,8}$ = .009V, y la resistencia es de 1166610 Ω :

	Vs =0.01/div	Vs = 03/div	Vs = .1/div	Vs = .3/div	Vs = 1/div	Vs = 3/div	Vs = 10/div	Vs = 30/div
Escala 8	[illegible]	0.03	0.1	0.3	1	3	10	30
Escala 7	0.007685602	0.009041683	0.03313961	0.331396096	0.331396096	0.994188289	3.31396096	9.941882891
Escala 6	0.003313961	0.002998689	0.009956284	0.099956284	0.099956284	0.299868851	0.99956284	2.998688508
Escala 5	0.000330959	0.000992877	0.003309589	0.009928768	0.033095893	0.09928768	0.33095893	0.992876797
Escala 4	9.95191E-05	0.000298557	0.000995191	0.002985574	0.009951912	0.029855736	0.09951912	0.298557359
Escala 3	9.95191E-05	3.26587E-05	0.000326587	0.000979762	0.003265873	0.009797619	0.03265873	0.097976187
Escala 2	3.26587E-05	9.51475E-06	9.51475E-05	0.000285442	0.000951475	0.002854424	0.009514746	0.028544244
Escala 1	9.51475E-06	8.48613E-06	2.82871E-05	8.48613E-05	0.000282871	0.0008486 13	0.00282871	0.008486126

Como se nota la relación que guarda es de 1/3 como factor de conversión para las escalas partiendo de 30 V hasta los 10 milivolts, en las 8 escalas.

Colocando una resistencia en paralelo de 6.8 MΩ tenemos una impedancia de 995774.6 Ω que nos sirve de carga para los voltajes de entrada, pero también necesitamos un capacitor que nos quite a componente de corriente directa en señales, así como un pequeño switch que nos permita acoplar corriente directa, esto lo logramos solo colocando en paralelo con el capacitor.

El capacitor que usamos es de C = 100nf a 250 V (puede ser menor el voltaje, se hizo de esta forma ya que se considera que el atenuador no puede crecer más allá de 250 V). Este dispositivo solo nos permitirá manejar frecuencias de 2 a 20khz, dado por la fórmula de filtro pasa altas definido por f = 1/RC, donde R varía de 330Ω (f = 30Khz) a 6.8 MΩ (f = 1.4hz).

En la figura 3.4, nos muestra el bloque de atenuación que usaremos en este proyecto, las características de éste nos permitirá medir valores desde 10 milivolt /div hasta 30 volts/div en 8 pasos: V = 10 mV para escala 8, V = 30 mV para escala 7, V = 100 mV para escala 6, V = 300 mV para escala 5, V = 1V para escala 4, V = 3V para escala 3, V = 10V para escala 2, V = 30V para escala 1.
La impedancia de entrada es aproximadamente 1M ohm (dada por el circuito acoplador de la siguiente etapa) , la salida de nuestro atenuador será de 80 mV pk-pk para una escala de 10mV/ div, considerando que las señales de quipos electrónicos no sobrepasan estos rangos, para medir señales superiores a los 30 V, se tendría que utilizar resistencias más altas y reduciría nuestra escala de precisión que sería el objetivo de utilidad de este tipo de equipos. El ancho de banda permisible es de 2 Hz a 20KHz.

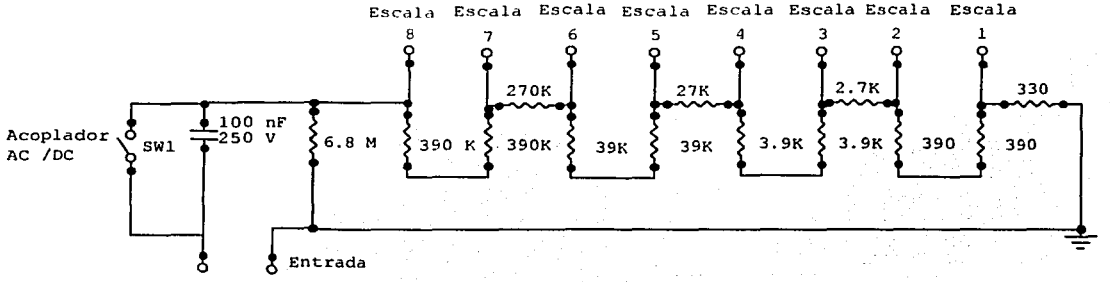


Fig. 3.4 Etapa de Entrada

Acoplador

El circuito acoplador (que es un seguidor de tensión) que utilizaremos con manejo de offset es como el que se muestra en la figura 3.5, como vemos utilizaremos un Amplificador Operacional, la explicación de su funcionamiento se dio en capítulo previo. Como se puede apreciar el circuito esta configurado

con ganancia unitaria con muy alta impedancia de entrada, contiene un offset de control nulo para asegurar que la salida es exactamente cero Volts cuando la entrada esta en corto circuito.

El circuito amplificador que utilizaremos es el TL 061, el cual tiene las caracteristicas del manejo de offset y además entrada de JFET, con una impedancia de entrada de 10^{12} M ohms, ancho de banda de 1 Mhz, Icc de 2.5 mA, Vcc = 15 V , –15 V, máxima ganancia 6 V/mv

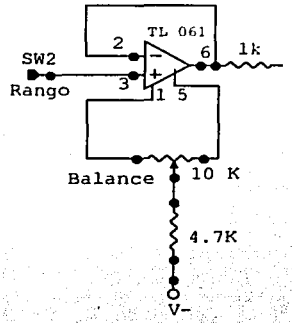


Fig. 3.5 Acoplador usando un Amplificador Operacional con entrada JFET

Nota: El arreglo de offset fue tomado del manual del fabricante de Texas Instruments.

Amplificador

Debido a que la señal que nos esta entregando el atenuador es muy pequeña es necesario amplificar esta a rangos que nos permita trabajar en valores permisibles para el convertidor AD, de tal forma, que utilizaremos un Amplificador Operacional como no inversor, con ganancia ajustable, así como lo muestra la figura 3.6. En esta etapa el primer circuito del cual la ganancia (A) es puesta alrededor de 62 y es ajustable por el potenciómetro de 50 kΩ, dando como resultado una amplificación de 80 mV pk-pk a 5 V pk-pk el cual es el rango total de entrada del convertidor A/D.

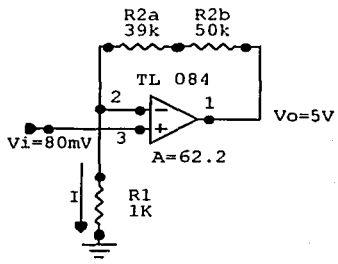


Fig. 3.6 Amplificador de señal.

La corriente a través de R_1 siempre determina la corriente a través de $R_2 = (R_{2a} + R_{2b})$ independientemente de R_2, por lo tanto R_2 puede utilizarse como un control de ganancia lineal, capaz de incrementar la ganancia desde la unidad mínima hasta el "infinito".

Para el cálculo de R_1 y R_2 tenemos que $Vi = IR_1$ y como $V_0 = I (R_1 + R_2)$ y $A = V_0/Vi = 1 + R_2/R_1$ entonces podemos calcular las resistencias, Considerando a $R_1 = 1$ KΩ debido a que es un valor

unitario y la entrada es en el orden de mili volts, así que la salida deberá de ser un factor multiplicativo de Mil.

Por lo tanto tenemos R_2 = (61-1) K Ω= 61 KΩ, considerando a R_2 como la suma de una resistencia constante y una variable por factor de ajuste y podemos tenerlo como R_{2a} de 39kΩ y una variable de R_{2b} de 50KΩ de tal forma que con esto estamos garantizando cualquier tipo de desajuste en nuestro equipo con un buen margen ($A_{mínimo}$ = 40 V0 = 3.2 V con 39 KΩ y $A_{máximo}$ = 90Vo = 7.2 V con 89 KΩ) y nos permitirá amplificar señales pequeñas que no estén en el rango establecido, pero serán poco confiables, así que se recomienda sólo para calibración la resistencia variable. Colocamos un sw3 que pone en corte la entrada del amplificador a tierra para ajustar la línea de base. El circuito final queda como lo muestra la figura 3.8.

El segundo circuito mostrado en la figura 3.7, permite el ajuste de posición de la línea de base, en esta fase tan bien se invierte la señal, utilizaremos un arreglo tipo diferencial, explicado en la parte teórica . Como se muestra en la figura 3.7 sólo es para alimentarlo en componentes de corriente directa y una tierra virtual. Mientras que la ganancia será unitaria, por ello podremos a R_1 = R_2 =1 K Ω y como es inversor A = -1 y entonces V_0 = Vi, nótese que V + y V- a través de R4 nos da una componente directa es decir Suman o Restan un voltaje directo al voltaje de entrada a través de R_2 la salida del amplificador operacional tendrá una componente de corriente directa más la señal que podemos mover ya sea negativa o positivamente.

Para los valores de R_3, R_4, R_5, consideramos un circuito a la mitad de tal forma que tendremos una tierra virtual y el voltaje en V(+) = 0 por ello la resistencia R_3 que pongamos en esta posición podemos considerarla = R_1 de tal forma que V_0 = $V_{inicial\ (+)}$ +$V_{inicial\ (-)}$, mientras que para R_4 deberá de ser una resistencia grande en el orden de KΩ ya que sólo nos permitirá tener una fuerte caída de voltaje que oscile entre los + 15 y – 15 volts podríamos poner 27 KΩ y la R_5 =a una resistencia variable de 10 que nos permitirá movernos de + 5 a 0 a –5 V, por consiguiente para el punto central de la resistencia variable tendríamos V + =15V = $V_{27KΩ\ =10V}$ + $V_{5kΩ\ =5V}$ y para V- = -15 V = $V_{27KΩ\ =-10V}$ + $V_{5kΩ}$ = - 5V. Por consiguiente la corriente es de 1 mA dando como resultado que este circuito de la componente directa que se entregará al convertidor analógico a digital.

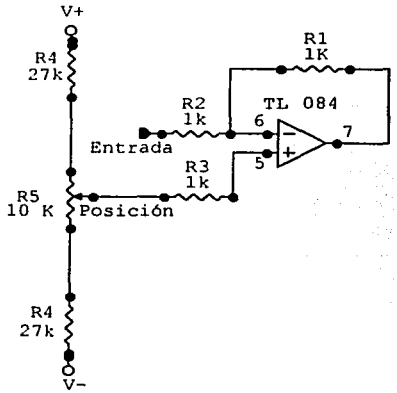


Fig. 3.7 Arreglo tipo diferencial para dar componente directa a la señal de entrada

En esta etapa se consideró el amplificador operacional TL084 que tiene entrada JFET, con una impedancia de entrada de 10^{12} M ohms, ancho de banda de 4 Mhz, Icc de 2.5 mA, Vcc = 15 V , –15 V, máxima ganancia 200 V/mv

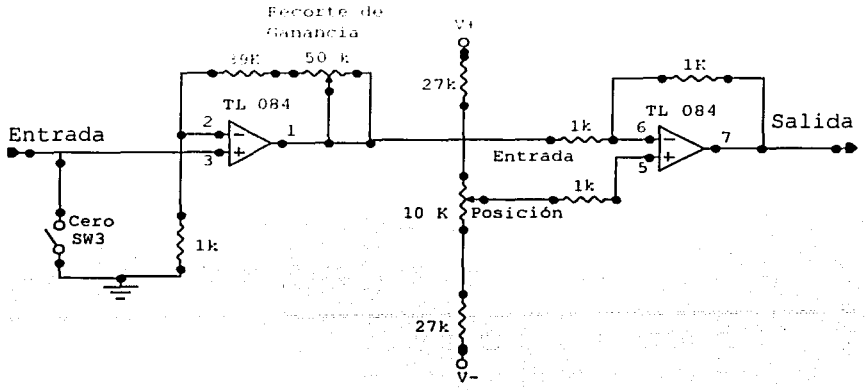


Fig. 3.8 Etapa de amplificación .

Convertidor Analógico Digital

En este caso tenemos finalmente la etapa que convierte la señal linealizada analógica a digital como lo muestra la figura 3.2. Hemos considerado para este caso el ADC 804 CMOS 8 bits, utiliza el método de aproximaciones sucesivas, del fabricante National, que por ser versátil, económico y de fácil de adquirir.

Características más importantes del ADC804 directamente de las hojas técnicas del fabricante:

- Tecnología CMOS 8-bits (2^8 = 256)
- Método de aproximaciones sucesivas.
- Rango de Vcc 4.5 Vdc a 6.3 Vdc , típico 5 V (el que estamos usando).
- Rango de voltaje analógico de entrada de 0 a 5 V.
- Tiempo de conversión de 100 μs.
- Modo de Conversión libre 8770 a 9708 conv /s
- Reloj 100khz mínimo, 1460 Khz máximo el típico es de 640 (exactitud garantizada)

La configuración que usaremos se muestra en la figura 3.9.

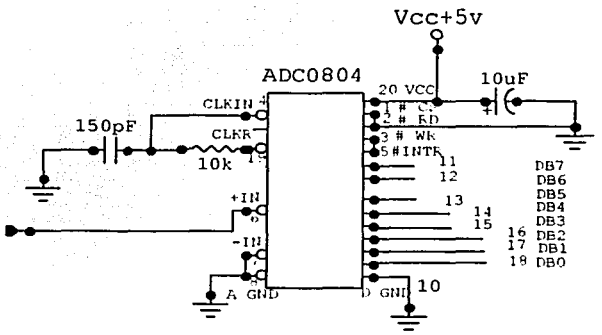


Fig. 3.9 Diagrama del D804 .

El convertidor consta de 20 pines, de los cuales del 11 al 18 son pines de salida de datos, donde el pin 11 es el más significativo (MSB) en el momento de hacer la lectura del valor en binario de un voltaje, y el pin 18 es menos significativo (LSB), ya que se empieza a tomar los datos a partir de este.

En el pin 20 se colocó el voltaje de alimentación, que fueron +5V, y en los demás pines se coloco una serie de componentes electrónicos según lo sugería el plano del circuito.

El convertidor utiliza un reloj para regular el tiempo entre cada conversión de un voltaje análogo a una señal digital. El A/D utiliza un reloj de entrada (CLK IN) y un reloj de respuesta (CLK R) el cual se puede sincronizar con el reloj interno de la CPU del computador o utilizando un circuito resistivo externo.

Este circuito externo utiliza una resistencia de 10k y un capacitor de 150pF; el primer elemento conectado entre el pin 19(CLK R) y el pin 4(CLK IN) y el segundo conectado entre tierra digital y el pin 4(CLK IN) para lograr una frecuencia de operación de 640 kHz.

El convertidor inicia su labor permitiendo la libre circulación de la corriente, para esto el pin 1 (CS) se conecta con tierra y el pin 3(WR) se une a un pulsador, como la seña esta presente de tiempo de conversión este pin estará referido a tierra para que dé inicio a la conversión de la señal.

Para modo de conversión libre INTR debe de estar conectado con WR y CS = 0 , que es la forma que estará operando el ADC.

Entre tierra y el pin 20 se ubica un capacitor de 10u F, cuya función es reducir el ruido en el voltaje de alimentación del A/D, este ruido puede causar errores en la conversión de la señal.

El voltaje ingresa directamente al convertidor por medio del pin 6(VIN(+)).

Para llevar los voltajes convertidos en números binarios al Circuito Controlador, los pines del convertidor, del bit menos significativo(LSB – pin 18) al bit más significativo(MSB – pin 11) se unen respectivamente al los pies del puerto del 2(LSB) al pin 9(MSB), cada uno de los pines del A/D se conecta con su correspondiente pin del puerto de memoria VRAM.

Para iniciar la conversión se debe de activar la señal #CS (poner a nivel bajo) y dentro del periodo de activación activar la señal #WR durante un mínimo de 100 ns. El convertidor activará la línea #INTR (el símbolo # indica que la señal es activa a nivel bajo) para indicar que ha terminado la conversión figura 3.10. A partir de ese momento se puede leer el dato figura 3.11.

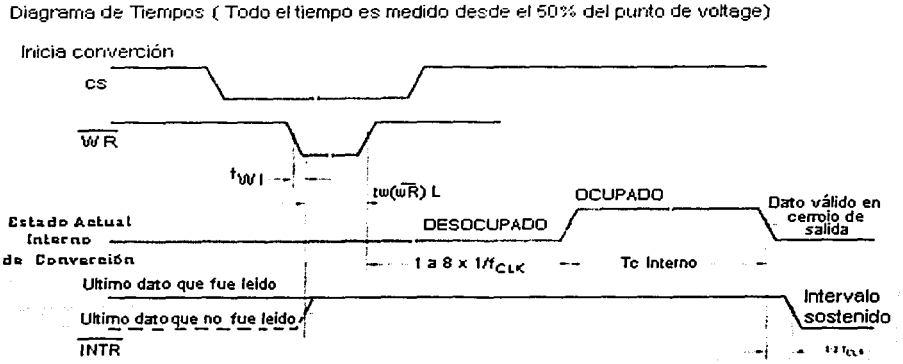
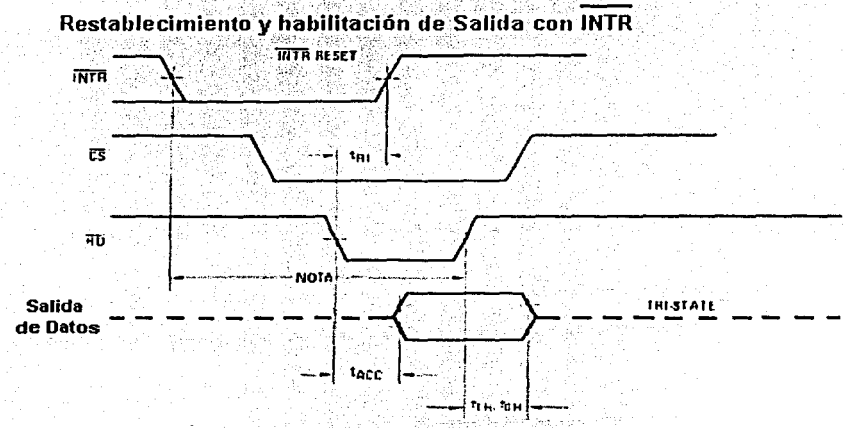


Fig. 3.10 Diagrama de tiempos para el comienzo de la conversión.



Nota: Lectura electroboscópica ocurre 8 períodos de reloj después de afirmación de interrupción garantiza un restablecimiento del INTR .

Fig. 3.11 Diagrama de tiempos para la lectura del dato

El proceso de lectura está representado en la figura 3.11. Se selecciona el convertidor a través de la línea #CS y se envía un pulso negativo por la línea #RD, tras unos 200 ns (tACC: tiempo de acceso) el dato ya se puede leer y se desactiva la señal de lectura. Estos ciclos se repetirán tantas veces como datos se quieran leer.

Después de lo anterior, unimos los bloques que se presentaron al inicio de este tema y tendremos nuestro circuito de entrada como lo muestra la figura 3.12.

La salida del convertidor nos dará los datos que nos permitirán usarlos en la entrada de la siguiente etapa Digital, perteneciente a los dispositivos lógicos programables complejos (CPLD) en nuestro caso el Flex de Altera.

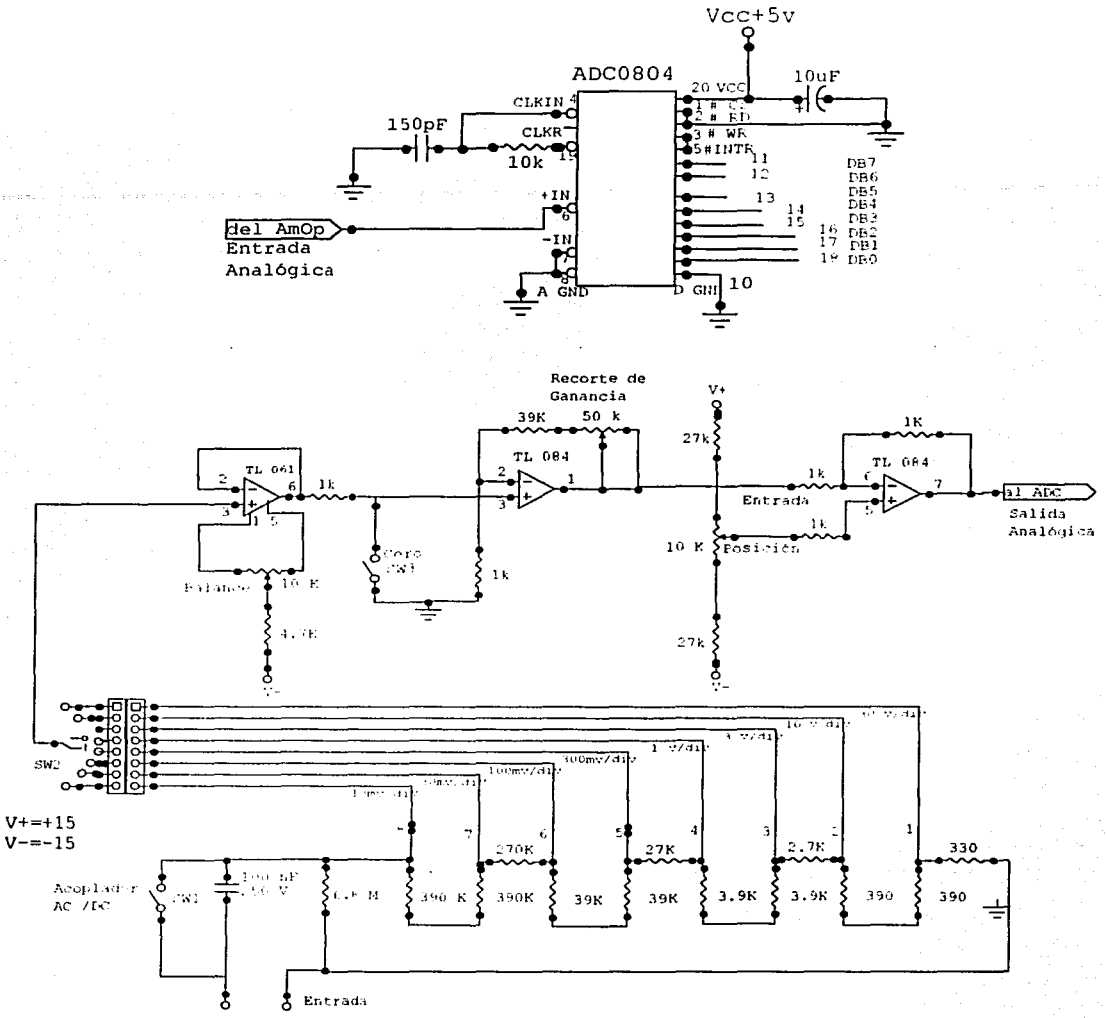


Fig. 3.12 Diagrama del acondicionador de señal.

III.2 ELECTRONICA DIGITAL DEL OSCILOSCOPIO

El bloque que comprende la electrónica digital del dispositivo se subdivide en tres módulos principales que se programaron en el dispositivo CPLD y que se denominan *Controlador de Video*, *Controlador de VRAM* y *Base de Tiempos*. El diagrama de bloques que se muestra en la Fig. 3.13, ilustra de una forma muy general cómo se interconectan entre sí y con los restantes bloques del proyecto, es decir, el acondicionador de señal, el monitor VGA y la memoria de video.

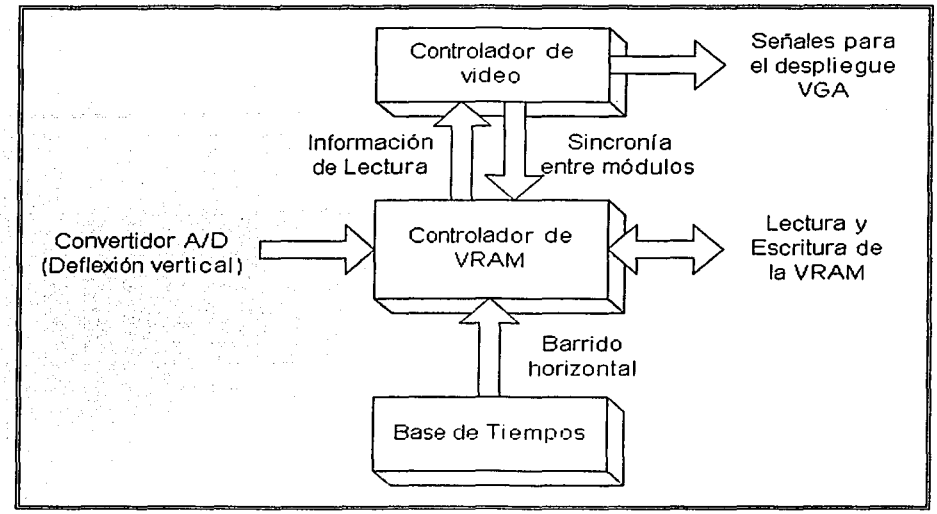


Figura 3.13 Diagrama a bloques de la Electrónica Digital del Osciloscopio.

A grandes rasgos, las funciones que cubre cada módulo se explican a continuación:

- *Controlador de Video.* Suministra a un monitor VGA la información que recibe del controlador de VRAM para cada pixel en la pantalla que se requiera actualizar, así como las señales de sincronía tanto horizontal como vertical que requiere para su correcto funcionamiento. Asimismo, proporciona señales de sincronización al módulo *controlador de VRAM* a fin de garantizar una perfecta comunicación entre ambos. Adicionalmente y como detalles de diseño, agregamos al módulo la función de desplegar la cuadrícula característica de los osciloscopios comerciales y una etiqueta que indica la escala que estamos empleando en la base de tiempos.

- *Controlador de VRAM.* Traduce la información del convertidor analógico digital como una deflexión vertical que la señal en pantalla debe mostrar con respecto a un valor de referencia, así como la información del módulo *Base de Tiempos* como un barrido horizontal, a fin de localizar y actualizar la información contenida en el mapa de memoria de video (VRAM) que corresponde a cada uno de los pixeles del monitor que forman la gráfica de la función de voltaje de la señal introducida con respecto al tiempo. También es responsable de suministrar la información almacenada en la VRAM al módulo *Controlador de Video* en el momento preciso en que éste lo requiere.

- *Base de Tiempos.* Mediante el empleo de contadores internos, se encarga de proporcionar al módulo *Controlador de VRAM* la señal de barrido horizontal con la velocidad correspondiente a la escala de tiempos seleccionada por el usuario, medida en milisegundos por división (ms / div).

El diseño e implementación de cada uno de los módulos que conforman la electrónica digital del osciloscopio así como la integración de los mismos como un solo bloque, se desarrolla con detalle en los subtemas siguientes.

III.2.1 Módulo Controlador de Video.

El módulo que controla el video, como ya se expuso en el marco teórico, debe proveer al monitor VGA de 5 señales perfectamente sincronizadas para realizar el barrido completo de los 480 renglones, conformados cada uno por 640 pixeles. Estas 5 señales son: pulso de sincronía horizontal, pulso de sincronía vertical y 3 señales de color.
Bajo la premisa de diseñar una interfaz monocromática, se requirió de un solo bit para indicar al monitor uno de dos estados posibles: encendido (o blanco) y apagado (o negro). Sin embargo, para ciertos despliegues especiales, es posible el empleo de color como se explica más adelante en éste mismo subtema.
Para trabajar correctamente, el módulo VGA debe contar con una entrada de reloj con frecuencia de 25.175 MHz, que es la misma con la que el cañón electrónico del monitor cambia de posición para iluminar otro pixel del renglón en el que se encuentra. Asimismo, en el momento preciso en el que un pixel específico es apuntado, se debe contar con la información del color con el que se iluminará. Esa información necesita de un dispositivo de almacenamiento (memoria RAM) para cada ciclo de actualización de la pantalla. Por lo tanto, la cantidad de espacio necesario para almacenar el estado del monitor está dada por la ecuación:

Espacio Necesario (en bits) = No. Renglones x No. Columnas x No. Bits por Pixel

Antes de comenzar la etapa de diseño, fue necesario tomar una decisión sobre el formato en el que presentaríamos la información en la pantalla. Inicialmente se consideró utilizar la totalidad de pixeles disponibles, por lo que la ecuación anterior nos daría:

Espacio Necesario = 480 x 640 x 1 bit = 307200 bits = 38400 bytes

El dispositivo CPLD a programar (EPF10K20) es capaz de proveer una cierta cantidad de memoria, limitada a un máximo de 12544 bits. Esto nos daría un área de trabajo bastante reducida: un cuadrado de 112 pixeles por lado. Por esa razón, se decidió utilizar una memoria externa preferentemente del tipo estático, para evitar la necesidad de constantes actualizaciones que requeriría el empleo de memoria dinámica.

Como la mayoría de los dispositivos de memoria están organizados en localidades de 8 bits, si quisiéramos utilizar la totalidad de la pantalla necesitaríamos disponer de una memoria con 2^16 localidades (65536), pues la potencia de 2 anterior, es decir 2^15 = 32768 resulta insuficiente para cubrir el resultado obtenido en la ecuación de Espacio Necesario. Sin embargo, tomando en cuenta que no es indispensable utilizar la totalidad de la pantalla, y que el espacio desperdiciado en una memoria con 2^16 localidades superaría el 40%, decidimos seleccionar un dispositivo con 32768 localidades para albergar 262144 pixeles.

A continuación nos confrontamos con la polémica de decidir cómo iríamos a distribuir sobre la pantalla el área disponible. Entre las posibles alternativas, discutimos las siguientes propuestas: la mayor área posible, un área proporcional a la de la pantalla del monitor, un área compatible con potencias de 2 y un área proporcional a la pantalla de los osciloscopios comerciales.

- Mayor área posible. Debido a que el paralelogramo con mayor área posible es el cuadrado, la medida de cada lado del cuadrado que debíamos colocar como área de trabajo se obtiene como la raíz cuadrada de 262144. Este resultado nos da un cuadrado de 512 pixeles por lado, medida que además resulta compatible con una potencia de 2. Desafortunadamente, el número máximo de renglones de los que disponemos se limita a 480. Por lo tanto, el área máxima de trabajo se obtiene con un rectángulo de 480 renglones por 546 columnas, como se muestra en la Fig. 3.14.

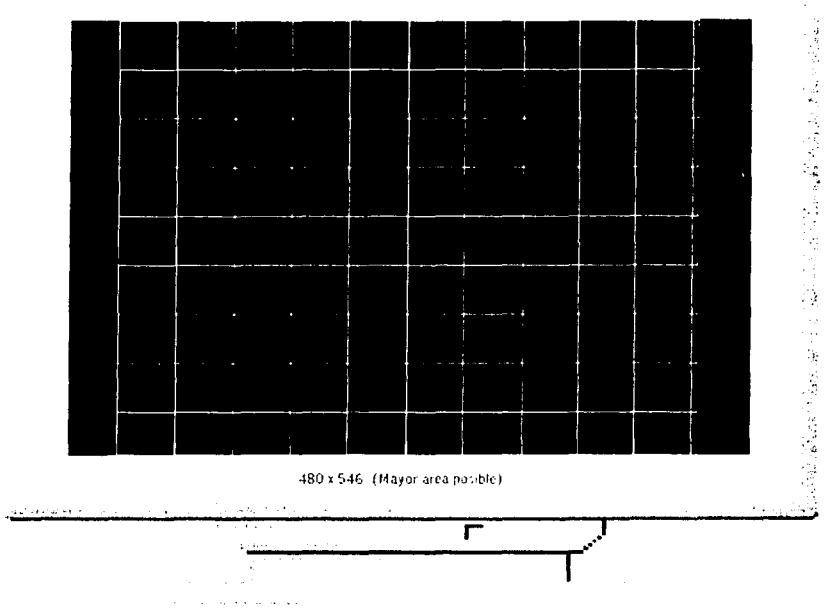


Fig. 3.14 Mayor área de trabajo posible, empleando una memoria con 2^{15} localidades de 8 bits.

- Área proporcional a la pantalla del monitor. La pantalla de los monitores de computadora generalmente guarda una proporción de 3 partes de alto por 4 partes de ancho, es decir, 75%. La ventaja de esta propuesta radica en el hecho de que, al centrar el área de trabajo, la pantalla deja un marco negro de un ancho uniforme, como lo muestra la Fig. 3.15.

  La disposición mostrada (384 x 512 pixeles) es además compatible con una potencia de 2 en el número de columnas.

- Área compatible con potencias de 2. A medida que la fase de diseño avanzó sobre el controlador de la memoria de video (VRAM), fue evidente la ventaja de disponer de medidas compatibles con potencias de 2 (específicamente, el número de columnas). La razón se debe a que la información de los pixeles dispuestos en una matriz de n renglones por m columnas, debe almacenarse en una matriz diferente: un mapa de memoria de 8 bits de ancho y un máximo de 2^15 direcciones. La ecuación para convertir las coordenadas (Renglón, Columna) de un pixel en específico en el área de trabajo a una dirección del mapa de memoria es la siguiente:

  Dirección = $Ent\left(\dfrac{Renglón \times m + Columna}{8}\right)$

  O bien:

  Dirección = Renglón x m / 8 + Ent(Columna / 8)

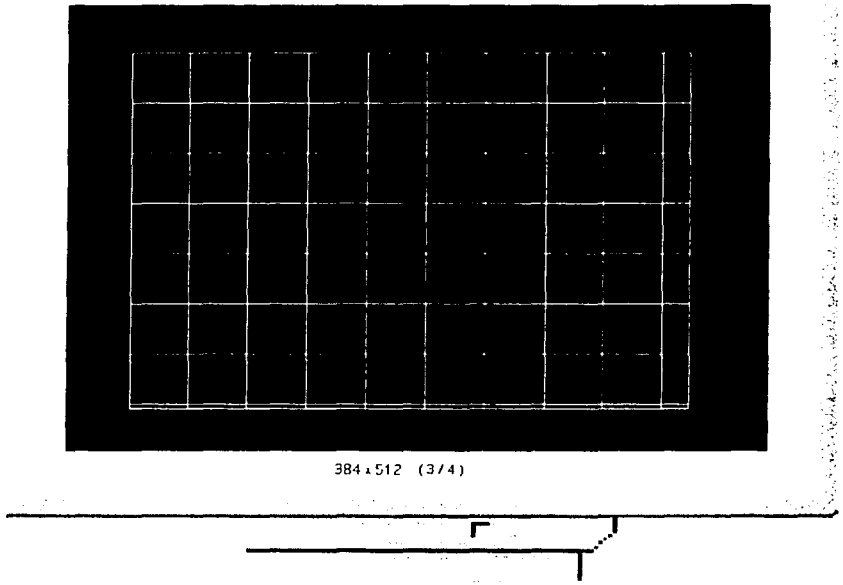


Fig. 3.15 Distribución del área de trabajo empleando 384 renglones y 512 columnas (proporción 3/4).

Donde las variables Dirección, Renglón y Columna se comienzan a contar desde cero; y la operación Ent(Columna / 8) significa obtener la parte entera de la división de la variable Columna entre el ancho del mapa de memoria. Ésta última cantidad (8 bits por palabra) también debe dividir al número de columnas que abarcan un renglón. De este modo, se recomienda que el número de columnas m del área de trabajo sea un número potencia de 2, ya que de lo contrario, la lógica aritmética para implementar la ecuación se vuelve más compleja. Algunas disposiciones para ésta propuesta se muestran en todas las figuras marcadas con la etiqueta (n x 512), así como en la Fig. 3.16:

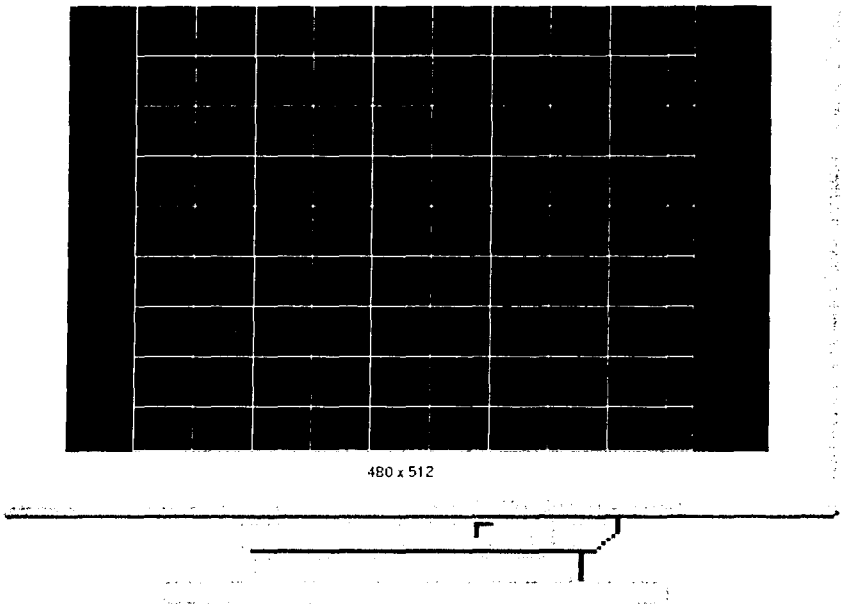


Fig. 3.16 Mayor área de trabajo con número de columnas compatible con potencia de 2, empleando una memoria con 2^{15} localidades de 8 bits.

- Área proporcional a la pantalla de los osciloscopios comerciales. Los osciloscopios comerciales están dispuestos en una conformación cuadriculada de 8 por 10 unidades, es decir, 4 / 5. La ventaja de ésta propuesta recae directamente sobre el usuario final, ya que al visualizar una pantalla estandarizada, le resulta más fácil familiarizarse con el manejo del instrumento. Las siguientes figuras muestran algunas de las distribuciones que son compatibles con ésta propuesta:

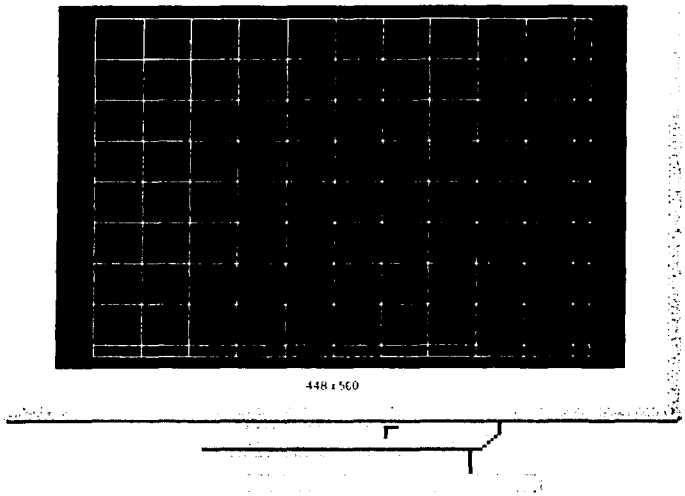


Fig. 3.17 Área de trabajo con 448 renglones por 560 columnas (proporción 4/5).

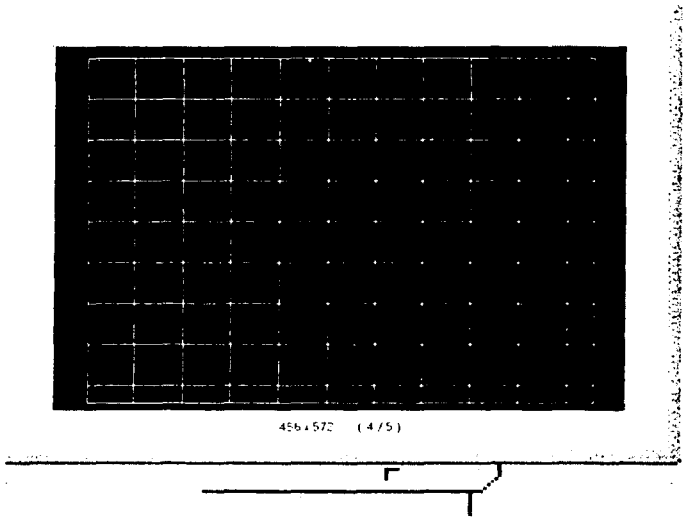


Fig. 3.18 Mayor área de trabajo con proporción 4/5, empleando una memoria con 2^{15} localidades de 8 bits.

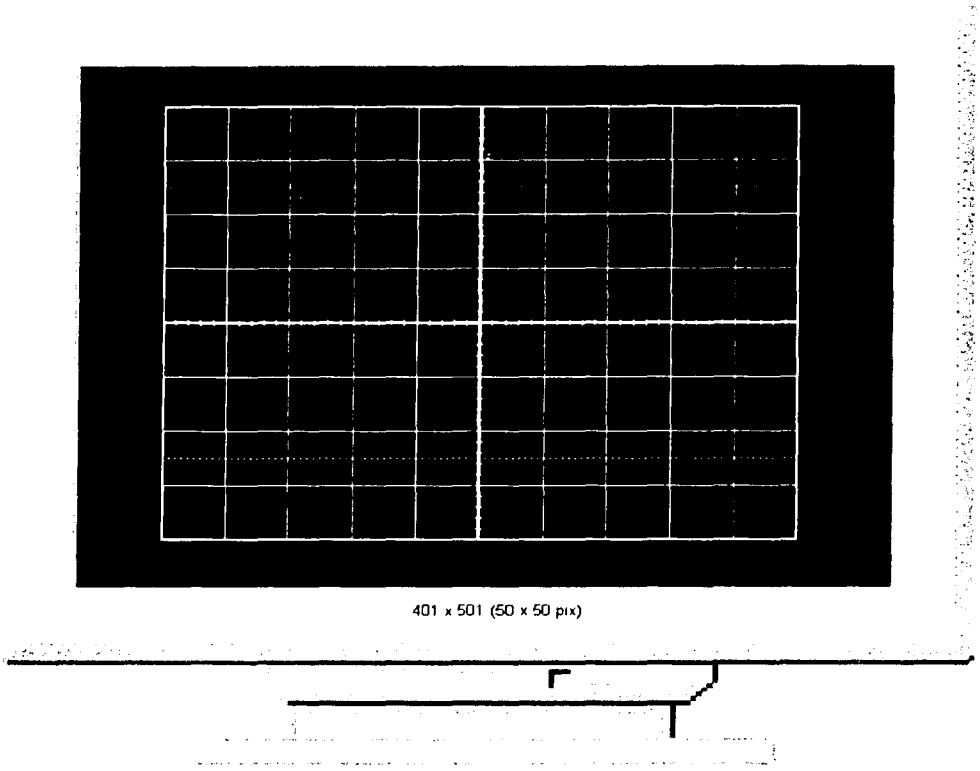


Fig. 3.19 Área de trabajo de 401 renglones por 501 columnas, dividida en cuadrados de 50 pixeles por lado, en proporción 4 / 5.

Finalmente, el área elegida fue la mostrada en la figura anterior, esto es, 401 x 501 pixeles, dividida en 80 cuadrados de 50 x 50. Además de la ventaja de ser un área estandarizada, ofrece otras cualidades como la de compatibilidad con potencia de 2, pues las últimas 11 columnas de cada renglón que faltan para completar el número 512, simplemente serán borradas al momento de desplegar la información en la pantalla. Otra ventaja más: la división en cuadros de 50 pixeles por lado simplifica el diseño de las escalas horizontal y vertical, así como la interpretación de la información por parte del usuario final. La columna y el renglón sobrantes al área de 400 x 500 sirven para delimitarla completamente con un marco de uno o dos pixeles de ancho.

Una vez hecha la decisión sobre la distribución de información sobre la pantalla, en este punto es importante diferenciar dos ideas importantes que se utilizarán en este subtema y en los posteriores. A manera de glosario, el concepto **Área de Trabajo** se refiere a la distribución de 401 renglones por 501 columnas que define la superficie visible en la pantalla del monitor; mientras que el concepto **Matriz de Datos Almacenados en la Memoria de Video** corresponde a una superficie ligeramente mayor: 401 renglones por 512 columnas, pues ya se resaltó la conveniencia de utilizar un número de columnas compatible con potencia de 2. El hecho de que ambos conceptos coexistan implica que en la memoria de video se almacenan más datos de los que se desplegarán en la pantalla, pero reditúa

en una mayor facilidad para vincular las coordenadas de cualquier pixel con su respectiva información dentro del mapa de memoria.

Regresando al problema del diseño del controlador de video debemos considerar que, independientemente del tamaño del área de trabajo que utilicemos sobre la pantalla, el monitor VGA requiere una serie ordenada de pulsos que utilizará para funcionar correctamente, esto es, pulsos de sincronia horizontal, pulsos de sincronía vertical y 3 señales de color que determinan la tonalidad que tendrá la gráfica desplegada. Sobre los pulsos de color se debe enfatizar que, al emplear un solo bit para almacenar la información de cada pixel, sólo contamos con dos opciones para iluminarlos, de un total de 8 posibilidades: negro, rojo, verde, amarillo, azul, magenta, cian y blanco. Aunque la obvia elección para "apagar" un pixel es con color negro y "encenderlo" es con blanco, podríamos escoger cualesquiera de los otros colores. El tiempo de actualización de cada pixel está dado por el inverso de la frecuencia 25.175 MHz (v. Marco Teórico), y la duración de los pulsos de sincronía tanto horizontal como vertical resultan ser múltiplos de ese periodo, por lo que el controlador de video diseñado tiene como base de casi todas sus operaciones al menos un contador.

Basándonos en las consideraciones anteriores y en algunas más que se tomaron en cuenta a lo largo de la implementación, el código en lenguaje VHDL para programar el CPLD y que constituye la interfaz VGA, se desarrolla de la siguiente forma:

El módulo es denominado *control_video*, y primeramente se designan las librerías necesarias. Seguidamente se define como una entidad con 3 señales de entrada y 8 de salida, como lo muestra la Fig. 3.20. En éste, como en todos los esquemas similares generados por el programa *MAX+PLUS II*, se suelen mostrar las entradas del lado izquierdo y las salidas del lado derecho.

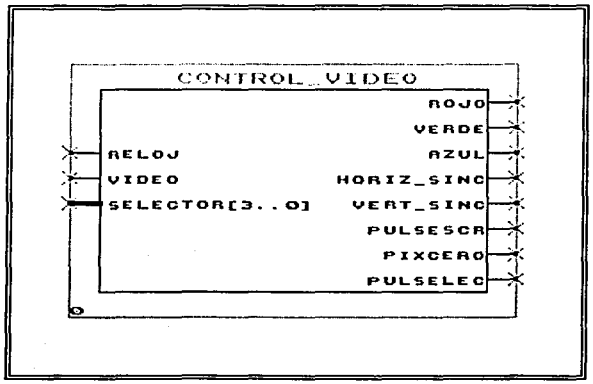


Fig. 3.20 Esquema de entradas y salidas del módulo Controlador de Video.

Entradas:

- *Reloj*. Entrada de reloj que requiere una señal cuadrada con una frecuencia de 25.175 MHz, que es la misma con la que el cañón electrónico actualiza la información de cada pixel en el monitor.
- *Video*. Recibe la información de "encendido" ('1' lógico) o "apagado" ('0' lógico) con la que debe contar el cañón electrónico en el momento de apuntar a un pixel dentro del área de trabajo.
- *Selector*. Entrada de 4 bits para seleccionar uno de 11 estados posibles, correspondientes a las etiquetas de milisegundos por división para desplegar la información sobre la escala de tiempos elegida por el usuario. Esta entrada también es compartida por el módulo *Base de Tiempos*.

Salidas:

- *Rojo*, *Verde*, *Azul*. Son las salidas de color, destinadas a las tres respectivas entradas del monitor empleado.

- *Horiz_sinc*. Indica al monitor cuándo debe regresar el cañón electrónico a la primera columna de pixeles, pero cambiando de renglón.
- *Vert_sinc*. Indica al cañón electrónico el momento en el que debe regresar a la primera posición de la pantalla, es decir, renglón '0' columna '0'.
- *Pulsolec*. Señal destinada al módulo *Controlador de VRAM* específicamente al submódulo *bufferpri*. Envía pulsos de la entrada *Reloj* únicamente cuando el cañón electrónico apunta a un pixel dentro del área de trabajo.
- *Pixcero*. Destinada también al submódulo "bufferpri". Sincroniza el inicio de lecturas en la memoria de video con el inicio del barrido izquierda-derecha en el primer renglón del área de trabajo por parte del cañón electrónico del monitor.
- *Pulsescr*. Señal empleada por el módulo *Controlador de VRAM*, que indica los intervalos de tiempo en los que el cañón electrónico del monitor se encuentra fuera del área de trabajo de la pantalla.

```
--control_video.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

entity control_video is
   port(signal Reloj: in std_logic;
        signal Video : in std_logic;
        signal Selector: in std_logic_vector(3 downto 0);
        signal Rojo, Verde, Azul : out std_logic;
        signal Horiz_sinc, Vert_sinc : out std_logic;
        signal Pulsolec : out std_logic;
        signal Pixcero, Pulsescr: out std_logic);
end control_video;
```

Después de definir el módulo como entidad, se procedió a establecer las funciones que realizaría en su arquitectura, comenzando por la definición de sus variables internas y constantes a utilizar:

- *H_cont* y *V_cont* son contadores binarios de 10 bits que deben ser capaces de realizar 800 y 525 conteos respectivamente, debido a las duraciones de los ciclos de actualización horizontal y vertical que se detallan más adelante.
- *Encendido* consta de un solo bit que sirve para inicializar algunas de las demás variables internas.
- *Info_Rojo*, *Info_Verde* y *Info_Azul* son variables de un bit donde se almacena la información de color de un pixel dependiendo de los valores de las variables *Ejes*, *Etiq* y *Area*, así como de la señal de entrada *Video*.
- *Info_Horiz_sinc* y *Info_Vert_sinc* almacenan temporalmente los pulsos de sincronía horizontal y vertical.
- *Matriz* , *Leclinea* y *Lecrenglon* indican los intervalos en los que el cañón electrónico apunta a un pixel correspondiente a la matriz de datos almacenados en la memoria de video.
- *Ejes*, *Area* y *Etiq* son variables de un bit para indicar respectivamente cuáles pixeles conforman la cuadricula de divisiones en la pantalla, cuál es el área de trabajo y dónde se encuentran los pixeles que despliegan la etiqueta de milisegundos por división de la escala de tiempos empleada.
- *H_max* y *V_max* son constantes que establecen las cuentas máximas horizontal y vertical de los contadores *H_cont* y *V_cont* respectivamente. La razón por la que adquieren los valores de 799 y 524 se expone más adelante
- *CentroH* y *CentroV* son las constantes que se emplean para centrar el área de trabajo dentro de la pantalla VGA: al disponer de 640 columnas y 480 renglones y como el área de trabajo se diseñó para contener prácticamente 500 columnas y 400 renglones, existe un excedente de 140 columnas y 80 renglones. Si repartimos el espacio sobrante en dos grupos de 70 columnas y dos grupos de 40 renglones, el área de trabajo quedará automáticamente centrada. Por esa razón, las constantes se establecen como *CentroH* = 70 y *CentroV* = 40.

```
architecture arq of control_video is
signal H_cont,V_cont: std_logic_vector(9 Downto 0);
signal Encendido: std_logic;
signal Info_Rojo,Info_Verde,Info_Azul: std_logic;
signal Info_Horiz_sinc,Info_Vert_sinc: std_logic;
signal Matriz, Leclinea, Lecrenglon: std_logic;
signal Ejes,Area,Etiq: std_logic;

constant H_max : std_logic_vector(9 Downto 0) := CONV_STD_LOGIC_VECTOR(799,10);
-- 799 es la cuenta máxima horizontal
constant V_max : std_logic_vector(9 Downto 0) := CONV_STD_LOGIC_VECTOR(524,10);
-- 524 es la cuenta máxima vertical
constant CentroH: std_logic_vector(9 Downto 0) := CONV_STD_LOGIC_VECTOR(70,10);
constant CentroV: std_logic_vector(9 Downto 0) := CONV_STD_LOGIC_VECTOR(40,10);

begin
Info_Rojo <= Etiq and Area;
Info_Verde <= (Video or Etiq) and Area;
Info_Azul <= (Video or Ejes or Etiq) and Area;
```

En la parte del código donde se asignan los valores para *Info_Rojo*, *Info_Verde* y *Info_Azul*, tomamos la decisión de distinguir con tres distintos colores la información que se presenta en la pantalla: con color azul los marcos de cada una de las 80 divisiones; con color cian (combinando azul y verde) la señal externa a medir; y con color blanco (combinando rojo, azul y verde) la etiqueta de milisegundos por división de la escala de tiempos empleada. La operación AND que conecta cada una de estas combinaciones con la variable *Area*, evita la iluminación de pixeles fuera del área de trabajo.

Cada uno de los tres renglones siguientes en el código, establece una relación importante para definir dos de las tres señales de sincronía con el módulo *Controlador de VRAM*. La primera sentencia indica que la señal *Matriz* sólo se activa cuando las señales *Leclinea* y *Lecrenglon* se encuentran ambas en estado lógico '1'. Al respecto podemos adelantar que esto ocurre cuando el cañón electrónico del monitor se encuentra apuntando a un pixel correspondiente a la matriz de datos almacenados en la memoria de video. En el siguiente renglón se define la salida *Pulsolec* a partir de la entrada *Reloj* y de la señal *Matriz* que se acaba de establecer, de tal modo que se produce una señal con frecuencia idéntica a la entrada *Reloj*, pero que se interrumpe cada vez que el cañón electrónico deja de apuntar dentro del área de trabajo, esto es, todo el tiempo que transcurre entre la actualización del último pixel de un renglón hasta el despliegue de información correspondiente al primer pixel del renglón siguiente. Finalmente, en la última sentencia se invierte la señal *Matriz* con el fin de indicar con estado activo alto a la salida *Pulsescr* que el cañón electrónico está fuera del área correspondiente a la matriz de datos almacenados en la memoria de video.

```
Matriz <= Leclinea and Lecrenglon;
Pulsolec <= Reloj and Matriz;
Pulsescr <= not Matriz;
```

En este punto y adelantándonos al subtema que detalla el diseño del *Controlador de VRAM*, podemos profundizar en la necesidad de implementar señales de sincronía entre ambos módulos. Hasta ahora, se ha establecido que la salida *Pulsescr* es una de ellas y tiene la función de indicar al *Controlador de VRAM* que no se requiere más información de la memoria de video. La importancia de ésta señal está dada por la definición de un tiempo que denominaremos "de descanso" a lo largo de éste capítulo, que no es mas que el tiempo de espera entre renglones descrito en el párrafo anterior que el cañón electrónico debe guardar. Así, el *Controlador de VRAM* puede aprovechar estos intervalos de tiempo para dejar de "leer" la información de la memoria y dedicarse a "escribir" la información que provenga en esos momentos por parte del bloque acondicionador de señal. En el subtema siguiente, estos conceptos definen un par de ciclos dentro de una máquina de estados denominados *Lectura* y *Escritura*. Por otro lado tenemos la salida *Pulsolec*, que debe indicar al Controlador de VRAM el momento preciso en que debe proporcionar la información de cada uno de los 512 pixeles que conforman alguno de los renglones del área que seleccionamos para trabajar. Esto sólo se logra suministrando un reloj como el que se describió en el párrafo anterior: trabajando a 25.175 MHz, pero interrumpiendo su operación durante los tiempos "de descanso". Finalmente, para garantizar que el *Controlador de Video* reciba la información de la memoria correspondiente al renglón que se está

actualizando, es necesario implementar una señal que indique al *Controlador de VRAM* que el cañón electrónico del monitor se encuentra a punto de iniciar la actualización del pixel correspondiente al primer bit dentro del mapa de memoria, a fin de activar en ese preciso instante el primer ciclo de Lectura de la máquina de estados que ya se ha mencionado. Esa señal se implementa más adelante con el nombre de *Pixcero*.

Así entonces, se inicia un proceso que tiene lugar cada vez que la entrada *Reloj* produce una transición bajo-alto. En él, se transmite la información almacenada en las variables *Info_Rojo*, *Info_Verde*, *Info_Azul*, *Info_Horiz_sinc* y *Info_Vert_sinc* directamente a las salidas de señal *Rojo*, *Verde*, *Azul*, *Horiz_sinc* y *Vert_sinc*. Esto evita que la información de dichas salidas se actualice antes de que el cañón electrónico del monitor apunte a otro pixel.

```
Process
Begin
        Wait Until Reloj'EVENT and Reloj = '1';
        Rojo <= Info_Rojo;
        Verde <= Info_Verde;
        Azul <= Info_Azul;
        Horiz_sinc <= Info_Horiz_sinc;
        Vert_sinc <= Info_Vert_sinc;
End process;
```

Para poder entender el algoritmo que define el siguiente proceso, es necesario recordar los tiempos que requiere un monitor VGA para trabajar apropiadamente (v. Capítulo 2). La totalidad del ciclo horizontal requiere un tiempo de 31.77 microsegundos, lo que equivale a 800 pulsos de un reloj de 25.175 MHz. De este modo, se justifica la implementación del contador *H_cont* de 10 bits, y también es por eso que la constante *H_max* establece que el contador *H_cont* debe iniciar su cuenta en '0' y reiniciarse al llegar a 799, incrementándose una sola unidad a cada periodo de la señal *Reloj*. Del mismo modo, se hace evidente la necesidad de disponer con un contador de ciclos horizontales, pues resulta que la duración del ciclo vertical resulta ser exactamente 525 veces mayor que el periodo de un solo ciclo horizontal, esto es, 16.6 ms. Por esa razón, la constante *V_max* adquiere el valor de 524.

Podemos resumir hasta ahora que el contador *H_cont*, al recopilar el número de pulsos que provienen de la entrada *Reloj*, funciona como un simulador de la posición del pixel al que el cañón electrónico del monitor se encuentra apuntando dentro de un renglón determinado, siempre y cuando esa posición se encuentre entre el conteo 0 y 639, ya que de ahí hasta finalizar el ciclo de 800 valores el monitor se encuentra a la espera del tiempo requerido para comenzar a actualizar el renglón siguiente, intervalo durante el cual se produce en un instante específico un pulso de sincronía horizontal. De igual manera el contador *V_cont* simula el número del renglón en el que se encuentra apuntando el cañón electrónico durante los primeros 480 conteos, pues a partir de ese punto se requiere de un pulso de sincronía vertical para posteriormente esperar un tiempo nominal y comenzar la actualización de toda una nueva pantalla. El establecimiento del número de conteos que se deben cumplir para lograr los tiempos nominales que definen la emisión, duración e interrupción de los pulsos de sincronía tanto horizontal como vertical, se detalla en cada una de las secciones de código correspondientes más adelante.

El proceso siguiente se inicia, al igual que el anterior, cada vez que la entrada *Reloj* produce una transición bajo-alto. Comenzamos diseñando una sección de inicialización de variables, aprovechando el hecho de que la variable *Encendido* se encuentra en '0' lógico al energizar el osciloscopio, por lo que las sentencias iniciales se cumplen al primer ciclo de *Reloj*: Los contadores *H_cont* y *V_cont* se inician en 654 y 493 respectivamente. El motivo por el que *V_cont* se inicializa en 493 se debe a que, como se explica más adelante, corresponde a la cuenta donde se emite el pulso activo bajo de sincronización vertical. Al mismo tiempo se inicializan las variables *Leclinea* y *Lecrenglon* en activo bajo, así como la salida *Pixcero*. Finalmente, para no repetir la inicialización, se establece que la variable *Encendido* se quede en '1' lógico durante el resto de la operación del osciloscopio.

```
VIDEO_DISPLAY: Process
Begin

Wait until(Reloj'Event) and (Reloj='1');
If Encendido = '0' Then
 H_cont <= CONV_STD_LOGIC_VECTOR(654,10);
 V_cont <= CONV_STD_LOGIC_VECTOR(493,10);
 Leclinea <= '0';
 Lecrenglon <= '0';
 Pixcero <= '0';
 Encendido <= '1';
```

A partir del segundo periodo de la señal *Reloj* se establece una serie de sentencias condicionales paralelas entre sí, a fin de cambiar el estado de las variables y con ello generar a tiempo las salidas requeridas. La siguiente parte del código establece que a cada periodo de *Reloj*, mientras el contador *H_cont* sea menor que la cuenta máxima de 799, se incremente en una unidad; y que al llegar al límite superior reinicie el conteo desde '0'.

```
Else
       If (H_cont >= H_max) then
         H_cont <= "0000000000";
       Else
         H_cont <= H_cont + "0000000001";
       End if;
```

Se establece la condición de que, cuando ambos contadores *H_cont* y *V_cont* se encuentren respectivamente en los valores 70 y 40 (que son las coordenadas de la primer columna, primer renglón del área de trabajo ya centrada), se emita un pulso activo alto en la salida *Pixcero* con duración de un solo ciclo de *Reloj*.

```
if H_cont = CONV_STD_LOGIC_VECTOR(70,10) and V_cont = CONV_STD_LOGIC_VECTOR(40,10) then
       Pixcero <= '1';
else
       Pixcero <= '0';
end if;
```

Se genera la señal de sincronía horizontal, que es un pulso activo bajo en *Horiz_Sinc* cuando el contador *H_cont* se encuentra entre 659 y 755. La diferencia entre ambos valores es de 96 conteos, lo que produce un ancho de pulso de 3.81 microsegundos aproximadamente. Este pulso de sincronía, como ya se ha mencionado en el marco teórico, es indispensable para indicar al cañón electrónico del monitor el momento en el que debe regresar a trazar el siguiente renglón desde el primer pixel.

```
       If (H_cont <= CONV_STD_LOGIC_VECTOR(755,10)) and
          (H_cont >= CONV_STD_LOGIC_VECTOR(659,10)) Then
              Info_Horiz_sinc <= '0';
       ELSE
              Info_Horiz_sinc <= '1';
       End if;
```

La siguiente sentencia condicional sirve para "recortar" los últimos 11 bits sobrantes en cada renglón, ubicados en la matriz de datos almacenados en la memoria de video, a fin de conformar el área de trabajo con 501 columnas.

```
       if (H_cont >= (conv_std_logic_vector(501,10) + CentroH)) then
              Area <= '0';
       else
              Area <= '1';
       end if;
```

Con respecto a los renglones, el contador *V_cont* incrementa una unidad cada vez que el contador *H_cont* llega al número 699, esto es, unos pocos cientos de nanosegundos antes de que el cañón electrónico comience a trazar el primer pixel del renglón siguiente. Pero cuando *V_cont* se encuentra en el conteo máximo se reinicia en el renglón '0'.

```
If (V_cont >= V_max) and (H_cont >= CONV_STD_LOGIC_VECTOR(699,10)) then
   V_cont <= "0000000000";
Else
   If (H_cont = CONV_STD_LOGIC_VECTOR(699,10)) Then
     V_cont <= V_cont + "0000000001";
   End if;
End if;
```

Al igual que la señal de sincronía horizontal, se debe generar una señal de sincronía vertical *Vert_Sinc* para indicar al cañón electrónico que debe regresar al renglón '0'. Después de completar el renglón 479, se deja transcurrir un tiempo nominal antes de emitir un pulso activo bajo con un ancho de 63.5 microsegundos, lo que se logra dejando pasar únicamente dos conteos de *V_cont*.

```
If (V_cont <= CONV_STD_LOGIC_VECTOR(494,10)) and
   (V_cont >= CONV_STD_LOGIC_VECTOR(493,10)) Then
       Info_Vert_sinc <= '0';
ELSE
       Info_Vert_sinc <= '1';
End if;
```

Mediante dos secuencias condicionales semejantes, se establece cuáles son los pixeles que corresponden a la matriz de datos almacenados en la memoria de video: 512 columnas por 401 renglones. Esto se logra estableciendo que, mientras el cañón electrónico apunte a un pixel que corresponda a la matriz de datos, las señales *Leclinea* y *Lecrenglon* permanecen en '1' lógico. Como ya se ha mencionado, ésta condición provoca la generación de la salida *Pulsolec* necesaria para incrementar la dirección en la VRAM. Es importante recordar que esta matriz de datos debe diferir del área de trabajo de la pantalla, ya que en la fase de diseño se decidió utilizar una matriz compatible con potencia de 2 con el fin de simplificar la conversión entre las coordenadas renglón-columna de un pixel con la dirección que ocupa su información dentro del mapa de la memoria de video.

```
If (H_cont <= CONV_STD_LOGIC_VECTOR(581,10)) and
   (H_cont >= CONV_STD_LOGIC_VECTOR(70,10)) Then
       Leclinea <= '1';
ELSE
       Leclinea <= '0';
End if;

If (V_cont <= CONV_STD_LOGIC_VECTOR(440,10)) and
   (V_cont >= CONV_STD_LOGIC_VECTOR(40,10)) Then
       Lecrenglon <= '1';
ELSE
       Lecrenglon <= '0';
End if;
```

Con las instrucciones hasta ahora expuestas, el módulo controlador de video funciona como interfaz entre el *Controlador de VRAM* y el monitor VGA sin ningún problema. Sin embargo, la información de los pixeles que conforman a los marcos de las 80 divisiones en pantalla, así como la de los pixeles que despliegan la etiqueta de milisegundos por división de la escala de tiempos empleada, decidimos agregarla dentro de éste mismo módulo sin emplear memoria, sino simplemente algoritmos de programación. A su vez, esta decisión nos permitió desplegar en pantalla a los marcos que conforman la rejilla y a la etiqueta de tiempos con dos colores diferentes a los de la gráfica correspondiente a la señal a medir por el osciloscopio.

Para comprender el diseño del algoritmo que da forma a la rejilla, debemos recordar que el área de trabajo se encuentra dividida en una configuración de 10 unidades de ancho por 8 de alto (v. Fig. 3.19). Cada unidad tiene 50 pixeles de longitud, pero la frontera entre cada unidad, es decir el marco, ocupa al menos un pixel de ancho. De esta manera y debido a que para establecer los límites de 10 columnas de divisiones se necesitan 11 fronteras verticales, se requerirían 500 + 11 pixeles en cada renglón; pero con el fin de no agregar espacio que provoque errores de medición, decidimos que el primer pixel de izquierda a derecha de cada división forme parte de la frontera. Como esto sólo nos da una frontera por columna de divisiones, hace falta un pixel más para establecer el marco lateral derecho de toda el área de trabajo. Por lo tanto, el algoritmo para dibujar las fronteras verticales debe

colorear los pixeles que pertenezcan a la serie 0, 50, 100, 150,..., 500. Asimismo, para resaltar las fronteras central, lateral izquierda y lateral derecha del área de trabajo, se agregan los pixeles 1, 249, 251 y 499.

```
if (H_cont = (conv_std_logic_vector(0,10) + CentroH)) or
   (H_cont = (conv_std_logic_vector(1,10) + CentroH)) or
   (H_cont = (conv_std_logic_vector(50,10) + CentroH)) or
   (H_cont = (conv_std_logic_vector(100,10) + CentroH)) or
   (H_cont = (conv_std_logic_vector(150,10) + CentroH)) or
   (H_cont = (conv_std_logic_vector(200,10) + CentroH)) or
   (H_cont = (conv_std_logic_vector(249,10) + CentroH)) or
   (H_cont = (conv_std_logic_vector(250,10) + CentroH)) or
   (H_cont = (conv_std_logic_vector(251,10) + CentroH)) or
   (H_cont = (conv_std_logic_vector(300,10) + CentroH)) or
   (H_cont = (conv_std_logic_vector(350,10) + CentroH)) or
   (H_cont = (conv_std_logic_vector(400,10) + CentroH)) or
   (H_cont = (conv_std_logic_vector(450,10) + CentroH)) or
   (H_cont = (conv_std_logic_vector(499,10) + CentroH)) or
   (H_cont = (conv_std_logic_vector(500,10) + CentroH)) or
```

De un modo semejante se agregan las fronteras horizontales, siguiendo la serie de renglones 0, 50, 100, 150,..., 400; sin olvidar resaltar las fronteras central, superior e inferior del área de trabajo.

```
   (V_cont = (conv_std_logic_vector(0,10) + CentroV)) or
   (V_cont = (conv_std_logic_vector(1,10) + CentroV)) or
   (V_cont = (conv_std_logic_vector(50,10) + CentroV)) or
   (V_cont = (conv_std_logic_vector(100,10) + CentroV)) or
   (V_cont = (conv_std_logic_vector(150,10) + CentroV)) or
   (V_cont = (conv_std_logic_vector(199,10) + CentroV)) or
   (V_cont = (conv_std_logic_vector(200,10) + CentroV)) or
   (V_cont = (conv_std_logic_vector(201,10) + CentroV)) or
   (V_cont = (conv_std_logic_vector(250,10) + CentroV)) or
   (V_cont = (conv_std_logic_vector(300,10) + CentroV)) or
   (V_cont = (conv_std_logic_vector(350,10) + CentroV)) or
   (V_cont = (conv_std_logic_vector(399,10) + CentroV)) or
   (V_cont = (conv_std_logic_vector(400,10) + CentroV)) or
```

Hasta este punto, la rejilla está completa, pero para facilitar la medición por parte del usuario final y siguiendo los estándares de los osciloscopios comerciales, se agrega sobre la frontera horizontal central una serie de 5 subdivisiones por unidad conformadas por cuatro pares de pixeles espaciados de 10 en 10. Del mismo modo, a las mitades del 2° y del 7° renglón de divisiones se introduce una línea punteada con el mismo espaciado:

```
(((H_cont = (conv_std_logic_vector(10,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(20,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(30,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(40,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(60,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(70,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(80,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(90,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(110,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(120,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(130,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(140,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(160,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(170,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(180,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(190,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(210,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(220,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(230,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(240,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(260,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(270,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(280,10) + CentroH)) or
  (H_cont = (conv_std_logic_vector(290,10) + CentroH)) or
```

```
        (H_cont = (conv_std_logic_vector(310,10) + CentroH)) or
        (H_cont = (conv_std_logic_vector(320,10) + CentroH)) or
        (H_cont = (conv_std_logic_vector(330,10) + CentroH)) or
        (H_cont = (conv_std_logic_vector(340,10) + CentroH)) or
        (H_cont = (conv_std_logic_vector(360,10) + CentroH)) or
        (H_cont = (conv_std_logic_vector(370,10) + CentroH)) or
        (H_cont = (conv_std_logic_vector(380,10) + CentroH)) or
        (H_cont = (conv_std_logic_vector(390,10) + CentroH)) or
        (H_cont = (conv_std_logic_vector(410,10) + CentroH)) or
        (H_cont = (conv_std_logic_vector(420,10) + CentroH)) or
        (H_cont = (conv_std_logic_vector(430,10) + CentroH)) or
        (H_cont = (conv_std_logic_vector(440,10) + CentroH)) or
        (H_cont = (conv_std_logic_vector(460,10) + CentroH)) or
        (H_cont = (conv_std_logic_vector(470,10) + CentroH)) or
        (H_cont = (conv_std_logic_vector(480,10) + CentroH)) or
        (H_cont = (conv_std_logic_vector(490,10) + CentroH))) and
       ((V_cont = (conv_std_logic_vector(75,10) + CentroV)) or
           (V_cont = (conv_std_logic_vector(198,10) + CentroV)) or
           (V_cont = (conv_std_logic_vector(202,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(325,10) + CentroV)))) or
```

Para completar la rejilla, se aplica el procedimiento anterior a la división vertical central:

```
      (((V_cont = (conv_std_logic_vector(10,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(20,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(30,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(40,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(60,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(70,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(80,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(90,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(110,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(120,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(130,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(140,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(160,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(170,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(180,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(190,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(210,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(220,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(230,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(240,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(260,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(270,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(280,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(290,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(310,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(320,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(330,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(340,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(360,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(370,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(380,10) + CentroV)) or
        (V_cont = (conv_std_logic_vector(390,10) + CentroV))) and
       ((H_cont = (conv_std_logic_vector(248,10) + CentroH)) or
        (H_cont = (conv_std_logic_vector(252,10) + CentroH)))) then
          Ejes <= '1';
    else
          Ejes <= '0';
    end if;
```

Por otro lado, el algoritmo para conformar la etiqueta de milisegundos por división requiere una serie de instrucciones diferentes dependiendo de la escala de tiempos seleccionada. Decidimos colocar la información de esta etiqueta dentro de la división inferior izquierda del área de trabajo, ocupando dentro de ella los pixeles que se muestran en la Fig. 3.21.

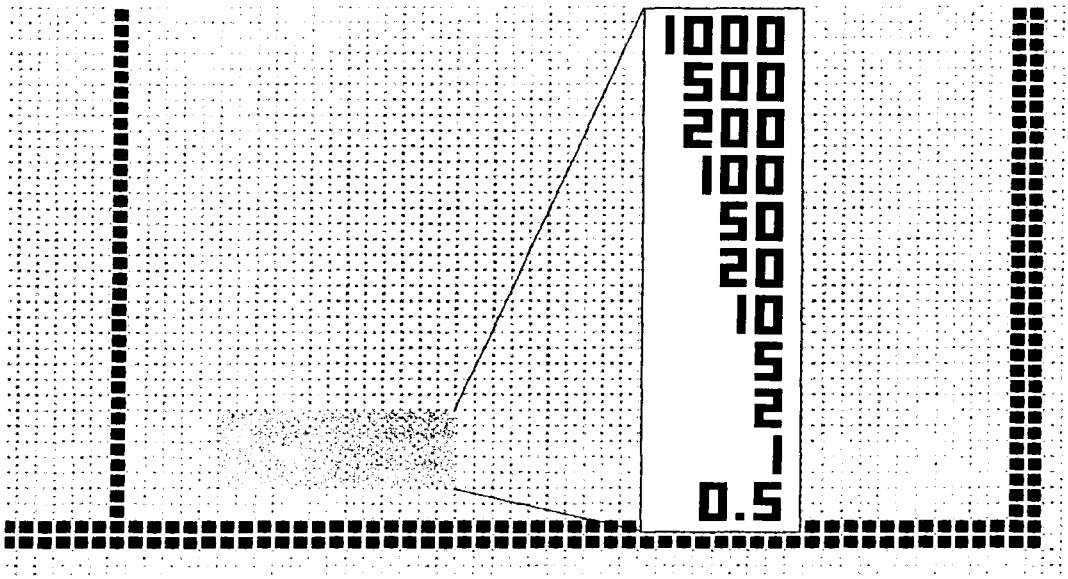


Fig. 3.21 Pixeles dentro del área de trabajo de la pantalla del monitor, destinados a desplegar la información sobre la escala de tiempo elegida por el usuario, en ms por división. El recuadro muestra las once posibles etiquetas y la disposición relativa de los pixeles que las conforman.

La sección de algoritmo para desplegar cada una de las 11 etiquetas de tiempo se muestra como sigue. Para la etiqueta de 500 (ms por división):

```
if Selector = "0001" then
        if ((H_cont = (conv_std_logic_vector(458,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(393,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
            ((H_cont = (conv_std_logic_vector(459,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
            ((H_cont = (conv_std_logic_vector(460,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(395,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or

            ((H_cont = (conv_std_logic_vector(462,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or
            ((H_cont = (conv_std_logic_vector(463,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
            ((H_cont = (conv_std_logic_vector(464,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or

            ((H_cont = (conv_std_logic_vector(466,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or
```

```
            ((H_cont = (conv_std_logic_vector(467,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
            ((H_cont = (conv_std_logic_vector(468,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) then
                Etiq <= '1';
        else
                Etiq <= '0';
        end if;
```

Para la etiqueta de 200 (ms por división):

```
    elsif Selector = "0010" then
        if ((H_cont = (conv_std_logic_vector(458,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(395,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
            ((H_cont = (conv_std_logic_vector(459,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
            ((H_cont = (conv_std_logic_vector(460,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(393,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or

            ((H_cont = (conv_std_logic_vector(462,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or
            ((H_cont = (conv_std_logic_vector(463,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
            ((H_cont = (conv_std_logic_vector(464,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or

            ((H_cont = (conv_std_logic_vector(466,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or
            ((H_cont = (conv_std_logic_vector(467,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
            ((H_cont = (conv_std_logic_vector(468,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) then
                Etiq <= '1';
        else
                Etiq <= '0';
        end if;
```

Para la etiqueta de 100 (ms por división):

```
    elsif Selector = "0011" then
        if ((H_cont = (conv_std_logic_vector(460,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or

            ((H_cont = (conv_std_logic_vector(462,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or
            ((H_cont = (conv_std_logic_vector(463,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
            ((H_cont = (conv_std_logic_vector(464,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or
```

```
            ((H_cont = (conv_std_logic_vector(466,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or
            ((H_cont = (conv_std_logic_vector(467,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
            ((H_cont = (conv_std_logic_vector(468,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) then
                Etiq <= '1';
        else
                Etiq <= '0';
        end if;
```

Para la etiqueta de 50 (ms por división):

```
    elsif Selector = "0100" then
        if ((H_cont = (conv_std_logic_vector(462,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(393,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
            ((H_cont = (conv_std_logic_vector(463,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
            ((H_cont = (conv_std_logic_vector(464,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(395,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or

            ((H_cont = (conv_std_logic_vector(466,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or
            ((H_cont = (conv_std_logic_vector(467,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
            ((H_cont = (conv_std_logic_vector(468,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) then
                Etiq <= '1';
        else
                Etiq <= '0';
        end if;
```

Para la etiqueta de 20 (ms por división):

```
    elsif Selector = "0101" then
        if ((H_cont = (conv_std_logic_vector(462,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(395,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
            ((H_cont = (conv_std_logic_vector(463,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
            ((H_cont = (conv_std_logic_vector(464,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(393,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or

            ((H_cont = (conv_std_logic_vector(466,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or
            ((H_cont = (conv_std_logic_vector(467,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
```

```
            ((H_cont = (conv_std_logic_vector(468,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) then
                Etiq <= '1';
        else
                Etiq <= '0';
        end if;
```

Para la etiqueta de 10 (ms por división):

```
    elsif Selector = "0110" then
        if ((H_cont = (conv_std_logic_vector(464,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or

           ((H_cont = (conv_std_logic_vector(466,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or
           ((H_cont = (conv_std_logic_vector(467,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
           ((H_cont = (conv_std_logic_vector(468,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) then
                Etiq <= '1';
        else
                Etiq <= '0';
        end if;
```

Para la etiqueta de 5 (ms por división):

```
    elsif Selector = "0111" then
        if ((H_cont = (conv_std_logic_vector(466,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(393,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
           ((H_cont = (conv_std_logic_vector(467,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
           ((H_cont = (conv_std_logic_vector(468,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(395,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) then
                Etiq <= '1';
        else
                Etiq <= '0';
        end if;
```

Para la etiqueta de 2 (ms por división):

```
    elsif Selector = "1000" then
        if ((H_cont = (conv_std_logic_vector(466,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(395,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
           ((H_cont = (conv_std_logic_vector(467,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
           ((H_cont = (conv_std_logic_vector(468,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(393,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) then
                Etiq <= '1';
```

```
        else
                Etiq <= '0';
        end if;
```

Para la etiqueta de 1 (ms por división):

```
elsif Selector = "1001" then
        if ((H_cont = (conv_std_logic_vector(468,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) then
                Etiq <= '1';
        else
                Etiq <= '0';
        end if;
```

Para la etiqueta de 0.5 (ms por división):

```
elsif Selector = "1010" then
        if ((H_cont = (conv_std_logic_vector(460,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or
           ((H_cont = (conv_std_logic_vector(461,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
           ((H_cont = (conv_std_logic_vector(462,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or

           ((H_cont = (conv_std_logic_vector(464,10) + CentroH)) and
                (V_cont = (conv_std_logic_vector(396,10) + CentroV))) or

           ((H_cont = (conv_std_logic_vector(466,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(393,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
           ((H_cont = (conv_std_logic_vector(467,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
           ((H_cont = (conv_std_logic_vector(468,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(394,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(395,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) then
                Etiq <= '1';
        else
                Etiq <= '0';
        end if;
```

Para la etiqueta de 1000 (ms por división), que es el valor que toma por defecto:

```
else
        if ((H_cont = (conv_std_logic_vector(456,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or

           ((H_cont = (conv_std_logic_vector(458,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or
           ((H_cont = (conv_std_logic_vector(459,10) + CentroH)) and
                ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
           ((H_cont = (conv_std_logic_vector(460,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or

           ((H_cont = (conv_std_logic_vector(462,10) + CentroH)) and
                (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
```

```
                    (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or
                ((H_cont = (conv_std_logic_vector(463,10) + CentroH)) and
                    ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                    (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
                ((H_cont = (conv_std_logic_vector(464,10) + CentroH)) and
                    (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                    (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or
                ((H_cont = (conv_std_logic_vector(466,10) + CentroH)) and
                    (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                    (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) or
                ((H_cont = (conv_std_logic_vector(467,10) + CentroH)) and
                    ((V_cont = (conv_std_logic_vector(392,10) + CentroV)) or
                    (V_cont = (conv_std_logic_vector(396,10) + CentroV)))) or
                ((H_cont = (conv_std_logic_vector(468,10) + CentroH)) and
                    (V_cont >= (conv_std_logic_vector(392,10) + CentroV)) and
                    (V_cont <= (conv_std_logic_vector(396,10) + CentroV))) then
                    Etiq <= '1';
            else
                    Etiq <= '0';
            end if;
    end if;

End if;
end process VIDEO_DISPLAY;
end arq;
```

En resumen, el módulo controlador de video proporciona las señales requeridas por el monitor VGA que servirá como interfaz de despliegue de información para la señal a medir por el osciloscopio digital. Estas señales, que deben ser proporcionadas en tiempos muy específicos y de manera síncrona, son las señales de color de un pixel en el área de trabajo así como las señales de sincronía horizontal y vertical que indican al cañón electrónico del monitor los momentos en que debe cambiar de renglón y/o actualizar la pantalla. Para ello, requiere una entrada de reloj de 25.175 MHz, así como la información de los pixeles por iluminar que se encuentra almacenada en la memoria de video (VRAM). Asimismo, el módulo controlador de video genera otras tres salidas destinadas a la sincronía con el módulo *Controlador de VRAM.* Adicionalmente, el código VHDL que lo define incluye algoritmos para trazar la rejilla estándar de los osciloscopios comerciales, así como instrucciones para desplegar una etiqueta de información sobre la escala de tiempos seleccionada por el usuario final (lo que requiere una entrada adicional de 4 vías).

Los recursos consumidos por el módulo *Controlador de Video* dentro del dispositivo lógico programable EPF10K20 programado, ocupan un total de 285 compuertas lógicas lo que corresponde a un 24% de la totalidad disponible. También cabe hacer notar que la compilación del módulo realizada por MAX+PLUS II en una PC con procesador Intel Pentium y 192 MB de memoria RAM, requiere un tiempo aproximado de 26 minutos. Tanto el porcentaje de recursos consumidos como el tiempo requerido por la compilación disminuyen drásticamente si suprimimos la información proporcionada por la etiqueta de la escala de tiempos seleccionada.

III.2.2 Módulo Controlador de VRAM.

Como se mencionó en el desarrollo del subtema que trata sobre el módulo *Controlador de Video*, decidimos emplear como memoria de video (VRAM) un dispositivo externo con capacidad para almacenar 2^15 bytes. Entre las opciones disponibles (y las que se encuentran en venta en nuestra ciudad), seleccionamos la memoria SRAM (Static Random Access Memory) 62256A con tecnología CMOS; la cual se ajustó a nuestros requerimientos debido a que algunas de sus características son:

- Completa operación estática y salida de tres estados
- Entradas y salidas compatibles con TTL
- Bajo consumo de potencia
- Disposición de 32786 x 8-bits
- 15 pines de dirección
- Un bus de datos bidireccional de 8 pines

- Pin Output Enable
- Pin Write Enable

El propósito del controlador VRAM es mantener constantemente la información lista para alimentar el módulo controlador de video, así como procesar los datos provenientes del convertidor analógico-digital y del módulo *Base de Tiempos* para almacenarlos en la VRAM, en el espacio que les corresponda. De este modo, nos encontramos con un par de funciones completamente distintas que debíamos implementar para lograr el control total de la memoria de video: en una de ellas debemos almacenar o "escribir" la información recibida por el módulo, que se interpreta como el par de coordenadas del pixel que se debe iluminar en la pantalla para formar parte de su gráfica y al que le corresponde un lugar específico dentro del mapa de memoria de la VRAM; y por otro lado, el módulo debe ser capaz de "leer" la información almacenada en la VRAM desde la primera localidad de memoria hasta la que corresponde al último pixel del área de trabajo del monitor, situado en su esquina inferior derecha. Con lo anterior queremos establecer que la matriz de pixeles que forman una imagen en el monitor y que es trazada de izquierda a derecha renglón por renglón, mantiene una correspondencia uno-a-uno con el conjunto de bits que conforman las primeras 25664 localidades de la memoria de video y que deben ser "leídas" desde el bit menos significativo hasta el más significativo, localidad por localidad. Para expresarlo más claramente, podemos decir que la información de los primeros 8 pixeles que deben ser actualizados por el cañón electrónico del monitor está almacenada en la primera localidad de la VRAM, la de los siguientes 8 en la segunda y así sucesivamente en paquetes de 8 hasta concluir que a todo el primer renglón de 512 pixeles (v. Módulo Controlador de Video) le corresponden las primeras 64 direcciones dentro del mapa de memoria, al segundo las siguientes 64, etc., hasta completar los 401 renglones que tiene el área de trabajo que elegimos implementar.

De ese modo es necesario diseñar una máquina de estados que nos ayude a definir las dos funciones principales que debe realizar el módulo: un *Ciclo de Lectura* de la VRAM y un *Ciclo de Escritura* sobre el mapa de memoria (Fig. 3.22).

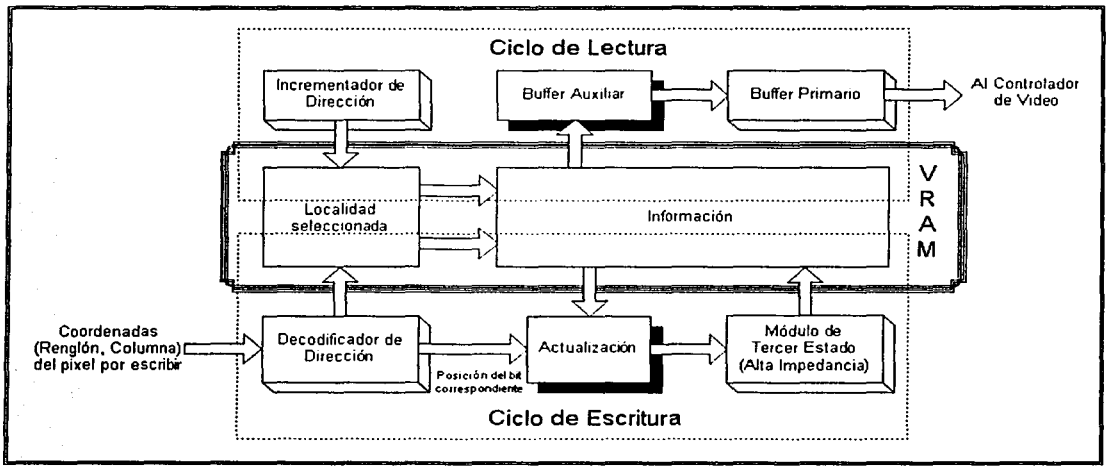


Fig. 3.22 Diagrama de bloques del módulo Controlador de VRAM, mostrando la interacción de los dos ciclos principales con la memoria de video.

Una de las primeras consideraciones que observamos para diseñar el Ciclo de Lectura fue tomar en cuenta la necesidad de programar un módulo que incrementara de uno en uno la dirección de memoria para seguir el orden expuesto en el párrafo anterior, desde la localidad cero hasta la 25663. Lo siguiente sería extraer el byte completo de la localidad de memoria seleccionada y así poderla

enviar al *Controlador de Video* bit por bit. Para lograr esto, diseñamos un proceso que involucra un buffer primario (o principal) y un buffer auxiliar (Fig. 3.23).

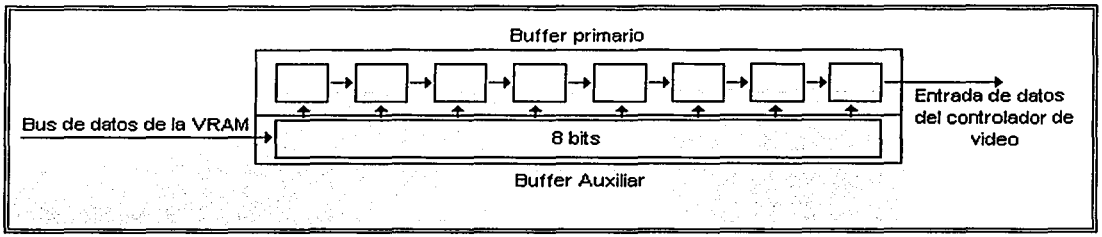


Fig. 3.23 Esquema de transferencia de información desde la memoria de video hasta la entrada de datos del controlador de video, a través de bufferes.

El bit menos significativo del buffer primario es el dato de "encendido" o "apagado" que el controlador de video debe recibir para aquel pixel que el cañón electrónico tiene apuntado dentro del área de trabajo. El buffer recibe un pulso proviniente del controlador de video (salida *Pulsolec*) cada vez que el cañón electrónico cambia de pixel, y en ese momento el buffer primario "dispara" el bit menos significativo al *Controlador de Video* (entrada *Video*) y realiza un corrimiento hacia la derecha con los demás bits. Si el buffer primario se vacía, en vez de recorrerse recibe la información almacenada en el buffer auxiliar, el que a su vez es actualizado por el controlador de la VRAM con datos provinientes de la memoria.

Por otro lado, para diseñar el Ciclo de Escritura partimos de la premisa de que siempre contamos con información proveniente tanto del convertidor A/D (coordenada Renglón) como del módulo *Base de Tiempos* (coordenada Columna) que nos indica indirectamente el bit dentro del mapa de memoria que debe ser actualizado. El primer problema es obtener el algoritmo de conversión que transforme el par de coordenadas a la dirección correspondiente y a la posición que ocupa el bit asociado dentro de la localidad seleccionada. Posteriormente se debe extraer completamente el byte almacenado en esa dirección con el fin de actualizar únicamente el bit que corresponde a la posición obtenida y finalmente devolverlo al lugar de donde se extrajo. Esta última operación plantea un problema adicional: como el bus de 8 pines que el chip SRAM elegido utiliza para permitir la salida de información es el mismo que se emplea para permitir la escritura del mismo (bus bidireccional), se requiere de un módulo que desconecte internamente el buffer empleado para actualizar la información hasta que estemos seguros de que el chip SRAM se encuentra en modo de operación de escritura. En otras palabras, requerimos un componente que sea capaz de emitir alta impedancia (tercer estado). Esto también significa que el chip no es capaz de trabajar en los dos modos de operación simultáneamente, por lo que ambos ciclos no pueden funcionar de modo paralelo, lo que repercute enormemente en la frecuencia de muestreo que caracteriza al osciloscopio diseñado.

Además del par de ciclos cuyas consideraciones han sido analizadas, consideramos conveniente la implementación de una tercera secuencia a la que denominamos *Ciclo de Reset*, cuya función consiste en almacenar en cada una de las localidades por utilizar un byte formado por ceros lógicos. Con esto se logra "limpiar" el mapa de memoria y por consiguiente, el área de trabajo en la pantalla del monitor.

Como los tres ciclos no pueden trabajar en forma paralela, elaboramos un diagrama de flujo que representa a la máquina de estados para un circuito secuencial síncrono (Fig. 3.24) que en resumen, funciona del siguiente modo: en el ciclo de Reset (al que se entra cuando se energiza el dispositivo) implementamos una secuencia para almacenar en todas las localidades de la VRAM el valor H00, a fin de que todos los pixeles se encuentren en estado "apagado"; tras lo cual se da inicio al ciclo de Lectura. Durante el ciclo de Lectura, el controlador extrae la información almacenada en el byte de la VRAM correspondiente al pixel apuntado y la transfiere al buffer auxiliar. Cada vez que el buffer auxiliar se vacía, el controlador comienza un nuevo ciclo de Lectura, empleando la información de la

siguiente localidad. Cuando no se encuentra ocupado en un ciclo de Lectura, el controlador de la VRAM verifica si el controlador de video se encuentra en un periodo "de descanso", es decir, cuando el cañón electrónico no apunta a ningún pixel dentro del área de trabajo. De ser así, el controlador entra al ciclo de Escritura, almacenando información en la VRAM hasta que el periodo "de descanso" llega a su fin. Cabe hacer notar que, si queremos actualizar la información de un solo pixel, debemos "leer" la información completa del byte de la VRAM al que pertenece, encender (o apagar) el bit correspondiente y escribir el byte de vuelta a la memoria.

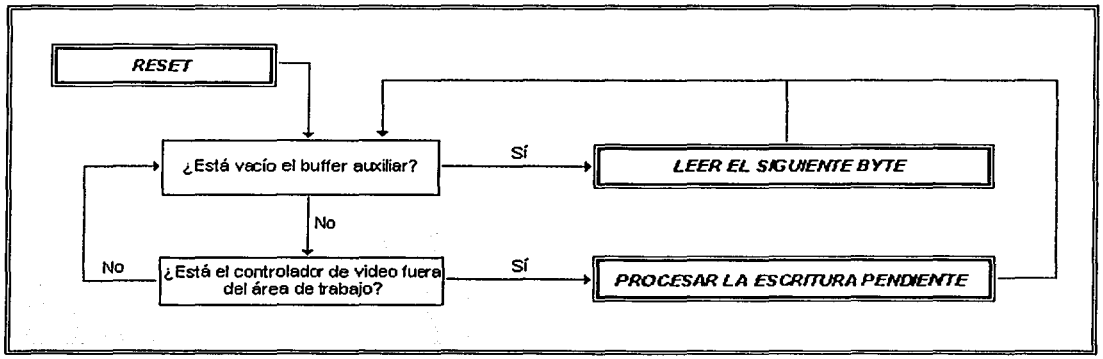


Fig. 3.24 Diagrama de flujo del Controlador de VRAM.

Para implementar el módulo controlador de VRAM, primeramente diseñamos una serie de componentes básicos que se explican a continuación:

Componente *triest*.

Representa un dispositivo con un bus de entrada de 8 datos, un bus de salida semejante y una entrada de un bit llamada *Conectar* como lo muestra la Fig. 3.25.

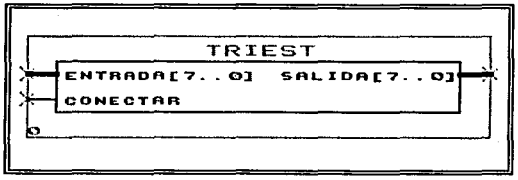


Fig. 3.25 Esquema de entradas y salidas del componente que suministra tercer estado (alta impedancia).

Su función es muy simple: al encontrarse en estado activo alto la señal *Conectar*, el dispositivo entrega a la salida la misma información que tiene a la entrada del bus; en caso contrario, a la salida presenta alta impedancia (tercer estado). Este submódulo es necesario para interactuar adecuadamente con los pines bidireccionales de datos del chip SRAM utilizado, ya que en algunos momentos se emplean como entradas y en otros como salidas de información.

```
-- triest.vhd

library ieee;
use ieee.std_logic_arith.all;
use ieee.std_logic_1164.all;

entity triest is
```

```
        port(Entrada: in std_logic_vector(7 downto 0);
             Conectar: in std_logic;
             Salida: out std_logic_vector(7 downto 0));
end triest;

architecture triarq of triest is
        component tri
                port(a_in, oe: in std_logic;
                     a_out: out std_logic);
        end component;
        begin
        tri0: tri port map(Entrada(0), Conectar, Salida(0));
        tri1: tri port map(Entrada(1), Conectar, Salida(1));
        tri2: tri port map(Entrada(2), Conectar, Salida(2));
        tri3: tri port map(Entrada(3), Conectar, Salida(3));
        tri4: tri port map(Entrada(4), Conectar, Salida(4));
        tri5: tri port map(Entrada(5), Conectar, Salida(5));
        tri6: tri port map(Entrada(6), Conectar, Salida(6));
        tri7: tri port map(Entrada(7), Conectar, Salida(7));
end triarq;
```

Como se puede observar, en el código se emplea a su vez un componente definido en la librería ieee.std_logic_1164 como "tri", el cual funciona del mismo modo que nuestro submódulo, pero empleando un solo bit de entrada y uno de salida.

Componente *decodirec.*

Sirve para traducir las coordenadas (Renglón, Columna) de un pixel dentro del área de trabajo, a su dirección correspondiente de 15 bits dentro del mapa de memoria y a su respectiva posición dentro del byte seleccionado. Es importante recordar del subtema anterior que, como decidimos emplear un área de trabajo compatible con potencia de 2 (ya que tenemos 512 pixeles por renglón), la ecuación para convertir las coordenadas de un pixel en específico en el área de trabajo a una dirección del mapa de memoria es la siguiente:

Dirección = Renglón x (512 / 8) + Ent(Columna / 8)

Donde las variables Dirección, Renglón y Columna se comienzan a contar desde cero; y la operación Ent(Columna / 8) significa obtener la parte entera de la división de la variable Columna entre el ancho del mapa de memoria. La división de (512 / 8) corresponde al número de pixeles que hay en un renglón entre el número de pixeles que caben en un byte; el resultado es 64 bytes por renglón.

Como las variables *Renglon* y *Columna* toman valores menores o iguales a 400 y 511 respectivamente, no sobrepasan la potencia 2^9 = 512, por lo que para definir el tamaño de cada una de ellas se requieren solamente 9 bits. La salida *Direccion* se refiere a la dirección que ocupa el pixel seleccionado dentro del mapa de memoria y por lo tanto requiere 15 bits, pues en la VRAM se almacenan 512 x 401 = 205312 pixeles en 25664 localidades, número que se encuentra entre 2^14 y 2^15. Finalmente se requiere una salida (*Posicion*) que indique una de los 8 posibles lugares dentro de la localidad de memoria que ocupa el bit correspondiente al pixel seleccionado. Ésta última salida debe ser de 3 bits (Fig. 3.26)

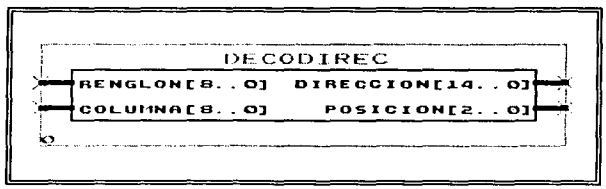


Fig. 3.26 Esquema de entradas y salidas del componente decodificador de dirección.

```
-- decodirec.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity decodirec is
       port(Renglon: in std_logic_vector(8 downto 0);
            Columna: in std_logic_vector(8 downto 0);
            Direccion: out std_logic_vector(14 downto 0);
            Posicion: out std_logic_vector(2 downto 0));
end decodirec;
```

Como se pretende utilizar en su totalidad 512 columnas, se requieren 64 bytes para almacenar un solo renglón (2^6 localidades). Por lo tanto, la arquitectura del módulo requiere los 9 bits de la entrada *Renglon* para formar la parte más significativa de la dirección, mientras que sólo los 6 bits más significativos de la entrada *Columna* se emplean para completarla. Los restantes bits de *Columna* se utilizan directamente para formar la salida *Posicion*, de tal modo, los pixeles situados en las columnas 0, 8, 16, 24, etc., comparten el mismo valor de *Posicion*.

```
architecture decoarq of decodirec is
       signal Coldirec: std_logic_vector(5 downto 0);
begin
       Coldirec <= Columna(8 downto 3);

       Direccion(14 downto 6) <= Renglon;
       Direccion(5 downto 0) <= std_logic_vector(Coldirec);

       Posicion <= Columna(2 downto 0);
end decoarq;
```

Para comprender mejor la explicación expuesta, podemos recurrir a un ejemplo: supongamos que queremos convertir las coordenadas del pixel (6,18) a su respectiva ubicación dentro del mapa de memoria, para lo cual utilizamos la ecuación:

Dirección = Renglón x (512 / 8) + Ent(Columna / 8)

Dirección = 6 x (512 / 8) + Ent(18 / 8)

pero empleando potencias de 2:

$Dirección_2 = 110_2 \times 2^6 + Ent(10010_2 \times 2^{-3})$

$Dirección_2 = 1{,}1000{,}0000_2 + Ent(10.010_2)$

es decir, se recorre el punto decimal tantos lugares como indica la potencia de 2. Por lo tanto:

$Dirección = 1{,}1000{,}0000_2 + 10_2 = 1{,}1000{,}0010_2 = 110,\ 000010_2$

Notamos que los bits más significativos corresponden al número del renglón ($110_2 = 6$), mientras que los menos significativos corresponden a la parte entera de la división del número de columna entre 8 (columnas por byte), esto es, Ent(18 / 8) = 2. Por otro lado, los bits que se perdieron al obtener la parte entera de la división entre 8, se emplean para indicar la posición que ocupa la información del pixel dentro de la localidad de memoria hallada (para nuestro ejemplo, es el bit en la 3ª posición).

Componente *incredirec.*

Necesitamos un módulo que, dada la dirección de la localidad dentro del mapa de memoria, nos dé a la salida la dirección incrementada en una unidad; y en caso de que la dirección proporcionada sea la última que almacena información, reiniciar desde la dirección H00 (Fig. 3.27)

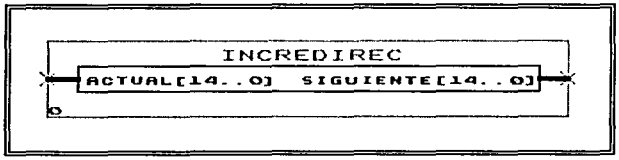


Fig. 3.27 Esquema de entrada y salida del componente que incrementa la dirección de lectura.

```
-- incredirec.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity incredirec is
        port(Actual: in std_logic_vector(14 downto 0);
             Siguiente: out std_logic_vector(14 downto 0));
end incredirec;
```

La arquitectura del componente es muy simple: como se explicó en el desarrollo del componente *decodirec*, los 9 bits más significativos de la dirección están directamente relacionados con la coordenada del renglón en el que se ubica el pixel dentro del área de trabajo. Por esa razón se define la constante *Maxren* como el valor máximo que puede tener un renglón: en nuestro caso, 400. Posteriormente se define un proceso que se inicia al cambiar la dirección en la entrada *Actual* del componente, que lo único que hace es mostrar en la salida esa misma dirección incrementada en '1', a menos que el la dirección proporcionada corresponda al último byte del renglón 400, en cuyo caso la salida *Siguiente* apunta a la primera dirección del mapa de memoria.

```
architecture increarq of incredirec is
        constant Maxren: std_logic_vector(8 downto 0):=
                                conv_std_logic_vector(400, 9);
begin
        process(Actual) begin
                if Actual(5 downto 0) = "111111" and Actual(14 downto 6) = Maxren then
                        Siguiente <= "000000000000000";
                else
                        Siguiente <= Actual + "000000000000001";
                end if;
        end process;
end increarq;
```

Componente *bufferpri.*

Como ya se mencionó, el controlador de VRAM debe proporcionar al *Controlador de Video* la información sobre el pixel que se está actualizando en pantalla. Esa información se encuentra en lo que llamamos buffer primario, implementado en éste componente como variable interna de la arquitectura con el nombre de *Prim.* El controlador de video proporciona una señal de reloj (*Pulsolec*) a 25.175 MHz, pero únicamente cuando el cañón electrónico del monitor envía su haz de electrones a un pixel correspondiente a un bit dentro de la matriz de datos del mapa de memoria. De este modo, por cada renglón se reciben 512 pulsos a la entrada *Pulsolec*, los que sirven para que éste componente le devuelva la información del buffer primario bit por bit a esa frecuencia, hasta quedar vacío. Cuando esto sucede, simplemente se toma la información presente en la entrada denominada

Auxiliar, proviniente precisamente del buffer auxiliar, para llenar nuevamente el buffer primario (Fig. 3.28.

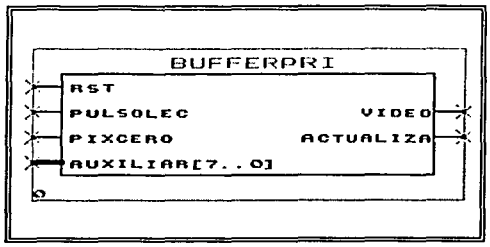


Fig. 3.28 Esquema de entradas y salidas del componente que controla al buffer primario.

```
-- bufferpri.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity bufferpri is
        port(Rst, Pulsolec, Pixcero: in std_logic;
             Auxiliar: in std_logic_vector(7 downto 0);
             Video, Actualiza: out std_logic);
end bufferpri;
```

La arquitectura del componente comienza por definir 4 variables internas: el buffer primario *Prim*, un contador de 3 bits, y dos señales que se emplean como banderas.

```
architecture bufarq of bufferpri is
        signal Prim: std_logic_vector(7 downto 0);
        signal Recorre: unsigned(2 downto 0);
        signal Vacio, Espera: std_logic;
```

La señal de salida *Video* recibe directamente el bit menos significativo del buffer primario, así como la salida *Actualiza* recibe la información de la bandera *Vacio*. Se establece un proceso que se activa cuando hay un cambio en la entrada *Pulsolec* o en *Rst*. Dándole prioridad a la entrada *Rst*, si se encuentra en '1' lógico, se envían '0's a las salidas *Video* y *Actualiza*. Al mismo tiempo se pone en '1' la bandera *Espera*, con el fin de prevenir una falta de sincronización entre los módulos controladores de video y VRAM.

```
begin
        Video <= Prim(0);
        Actualiza <= Vacio;

        process(Pulsolec, Rst) begin
                if Rst = '1' then
                        Prim(0) <= '0';
                        Vacio <= '0';
                        Espera <= '1';
```

Seguidamente se establece dentro de la secuencia que si *Rst* no es '1', a cada transición bajo-alto de la entrada *Pulsolec* se cumpla la serie de eventos que a continuación se detallan: se incrementa en una unidad el contador interno *Recorre*, se desplazan todos los bits del buffer primario una posición hacia la derecha mientras que el bit más significativo se vacía. En caso de que el contador llegue al valor máximo (111_2), la salida *Actualiza* cambia de estado y el buffer primario se renueva con los valores presentes en la entrada *Auxiliar*.

```
                elsif Pulsolec = '1' and Pulsolec'event then
                        if Espera = '0' then
```

```
                        if std_logic_vector(Recorre) = "111" then
                                Vacio <= not Vacio;
                                Prim <= Auxiliar;
                        else
                                Prim(6 downto 0) <= Prim(7 downto 1);
                                Prim(7) <= '0';
                        end if;
                Recorre <= Recorre + 1;
```

Lo anterior sólo se cumple con la condición de que la bandera *Espera* se encuentre en '0' lógico, pero si nos remitimos al estado que *Rst* establece cuando es '1' lógico, recordamos que *Espera* se mantiene en nivel activo alto hasta que se presenta el siguiente condicional: si en la entrada *Pixcero* se mantiene un '1' lógico durante al menos un ciclo de reloj, no sólo se da inicio al proceso descrito en el párrafo anterior, sino que se inicializan las variables que intervendrán en él. Para entender esto, es necesario mencionar que la entrada *Pixcero* recibe su información de la salida homónima del controlador de video, la cual emite un pulso activo alto cada vez que el cañón electrónico del monitor apunta al primer pixel del primer renglón dentro del área de trabajo (pixel (0,0)). De ese modo, la información de ese pixel se considera perdida, ya que al siguiente ciclo de reloj el cañón necesita la información del segundo pixel. Por esa razón, el contador *Recorre* no debe iniciar en cero sino en "001" y el buffer primario debe estar recorrido una posición hacia la derecha. Al mismo tiempo, la salida *Actualiza* se establece en estado '1' lógico.

```
                elsif Pixcero = '1' then
                        Espera <= '0';
                        Vacio <= '1';
                        Recorre <= "001";
                        Prim(6 downto 0) <= Auxiliar(7 downto 1);
                        Prim(7) <= '0';
                end if;
        end if;
    end process;
end bufarq;
```

Una vez explicados los componentes que lo integran, procederemos a la explicación del funcionamiento y desarrollo del código correspondiente al módulo controlador de la VRAM. Como se muestra en el código y en la Fig. 3.29 el módulo consta de 8 entradas, 5 salidas y un bus bidireccional:

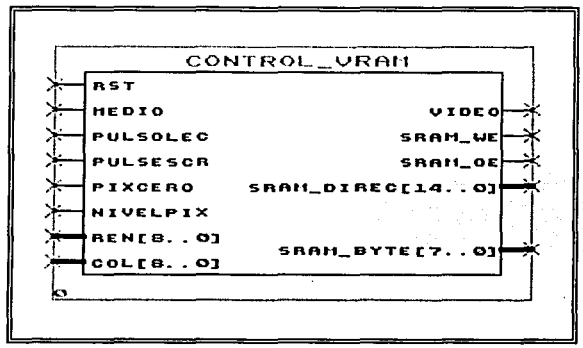


Fig. 3.29 Esquema de entradas y salidas del módulo Controlador de VRAM, junto con el puerto bidireccional *sram_byte*.

```
-- control_vram.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

```
entity control_vram is
        port(Rst, Medio, Pulsolec, Pulsescr, Pixcero, Nivelpix: in std_logic;
             Ren: in std_logic_vector(8 downto 0);
             Col: in std_logic_vector(8 downto 0);
             Video: out std_logic;
             sram_we, sram_oe: out std_logic;
             sram_direc: out std_logic_vector(14 downto 0);
             sram_byte: inout std_logic_vector(7 downto 0));
end control_vram;
```

Se definen las variables internas de la arquitectura y seguidamente los componentes *triest*, *bufferpri*, *decodirec* e *incredirec*. Se establece la constante *Direcmax* con el valor 25663 para indicar la dirección máxima dentro del mapa de memoria VRAM, ya que, como cada renglón requiere 64 bytes y necesitamos 401 renglones, el producto nos da ese valor.

```
architecture vramarq of control_vram is
        signal oe, we, Inhabilbuf, Actuabuf, Bandebuf, Enviaescr: std_logic;
        signal Estado: std_logic_vector(2 downto 0);
        signal Direcactual, Almacedirec, Direcsig: std_logic_vector(14 downto 0);
        signal Bufaux, Byteant, Bytenue: std_logic_vector(7 downto 0);
        signal Posdeco, Almacepos: std_logic_vector(2 downto 0);
        signal Direcdeco: std_logic_vector(14 downto 0);
        component triest
                port(Entrada: in std_logic_vector(7 downto 0);
                     Conectar: in std_logic;
                     Salida: out std_logic_vector(7 downto 0));
        end component;
        component bufferpri
                port(Rst, Pulsolec, Pixcero: in std_logic;
                     Auxiliar: in std_logic_vector(7 downto 0);
                     Video, Actualiza: out std_logic);
        end component;
        component decodirec
                port(Renglon: in std_logic_vector(8 downto 0);
                     Columna: in std_logic_vector(8 downto 0);
                     Direccion: out std_logic_vector(14 downto 0);
                     Posicion: out std_logic_vector(2 downto 0));
        end component;
        component incredirec
                port(Actual: in std_logic_vector(14 downto 0);
                     Siguiente: out std_logic_vector(14 downto 0));
        end component;
        constant Direcmax: std_logic_vector(14 downto 0):= conv_std_logic_vector(25663, 15);
```

Se establecen las variables internas que intervienen en cada uno de los componentes. Con base en el desarrollo anteriormente expuesto, la explicación de cada una de ellas queda como sigue:

- En *triest*, *Bytenue* representa la entrada de 8 bits, que se obtiene en la salida *sram_byte* siempre y cuando *Enviaescr* se encuentre en '1' lógico.
- En *bufferpri*, la variable *Inhabilbuf* representa la entrada de *Rst*; *Pulsolec* es el reloj a 25.175 MHz. con el cual se actualiza la información de cada pixel, dentro del área de trabajo en el monitor; *Pixcero* es directamente una de las señales de entrada de todo el módulo la cual, como se ha explicado, sincroniza el inicio de suministro de información sobre los pixeles en el área de trabajo con el momento en el que el primer pixel del primer renglón es actualizado por el cañón electrónico del monitor; *Bufaux* representa al buffer auxiliar, encargado de alimentar al buffer primario cuando éste se vacía; *Video* contiene la información de "encendido" o "apagado" del pixel que está siendo actualizado, y es directamente una de las señales de salida de todo el módulo; finalmente *Actuabuf* es la señal que indica cambiando de nivel lógico, el momento en el que el buffer primario se ha vaciado por completo.
- En *decodirec*, *Ren* y *Col* son directamente entradas de todo el módulo, que respectivamente representan las coordenadas renglón y columna del pixel que se desea encender o apagar;

Direcdeco y *Posdeco* son respectivamente el valor de dirección dentro del mapa de memoria y la posición que tiene dentro de esa localidad el bit que contendrá la información de dicho pixel.

- En *incredirec*, *Direcactual* es la variable que almacena la dirección actual de la memoria, mientras que *Direcsig* es el valor de la siguiente dirección.

```
begin
        Altaimp: triest port map (Bytenue, Enviaescr, sram_byte);
        Primario: bufferpri port map (Inhabilbuf, Pulsolec, Pixcero, Bufaux, Video,
                                      Actuabuf);
        Decoescr: decodirec port map (Ren, Col, Direcdeco, Posdeco);
        Incremento: incredirec port map (Direcactual, Direcsig);
```

Se establecen las equivalencias entre las variables internas con algunas de las salidas del módulo: la salida *sram_direc* recibe su información de la variable interna *Direcactual* e irá directamente conectada a los pines de dirección del chip SRAM, al igual que las salidas *sram_oe* y *sram_we*, pines que respectivamente controlan los modos de Salida y Entrada en los que puede trabajar el chip SRAM para la información que contiene. Internamente, las variables relacionadas simplemente se llaman *oe* y *we*.

```
        sram_direc <= std_logic_vector(Direcactual);
        sram_oe <= oe;
        sram_we <= we;
```

A continuación se da inicio al proceso para implementar la máquina de estados con sus ciclos Reset, Lectura y Escritura. Este proceso tiene lugar cada ciclo de la entrada general *Medio*, la cual debe tener la mitad de la frecuencia que el reloj del módulo controlador de video, es decir, 12.5875 MHz. El motivo por el cuál se debe emplear tal frecuencia está dado por una de las limitaciones del chip SRAM elegido: una velocidad típica de operación de 70 ns, lo que nos da un equivalente en frecuencia aproximado a 14.28 MHz. Significa que con un valor de frecuencia menor o igual al obtenido, se puede garantizar el correcto funcionamiento de la memoria. Así entonces, la forma más sencilla en la que podemos cumplir esa condición es implementando un divisor de frecuencia compatible con lógica binaria, esto es, simplemente entre 2.

Durante el funcionamiento de la máquina de estados se le da precedencia a la señal presente en la entrada *Rst*, de tal modo que si está en '1' lógico (que es el estado en el que se encuentra cuando se energiza el dispositivo) y sin importar el estado que tenía en el ciclo de reloj anterior, se inicializa la serie de variables que a continuación se detalla:

- La variable interna *Estado*, que puede tener uno de 8 valores, se inicia e ceros para dar inicio a la máquina de estados en el ciclo Reset.
- La dirección *Direcactual* apunta a la localidad H00 en el mapa de memoria, mientras que las salidas *sram_we* y *sram_oe* se mantienen en estado activo alto ya que como estas señales son conectadas al chip SRAM en los pines Write Enable y Output Enable (que son activos bajos), se inhabilitan tanto la escritura como la lectura de la memoria.
- La variable *Enviaescr* se inicializa en '0' lógico. Esto significa que se obtiene alta impedancia a la salida del componente triest, o dicho con otras palabras, se desconecta el bus bidireccional de datos de la memoria.
- Se inicia en '1' lógico la variable *Inhabilbuf*, es decir, la entrada de *Rst* del componente *bufferpri*, a fin de evitar salida de información hacia el monitor.
- Se establece en '0' lógico la bandera *Bandebuf*.
- Se inicializa la variable *Bytenue* en ceros, esto es, la entrada del componente *triest*.

```
process(Rst, Medio) begin
              if Rst = '1' then
                      Estado <= "000";
                      Direcactual <= "0000000000000000";
                      we <= '1';
```

```
oe <= '1';
Enviaescr <= '0';
Inhabilbuf <= '1';
Bandebuf <= '0';
Bytenue <= "00000000";
```

Una vez que la entrada *Rst* cambia de nivel lógico, a cada pulso de la entrada *Medio* comienza el cambio de estados que se ilustra en la Fig. 3.30, donde los estados "000" y "001" comprenden el ciclo Reset, el estado "011" es el ciclo de Lectura y los estados "100", "101", "110" y "111" conforman el ciclo de Escritura. El estado "010" se explica más adelante como un ciclo donde se revisa el estado del buffer primario.

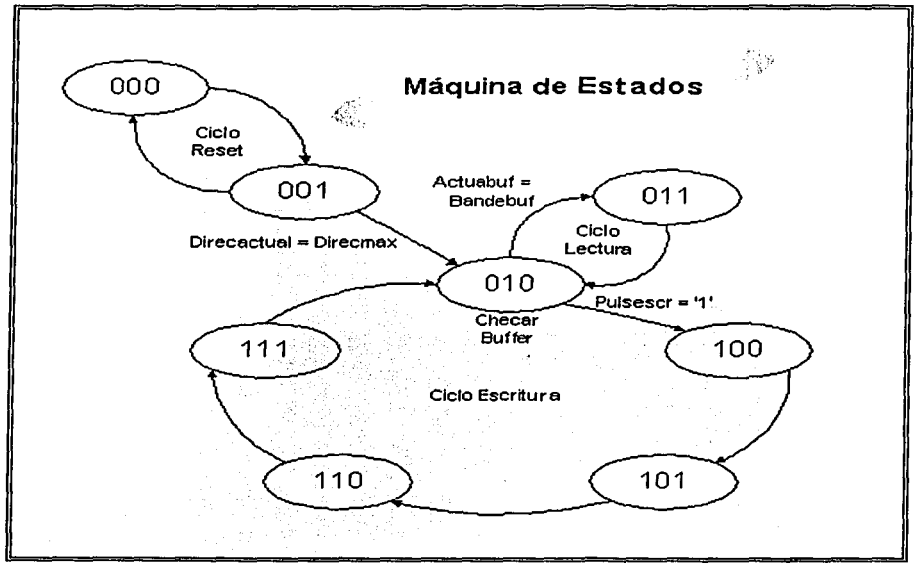


Fig. 3.30 Máquina de Estados que muestra los ciclos Reset, Lectura y Escritura del módulo *Controlador de VRAM.*

Ciclo Reset.

En el ciclo Reset, se puede escribir en la VRAM enviando un pulso activo bajo a la salida *sram_we*, y un pulso activo alto a *Enviaescr* para habilitar el módulo *triest*. Como la entrada de *triest* es *Bytenue* y está en ceros (debido a las condiciones iniciales), la salida *sram_byte* recibe esa información y con ello vacía la localidad de memoria H00. Al siguiente pulso de reloj se inhabilita la escritura, se desconecta la salida y se incrementa la dirección. Este ciclo se realiza para las 25664 direcciones empleadas, durante un tiempo de 4.078 ms (25664 / 12.5875 MHz x 2 periodos). Una vez alcanzada la dirección máxima, se procede al estado que revisa el buffer.

```
elsif Medio = '1' and Medio'event then
                    case Estado is
                        when "000" =>  we <= '0';
                                       Enviaescr <= '1';
                                       Estado <= Estado + "001";
                        when "001" =>  we <= '1';
                                       Enviaescr <= '0';
                                       Direcactual <= Direcsig;
                                       if Direcactual = Direcmax then
```

```
                Estado <= "010";
        else
                Estado <= "000";
        end if;
```

Ciclo de revisión del buffer primario.

Para explicar la función de éste ciclo, debemos hacer notar que *Actuabuf* es una variable que proviene directamente del submódulo *bufferpri*, y que cambia de nivel lógico cada 8 periodos de *Pulsolec*, a menos que la variable *Inhabilbuf* esté en '1' lógico. Como *Inhabilbuf* se inicializa precisamente en '1', la variable *Actuabuf* está en '0' (v. Componente *bufferpri* en éste mismo subtema) y por lo tanto es equivalente al estado inicial de *Bandebuf*. Esta condición provocará la entrada al ciclo de Lectura en el siguiente pulso de la entrada de reloj *Medio*, por lo que se deben habilitar como salidas los pines de la SRAM para introducir información al buffer auxiliar *Bufaux*. También se debe modificar el nivel de *Bandebuf* para que la siguiente vez que se regrese al estado de revisión del buffer primario, no nos conduzca inmediatamente de nuevo al ciclo de lectura, sino que se espere hasta que se cumplan las siguientes tres condiciones:

1. La variable interna *Inhabilbuf* en activo bajo, lo que se logra al pasar al menos una vez por el ciclo de lectura.
2. Un pulso positivo en la entrada *Pixcero* para iniciar la función del componente *bufferpri*, pulso que recordamos proviene del controlador de video cuando se encuentra apuntado al pixel (0,0). Es con ésta condición que se asegura la sincronía del controlador de VRAM con la información que debe desplegar el controlador de video en la pantalla.
3. Un cambio de nivel lógico de la variable interna *Actuabuf*, producido por el componente *bufferpri* cuando se vacía el buffer primario y necesita nuevos datos del buffer auxiliar *Bufaux*, el que a su vez obtendrá información de la VRAM.

Las tres condiciones mencionadas deben cumplirse en estricto orden, aunque cabe mencionar que debido al diseño del programa, una vez que se cumplen las dos primeras ya no es posible detener la operación del módulo *bufferpri* (a menos que se reciba un pulso activo alto en la entrada *Rst*), por lo que solo se espera la tercera condición para que ocurra un cambio de estado e ingresar al ciclo de Lectura. Por otra parte, si nos encontramos en un tiempo "de descanso" (cuando el cañón electrónico del monitor debe hacer un cambio entre renglones o entre pantallas), el controlador de video mantiene un pulso activo alto en la entrada *Pulsescr*, el reloj *Pulsolec* se detiene, y ya no es necesario llenar el buffer primario con datos de la VRAM sino hasta que termina dicho descanso. Por lo tanto, cuando *Pulsescr* = '1' y si las variables internas *Bandebuf* y *Actuabuf* son diferentes, el módulo se prepara para entrar al ciclo de Escritura, por lo que se almacena la dirección actual de lectura en la variable *Almacedirec*, se apunta a una dirección del mapa de memoria utilizando la variable interna *Direcdeco* proveniente del componente *decodirec*, se almacena la posición del pixel que queremos escribir en *Almacepos*, se provoca la entrada al ciclo de escritura y se habilita la lectura de la dirección seleccionada en la VRAM. Esto último tiene el fin de almacenar el byte completo como variable interna y sustituir únicamente el bit que queremos iluminar (o borrar). Cabe hacer notar que, dadas las condiciones anteriores, nos encontramos en el preciso momento en el que se realiza un muestreo de las entradas *Ren* y *Col*, lo cual, si analizamos la máquina de estados podemos observar que no volverá a ocurrir al menos durante 5 cambios de estado adicionales. Esto significa que, encontrándonos en un periodo "de descanso", el tiempo de muestreo es de aproximadamente 0.4 microsegundos (5 / 12.5875 MHz). Más aún: en el peor de los casos el tiempo que hay que esperar incluye 64 ciclos de Lectura equivalentes a 512 periodos de la entrada *Pulsolec* a 25.175 MHz, lo que limita el tiempo de muestreo a 20.73 microsegundos (48.228 KHz).

```
when "010" =>  if Actuabuf = Bandebuf then
                        Bandebuf <= not Bandebuf;
                        oe <= '0';
                        Estado <= "011";
                elsif Pulsescr = '1' then
```

```
                Almacedirec <= Direcactual;
                Direcactual <= Direcdeco;
                Almacepos <= Posdeco;
                Estado <= "100";
                oe <= '0';
        end if;
```

Ciclo de Lectura.

Consiste en un solo periodo de reloj al cual normalmente se accede sólo cuando el cañón electrónico está escribiendo en una línea de la pantalla dentro del área de trabajo. El dato presente en la dirección actual de lectura, que al terminar el ciclo Reset se encuentra al inicio del mapa de memoria, se transfiere en su totalidad al buffer auxiliar *Bufaux*, tras lo cual se inhabilita el modo de salida del chip SRAM enviando un pulso activo alto a la salida *sram_oe* y se emplea el componente *incredirec* para incrementar el valor de la dirección con el fin de dejarla lista para el siguiente ciclo de lectura. Los siguientes 3 periodos de reloj, la máquina de estados permanecerá en el estado que revisa el buffer, esto es, hasta que se vacíe el buffer primario y *Actuabuf* cambie de nivel lógico, o bien, que se entre a un periodo "de descanso". Adicionalmente, a la variable interna *Inhabilbuf* se le asigna un valor activo bajo con el fin de proporcionar una de las condiciones necesarias para hacer funcionar al componente *bufferpri*.

```
when "011" =>  Bufaux <= sram_byte;
               Direcactual <= Direcsig;
               Inhabilbuf <= '0';
               oe <= '1';
               Estado <= "010";
```

Ciclo de Escritura.

Como ya se ha mencionado, éste ciclo que contiene cuatro estados sólo puede ocurrir si nos encontramos en un intervalo "de descanso", es decir que se aprovechan los ciclos de reloj en los que el cañón electrónico del monitor no requiere la información de la VRAM por encontrarse fuera del área de trabajo. Esa condición se manifiesta por la activación de la entrada *Pulsescr* en '1' lógico. La función general de este ciclo de Escritura consiste en almacenar en un bit específico dentro del mapa de memoria la información que proviene de la entrada *Nivelpix*, esto es, actualizar la información que el controlador de la VRAM obtendrá de la memoria en el ciclo de lectura para el pixel cuyas coordenadas están dadas por las entradas *Ren* y *Col*.

En el ciclo que revisa el estado del buffer, cuando se detecta que *Pulsescr* = '1' se preparan las condiciones para ingresar al ciclo de Escritura: se almacena la dirección actual del ciclo de Lectura, se habilita el chip SRAM en modo de salida y se le indica al mismo chip cuál es la localidad donde se encuentra el pixel que deseamos actualizar mediante la conversión proporcionada por el componente *decodirec*. Además, con la misma conversión mencionada, se almacena en la variable interna *Almacepos* el lugar que ocupa la información del pixel dentro de la localidad. Con la dirección de escritura apuntando a la SRAM, durante el estado "100" se obtiene la información de todo el byte en un ciclo de reloj y se almacena en la variable interna llamada *Byteant*.

```
when "100" =>  Byteant <= sram_byte;
               Estado <= Estado + "001";
```

Al siguiente ciclo de reloj, se inhabilita la lectura del chip SRAM y se pregunta por la posición del pixel en el byte, almacenada en la variable interna *Almacepos* y dependiendo del caso, se sobrescribe en la posición correspondiente de la variable *Byteant* la información de *Nivelpix* (prendido o apagado).

```
when "101" =>  oe <= '1';
               case Almacepos is
                      when "000" => Byteant(0) <= Nivelpix;
                      when "001" => Byteant(1) <= Nivelpix;
                      when "010" => Byteant(2) <= Nivelpix;
```

```
                    when "011" => Byteant(3) <= Nivelpix;
                    when "100" => Byteant(4) <= Nivelpix;
                    when "101" => Byteant(5) <= Nivelpix;
                    when "110" => Byteant(6) <= Nivelpix;
                    when others => Byteant(7) <= Nivelpix;
                end case;
                Estado <= "110";
```

Al ciclo de reloj siguiente se habilita el chip SRAM en modo de escritura, la variable *Byteant* actualizada se asigna a la entrada del componente triest, y se dispara la salida a la misma localidad del chip SRAM que se leyó para sobrescribirla.

```
            when "110" =>  we <= '0';
                           Bytenue <= Byteant;
                           Enviaescr <= '1';
                           Estado <= Estado + "001";
```

Finalmente, se espera otro ciclo de reloj para desconectar el componente *triest*, se inhabilita el modo de escritura de la memoria y se devuelve la dirección de lectura almacenada al apuntador *Direcactual*. Por último se provoca el ingreso al ciclo que revisa el buffer primario.

```
            when others => we <= '1';
                           Enviaescr <= '0';
                           Direcactual <= Almacedirec;
                           Estado <= "010";
            end case;
        end if;
    end process;
end vramarq;
```

El hecho de que el ciclo de Escritura necesite 5 estados para completarse (contando el ciclo que revisa el buffer primario) imposibilita la opción de intercalarlo con los ciclos de Lectura que se presentan a lo largo del periodo donde se actualiza la información de un solo renglón dentro del área de trabajo de la pantalla, ya que el número de ciclos de reloj que es necesario esperar para requerir un nuevo ciclo de Escritura equivale a 3 cambios de estado únicamente: cuando la variable interna *Actuabuf* que proviene del componente *bufferpri* cambia de nivel lógico y suponiendo que la máquina de estados se encuentra en el ciclo que revisa el buffer, debemos esperar otro periodo de la entrada *Medio* a fin de reconocer tal condición. Una vez reconocida ingresamos al ciclo de Lectura en el siguiente periodo. Actualizado el buffer auxiliar, se procede nuevamente al estado que revisa el buffer y suponiendo que *Pulsescr* = '0' no ingresaremos al ciclo de Escritura ni al de Lectura, puesto que hasta el momento sólo se llevan 6 conteos de la entrada *Pulsolec*. Finalmente se produce otro periodo de *Medio* que al tener la mitad de la frecuencia de *Pulsolec* indica que se han completado los 8 conteos que necesita el componente *bufferpri* para realizar un nuevo cambio de nivel lógico en la variable *Actuabuf*, con lo que se cierra el círculo. Descrito de ésta manera, podemos notar que la máquina de estados permanece 3 periodos en el estado que revisa el buffer primario y sólo 1 en el estado de Lectura.

En conclusión, el controlador de VRAM es un módulo destinado para interactuar con la memoria de video que por problemas de espacio fue necesario implementar en un dispositivo SRAM externo al CPLD. Sus funciones principales son:

- Limpiar la VRAM mediante un ciclo denominado Reset.
- Suministrar la información requerida por el controlador de video para iluminar o apagar los pixeles de la pantalla del monitor que conformarán la señal a medir por el usuario final. Dicha información debe ser "leída" de la VRAM por el módulo en un ciclo denominado Lectura.
- Aprovechar los lapsos de tiempo en los que no se encuentra en ninguno de los dos ciclos anteriores para actualizar la información contenida en la VRAM y por ende, en la pantalla del monitor. Lo anterior define un nuevo ciclo denominado Escritura.

Es importante hacer notar que los tres ciclos trabajan en forma serial, o dicho con otras palabras, ningún par de ciclos puede funcionar simultáneamente. Esto implica que el ciclo de Escritura solo

puede realizarse en los intervalos de tiempo en los que el controlador no está inmerso en el ciclo de Lectura, el cuál tiene una duración aproximada de 20 x 10^{-6} segundos (512 pixeles / 25.175 MHz, en cada renglón). Si durante ese tiempo se requiere actualizar la información de algunos pixeles por parte del convertidor analógico digital y/o por el módulo Base de Tiempos, ello resulta imposible y dicha información se pierde. Por ese motivo, la frecuencia máxima teórica a la que el osciloscopio puede obtener muestras de una señal sin perder información está dada por el recíproco del tiempo de Lectura calculado: 50 KHz.

Los recursos consumidos por el módulo *Controlador de VRAM* dentro del dispositivo lógico programable EPF10K20, ocupan un total de 229 compuertas lógicas lo que corresponde a un 19% de la totalidad disponible. También cabe hacer notar que la compilación del módulo realizada por MAX+PLUS II en una PC con procesador Intel Pentium y 192 MB de memoria RAM, requiere un tiempo menor a 2 minutos.

III.2.3 Módulo Base de Tiempos

En el desarrollo del subtema anterior, se explicó que el módulo controlador de VRAM requiere dos entradas especiales, cada una de 9 bits denominadas *Ren* y *Col*, las cuales indican al controlador de VRAM las coordenadas (Renglón, Columna) del pixel que se desea actualizar. La entrada *Ren* obtiene directamente la información del convertidor analógico digital, lo que se traduce en la pantalla como un incremento o decremento de la altura de los valores de muestra de la señal a medir y que es proporcional al voltaje de ésta.

Por otro lado, la señal *Col* se debe utilizar para incrementar la coordenada Columna del pixel correspondiente a la muestra tomada en intervalos regulares de tiempo, a fin de realizar un barrido horizontal de la señal a través del área de trabajo. El control del barrido horizontal así como la generación de una señal para "limpiar" el área de trabajo una vez que ha concluido el trazado de izquierda a derecha, son las funciones principales del módulo denominado *Base de Tiempos*. Adicionalmente, ante la falta de una sección de disparo que estabilice las señales repetitivas, se implementó en éste mismo módulo un algoritmo para desplegar por tiempo ilimitado a conveniencia del usuario la señal almacenada en la VRAM hasta el momento en que termina el trazado. En otras palabras, se obtiene una instantánea de la señal o un "congelamiento" para su medición detallada.

Los osciloscopios comerciales manejan una base de tiempos que se encuentra dentro del rango de 200 ms por división hasta 0.5 microsegundos por división y en algunos casos hasta el orden de unidades de nanosegundo, pasando por las potencias negativas de 10 multiplicadas por los valores 5, 2 y 1. Sabiendo que las limitantes de nuestro osciloscopio lo restringen a bajas frecuencias, decidimos comenzar la base de tiempos con una escala de 1 seg / div, y definir las demás siguiendo el modelo estandarizado de los osciloscopios comerciales.

En el subtema en el que se detalla el funcionamiento del controlador de video, se define la extensión de una división sobre la pantalla del monitor: 50 x 50 pixeles. De ese modo, si queremos definir el intervalo de tiempo que debe durar la señal por medir en una columna determinada antes de pasar a la siguiente y realizar un barrido horizontal de 1 seg / div, realizamos el cociente: 1 seg / 50 pixeles = 20 ms. El módulo Base de Tiempos simplemente tiene que incrementar el valor de la coordenada *Col* a la entrada del controlador de VRAM cada vez que transcurra este periodo, para lo cual necesitamos emplear el reloj proporcionado por la tarjeta de desarrollo: 25.175 MHz. Como 20 ms es el periodo de tiempo más grande que se va a utilizar para incrementar el valor de la coordenada Columna, se debe definir un contador de pulsos de reloj con un tamaño de palabra suficiente para completar dicho periodo, y como el número de pulsos que debe contabilizar está dado por el producto 20 x 10^{-3} seg x 25.175 x 10^{6} = 503500, el contador implementado debe ser de 19 bits de longitud, ya que 2^{19} = 524288, cantidad suficiente para cubrir el conteo. De la misma manera podemos calcular los demás periodos de incremento de columna para llenar la siguiente tabla:

Escala de tiempo requerida	Código binario de selección	Periodo de incremento de columna	Número de pulsos por contabilizar	Escala real de tiempo obtenida
1 seg / div	0000	20 ms	503500	1 seg / div
500 ms / div	0001	10 ms	251750	500 ms / div
200 ms / div	0010	4 ms	100700	200 ms / div
100 ms / div	0011	2 ms	50350	100 ms / div
50 ms / div	0100	1 ms	25175	50 ms / div
20 ms / div	0101	0.4 ms	10070	20 ms / div
10 ms / div	0110	0.2 ms	5035	10 ms / div
5 ms / div	0111	0.1 ms	2518	5.001 ms / div
2 ms / div	1000	40 µs	1007	2 ms / div
1 ms / div	1001	**20 µs**	504	1.001 ms / div
0.5 ms / div	1010	**10 µs**	252	0.5005 ms / div

Los valores en negrita correspondientes a los periodos de incremento de columna para las escalas de 1 ms/div y 0.5 ms/div, son menores que la duración del ciclo de Lectura para un renglón de pixeles en la pantalla: 20.73 µs (v. Ciclo de revisión del buffer primario). Esto significa que en estas dos escalas ya estamos excediendo los límites de funcionamiento del osciloscopio, ya que en algunos puntos de la señal introducida se pueden perder algunas muestras (Fig. 3.31). Por otro lado, para el número de pulsos por contabilizar para las escalas de 5 ms/div, 1 ms/div y 0.5 ms/div, resultaron valores fraccionarios de pulso. Como sólo se realiza el conteo de números enteros, se redondearon a los valores más próximos. La tabla muestra una columna adicional que (a reserva de la exactitud del reloj empleado) contiene los valores reales de las escalas de la base de tiempos, donde podemos notar que el porcentaje máximo de error no excede el 0.1 %.

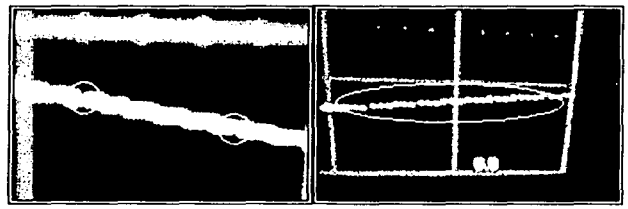


Fig. 3.31 Pérdida de información para las escalas 1 ms/div y 0.5 ms/div.

Tanto en la Fig. 3.32, como en el cuerpo de la entidad definida en el código VHDL, podemos observar que se establecen 4 entradas y 2 salidas para el módulo denominado Base de Tiempos (o simplemente *tiempos*). La entrada *Reloj* se refiere a la señal de reloj de 25.175 MHz. necesaria para la exactitud del barrido horizontal. Las entradas *Rst* y *Congela* serán directamente definidas por el usuario a fin de restablecer las condiciones iniciales (junto con las del módulo *Controlador de VRAM*) y desplegar una imagen fija de la señal capturada, respectivamente. El bus de entrada *Selector* se relaciona con las 11 diferentes opciones que puede realizar el usuario para seleccionar una escala dentro de la base de tiempos. La salida *Limpia* está destinada al controlador de VRAM, con el fin de dar inicio al ciclo Reset y con ello evitar la saturación de la pantalla, por sobrescribir una nueva señal una vez que ha concluido el trazado de izquierda a derecha. Finalmente, el bus de salida *Col* tiene relación directa con la entrada homónima del módulo *Controlador de VRAM*, e incrementa su valor cada que se cumple el periodo de tiempo calculado para cada escala.

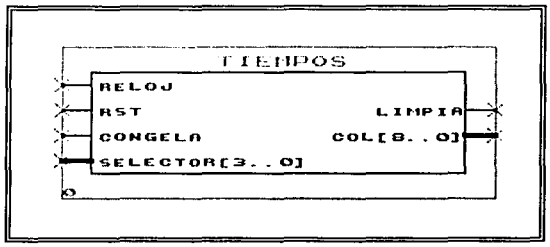


Fig. 3.32 Esquema de entradas y salidas del módulo *Base de Tiempos*.

```
--tiempos.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity tiempos is
       port(Reloj, Rst, Congela: in std_logic;
            Selector: in std_logic_vector(3 downto 0);
            Limpia: out std_logic;
            Col: out std_logic_vector(8 downto 0));
end tiempos;
```

La arquitectura del módulo comienza con la definición de las variables y constantes internas: *Contcol* de 9 bits almacena la coordenada Columna que más adelante se asigna a la salida *Col*; *Contreloj* es el contador de 19 bits que incrementa su valor a cada pulso de *Reloj* hasta llegar al valor definido en *Maxreloj*, variable que establece el número de cuentas necesario para alcanzar el tiempo de espera calculado en cada escala de la base de tiempos; *Contrst* es un contador adicional de 17 bits, implementado para impedir la pérdida de información en el intervalo de tiempo necesario para completar el ciclo Reset del módulo controlador de VRAM. Por último, la constante *Maxcol* define la coordenada máxima de columna en 501, ya que cabe recordar que las columnas 501 a 511 están suprimidas del área de trabajo.

```
architecture arq of tiempos is
       signal Contcol: std_logic_vector(8 downto 0);
       signal Contreloj, Maxreloj: std_logic_vector(18 downto 0);
       signal Contrst: std_logic_vector(16 downto 0);
       constant Maxcol: std_logic_vector(8 downto 0):= conv_std_logic_vector(501, 9);

begin
       Col <= Contcol;
```

Las funciones del módulo están definidas bajo un proceso regido por las transiciones bajo-alto de la entrada *Reloj*, pero dando preferencia al estado de la entrada *Rst*, de tal modo que si ésta última se encuentra en nivel '1' lógico se establecen las condiciones iniciales siguientes: *Contcol*, *Contrst* y *Limpia* en ceros mientras que la variable *Maxreloj* en 503500, valor obtenido para la escala 1 seg/div.

```
       process(Reloj, Rst) begin
              if Rst = '1' then
                     Contcol <= "000000000";
                     Contrst <= conv_std_logic_vector(0,17);
                     Maxreloj <= conv_std_logic_vector(503500,19);
                     Limpia <= '0';
              elsif Reloj = '1' and Reloj'event then
```

Enseguida se establece una secuencia de varias sentencias condicionales, cada una de ellas anidada en la anterior que a continuación se detallan.

Primeramente, se establece el condicional

```
if Contreloj = Maxreloj then
```

de tal manera que si se cumple que el contador *Contreloj* ha llegado a su máxima cuenta, es decir, si ha transcurrido el tiempo necesario para incrementar el valor de la coordenada Columna, el programa accede al siguiente condicional. En caso contrario simplemente se incrementa el valor del contador *Contreloj* en una unidad, mediante la sentencia:

```
                    else
                        Contreloj <= Contreloj + '1';
                    end if;
                end if;
        end process;
end arq;
```

El siguiente condicional anidado pregunta por el estado del contador *Contcol*, a modo de saber si se ha llegado al final del barrido horizontal.

```
        if Contcol = Maxcol then
```

Si no es así se incrementa el valor de *Contcol*, se reinicia el contador *Contreloj* (no en cero sino en '1') y se asegura que el pin de salida *Limpia* no indique al módulo *Controlador de VRAM* el ingreso al ciclo Reset.

```
        else
                Limpia <= '0';
                Contcol <= Contcol + '1';
                Contreloj <= conv_std_logic_vector(1,19);
        end if;
```

Para establecer los siguientes dos condicionales, tomamos algunas decisiones de diseño después de realizar algunas pruebas con la parte del programa que ya teníamos implementada. Uno de los resultados obtenidos nos llevó a la conclusión de que la falta de una sección de disparo representaba un problema, pues al medir una señal repetitiva no se aseguraba que el nuevo trazado comenzara en el mismo punto que el anterior. Cuando esto sucede, se provoca un efecto de animación de la señal que para las escalas más bajas de la base de tiempos imposibilita la medición en el eje horizontal.

A fin de resolver el problema planteado, deliberamos un par de soluciones: una de ellas consiste en adicionar un control accesible al usuario final para incrementar o decrementar el número de columnas que conforman la cuenta máxima *Maxcol*, y de ese modo recortar la señal en el momento exacto en el que se provoca un inicio en el mismo punto que el trazado anterior. La otra opción planteada aprovecha una de las propiedades de los osciloscopios digitales: el dispositivo de almacenamiento. Para facilitar la medición agregamos un interruptor que también es controlado por el usuario final con el objetivo de "detener" la animación de la señal introducida, desplegando en la pantalla únicamente la información almacenada sin actualizar más muestras. Debido a la sencillez de implementación así como de su control por el usuario, nos inclinamos por ésta última opción. Decidimos también que el despliegue de la información almacenada sólo se realizaría después de finalizar el trazado de izquierda a derecha a fin de presentar una pantalla completa. Esto implica que el incremento de la coordenada Columna ha llegado a su cuenta máxima dada por la variable *Maxcol*.

Otro resultado que obtuvimos fue la pérdida de información en las primeras columnas del área de trabajo, debido al ingreso al ciclo Reset del controlador de VRAM con el que debemos "limpiar" la pantalla antes de comenzar un nuevo trazado. Sucede que el ciclo mencionado requiere un periodo de

4.078 ms (v. Ciclo Reset) para completarse, y dicho periodo ocurría simultáneamente con el inicio del incremento de la variable *Contcol*, es decir, con el inicio del nuevo trazado. De ese modo, la pantalla mostraba un recorte al principio del trazo, situación que empeoraba a medida que reducíamos la escala de tiempo por división.
La solución al problema consiste en implementar un contador independiente de los otros llamado *Contrst* para alcanzar un conteo de la señal de *Reloj* equivalente al periodo de tiempo que dura el ciclo Reset, antes de dar inicio a un nuevo trazado. Tal contador requiere 17 bits para registrar 102656 pulsos (4.078 ms x 25.175 MHz).
Una vez expuestos los problemas encontrados y sus soluciones, procedimos a la implementación del algoritmo:

```
if Congela = '0' then
        if Contrst = conv_std_logic_vector(102655,17) then
```

El par de condicionales indicados, se refieren en orden de aparición primeramente al estado del interruptor para desplegar la imagen fija (variable interna *Congela*), de tal modo que si éste se encuentra en nivel lógico '1' se activa la función y se detienen todos los contadores del módulo, y aunque el convertidor analógico digital siga proporcionando muestras que desvíen el haz de electrones de modo vertical, dichas muestras ya no se despliegan en la pantalla, pues se localizan en la columna máxima (501) que ya se encuentra fuera del área de trabajo. Ahora bien, si se cambia la posición del interruptor de congelamiento, al siguiente pulso de *Reloj* nos encontramos con la pregunta condicional sobre si el contador *Contrst* ha llegado a la cuenta máxima calculada. Si no es así se procede a incrementarlo y a generar la señal que indica al módulo controlador de VRAM que dé inicio al ciclo Reset, mediante la siguiente sección de código:

```
        else
                Contrst <= Contrst + '1';
                if Contrst = conv_std_logic_vector(1,17) then
                        Limpia <= '1';
                else
                        Limpia <= '0';
                end if;
        end if;
end if;
```

Mas si se ha llegado a la cuenta calculada de espera antes de iniciar el siguiente trazado, reiniciamos los tres contadores: *Contreloj*, *Contrst* y *Contcol*. Es también en estas condiciones que se redefine la cuenta máxima del contador *Contreloj* dependiendo de la selección de escala hecha por el usuario y con el fin de no mostrar en una misma pantalla dos o más secciones que tengan diferentes escalas de tiempo.

```
Contreloj <= conv_std_logic_vector(1,19);
Contrst <= conv_std_logic_vector(0,17);
Contcol <= "000000000";

if Selector = "0001" then
        Maxreloj <= conv_std_logic_vector(251750,19);       --500 ms/div
elsif Selector = "0010" then
        Maxreloj <= conv_std_logic_vector(100700,19);       --200 ms/div
elsif Selector = "0011" then
        Maxreloj <= conv_std_logic_vector(50350,19);        --100 ms/div
elsif Selector = "0100" then
        Maxreloj <= conv_std_logic_vector(25175,19);        -- 50 ms/div
elsif Selector = "0101" then
        Maxreloj <= conv_std_logic_vector(10070,19);        -- 20 ms/div
elsif Selector = "0110" then
        Maxreloj <= conv_std_logic_vector(5035,19);         -- 10 ms/div
elsif Selector = "0111" then
        Maxreloj <= conv_std_logic_vector(2518,19);         --  5 ms/div
elsif Selector = "1000" then
        Maxreloj <= conv_std_logic_vector(1007,19);         --  2 ms/div
elsif Selector = "1001" then
        Maxreloj <= conv_std_logic_vector(504,19);          --  1 ms/div
```

```
elsif Selector = "1010" then
        Maxreloj <= conv_std_logic_vector(252,19);          -- .5 ms/div
else
        Maxreloj <= conv_std_logic_vector(503500,19);       --  1 seg/div
end if;
```

Para comprender los valores asignados en ésta última parte del código presentado, es necesario remitirnos a la tabla expuesta al principio del desarrollo de éste subtema.

En resumen, la función del módulo Base de Tiempos consiste en suministrar la información requerida por el módulo controlador de VRAM para realizar el barrido horizontal de la pantalla del osciloscopio. Dicha información consta del incremento a intervalos uniformes de tiempo de la columna de pixeles que conforman la señal introducida, donde la duración de los citados intervalos depende de una de las 11 selecciones posibles proporcionada por el usuario final, con respecto a la escala de tiempos. Asimismo, el módulo Base de Tiempos emite una señal para renovar la pantalla una vez terminado el trazado de izquierda a derecha o incluso, si el usuario lo requiere, proporciona la opción de mantener fija la imagen hasta ese momento capturada.

Los recursos consumidos por el módulo *Base de Tiempos* dentro del dispositivo lógico programable EPF10K20, ocupan un total de 219 compuertas lógicas lo que corresponde a un 19% de la totalidad disponible. También cabe hacer notar que la compilación del módulo realizada por MAX+PLUS II en una PC con procesador Intel Pentium y 192 MB de memoria RAM, requiere un tiempo menor a 1 minuto.

III.3 INTEGRACIÓN DE LOS MODULOS.

Hasta ahora se ha detallado el funcionamiento de los módulos controlador de video, controlador de VRAM y Base de Tiempos. Dentro del diseño del código VHDL, sólo queda explicar las funciones que realiza el programa destinado a integrar los tres módulos principales en el bloque que conforma la electrónica digital del proyecto. Además de la integración, el programa denominado simplemente *Osciloscopio* tiene la tarea de relacionar cada una de las entradas y salidas de los módulos citados con los pines del chip FLEX EPF10K20RC240-4 que necesitarán para su correcta interacción con la memoria SRAM, con el convertidor analógico digital y con los controles que el usuario final tendrá a su disposición.

Las entradas y salidas, así como el puerto bidireccional del bloque que conforma la electrónica digital del osciloscopio, se definen de la siguiente manera (Fig. 3.33):

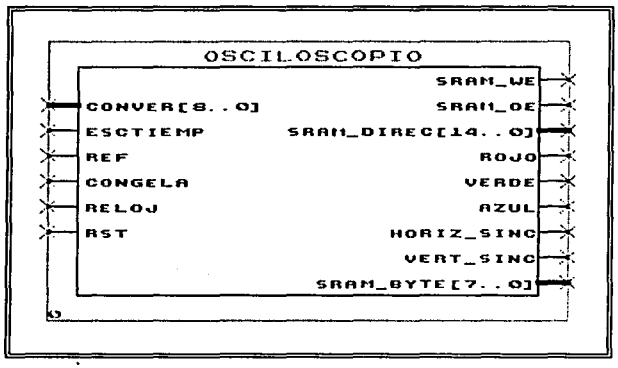


Fig. 3.33 Esquema de entradas, salidas y bus bidireccional del bloque de Electrónica Digital del osciloscopio.

Entradas:

- *Conver*, de 9 bits que requiere información del convertidor analógico digital y la destina al módulo *Controlador de VRAM* a fin de realizar el equivalente a la deflexión vertical del cañón electrónico del monitor. Al mencionar esta entrada, cabe hacer notar que el convertidor utilizado (ADC0804) proporciona únicamente una salida de 8 bits, lo que se traduce en la pantalla del monitor como una deflexión vertical máxima de 256 pixeles de amplitud, a condición de que el bit más significativo de *Conver* se encuentre en nivel '0' lógico.
- *Esctiemp* se refiere a una señal suministrada por el usuario final con el objetivo de modificar la escala en la base de tiempos, de tal modo que por cada pulso emitido (en un push button) se decrementa el número de milisegundos por división del modo indicado en el desarrollo del subtema referente al módulo *Base de Tiempos*.
- *Ref* es también una señal controlada por el usuario final que modifica la línea de referencia con respecto al eje vertical de la cuadrícula mediante pulsos emitidos por un push button. El efecto que produce es que la señal se desplaza hacia la parte superior de la pantalla hasta desaparecer, pudiendo incluso emerger nuevamente por el límite inferior del área de trabajo.
- *Congela* es la entrada homónima requerida por el módulo *Base de Tiempos* para mostrar una imagen fija a conveniencia del usuario (v. subtema *Base de Tiempos*).
- *Reloj* es la entrada un tren de pulsos a 25.175 MHz. requerida por los tres módulos.
- *Rst* es un interruptor controlado por el usuario, que en caso de presentar nivel '1' lógico, reinicia la operación del osciloscopio. Se utiliza por los módulos *Controlador de VRAM* y *Base de Tiempos* para inicializar las variables internas, ya que al considerarse como un registro de tipo presettable, al energizarse el osciloscopio presenta estado activo alto (Fig. 3.34).

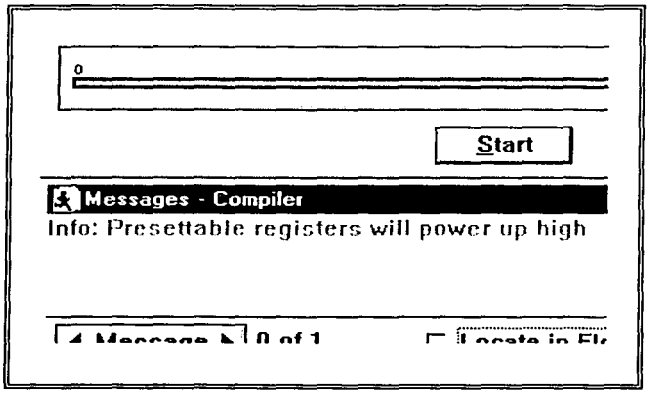


Fig. 3.34 Mensaje del compilador de MAX+PLUS II, que se interpreta como "Los registros de tipo *preset* se energizan en estado activo alto".

Salidas:

- *Sram_we*, *sram_oe* y *sram_direc* son respectivamente las salidas dirigidas al chip SRAM que almacena la memoria de video, para los pines *we*, *oe* y los 15 bits que definen la dirección dentro del mapa de memoria. Se utilizan para las funciones detalladas en el subtema sobre el *Controlador de VRAM*.
- *Rojo*, *Verde* y *Azul* provienen del módulo *Controlador de Video* (v. subtema correspondiente) y son requeridas por el monitor para suministrar la información sobre el color que cada pixel debe desplegar dentro del área de trabajo de la pantalla.
- Del mismo modo *Horiz_sinc* y *Vert_sinc* son requeridas por el monitor para sincronizar los barridos horizontal y vertical del cañón electrónico sobre la pantalla (v. subtema *Módulo Controlador de Video*).

Bidireccional:

- *Sram_byte* funciona como puerto bidireccional que une al chip SRAM con el módulo *Controlador de VRAM* (v. subtema correspondiente), a fin de suministrar datos a la memoria de video para posteriormente leerlos de la misma.

A continuación se presenta la sección de código VHDL que define la entidad *osciloscopio* como se explicó en los párrafos anteriores:

```
--osciloscopio.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

entity osciloscopio is
       port(Conver: in std_logic_vector(8 downto 0);
            signal Esctiemp, Ref, Congela: in std_logic;
            signal Reloj, Rst: in std_logic;
            sram_we, sram_oe: out std_logic;
            sram_direc: out std_logic_vector(14 downto 0);
            signal Rojo, Verde, Azul : out std_logic;
            signal Horiz_sinc, Vert_sinc : out std_logic);
            sram_byte: inout std_logic_vector(7 downto 0);
end osciloscopio;
```

Posteriormente se definen las variables internas de la arquitectura que intervienen en el programa, y se incluyen como componentes los tres módulos analizados: *control_video*, *control_vram* y *tiempos*.

```
architecture arq of osciloscopio is
              signal Medio: std_logic;
              signal VRrst, VRnivelpix: std_logic;
              signal VRren, Increren: std_logic_vector(8 downto 0);
              signal Limpia: std_logic;
              signal Selector: std_logic_vector(3 downto 0);
              signal Retardo: std_logic_vector(1 downto 0);
              signal Fincont: std_logic;

component control_video
         port(signal Reloj: in std_logic;
              signal Rojo, Verde, Azul : out std_logic;
              signal Horiz_sinc, Vert_sinc : out std_logic;
              signal Pulsolec : out std_logic;
              signal Video : in std_logic;
              signal Selector: in std_logic_vector(3 downto 0);
              signal Pixcero, Pulsescr: out std_logic);
end component;

component control_vram
          port(Rst, Medio, Pulsolec, Pulsescr, Pixcero, Nivelpix: in std_logic;
               Ren: in std_logic_vector(8 downto 0);
               Col: in std_logic_vector(8 downto 0);
               Video: out std_logic;
               sram_we, sram_oe: out std_logic;
               sram_direc: out std_logic_vector(14 downto 0);
               sram_byte: inout std_logic_vector(7 downto 0));
end component;

component tiempos
          port(Reloj, Rst, Congela: in std_logic;
               Selector: in std_logic_vector(3 downto 0);
               Limpia: out std_logic;
```

```
                Col: out std_logic_vector(8 downto 0));
end component;
```

Se establecen las interconexiones entre los tres módulos, así como con las entradas y salidas generales del osciloscopio:

```
begin
        video:  control_video port map (Reloj, Rojo, Verde, Azul, Horiz_sinc,
                Vert_sinc, Pulsolec, Video, Selector, Pixcero, Pulsescr);

        vram:   control_vram port map (VRrst, Medio, Pulsolec, Pulsescr, Pixcero, VRnivelpix,
                VRren, Col, Video, sram_we, sram_oe, sram_direc, sram_byte);
        osc:    tiempos port map (Reloj, Rst, Congela, Selector, Limpia, Col);

        VRrst <= Limpia or Rst;
        VRren <= not ((Conver + Increren) + conv_std_logic_vector(183,9));
        VRnivelpix <= '1';
```

Analizando con detalle la sección de código donde se definen los componentes y si la comparamos con la sección de interconexión, podemos notar que los tres módulos, así como el proyecto en general, comparten variables en sus entradas y salidas:

- *Reloj* y *Selector* son entradas para los módulos *control_video* y *tiempos*. Más adelante se explica que la entrada *Medio* en el módulo restante está directamente relacionada con *Reloj*, así como *Selector* se genera por un algoritmo que utiliza la entrada general *Esctiemp*.
- *Rojo*, *Verde*, Azul, *Horiz_sinc* y *Vert_sinc*, salidas del módulo *control_video* se convierten en salidas generales del osciloscopio, dirigidas al monitor VGA.
- Las salidas restantes del mismo *control_video*, esto es, *Pulsolec*, *Pixcero* y *Pulsescr* son requeridas como entradas para el módulo *control_VRAM*.
- A su vez, la salida *Video* que proviene de *control_VRAM* es requerida por el módulo *control_video*.
- La entrada *Col* del *Controlador de VRAM*, proviene de la salida homónima del módulo *Base de Tiempos*.
- Algunas de las salidas provinientes del módulo *control_VRAM*, esto es, *sram_we*, *sram_oe* y *sram_direc*, del mismo modo que el puerto bidireccional *sram_byte*, se conectan directamente a los pines externos del chip FLEX, a fin de lograr la correcta intercomunicación entre éste y el chip SRAM que alberga la memoria de video.
- Las entradas generales del osciloscopio denominadas *Rst* y *Congela* son directamente conectadas a las entradas homónimas del módulo Base de tiempos.

Las condiciones anteriores y algunas otras que se seguirán desarrollando, quedan claramente ilustradas en la Fig. 3.35.

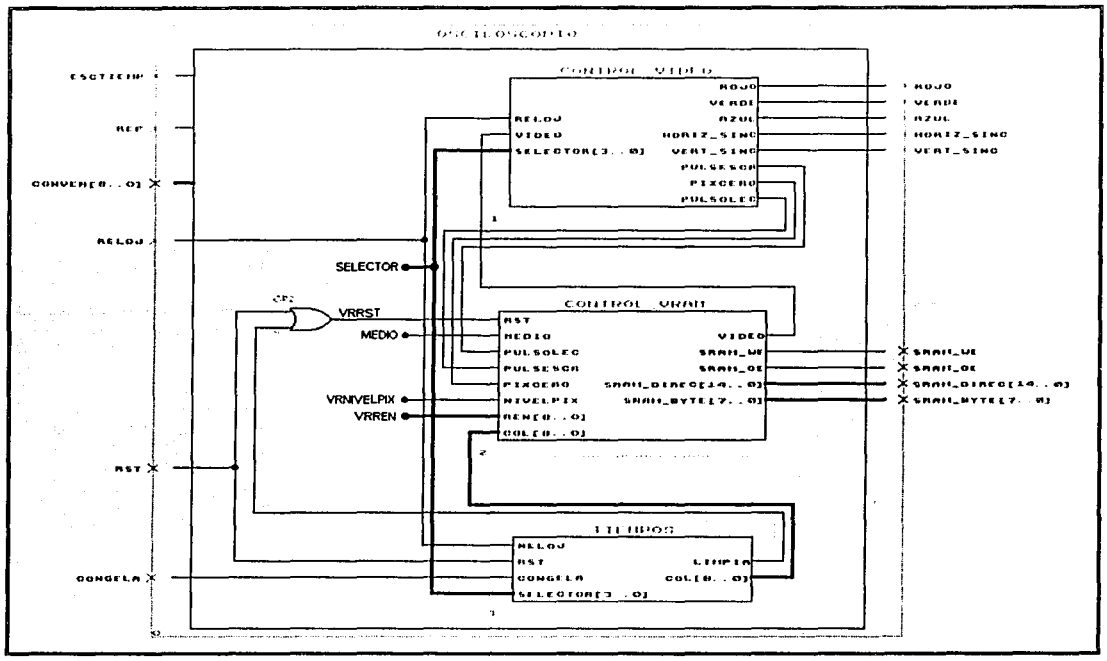


Fig. 3.35 Esquema de interconexión entre los tres módulos que conforman el bloque de Electrónica Digital del osciloscopio.

Adicionalmente, explicaremos la función de la salida *Limpia*, así como los algoritmos que producen las entradas *VRrst*, *VRren* y *VRnivelpix* del módulo controlador de la VRAM:

VRRST. Se activa cuando existe un valor '1' lógico en cualquiera de las dos señales *Limpia* o *Rst*, o en ambas. La salida *Limpia*, proviniente del módulo *tiempos* (v. subtema *Módulo Base de Tiempos*), envía un pulso activo alto cada vez que es necesario borrar la información de la memoria de video para borrar de la pantalla la señal registrada en el barrido anterior, tarea que también se efectúa al energizar el dispositivo o cuando el usuario activa la entrada general *Rst.*

VRREN. Inicialmente ésta variable estaba directamente conectada a la entrada general *Conver*, pero por las características del contador de renglones en el módulo *control_video* (v. subtema *Módulo Controlador de Video*) fue necesario invertir dicha entrada a fin de, al momento de que el convertidor analógico digital entregara una cuenta creciente, no se desplegara en la pantalla del monitor una línea descendente. Dicho con otras palabras, la señal estaría "de cabeza". Una vez invertida *Conver*, la línea de referencia que indicaría ceros en el convertidor analógico digital estaría en el renglón 511, esto es, fuera del área de trabajo. Por tal motivo, le sumamos una cantidad calculada en 183 renglones siguiendo el siguiente razonamiento: el acondicionador de señal debe ser capaz de registrar diferencias de potencial tanto positivas como negativas, por lo que el convertidor analógico digital deberá mantener un voltaje de referencia de 2.5 volts, lo que equivale a la mitad del voltaje que es capaz de registrar sin llegar al punto de saturación. Ese voltaje de referencia provoca a la salida del convertidor también la mitad de su rango de conteo máximo, que es de 256 niveles diferentes. De esta manera, la línea de referencia que indica ceros en el convertidor analógico digital está directamente relacionada con el voltaje negativo máximo que éste es capaz de registrar, y debe estar 128 renglones por debajo del eje central horizontal de la rejilla que divide el área de trabajo. Contando de arriba hacia

abajo, el eje central se ubica en el renglón 200, y la línea de referencia en el 328; pero como ya implementamos en el programa que ese número debe pasar por un inversor de 9 bits, realizamos la operación complemento para obtener 511 - 328 = 183, número que se le suma a la entrada *Conver* para mantener la señal dentro de los 128 renglones por arriba y por debajo del eje central. A conveniencia del usuario, se adiciona la variable *Increren* cuyo algoritmo se explica más adelante y que sirve para modificar esta condición.

VRNIVELPIX. Es la entrada *Nivelpix* explicada en el subtema sobre el controlador de VRAM, la cual es requerida para "encender" o "apagar" el pixel especificado en las entradas del mismo módulo con sus coordenadas (Renglón, Columna). Pues bien, como el único momento necesario para "apagar" los pixeles es cuando se ingresa al ciclo Reset (v. subtema *Módulo Controlador de VRAM*) y como para ello no se requiere el empleo de dicha variable, la entrada *VRnivelpix* tiene permanentemente asignado un nivel '1' lógico.

La siguiente sección de código indica que se da inicio a un proceso activado por la modificación de la entrada *Rst* o de la variable *Pixcero*. Si la entrada *Rst* se encuentra en '1' lógico, se inicializan las variables internas *Selector*, *Fincont* y *Retardo* con ceros.

```
process(Rst, Pixcero) begin
        if Rst = '1' then
                Selector <= "0000";
                Fincont <= '0';
                Retardo <= conv_std_logic_vector(0,2);
```

En caso contrario, y si la variable *Pixcero* realiza una transición alto-bajo (lo que ocurre 60 veces por segundo, es decir, cada 16.6 milisegundos aprox.) se da inicio a una serie de sentencias condicionales anidadas: primeramente se pregunta por el estado de la entrada *Esctiemp* la cual deberá ser conectada a un push button N/A conectado a tierra a fin de producir '0' lógico cuando es activado por el usuario. Si esa condición ocurre se pregunta por el estado de la variable *Fincont*, que de estar en '0' lógico nos lleva a otro condicional. Se pregunta si el contador de 2 bits *Retardo* ha llegado a su cuenta máxima, de tal modo que de no ser así, se incrementa una unidad y se espera hasta la siguiente transición alto-bajo de *Pixcero*. Todo lo anterior tiene la finalidad de que el usuario final mantenga presionado el push button correspondiente un mínimo de tiempo de 66.6 milisegundos antes de que se produzca la última parte del proceso: alcanzado el conteo máximo, se reinicia el contador, se levanta la bandera *Fincont* para impedir un nuevo ingreso al ciclo y se pregunta por el estado de la variable *Selector* que se incrementa en una unidad a menos que se encuentre en el estado máximo "1010", en cuyo caso se reinicia en ceros. Recordemos que *Selector* tiene que ver con una de las 11 escalas utilizadas en el módulo *Base de Tiempos* así como con la etiqueta correspondiente desplegada por el módulo *Controlador de Video* (v. subtemas correspondientes).

```
        elsif Pixcero = '0' and Pixcero'event then
                if Esctiemp = '0' then
                        if Fincont = '0' then
                                if Retardo = conv_std_logic_vector(3,2) then
                                        Retardo <= conv_std_logic_vector(0,2);
                                        Fincont <= '1';
                                        if Selector = "1010" then
                                                Selector <= "0000";
                                        else
                                                Selector <= Selector + "0001";
                                        end if;
                                else
                                        Retardo <= Retardo + '1';
                                end if;
                        end if;
                else
                        Retardo <= conv_std_logic_vector(0,2);
                        Fincont <= '0';
                end if;
        end if;
end process;
```

Con lo anterior, queremos establecer que cada vez que el usuario presiona el botón correspondiente a la entrada *Esctiemp*, se modifica la escala de tiempo de la manera indicada en el subtema sobre el módulo *Base de Tiempos*. Otro dato de importancia es que el retardo implementado en el algoritmo evita que el osciloscopio "brinque" aleatoriamente a otra escala que no es la esperada.

Otro proceso menos complejo da inicio al modificarse el estado de cualesquiera de las dos entradas *Rst* o *Ref*, de tal modo que si la primera está en nivel '1' lógico se inicializa la variable *Increren* con ceros; y en caso contrario si la segunda realiza una transición bajo-alto, la mencionada variable *Increren* incrementa en una unidad. Recordemos que la variable *Increren* se adicionó a la entrada *Conver* para establecer la línea de referencia y ubicar la señal registrada dentro de una franja horizontal centrada con respecto al área de trabajo. A conveniencia del usuario, dicha franja puede ubicarse más arriba a cada pulso de la entrada *Ref* hasta sobrepasar el límite superior de la pantalla e incluso emerger por el límite inferior. A diferencia del proceso análogo seguido por la modificación de la variable *Selector*, no implementamos un algoritmo para impedir que se "brinque" más de un renglón a la vez.

```
process(Rst, Ref) begin
        if Rst = '1' then
                Increren <= conv_std_logic_vector(0,9);
        elsif Ref = '1' and Ref'event then
                Increren <= Increren + "000000001";
        end if;
end process;
```

Finalmente, el último proceso define el algoritmo necesario para crear la entrada de reloj denominada *Medio* para el módulo *control_VRAM*. Dicha entrada requiere una frecuencia de 12.5875 MHz, es decir, la mitad de la frecuencia de la entrada general *Reloj* (v. subtema *Módulo Controlador de la VRAM*):

```
        process(Reloj) begin
                if Reloj = '1' and Reloj'event then
                        Medio <= not Medio;
                end if;
        end process;
end arq;
```

El proceso es muy simple: cada vez que se produce una transición bajo-alto de la señal *Reloj*, se invierte el estado lógico de la variable *Medio*. Esto produce un ancho de pulso equivalente a todo un periodo de la señal *Reloj* y se obtiene la frecuencia requerida.

En conclusión, la integración de los módulos *Base de Tiempos*, *Controlador de Video* y *Controlador de VRAM*, está a cargo del bloque denominado *Osciloscopio* que realiza las interconexiones entre los otros tres, les proporciona algunas de las variables que solicitan, canaliza tanto los paquetes de información que son requeridos por el monitor como los que son empleados por la memoria de video, y finalmente, establece una interface digital simple entre el usuario y el equipo a fin de manipular la escala de tiempo, una señal de reinicio y algunas características adicionales de la señal registrada, específicamente, la altura de esta con respecto al área de trabajo así como la posibilidad de mantener fija una imagen registrada durante un tiempo indefinido.

Los recursos consumidos por el bloque de electrónica digital del osciloscopio dentro del dispositivo lógico programable EPF10K20, ocupan un total de 771 compuertas lógicas lo que corresponde a un 66% de la totalidad disponible. También cabe hacer notar que la compilación del módulo realizada por MAX+PLUS II en una PC con procesador Intel Pentium y 192 MB de memoria RAM, requiere un tiempo mayor a 21 minutos.

IV. PRUEBAS Y RESULTADOS.

IV.1 CARACTERIZACIÓN DEL INSTRUMENTO.

El prototipo que diseñamos se puede considerar como un instrumento de medición para señales oscilatorias de voltaje capaz de registrar frecuencias relativamente bajas, esto es, desde fracciones de Hertz hasta unas pocas decenas de KHz. Para ello, el osciloscopio cuenta con 11 escalas en una base de tiempos que el usuario puede seleccionar para visualizar en la cuadrícula que divide a la pantalla del monitor VGA, el comportamiento de la señal suministrada con respecto al tiempo que representa cada división horizontal para la escala elegida.

Las escalas en la base de tiempos del osciloscopio son:

Escala de tiempos
1 seg / div
500 ms / div
200 ms / div
100 ms / div
50 ms / div
20 ms / div
10 ms / div
5 ms / div
2 ms / div
1 ms / div
0.5 ms / div

El límite que caracteriza la velocidad máxima de barrido horizontal de la señal está dado por el periodo de tiempo que requiere la máquina de estados del osciloscopio para finalizar los ciclos de lectura necesarios para transmitir la información de la memoria de video a uno de los renglones de la pantalla (v. Módulo *Controlador de VRAM*). Ese periodo es de 20.73 μs y durante él no es posible registrar la información que pudiera ser suministrada por el convertidor A/D. Esto significa que la velocidad máxima de muestreo que el osciloscopio es capaz de manejar está dada por el recíproco del periodo calculado: aproximadamente 50 Kilomuestras por segundo (48.24 KHz), a condición de que el convertidor A/D empleado trabaje a una frecuencia de muestreo mayor o igual a ésta. Más allá de las frecuencias de medición empleadas para obtener estas escalas, parte de la información de la señal se pierde y por lo tanto se dificulta su interpretación. La exactitud de la base de tiempos como ya se indicó, tiene un porcentaje máximo de error de 0.1%

En cuanto a la sección de deflexión vertical, el prototipo actual tiene una sensibilidad máxima de 1 V/div, y una exactitud en la ganancia con un 10% como porcentaje máximo de error. La resolución vertical está determinada por el convertidor analógico digital que tiene una precisión de 8 bits, lo que se interpreta como 256 valores digitales diferentes que se pueden almacenar en la memoria.

Otro de los parámetros que se caracterizó fue el ancho de banda, que se calculó desde cero hasta 50 KHz. Por otro lado, para la reconstrucción de la forma de onda y entendiendo como longitud de registro el parámetro que indica el número de puntos que se memorizan, podemos establecer que se obtiene realizando un producto con el número de localidades empleadas en el mapa de memoria (25664) por el número de pixeles almacenados en cada una de ellas (8), lo que nos da una longitud máxima de 205312 puntos.

El prototipo carece de una sección de disparo que estabilice señales repetitivas. Sin embargo, a fin de remplazar esa función, tiene implementado un interruptor para desplegar por tiempo indefinido la imagen capturada en la memoria de video al momento de concluir un ciclo de barrido horizontal. Esto

equivale a tomar una fotografía con respecto al tiempo de la señal suministrada para su estudio detallado, pero presenta la desventaja de que tal imagen no puede ser manipulada por parte del usuario, esto es que no se modifica bajo ninguna circunstancia a menos que se inhabilite el interruptor.

Cuenta además con un control de posición vertical, que a pesar de haberse implementado con bastantes limitaciones para esta versión del proyecto, cumple con su función primordial: desplazar verticalmente a través de la pantalla la señal registrada. Carece de un control de posición horizontal, pues como ya se mencionó, no cuenta con una sección de disparo. Los restantes controles que afectan la visualización (intensidad y contraste) dependen exclusivamente del monitor VGA empleado.

IV.2 PRUEBAS DE DIVERSAS MEDICIONES

Una vez que se probaron por separado tanto el bloque acondicionador de señal como el bloque de electrónica digital del osciloscopio, se procedió a la interconexión de ambos a través del convertidor analógico digital con el fin de realizar las pruebas definitivas. Con la ayuda de un generador de funciones, registramos tanto en el prototipo como en un osciloscopio analógico comercial primeramente una señal sinusoidal de voltaje alterno con 4 Vpp, a una frecuencia de 14.7 Hz, como se muestra en la Fig. 4.1.

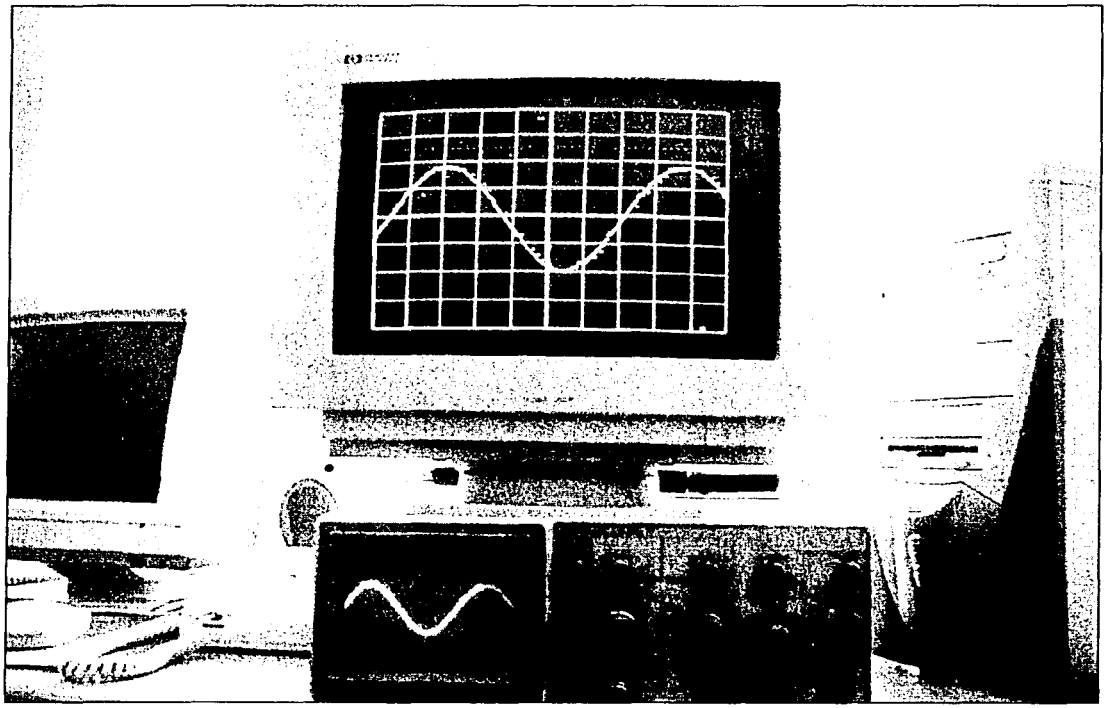
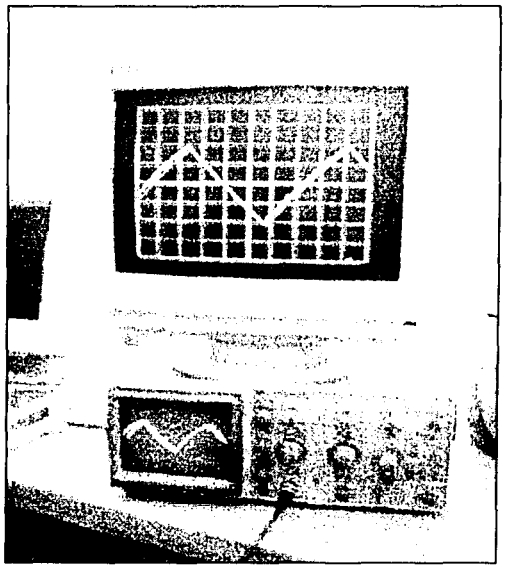
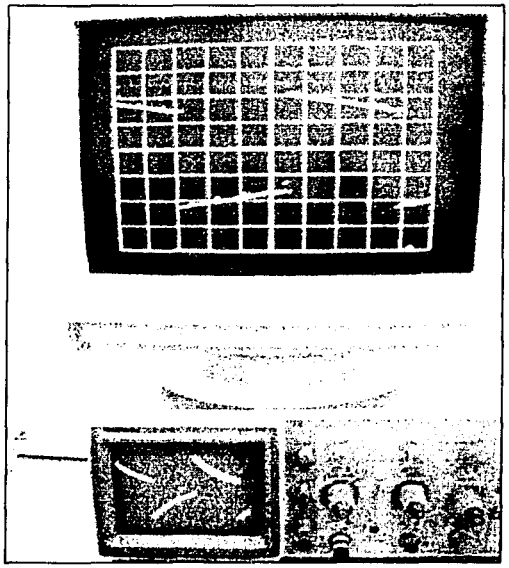


Fig. 4.1 Señal sinusoidal de 4 Vpp a 14.7 Hz, registrada por un osciloscopio comercial y por el prototipo diseñado.

Para observar la señal introducida, en ambos osciloscopios ajustamos la escala de tiempo a 10 ms/div, y mientras que en el osciloscopio analógico seleccionamos la escala de 1 V/div, en el prototipo

utilizamos la etapa de atenuación más alta y ajustamos la ganancia para calibrar la deflexión vertical. Dejando fijos estos parámetros, simplemente modificamos la forma de la señal entregada por el generador de funciones para comparar los resultados obtenidos con ambos osciloscopios al introducir una señal triangular y una señal de tipo cuadrado (Figs. 4.2 y 4.3).





Figs. 4.2 y 4.3 Comparación de mediciones de una señal triangular y una señal cuadrada a 14.7 Hz introducidas en los osciloscopios digital y analógico.

Algunos resultados que obtuvimos de estas comparaciones demostraron que el osciloscopio diseñado trabaja de acuerdo a las expectativas de diseño, para cualquier tipo de señal con frecuencia inferior al ancho de banda caracterizado. Para escalas de tiempo altas, la señal se despliega como una gráfica bien delineada y con alto grado de definición, pero las pruebas que realizamos con señales de más alta frecuencia mostraron una discontinuidad vertical entre los puntos que la forman, pues desafortunadamente el método de muestreo que el osciloscopio emplea carece de técnica de interpolación o de reconstrucción en tiempo equivalente.

Otro de los resultados obtenidos al realizar las pruebas fue la aparición de algunos puntos fuera de la línea esperada que conforma la gráfica. Descartados como ruido, hasta el momento en el que redactamos éste capítulo manejamos la hipótesis de que tales distorsiones son provocadas por una falta de sincronización en el ciclo de Escritura dentro del módulo *Controlador de VRAM*, problema que por falta de tiempo no hemos resuelto y dejaremos dentro de la lista que conforma los trabajos posteriores.

IV.3 DISPOSITIVOS UTILIZADOS

Familia	Subfamilia	Modelo	Descripción	Cantidad	Precio Unitario (pesos)	Subtotal
Capacitor	Electrolítico	E10-63R	Capacitor electrolítico	1	2.00	2.00
Capacitor	Cerámico	C.047-50	Capacitor cerámico de disco	1	2.00	2.00
Capacitor	Polyester	P.1-250	Capacitor de poliéster metalizado	1	4.00	4.00
Resistencia	0.25 W	1/4	Resistencia de carbón 5 % tolerancia	24	0.50	12.00
Potenciómetros y Perillas	Presets preajustables	110-XK	Potenciómetro preset vertical 10 mm	3	2.50	7.50
Semiconductores	Microcontroladores	ADC0804	Convertidor CMOS 8 bits A/D	1	39.00	39.00
Semiconductores	CI Lineales	TL084CN	CI Lineal	1	6.00	6.00
Semiconductores	CI Lineales	TL081CP	CI Lineal	1	5.00	5.00
Switches y relevadores	Switch rotatorio	1P 11T	Switch rotatorio de 1 polo 11 tiros	1	8.50	8.50
Switches y relevadores	Switch On-Off	AU-106	Switch on-off 125 V, 3 A	1	9.00	9.00
Push button	--	AU-102R	Push button	3	3.00	9.00
Push button	--	AU-103N	Push button	1	9.00	9.00
Memoria	CMOS SRAM	HY62256	Memoria SRAM 32K X 8	1	43.30	43.30

V. BIBLIOGRAFÍA

Valeriano, J., Chávez, N., Haro, A. (2001) *Lenguaje de Descripción de Hardware VHDL*
Facultad de Ingeniería
Ciudad Universitaria, México, D.F.

Chávez, N., Valeriano, J., Haro, A. (2001) *Entorno de Diseño Max+Plus II*
Facultad de Ingeniería
Ciudad Universitaria, México, D.F.

Taub, H., Schilling, D. (1985) *Digital Integrated Electronics*
International Student Edition
Editorial McGraw-Hill
11th printing

Linear Data Book. National Semiconductor Corporation

Circuitos Básicos de Ordenador
E.A.PARR
Monografías CEAC de Informática
3ª Edición Marzo 1986

Holman, J. (1981) *Métodos Experimentales Para Ingenieros*
Editorial McGraw-Hill
1ª Edición

Tocci, R. *Sistemas Digitales Principios y Aplicaciones*
PHH Prentice Hall
Tercera Edición

Referencias de Internet

http://www.altera.com

http://www.cs.swarthmore.edu/~sproul/cs23/finalProject/finalProject.html
Sproul, D. *The Most Unnecessarily Elaborate Oscilloscope in the World.*
Swarthmore College, 2001.

http://users.ece.gatech.edu/~hamblen/ALTERA/onedge/gatech/video.htm
McAlister, D; Philippart, G.; Sugg, M. *Using the Video on the Altera UP1 Board.*
School of Electrical and Computer Engineering
Georgia Institute of Technology.

http://www.optimagic.com/faq.html
Frequently-Asked Questions (FAQ) About Programmable Logic.
OptiMagic™, Inc., 2000.

http://users.ece.gatech.edu/~hamblen/ALTERA/altera.htm
Hamblen, J. *Video and VHDL Demo Files for Altera's UP 1 and UP 2 Education Boards.*
School of Electrical and Computer Engineering
Georgia Institute of Technology.

http://users.ece.gatech.edu/~hamblen/ALTERA/wcae98.PDF
Hamblen, J. *Using Large CPLDs and FPGAs for Prototyping and VGA Video Display Generation in Computer Architecture Design Laboratories.*
School of Electrical and Computer Engineering
Georgia Institute of Technology.

http://www.erc.msstate.edu/~reese/vhdl_synthesis/
Reese, B. *VHDLSynthesisTutorial.*
Electrical Engineering Department
Mississippi State University.

http://www.mitronet.com/chipdir/f/sram.htm
Chips in the functional section 'sram'.
Chip directory (Search site)

http://www.national.com

http://www.ii.uam.es/~gdrivera/labetcii/curso0001/pract2.html

http://us.st.com/

http://www.ifent.org/Lecciones/digitales/secuenciales/ConvertA_D.htm