DIPLOMADO EN MICROCONTROLADORES (SISTEMAS EMBEBIDOS)

CA 213 MÓDULO VII

MICROCONTROLADORES E INTERNET

TEMA:

INSTRUMENTACIÓN DIGITAL UTILIZANDO LÓGICA PROGRAMABLE
INTRODUCCIÓN AL MANEJO DE DISPOSITIVOS MÓVILES VÍA INTERNET

INSTRUCTOR: ING. NORMA ELVA CHÁVEZ RODRÍGUEZ

DEL 22 AL 29 DE NOVIEMBRE DE 2004

PALACIO DE MINERÍA

Palacio de Mineria, Calle de Tacuba No 5, Primer piso, Delegación Cuauhtémoc, CP 06000, Col. Centro, México D.F.,
APDO Postal M-2285 • Tels 5521 4021 al 24, 5623 2910 y 5623.2971 • Fax: 5510.0573

CAPÍTULO 5

GENERADOR DIGITAL DE TREN DE PULSOS PROGRAMABLE

CONTENIDO

5.1 INTRODUCCIÓN.

En el contexto de la instrumentación y más aún en cualquier laboratorio de electrónica digital es muy importante disponer de un generador de tren de pulsos, ya que es indispensable para realizar un trabajo metódico tanto para la operación de equipos como para efectuar pruebas diversas

Este dispositivo tiene por objetivo generar un tren de pulsos a partir de una frecuencia base con la cual es posible generar frecuencias menores mediante divisores de frecuencia. Resulta muy útil que la frecuencia pueda ser variada desde el intervalo de la frecuencia base hasta por ejemplo un hert, además de poder variar el ancho del pulso

Un equipo de este tipo es muy útil, ya que puede ser utilizado para proveer una simple señal de reloj hasta ser utilizado como modulador de ancho de pulso, por ejemplo para control de velocidad de un motor o para regular la intensidad de un sistema de iluminación.

El objetivo de este capítulo es diseñar e implementar un generador digital de tren de pulsos que tenga la capacidad de programar su frecuencia de operación así como su ancho de pulso, ambos digitalmente, programándolo en VHDL como un módulo que será muy útil en el diseño de instrumentación digital. De esta forma, la Metodología objeto de la presente tesis agrega este componente a la librería de módulos para el diseño de instrumentación digital.

5.2 DISEÑO DE UN GENERADOR DIGITAL DE TREN DE PULSOS PROGRAMABLE EN FRECUENCIA Y ANCHO DE PULSO.

En la Fig 5 1 se muestra una señal cuadrada con tres parámetros que podemos manipular para controlar la frecuencia de la señal así como el ancho del pulso El parámetro "T" define el periodo de la señal, de modo que "1/T" define la frecuencia (F); por lo tanto el control de "T" nos permite alterar la frecuencia de la señal Los parámetros "Ton" y "Toff" definen el tiempo en alto y bajo de la señal, entonces mediante la manipulación de uno de ellos es posible variar el ancho del pulso o ciclo de trabajo sin alterar la frecuencia de la señal, dado que se cumple que "T=Ton+Toff".

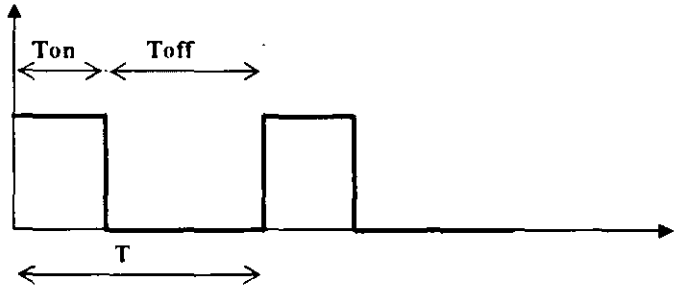


Fig. 5.1 Parámetros programables del tren de pulsos.

El módulo que vamos a diseñar es capaz de manipular digitalmente la duración del parámetro "T" y "Ton" mostrados en la Fig 5.1 utilizando una palabra de control para cada uno de ellos. El diagrama a bloques que ilustra la estructura del generador de pulsos se muestra en la Fig. 5.2.

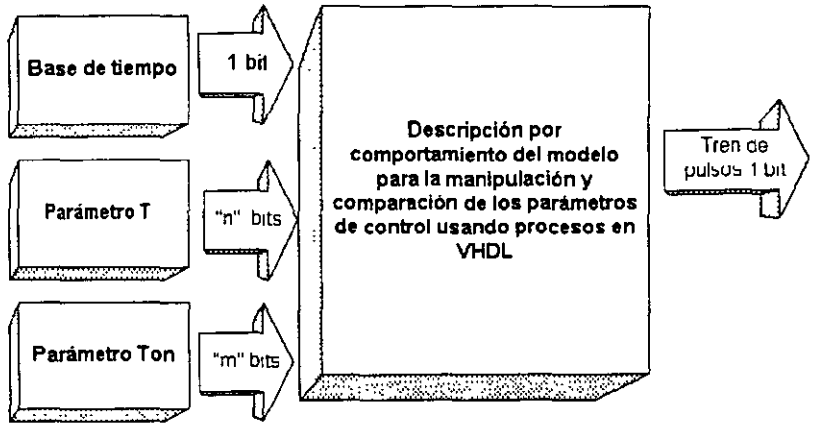


Fig 5 2. Diagrama a bloques del generador de tren de pulsos

Refiriéndonos a la Fig. 5.2 tenemos que la base de tiempo es un oscilador que nos entrega una señal cuadrada con ciclo de trabajo del 50 %, cuyo periodo "Tosc=1/Fosc" es conocido y en función de la cual se generarán frecuencias en el intervalo Fosc/2 hasta una frecuencia "Finferior" esta última está en función del número de bits empleados para representar el parámetro "T". El ancho del pulso "Ton" está en función del número de bits del parámetro "Ton" y puede estar en el intervalo "Tosc" que es el periodo del oscilador hasta "Toff =Tosc". Es decir el ancho del pulso en alto o en bajo nunca puede ser menor a un periodo de la señal del oscilador

Los parámetros anteriores se programan en registros contadores de "n" y "m" bits como se observa en la Fig 5.2, los cuales se ingresan al algoritmo descrito en VHDL que genera el tren de pulsos correspondiente Es importante señalar que los parámetros de programación "T" y "Ton" pueden ser manipulados con el dispositivo funcionando lo cual permite variar la frecuencia y ancho del pulso del tren de pulsos en cualquier instante; o por el contrario si la aplicación lo requiere, pueden ser establecidos como constantes con lo cual se reduce el espació de silicio requerido. Adicionalmente la precisión de los parámetros están a libertad del usuario asi como en función de la frecuencia del oscilador de la base de tiempos.

A continuación presentamos el código VHDL que implementa el generador digital programable de tren de pulsos

```
library ieee,
use ieee std_logic_1164.all;
use ieee.std_logic_arith.all,
ENTITY generador4 IS
PORT(                                                    -- (A)
        up, down        : IN    bit;
        clk             : IN  bit,
        escala            IN  bit_vector (2 downto 0);
        trabajo         : IN  unsigned (4 downto 0);
        salida          : OUT bit);

END generador4;
```

```
ARCHITECTURE comportamiento OF generador4 IS

signal ttl: bit;                                        -- (B)
signal reloj :std_logic;
signal incremento: unsigned(25 downto 0);

BEGIN

PROCESS (reloj)                                         -- (C)
BEGIN
  if(reloj'event and reloj='1') then
    if(up = '0') then
      incremento<= incremento + "00000000000000000000000001";
    else if(down ='0') then
          incremento<= incremento - "00000000000000000000000001";
        end if;
    end if;
  end if;
END PROCESS;

process (clk)                                           -- (D)
variable temporal : unsigned (25 downto 0);
begin
  if(clk'event and clk ='1') then
    if(temporal < (trabajo & "00000000000000000000")) then   --33554432
      ttl<='1';
      temporal:=temporal + incremento;--"10000000000000000";
    else
      ttl <='0';
      temporal:=temporal + incremento,--"10000000000000000";
    end if;
   end if;
end process;

process (clk)                                           -- (E)
variable temporal: unsigned(24 downto 0);
variable auxiliar: std_logic_vector(24 downto 0);
begin
  if(clk'event and clk='1') then
    temporal:=temporal+1;
    auxiliar:=CONV_STD_LOGIC_VECTOR(temporal,25),
    case escala is
      when "000" => reloj <= auxiliar(24);
      when "001" => reloj <= auxiliar(20);
      when "010" => reloj <= auxiliar(18);
      when "011" => reloj <= auxiliar(14);
      when "100" => reloj <= auxiliar(8);
      when "101" => reloj <= auxiliar(6);
      when "110" => reloj <= auxiliar(4),
      when "111" => reloj <= auxiliar(2);
    end case,
  end if,
end process;
salida <= ttl;
END comportamiento,
```

El código anterior maneja los controles programables de frecuencia y tiempo de pulso de acuerdo a las necesidades del usuario, dependiendo de la aplicación estos valores podrán ser constantes con lo que se ahorra espacio de semiconductor. En la sección (A) declaramos los puertos de entrada, para fines de pruebas, este código ajusta el control programable de la frecuencia de operación mediante un par de botones de incremento y decremento, permitiendo un aumento o disminución rápido si se mantiene pulsada la tecla correspondiente. El puerto "trabajo" permite programar la duración en alto del pulso mediante una combinación binaria, el puerto "escala" permite elegir la sensibilidad de la botonera de ajuste de la frecuencia; el puerto salida entrega la señal que es calculada por el módulo El sistema emplea una señal de reloj de 25.175 MHz que corresponde al oscilador disponible en la tarjeta prototipo UP1 donde se realizo la prueba.

En la sección (B) del código se declaran señales internas para controlar el incremento del contador que divide la frecuencia y así poder variar la frecuencia de la señal a generar mediante cambios en el valor del incremento de los contadores.

El proceso de la sección (C) modifica el contenido del contador donde se programa el valor de la frecuencia en función de la botonera de incremento y decremento, la actualización de este registro se realiza en incrementos unitarios con la frecuencia de la señal interna "reloj" que se genera de acuerdo a "escala" para controlar la sensibilidad de la botonera, de manera que si se mantiene presionado alguno de los controles, la frecuencia aumenta o disminuye rápidamente.

El proceso de la sección (D) actualiza los contadores de la división de frecuencia sumando el contenido del dato programado en la señal "incremento", dejando así establecida una cierta frecuencia. La sentencia "if" contenida en el mismo proceso controla el tamaño del pulso en función de la señal de entrada "trabajo". Finalmente el proceso de la sección (E) ajusta la sensibilidad de la botonera de acuerdo a la combinación de la señal de entrada "escala".

5.3 RESULTADOS.

El diseño presentado en la sección anterior se programó en la tarjeta de desarrollo UP1, haciendo pruebas para distintas combinaciones de frecuencia y ancho de pulso, revisando los resultados utilizando un osciloscopio comercial. Las características del generador digital programable de tren de pulsos se presentan en la Tabla 5.1, programado sobre la UP1 utilizando el dispositivo FLEX EPF10K20, con el oscilador de 15.175 MHz

Característica	Valor mínimo	Valor máximo
Niveles de Salida	TTL	
Frecuencia de salida del tren de pulsos (Fsal)	1 Hz	12.5875 MHz
Duración del pulso en alto	40 ns	(1/Fsal) – 40 ns
Escala de sensibilidad	0.75 Hz	3.146875 MHz
Porcentaje utilizado del chip	15 %	
Frecuencia máxima de operación base	31.23 MHz	

Tabla 5 1 Características del generador

En las Fig 5.3 se ilustra el equipo completo después de programar el dispositivo, se puede observar en el osciloscopio una señal de prueba En la Fig. 5.4 se muestra una señal de salida del generador con un ciclo de trabajo del 50 % En la Fig. 5.5 se ilustran los oscilogramas para los casos de pulsos con ciclos de trabajo extremos en alto y bajo. Por tanto podemos observar que el funcionamiento del generador es óptimo.

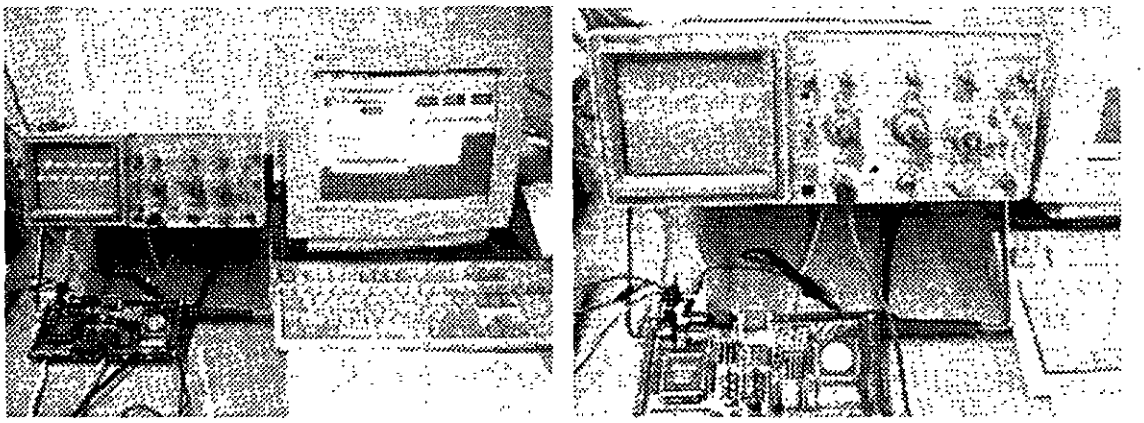


Fig. 5.3 Equipo completo después de programar la UP1 con el generador

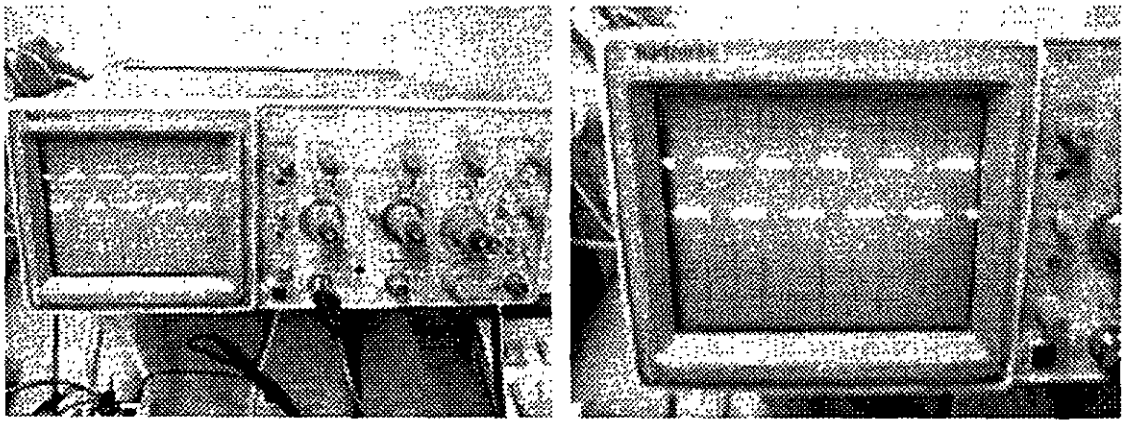


Fig. 5.4. Tren de pulsos con ciclo de trabajo del 50 %

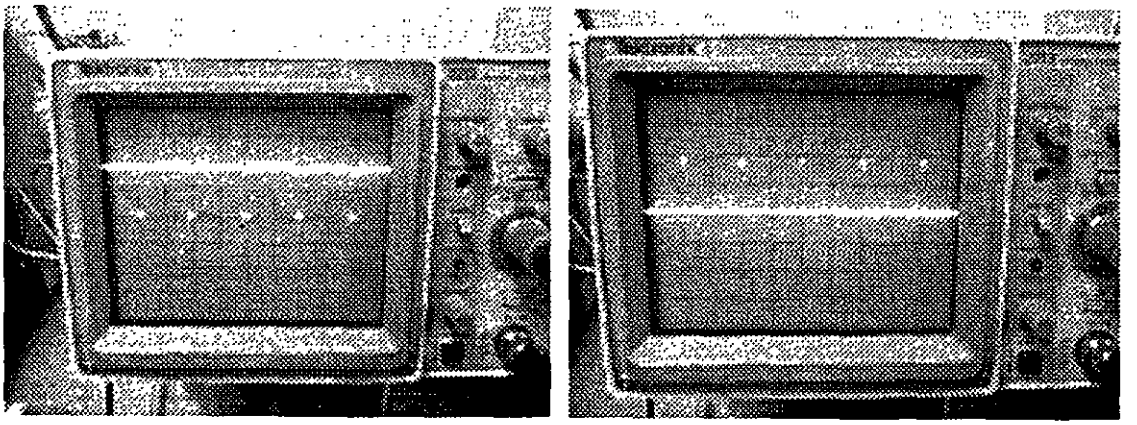


Fig. 5.5 Tren de pulsos con ciclos de trabajo extremos

CAPÍTULO 3

CASO DE APLICACIÓN FRECUENCÍMETRO DIGITAL

CONTENIDO DEL CAPÍTULO

3.1 INTRODUCCIÓN

En este capítulo vamos a mostrar un caso de aplicación para el diseño e implantación de un instrumento conocido como *frecuencímetro* Este caso nos va a permitir ilustrar el flujo de diseño completo de nuestra metodología propuesta

Un frecuencímetro en esencia es un instrumento cuyo objetivo es comparar la frecuencia de una señal base conocida contra la frecuencia de una señal desconocida y obtener el valor de esta última

Si recordamos algunos aspectos del capítulo anterior nos daremos cuenta que nuestra metodología se describe en un diagrama de flujo que representa cada uno de los pasos a seguir y recordemos que hicimos mucho énfasis en realizar modelado por comportamiento, más que modelado estructural Con el objetivo de demostrar la inclinación de nuestra metodología por un modelado por comportamiento, vamos a realizar a lo largo de este capítulo el diseño en forma estructural así como por comportamiento, para entender la importancia de modelar por comportamiento

Se eligió como caso de aplicación este instrumento por varias razones que a continuación presentamos

- Es un instrumento que además del *núcleo* digital requiere de una etapa previa de procesamiento analógico, que no es parte de la metodología
- Requiere de estrategias de optimización para lograr cumplir con los requerimientos especificados, por tanto requiere de una interfaz de salida optimizada como la propuesta en la metodología
- Tiene la complejidad suficiente para comparar las ventajas y desventajas entre modelado estructural y por comportamiento
- Existe un frecuencímetro digital creado en el Centro de Instrumentos y diseñado a la manera tradicional usando componentes discretos TTL, contra el cual se puede comparar las ventajas y desventajas de una implantación de instrumentos utilizando Dispositivos Lógicos Programables empleando la metodología del capítulo anterior

3.2 ESPECIFICACIÓN Y ANÁLISIS DEL FRECUENCÍMETRO

En esta etapa evaluaremos las especificaciones del instrumento que son proporcionadas por quien solicita la creación del mismo con el objetivo de considerar la factibilidad de las mismas. Adicionalmente se hará un análisis del instrumento que nos permita aislar perfectamente lo que será el núcleo del mismo

ESPECIFICACIÓN DEL INSTRUMENTO

Las especificaciones a continuación mostradas son proporcionadas por quien solicita la creación del instrumento

Se requiere crear un frecuencímetro digital que es un instrumento electrónico de medición que compara el valor de la frecuencia de una señal periódica desconocida contra la frecuencia de otra señal periódica conocida, permitiendo así determinar la frecuencia de la señal desconocida
La filosofía del instrumento requerido está fundamentada en optimizar la lógica del frecuencímetro con el objetivo de liberar recursos y poder así medir un intervalo de frecuencia más amplio utilizando los mismos recursos. Se requiere medir frecuencias de señales periódicas de forma de onda desconocida en el intervalo de 1Hz a 10 MHz y voltajes en el intervalo 2Vpp a 20 Vpp. Se requiere adicionalmente bajo costo

Analizando las especificaciones proporcionadas podemos extraer las siguientes características que considera nuestra metodología

? **Velocidad de operación del instrumento (Ancho de Banda)**

Observamos que requerimos medir una variable física llamada frecuencia, que es una señal de voltaje con un ancho de banda de hasta 10 MHz. Dado que vamos a implantar el instrumento en un Dispositivo Lógico Programable es necesario saber si soporta el ancho de banda requerido

Remitiéndonos a las hojas de datos técnicos de los Dispositivos Lógicos Programables de Altera [3], resulta claro que uno de los dispositivos mas comerciales como el de la Familia MAX 7000S, tiene posibilidad de operar hasta un límite crítico de 100 MHz, por tanto puede cumplir con toda facilidad la especificación requerida

? **Resolución de las variables de entrada al instrumento.**

De las especificaciones dadas es claro que no se trata de un instrumento 100 % digital porque la señal de frecuencia desconocida es de forma de onda no determinada y de amplitud comprendida entre 2Vpp y 20 Vpp. Esto plantea la necesidad de una etapa analógica de procesamiento no considerada como parte de la metodología

Sin embargo en esta parte el objetivo es determinar la resolución de la variable que finalmente ingresará al "núcleo" del instrumento. La especificación implica un frecuencímetro de solo un canal de medición, por tanto después del adecuamiento de la señal el "núcleo" recibirá únicamente una señal TTL, que es una secuencia de 1's y 0's por tanto el instrumento tiene entrada de un bit. No se requiere de ningún tipo de convertidor A/D sino solamente del adecuador de señal. De esta manera las entradas al núcleo del instrumento no representan problema alguno en cuanto al numero de pines requeridos

? **Precisión y exactitud del instrumento.**

Estas características inherentes a cualquier instrumento no fueron especificadas por quien solicita el instrumento sin embargo es necesario especificarlas, por ejemplo la precisión del instrumento debe ser tal que lecturas sucesivas de una señal con frecuencia estable deben ser repetibles (poco cambiantes una de la otra). La exactitud debe ser tal que no se desvíe del valor real de la variable medida

En este caso de aplicación vamos a permitir que estas tres características queden en función exclusivamente del modelo de solución generado y verificaremos que al final estén dentro de los límites razonables y tolerados por instrumentos de medición parecidos

ANÁLISIS DEL INSTRUMENTO

Nuestro objetivo en esta parte de la metodología es separar el "núcleo" del instrumento del resto de la arquitectura y manifestar por separado las características de cada uno

? **Conocimiento acerca del instrumento a construir.**

Entendidas las especificaciones requeridas por el instrumento es necesario conocer la teoría básica del instrumento que vamos a construir. Sabemos que un frecuencímetro es un instrumento comun en el área de la instrumentación

Esencialmente el frecuencímetro digital es un instrumento electrónico de medición que compara el valor de la frecuencia de una señal periódica desconocida contra la frecuencia de otra señal periódica conocida, permitiendo así determinar la frecuencia de la señal desconocida

La comparación consiste en contar el número de pulsos que ocurren durante el tiempo de duración de una ventana de comparación obtenida a partir de una señal de base. Dado que el tiempo de comparación depende de la señal base, entonces la precisión, exactitud y tolerancia finales del instrumento dependen de la ventana de comparación

Por tanto es necesario que esta señal base sea una señal de entrada con características de alta precisión y exactitud a partir de la cual se genere la ventana de comparación. Una señal de esta calidad se puede obtener a partir de un oscilador de cristal dado que su frecuencia es muy estable. De la característica de este cristal y de la ventana de comparación generada dependerá la calidad de nuestro instrumento

Los detalles del conteo de pulsos, retención de la cuenta y ventana de comparación son parte ya del modelado del "núcleo" del instrumento

? **Definición del "núcleo" del instrumento**

Aquí es esencial determinar el núcleo del instrumento, es decir, separar las secciones que no forman parte del algoritmo de procesamiento

Hemos determinado que el algoritmo esencial del instrumento es el conteo de pulsos de la señal desconocida durante una ventana de comparación generada por la señal base. De esta manera nuestro núcleo va a recibir tanto la señal desconocida ya adecuada a TTL así como la señal de referencia y nos va a entregar el número de pulsos ocurridos en la señal desconocida durante la ventana de comparación. Esta idea se ilustra gráficamente en la Fig. 3.1

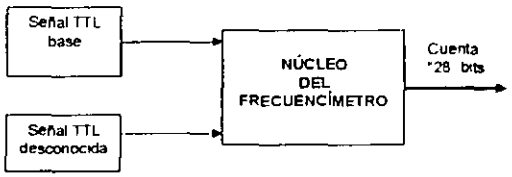


Fig 3 1 Núcleo del frecuencímetro

? **Definición de la interfaz de entrada**

Nuestro instrumento en realidad solo tiene una variable de entrada que es la señal de forma de onda y amplitud desconocida que deseamos medir y que después de ser acondicionada por un módulo analógico es ingresada al nucleo del instrumento Adicionalmente tenemos otra variable de entrada que es generada dentro del instrumento y proporcionada al núcleo como señal de entrada Como observamos en la Fig 3 1 ambas son señales de un bit y la interfaz de entrada no requiere ningún tipo de optimización como en el caso de una multiplexión de muchas variables de entrada de resoluciones variables

El unico aspecto importante incluido en esta etapa es el procesamiento previo requerido por la señal de entrada al instrumento, el cual debe ser realizado por un módulo analógico que no es objeto de análisis en el presente trabajo

? **Definición de la interfaz de salida.**

Este denominador común nos plantea la necesidad de definir la forma en que la salida del núcleo, en nuestro caso el valor de frecuencia de la señal desconocida, será visualizado por el usuario del instrumento

La especificación del instrumento no establece la forma especifica en la cual visualizar el conteo de la frecuencia Sin embargo si ponen la restricción de "bajo costo" por tanto podemos considerar un despliegue utilizando displays de siete segmentos

Esta consideración afecta el diseño del nucleo porque entonces estamos requiriendo que la cuenta entregada por el núcleo sea en código BCD, es decir cuatro bits por cada digito de la cuenta y dado que la especificación solicita una frecuencia máxima medible de 10 MHZ, requerimos de siete digitos y por tanto el núcleo tendrá 28 bits de salida

3.3 MODELADO DEL FRECUENCÍMETRO

En esta sección vamos a modelar el núcleo del instrumento asi como las interfaces de entrada y salida que fueron analizadas en la sección anterior Recuerde que nuestra metodologia recomienda usar siempre un modelado por comportamiento, sin embargo en esta sección utilizaremos también el modelado estructural Esto única y exclusivamente con el fin de demostrar las ventajas que tiene el modelado por comportamiento sobre cualquier otro tipo de modelado en este caso el estructural El modelado por comportamiento lo haremos utilizando el lenguaje de descripción de hardware VHDL y el estructural utilizando el ambiente gráfico de la herramienta Todo esto con el software MAX+PLUS II y CPLDs de Altera

Como sabemos tenemos que modelar tres partes que son

? Núcleo del instrumento
? Interfaz de entrada
? Interfaz de salida

Sabemos que este instrumento requiere de una etapa de procesamiento previo a la señal de entrada y por tanto tiene que ser diseñado separadamente ajustándose a las especificaciones del instrumento y a los requerimientos del nucleo del instrumento, por un equipo de trabajo con experiencia en el área de adecuadores de señal

3.3.1 Modelado estructural del núcleo del frecuencímetro.

Modelar el núcleo del instrumento ilustrado en la Fig 3 1 en forma estructural, consiste en diseñar una "*arquitectura*" que permita realizar el conteo de pulsos, la generación de la ventana de tiempo y la retención de la medición

Para lograrlo podemos construir un esquemático basado en compuertas lógicas, unidades funcionales como contadores y todos los demás elementos del diseño digital arreglados de forma tal que realicen las funciones requeridas por el núcleo del instrumento que son

? Generar la ventana de comparación
? Realizar el conteo de pulsos durante la ventana de comparación
? Retener la lectura del conteo
? Lógica de control

La estructura del núcleo del frecuencimetro aislado de las interfaces de entrada, salida, y acondicionador de señal se muestra en la Fig 3 2 a nivel de diagrama de bloques La señal periódica TTL de entrada cuya frecuencia es desconocidas se alimenta al núcleo del frecuencimetro Una segunda señal TTL proveniente de un "Oscilador de cristal a 8 MHz" sirve como la señal periódica conocida para generar la ventana de comparación, el valor de la frecuencia se elige asi por el bajo costo del cristal. En el interior del núcleo tenemos un bloque llamado "Divisores por décadas" cuya función es recibir la señal del oscilador a 8MHz y generar la señal base de un segundo, se elige una ventana de comparación de un segundo para facilitar la conversión de la cuenta final El bloque denominado "Lógica de Control" recibe la señal base de un segundo y la señal TTL cuya frecuencia es desconocida y genera un conjunto de señales de control que permiten al bloque denominado "Contadores BCD" iniciar la cuenta de flancos de subida a la frecuencia de la señal TTL desconocida durante el intervalo de duración de la señal base de un segundo La lógica de control también manipula el bloque llamado "Registros de retención" el cual permite retener el valor de la cuenta obtenida en los contadores al final de la señal base de un segundo

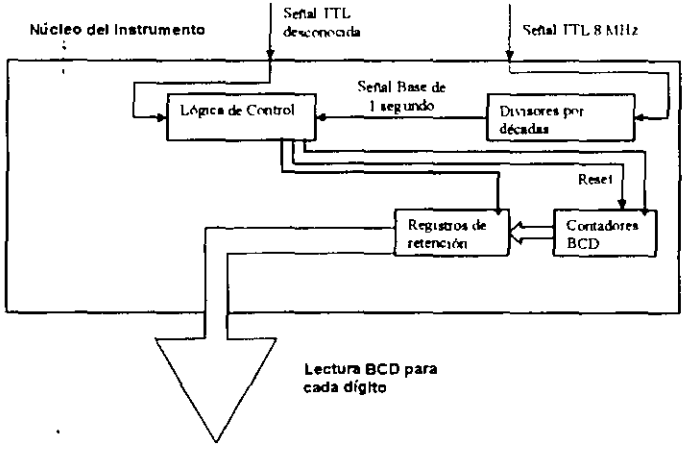


Fig 3 2 Diagrama a bloques del núcleo estructural del frecuencímetro

La arquitectura de la Fig 3 2 es traducida al ambiente de captura esquemática de Max+Plus II que es del tipo gráfico Cada uno de los bloques funcionales pueden ser diseñados a nivel de compuertas o buscar equivalentes en las librerías incluidas con la herramienta para reducir tiempos de diseño y por tanto limitarse a interconectar los diferentes bloques propuestos por la arquitectura

Ventana de Comparación para generar la Señal Base de 1 segundo.

Característica esencial para el funcionamiento preciso y exacto del instrumento es la generación de una base de tiempo muy estable, razón por la cual se utiliza un oscilador de cristal de cuarzo Para nuestra aplicación requerimos una base de tiempo de 1 segundo de duración esto significa que debemos bajar la frecuencia del oscilador de 8 MHz hasta 1 Hz Este proceso de dividir la frecuencia es realizado por el bloque titulado "Divisores por décadas" de la Fig 3 2 La Fig 3 3 ilustra el esquemático del circuito que genera la señal base de 1 segundo a partir del oscilador de 8 MHz, este esquemático está creado en MAX+PLUS II El "Divisor por décadas" está constituido por dos etapas, la primera es ilustrada en la Fig 3 3 (a) que es un divisor entre ocho con un ciclo de trabajo del 50 %, de manera que a la salida de esta etapa tenemos una señal con una frecuencia de 1 MHz, la segunda es ilustrada en la Fig 3 3 (b) que es un divisor entre diez mediante el uso de contadores Johnson y posteriormente con el latch SR a la salida del contador es posible generar una señal con un nivel negativo de solo 1 ?s Como se requiere una señal base de 1 segundo entonces son necesarias seis etapas como la mostrada en la Fig 3 3 (b) En la Fig 3 4 se muestra la forma de la señal base de 1 segundo después de los divisores por décadas, como se observa es

una señal de 1 segundo con un nivel negativo de 1 ?s Un problema encontrado al implantar en forma estructural esta arquitectura resulta en que el contador Jonson no existe en las librerías de Altera, debido a que no se fabrica comercialmente como integrado TTL, sino solamente como integrado CMOS, de forma tal que fue necesario implementarlo a nivel de compuertas y flip-flops como se ilustra en la Fig 3 5

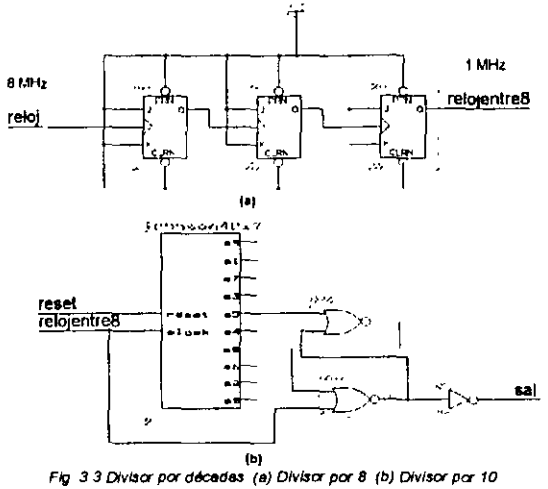


Fig 3 3 Divisor por décadas (a) Divisor por 8 (b) Divisor por 10

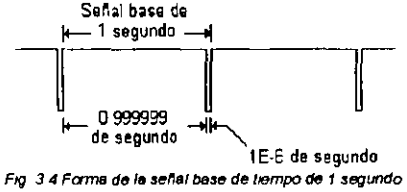


Fig 3 4 Forma de la señal base de tiempo de 1 segundo

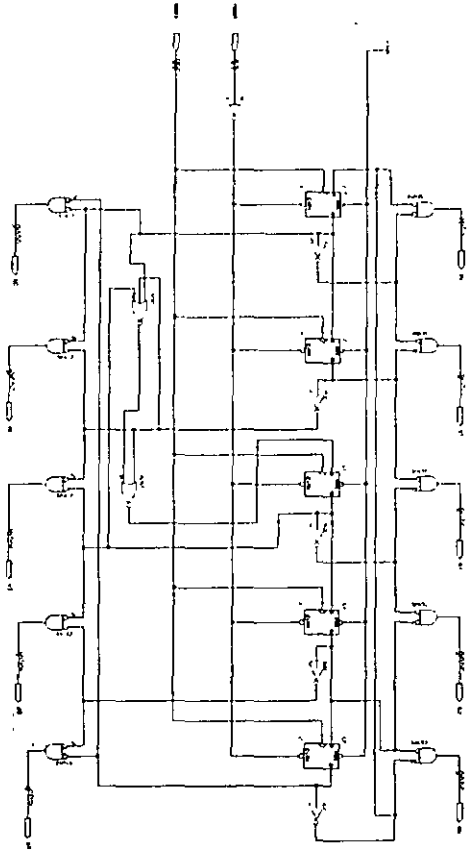


Fig 3 5 Implementación del contador Johnson.

Conteo de Pulsos durante la Ventana de Comparación

Después de generar la base de tiempo de un segundo, el objetivo del instrumento es contar el numero de pulsos de la señal TTL de frecuencia desconocida durante un segundo, este número corresponderá directamente con la frecuencia de la señal desconocida en unidades Hz Como nuestro objetivo es poder medir hasta 10 MHz haciendo un despliegue en display de 7 segmentos entonces requerimos la utilización de siete contadores BCD conectados en cascada Esta etapa del conteo de pulsos está etiquetada como "Contadores BCD" en la Fig 3 2 En la Fig 3 6 se ilustra el esquemático de la etapa de conteo utilizando MAX+PLUS II Los siete contadores BCD están conectados en cascada La frecuencia de conteo está controlada directamente por la frecuencia de la señal TTL que se desea medir El intervalo de tiempo durante el cual los contadores funcionan está controlado por la señal base de tiempo de 1 segundo De esta forma al final de la señal base de 1 segundo cada uno de los contadores BCD contienen cada una de las cifras de la frecuencia medida desde la menos significativa (LSB) hasta la más significativa (MSB) en unidades Hertz

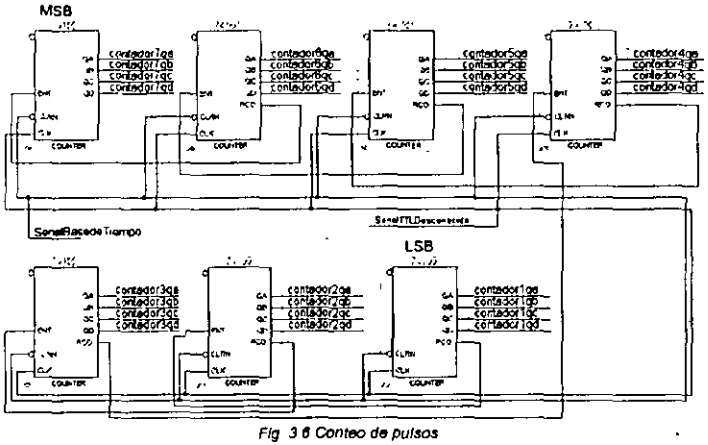


Fig 3 6 Conteo de pulsos

Como podemos observar en la Fig 3 6 estamos utilizando contadores BCD típicos de la serie 74, en específico el contador 74160 que entre sus características de interés para nuestra arquitectura esta su capacidad para conectarse en cascada, así como el reset asincrono Los contadores están conectados en forma síncrona

Retención de la Lectura de Conteo

Después del conteo de pulsos y por tanto de la obtención de la frecuencia de la señal TTL que está siendo medida es necesario retener esta lectura para proceder a su despliegue posterior en display de siete segmentos Esta etapa está conformada por registros constituidos por Flip-Flops tipo D El circuito mostrado con la etiqueta "Registros de retención" en la Fig 3 2 es detallado en la Fig 3 7

Estos circuitos de retención son controlados por la señal base de tiempo de un segundo pero invertida Esto significa que antes de iniciar la siguiente cuenta en la etapa de conteo primero se retiene la cuenta actual en el circuito de retención

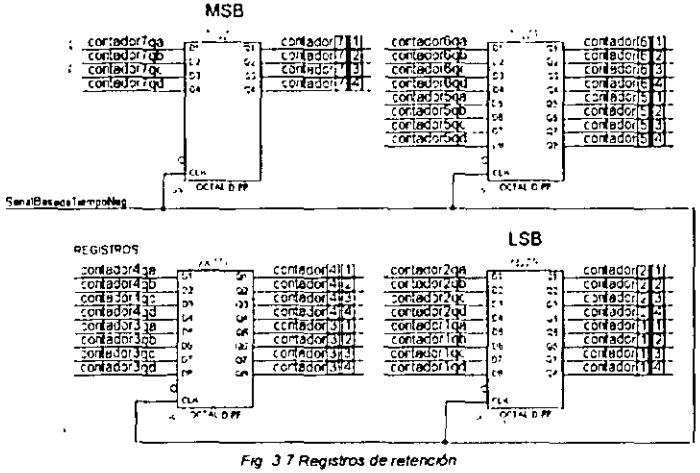


Fig 3 7 Registros de retención

Como podemos observar en la Fig 3 7 estamos utilizando Flip-Flop "D" tipicos de la serie 74, en especifico el 74273 conectados en forma síncrona

Lógica de Control

El bloque etiquetado "Lógica de Control" en la Fig 3 2, es el encargado de controlar y manipular la arquitectura descrita en las secciones anteriores En la Fig 3 8 se ilustra en detalle el circuito para la lógica de control El circuito recibe una señal de 1 KHz que se ingresa a un contador módulo 3, este contador controla un decodificador 3 a 8 el cual finalmente controla tanto los circuitos tres estados de la etapa posterior así como los transistores que habilitan los display de 7 segmentos de la misma etapa, con el objeto de compartir el decodificador BCD – 7 segmentos como se explicará en la interfaz de salida Adicionalmente se generan las señales de reloj y reloj invertido para el resto de los componentes ilustrados en la Fig 3 2 mediante el uso de la base de tiempo de un segundo así como de la señal TTL a medir a través de una compuerta AND

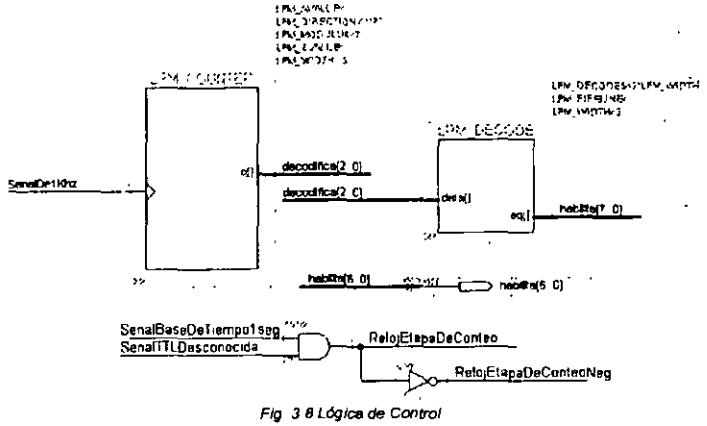


Fig 3 8 Lógica de Control

3.3.2 Modelado estructural de la interfaz de entrada del frecuencimetro.

Como podemos observar en la Fig 3 1 el núcleo solo tiene dos entradas cada una de un bit, por tanto no plantea la necesidad de ningun esquema de multiplexión a la entrada Sin embargo estas dos señales que recibe el nucleo requieren ser previamente acondicionadas y aunque no es considerado parte del presente trabajo, se muestra el acondicionador diseñado por *Quintana*[10]; tomamos unos segundos para presentar este módulo acondicionador

Acondicionador a niveles TTL

El bloque "Acondicionador a niveles TTL" tiene por función adecuar cualquier señal periódica de frecuencia, amplitud y offset desconocidos a una señal TTL de la misma frecuencia y sin offset. Esta etapa analógica está diseñada para recibir señales periódicas en el intervalo 2 Vpp a 20 Vpp con posible offset positivo o negativo

Esta etapa es importante porque finalmente genera una señal TTL de frecuencia igual a la señal que se desea medir, de forma tal que la señal TTL pueda ser manipulada por la lógica del frecuencimetro contenida en el CPLD

La Fig 3 9 ilustra el diseño de la etapa analógica para el acondicionamiento de la señal de entrada a niveles TTL Como primer tratamiento de la señal a medir, tenemos a la entrada un capacitor de 47?F para eliminar la componente de DC, a continuación se conecta una resistencia de 1k, la cual restringe la señal a una corriente que no resulte peligrosa para el resto del circuito, un par de

diodos 1N4148 recortan la señal a que permanezca dentro del rango de polarización Finalmente está señal entra al comparador LM360 que se conectó a tierra por la entrada positiva para dtectar cruces por cero, de esta manera está comparando contra una referencia constante de 0V el resto de los componentes los recomienda el fabricante del comparador

Esta interfaz logra un ancho de banda de 0 1 Hz hasta 10 MHz para señales de entrada de 2Vpp a 20Vpp gracias a que el comparador LM360 es de alta velocidad

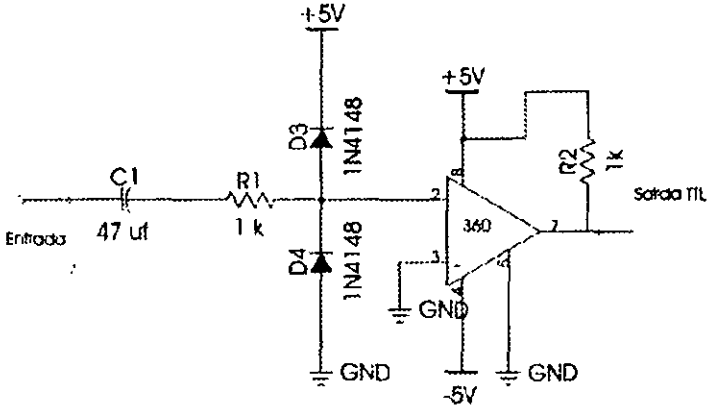
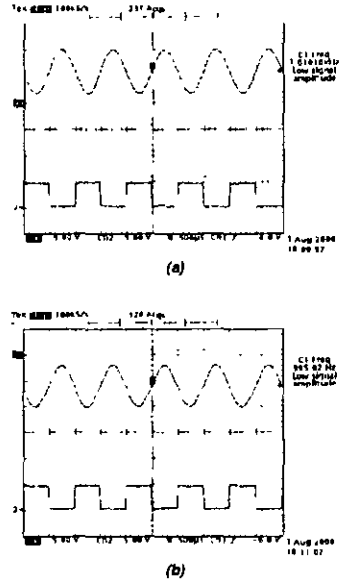


Fig 3 9 Acondicionador de la señal de entrada a niveles TTL

En la Fig 3 10 se muestran los osciloramas resultado de pruebas diversas del circuito acondicionador a niveles TTL de la Fig 3 9 En la Fig 3 10 (a) se tiene como señal de entrada una senoidal de 1 KHz, 8 Vpp y un offset de +6 5 V y como señal de salida del acondicionador una señal TTL también de 1 KHz En la Fig 3 10 (b) se tiene como señal de entrada una senoidal de 1 KHz, 8Vpp y un offset de – 6 5 V y como señal de salida del acondicionador una señal TTL también de 1 KHz Como podemos observar el circuito diseñado cumple con su función de acondicionar cualquier señal periódica de entrada de frecuencia "Fx" a una señal de salida TTL de frecuencia "Fx"



(a)

(b)

Fig 3 10 Resultados del circuito acondicionador a niveles TTL (a) Señal senoidal con offset positivo (b) Señal senoidal con offset negativo

3.3.3 Modelado estructural de la Interfaz de salida del frecuencímetro.

Como podemos observar en la Fig 3 1 y 3 2 el núcleo solo tiene una salida conformada por siete grupos codificados en BCD, por tanto tenemos 28 bits de salida que deben ser desplegados en displays de siete segmentos

Dado que la señal base de un segundo es utilizada para realizar la cuenta durante ese tiempo, la actualización de las lecturas se efectua a una frecuencia de un Hertz Por esta razón no requerimos de una interfaz de salida rápida, por consiguiente es conveniente multiplexar los siete grupos de salidas con el objetivo de compartir un solo convertidor BCD – Siete Segmentos optimizando espacio de silicio en el CPLD

La arquitectura de la interfaz de salida se ilustra en el diagrama a bloques de la Fig 3 11. Observamos que los circuitos de retención localizados en el núcleo envían la información a la interfaz de salida que está constituida por circuitos tres estados cuya función es multiplexar los siete grupos de cuatro bits con el objetivo de compartir el decodificador BCD – 7 Segmentos

La salida del decodificador es enviada usando un bus comun a todos los displays los cuales son controlados por las lineas de selección generadas en el módulo de lógica de control ubicada en el nucleo de la aplicación

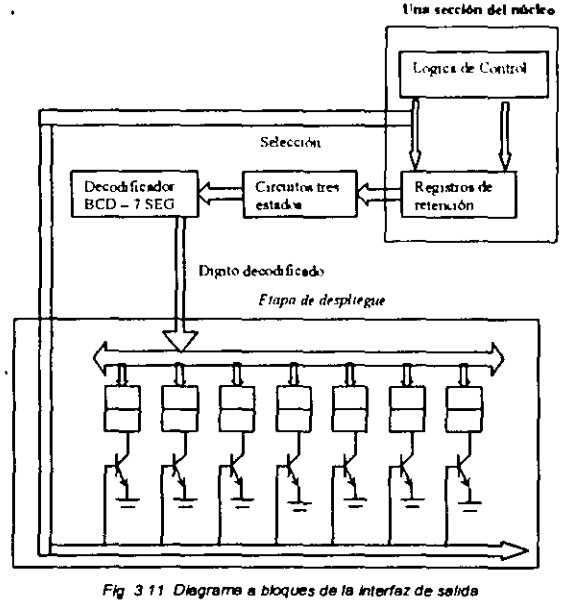


Fig 3 11 Diagrama a bloques de la interfaz de salida

En la Fig 3 12 se ilustra el esquemático de detalle diseñado en Max+Plus II para implementar el diagrama a bloques de la interfaz de salida ilustrada en la Fig 3 11

Para lograr optimizar recursos del CPLD que nos permitan utilizar un solo dispositivo EPM7128SLC84-15 para toda la lógica del frecuencimetro es necesario que los siete digitos contenidos en los registros de retención compartan el mismo decodificador BCD – 7 Segmentos Este requerimiento nos obliga a multiplexar el decodificador mediante la utilización de circuitos tres estados disponibles en los pines de salida del dispositivo Como se puede observar en la Fig 3 12 las salidas de los circuitos de retención están conectadas a circuitos tres estados los cuales conforman un bus común que dependiendo del estado de las líneas de comando de los circuitos tres estados se elige el correspondiente digito BCD convertido a 7 Segmentos Es conveniente referirse a la Fig 3 8 para recordar la frecuencia de multiplexión

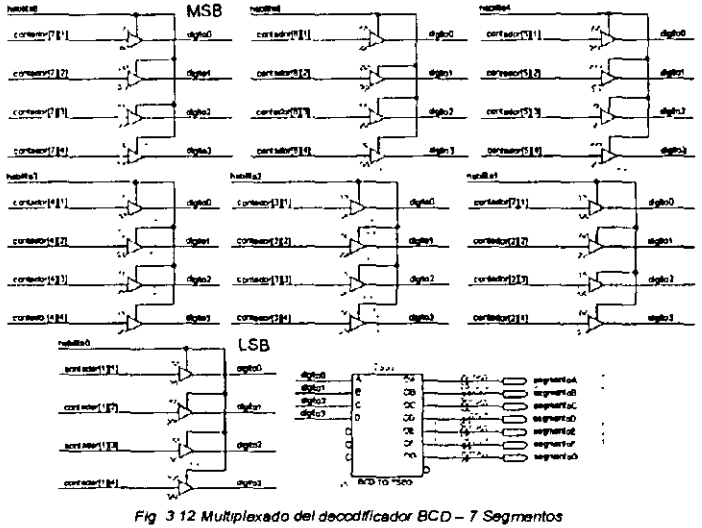


Fig 3 12 Multiplexado del decodificador BCD – 7 Segmentos

3.3.4 Modelado por comportamiento del núcleo del frecuencímetro.

Modelar el núcleo del instrumento ilustrado en la Fig 3 1 por comportamiento, consiste en describir la funcionalidad del núcleo, que textualmente es "contar el número de pulsos de la señal desconocida durante un segundo" Esta descripción se presenta en el siguiente listado utilizando el lenguaje VHDL

```
-- Nucleo del frecuencimetro
LIBRARY ieee,
USE ieee.std_logic_1164.all,
USE ieee.std_logic_arith.all,

ENTITY frecuencimetro IS
   PORT(
      reloj, desconocida          : IN    STD_LOGIC,
      dividido                    : OUT   STD_LOGIC,
      m0, m1, m2, m3, m4, m5, m6  : OUT   INTEGER RANGE 0 TO 9),
  END frecuencimetro;

ARCHITECTURE nucleo OF frecuencimetro IS
   SIGNAL relojbase : STD_LOGIC;
   SIGNAL cero, uno, dos, tres, cuatro, cinco, seis : INTEGER RANGE 0 TO 9;
```

```
BEGIN
        PROCESS (reloj)         -- proceso "A"
        VARIABLE cuenta : INTEGER RANGE 0 TO 8000000
        BEGIN
          IF (reloj'EVENT AND reloj = '1') THEN
             IF (cuenta < 8000000) THEN
               relojbase <= '0';
               cuenta  = cuenta+1
             ELSE
               relojbase <= '1';
               cuenta  = 0;
             END IF;
          END IF;
END PROCESS;

dividido <= relojbase           -- asignación "B"

PROCESS (desconocida, relojbase)      -- proceso "C"
  BEGIN
    IF (desconocida'EVENT AND desconocida = '1') THEN
           IF (relojbase = '0') THEN
        cero <= cero+1;
      END IF;

      IF (relojbase = '0' and cero=9) THEN
        uno <= uno+1;
        cero <= 0;
      END IF;

      IF (relojbase = '0' and uno=10) THEN
        dos <= dos+1;
        uno <= 0;
      END IF;

      IF (relojbase = '0' and dos=10) THEN
        tres <= tres+1;
        dos <= 0;
      END IF;

      IF (relojbase = '0' and tres=10) THEN
        cuatro <= cuatro+1;
        tres <= 0;
      END IF;

      IF (relojbase = '0' and cuatro=10) THEN
        cinco <= cinco+1;
        cuatro <= 0;
      END IF;

      IF (relojbase = '0' and cinco=10) THEN
        seis <= seis+1;
        cinco <= 0;
      END IF;

      IF (relojbase = '0' and seis=10) THEN
        seis <= 0;
```



```
      END IF;

      END IF;
      IF (relojbase'EVENT and relojbase = '1') THEN
        m0 <= cero; m1 <= uno; m2 <= dos; m3 <= tres; m4 <= cuatro; m5 <= cinco; m6 <= seis;
      END IF;
 END PROCESS
END nucleo;
```

La descripción define una entidad que recibe dos señales de entrada llamadas "reloj" y "desconocida", la primera de ellas es la señal de reloj utilizada para el funcionamiento del instrumento y la segunda constituye la señal de frecuencia desconocida Se definen dos salidas llamadas "dividido" y el conjunto "m0, m1, m2, m3, m4, m5, m6", la primera entrega la base de tiempo generada de un segundo y el conjunto de salidas "m[0 6]" entregan el valor de la frecuencia de la señal "desconocida" en formato BCD

La arquitectura definida describe el funcionamiento del núcleo mediante dos procesos y una sentencia de asignación que funcionan paralelamente entre si El proceso "A" describe el funcionamiento necesario para generar la base de tiempo de un segundo a partir de la señal de "reloj" de entrada de 8 MHz, utilizando una variable local al proceso llamada "cuenta" la cual lleva el conteo del número de pulsos Este proceso por tanto genera un pulso en alto por cada 8000000 de pulsos en la señal de "reloj"

La sentencia de asignación "B" se encarga de hacer visible el valor de la señal base hacia el exterior del núcleo para fines de simulación

El proceso "C" describe el conteo de pulsos de la señal "desconocida" durante la duración de la señal "relojbase" generada en el proceso "A", esta señal es de 1 seg Este proceso describe siete contadores tipo BCD conectados en cascada Al final del la señal base de 1 seg El proceso hace visible el valor de la frecuencia de la señal "desconocida" hacia el exterior del nucleo a través de las salidas "m[6 0]"

3.3.5 Modelado por comportamiento de la interfaz de entrada del frecuencimetro.

Como podemos observar el núcleo solo tiene dos entradas cada una de un bit, por tanto no plantea la necesidad de ningún esquema de multiplexión a la entrada Sin embargo estas dos señales que recibe el nucleo requieren ser previamente acondicionadas y aunque no es considerado parte de la metodología, podemos remitirnos a la sección 3 3 2 para recordar el módulo acondicionador

3.3.6 Modelado por comportamiento de la interfaz de salida del frecuencimetro.

Como podemos observar el núcleo solo tiene una salida conformada por siete grupos codificados en BCD, por tanto tenemos 28 bits de salida que deben ser desplegados en displays de siete segmentos

Dado que la señal base de un segundo es utilizada para realizar la cuenta durante ese tiempo, la actualización de las lecturas se efectúa a una frecuencia de un Hertz Por esta razón no requerimos de una interfaz de salida rápida, por consiguiente es conveniente multiplexar los siete grupos de salidas con el objetivo de compartir un solo convertidor BCD – Siete segmentos optimizando espacio de silicio en el CPLD
La descripción funcional de la interfaz de salida se lista a continuación

```
-- Interfaz de Salida del Frecuencimetro
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY interfazsalida IS

        PORT
        (,
                reloj           IN STD_LOGIC;
                valores         IN STD_LOGIC_VECTOR (0 to 27);
                salida_bcd      OUT     STD_LOGIC_VECTOR (0 to 6);
                selector        OUT   STD_LOGIC_VECTOR (0 to 6)
        );

END interfazsalida;

ARCHITECTURE interfaz OF interfazsalida IS
SIGNAL cuenta  INTEGER RANGE 0 to 6;
SIGNAL temporal  STD_LOGIC_VECTOR (0 to 3);
BEGIN
  PROCESS (reloj)  --proceso "A"
  BEGIN
    IF (reloj'EVENT and reloj='1') THEN
      cuenta <= cuenta + 1;
    END IF;
    IF (cuenta = 7) THEN
      cuenta <= 0;
    END IF;
  END PROCESS;

  PROCESS (reloj)  --proceso "B"
  BEGIN
    IF (reloj'EVENT and reloj='1') THEN
      CASE cuenta IS
              WHEN 0 => selector <= "0000001";
               temporal <= valores(0 to 3);
              WHEN 1 => selector <= "0000010";
               temporal <= valores(4 to 7);
              WHEN 2 => selector <= "0000100";
               temporal <= valores(8 to 11);
              WHEN 3 => selector <= "0001000";
               temporal <= valores(12 to 15);
              WHEN 4 => selector <= "0010000";
               temporal <= valores(16 to 19);
              WHEN 5 => selector <= "0100000";
               temporal <= valores(20 to 23);
              WHEN 6 => selector <= "1000000";
               temporal <= valores(24 to 27);
      END CASE;
    END IF;
  END PROCESS;

  PROCESS (temporal)  -- Proceso "C"
  BEGIN
    CASE temporal IS
      WHEN "0000" => salida_bcd <= "0000001";
```



```
      WHEN "0001" => salida_bcd <= "1001111";
      WHEN "0010" => salida_bcd <= "0010010";
      WHEN "0011" => salida_bcd <= "0000110";
      WHEN "0100" => salida_bcd <= "1001100";
      WHEN "0101" => salida_bcd <= "0100100";
      WHEN "0110" => salida_bcd <= "1100000";
      WHEN "0111" => salida_bcd <= "0001111";
      WHEN "1000" => salida_bcd <= "0000000";
      WHEN "1001" => salida_bcd <= "0001100";
      WHEN others => salida_bcd <= "1111111";
    END CASE;
  END PROCESS;

END interfaz;
```

La entidad define como entradas una señal llamada "reloj" que controla la velocidad de multiplexión y un vector de entradas llamado "valores" de 28 bits que recibe las siete cifras de conteo BCD entregadas por el núcleo. Define una única señal de salida llamada "salida_bcd" de siete bits que contiene el código siete segmentos del dígito BCD convertido. Define una señal de salida "selector" de siete bits que habilita el correspondiente displays de siete segmentos para la visualización de la cuenta. La arquitectura describe el comportamiento de la interfaz de entrada utilizando tres procesos trabajando paralelamente.

El proceso "A" describe un contador de tres bits en la señal "cuenta" que será utilizado en el proceso "B". El proceso "B" describe la habilitación en cada pulso de la señal de "reloj" de una de las siete señales del "selector" que controlan la elección del display, esto en función de la señal "cuenta". Este proceso también para cada caso, asigna a la señal temporal el grupo BCD correspondiente recibido a la entrada del módulo. El proceso "C" utiliza la señal "temporal" que contiene un grupo BCD y lo transforma en salida 7 Segmentos correspondiente, la cual es enviada a la salida del módulo llamada "salida_bcd".

3.3.7 Escalas del Instrumento.

Como lo mencionamos en las especificaciones del instrumento, el intervalo de medición va de 1 Hz hasta 10 MHz, por tanto es necesario disponer de por lo menos dos escalas que nos permitan lecturas confiables tanto a bajas frecuencias como a altas frecuencias. Nuestro instrumento implementa dos escalas de medición:

1. La primera escala genera una ventana de comparación de diez segundos utilizando el esquema de la Fig. 3.6, de forma tal que se requiere de otra etapa divisora por diez. Esta escala permite medir con una precisión de hasta 0.1 Hz., por tanto el objetivo es utilizarla para medir frecuencias bajas. Teóricamente esta escala puede ser utilizada para medir hasta una frecuencia máxima de 1 MHz; sin embargo la recomendamos para mediciones de 1 Hz hasta 10 Khz. Por ejemplo al medir una señal de 10 Hz nos dará la información intermedia entre 10 y 11 Hz.

2. La segunda escala genera una ventana de comparación de un segundo utilizando exactamente el esquema de la Fig. 3.6. Esta escala permite medir con una precisión de 1 Hz hasta una lectura máxima de 10 MHz. El objetivo de la escala es ser utilizada para medir frecuencias arriba de los 10 KHz, sin embargo, teóricamente puede medir en el intervalo de 1Hz hasta 10Mhz.

El instrumento puede recibir en los bornes de entrada una señal periódica en el intervalo 2Vpp – 20 Vpp o por el contrario una señal TTL, para lo cual se elige un interruptor de selección manual.

3.4 ELECCIÓN DEL DISPOSITIVO CPLD

Llegamos a una sección de la metodología que impacta directamente el costo del sistema debido a la elección del CPLD a utilizar. Como lo indica nuestra metodología debemos considerar tres características esenciales.

Restricciones de tiempo

Nuestro instrumento debe ser capaz de medir frecuencias desconocidas de hasta 10 Mhz, por tanto el dispositivo elegido debe ser capaz de recibir un tren de pulsos de esa frecuencia sin ningun problema. La familia MAX de Altera permite un funcionamiento cerca de los 100 Mhz, por tanto no representa problema alguno.

Podemos utilizar una herramienta incluida en el software MAX+PLUS II llamada Timing Analyser la cual permite calcular la frecuencia máxima de operación de nuestro diseño sobre el dispositivo utilizado.

Restricciones de espacio

El tamaño en numero de macroceldas o elementos lógicos del dispositivo afecta directamente el costo del instrumento. Sabemos que cualquier diseño implícitamente pretende desde el punto de vista de la ingeniería ser optimo en costo. Es por esta razón que la interfaz de salida se elige del tipo multiplexado utilizando los circuitos tres estados disponibles en todos los pines de salida de la familia MAX. Tomando en cuenta el concepto de macrocelda expuesto en el capítulo 1 tenemos que cada Flip Flop involucrado en el diseño implica una macrocelda, por tanto debemos contar todas las etapas de los contadores y divisores de frecuencia. Adicionalmente cada convertidor BCD a 7 Segmentos requiere de siete funciones lógicas cada una de cuatro variables, por tanto de siete macroceldas. Si la interfaz de salida no fuera multiplexada y utilizaramos siete convertidores BCD 7 Segmentos requeriríamos de 49 macroceldas solo para el despliegue, sin embargo con el esquema propuesto solo se requieren siete macroceldas.

Con este conteo de macroceldas resulta que estamos cerca de las 128, por tanto el dispositivo que elegimos es el EPM7128SLC84-15 de la Familia MAX 7000S.

Restricciones de costos

En este caso no existe un costo tope para nuestro instrumento impuesto por el cliente simplemente se busca el menor costo. El costo estimado del prototipo que incluye el CPLD y un PCB, así como la etapa de adecuación de la señal junto con las interfaces de entrada y salida se estima en 400 pesos. Esto por supuesto incluye todos los componentes requeridos como la base para el CPLD, capacitores, resistencias y demás componentes.

3.5 COMPILACIÓN Y SÍNTESIS

Una vez disponible el archivo de diseño ya sea en forma estructural (gráfico) o por comportamiento (formato texto) se le proporciona al compilador de la herramienta Max + Plus II. Esta aplicación al igual que los compiladores como "C" y "Pascal" verifica la sintaxis correcta en el archivo de diseño, introduciéndonos en una fase de "prueba y error" para la corrección de los errores sintácticos.

Como el resultado de la etapa anterior es satisfactorio, el compilador realiza la síntesis del diseño para la tecnología elegida, así como genera la extracción de tiempos del diseño que podrán ser utilizados por la herramienta de simulación.

El compilador incluso genera el archivo de programación que ya puede ser utilizado para programar el dispositivo CPLD elegido.

En nuestro caso de estudio (frecuencimetro) es necesario resaltar las diferencias encontradas al compilar el modelo estructural y el modelo por comportamiento.

Modelo estructural

La fase de compilación generalmente detecta muchos errores, incluso cuando el diseño está bien pensado, esto se debe a muchos factores entre ellos:

- La descripción es gráfica de manera que se producen malas conexiones producto de la apreciación visual, o por el contrario se realizan conexiones donde no se desean, debido a errores de manejo al mover los componentes gráficos.
- El esquemático en nuestra aplicación es medianamente denso, de manera que navegar en el archivo es confuso, esto ocasiona mayor complejidad al corregir los errores por mal conexionado o por violaciones eléctricas.
- La compilación detecta en general bastantes errores dado que la mayor parte de la estructura del diseño queda en manos del diseñador.

Modelado por comportamiento

La fase de compilación generalmente produce mucho menos errores en relación al modelado estructural, siempre y cuando se tenga conocimiento suficiente de la sintaxis del lenguaje de descripción de hardware que se está utilizando, esto se debe a los siguientes factores:

- La descripción por comportamiento es mucho más compacta que la estructural.
- Para alguien que alguna vez en su vida a utilizado medianamente un lenguaje de programación, le resulta más amigable y menos laborioso utilizar descripción por comportamiento.
- En una descripción por comportamiento la mayor parte de las estructuras del diseño son implementadas por el compilador.
- La depuración de errores es más sencilla, debido a que es más claro analizar un archivo de texto pequeño.

Lo anterior significa que el utilizar diseño por comportamiento y teniendo conocimiento suficiente del lenguaje que se utiliza es posible reducir el tiempo de prueba y error requerido en la fase de compilación y síntesis.

Claramente podemos observar en la Tabla 3.1 que el diseño por comportamiento permite hacer mas eficiente el proceso de desarrollo al reducir el tiempo requerido para tener el diseño final, por otro lado es mas sencillo el proceso de depuración, pruebas y detección de errores. Sin embargo por ser el modelado por comportamiento una descripción de alto nivel permite un menor grado de

optimización en cuanto espacio de silicio, así como en cuanto a tiempo máximo de operación dado que se tiene menor acceso a recursos de bajo nivel Una de las grandes ventajas del modelado estructural es que permite mayor optimización de espacio y eficiencia en tiempo tal y como sucede al utilizar lenguaje ensamblador en alguna aplicación de software Para nuestro caso de aplicación que consiste en el diseño de un frecuencimetro tenemos la siguiente información arrojada por los resultados de compilación y síntesis en relación a los dos modelos como se muestra en la Tabla 3 2

Característica vs. Modelado	Modelado estructural	Modelado por comportamiento
Menor tiempo de desarrollo	??	?????
Mayor facilidad para depuración, pruebas y detección de errores	??	?????
Menores conocimientos del lenguaje HDL	?????	?
Mayor portabilidad con otras plataformas	?	????
Mayor acceso para optimizaciones de bajo nivel	?????	??
Requiere menor espacio en silicio	????	??
Mayor facilidad para ser modificado por otro diseñador con un mínimo de documentación	??	?????
Mayor facilidad de aprender si se tienen conocimientos en lenguajes de programación	??	????
Produce diseños más eficientes en tiempo	?????	???

Tabla 3 1 Comparación entre las características del modelado estructural y por comportamiento

Característica vs. Modelado	Modelado estructural	Modelado por comportamiento
Tiempo de desarrollo (T)	T	0 25T
Facilidad para hacer cambios y simular nuevamente	Complicado	Sencillo
Macroceldas utilizadas para el chip MAX7000S EPM7128SLC	118 macroceldas 92 % del chip	128 macroceldas 99 % del chip
Frecuencia máxima de operación, según la herramienta Timing Analyser	45 45 MHz 22 ns	32 25 MHz 31 ns
Costos totales	C	C

Tabla 3 2 Resultado del modelado estructural y por comportamiento para el caso frecuencímetro

Como se observa en la Tabla 3 2, es claro que el modelado estructural produce diseños más eficientes en espacio de silicio y en frecuencia máxima de operación, sin embargo para los fines de la aplicación el diseño por comportamiento no ocasiona la utilización de un chip mas grande, de

forma tal que los costos no se elevan En los casos que el modelado por comportamiento requiera de la utilización de un chip de mayor capacidad entonces es recomendable optimizar los cuellos de botella mejorando el diseño por comportamiento y en casos extremos utilizando modelado estructural para las secciones críticas del diseño

3.6 SIMULACIÓN DIGITAL

El proceso de simulación es indispensable y sumamente importante para el diseño de nuestro frecuencímetro Es indispensable pasar por esta etapa antes de programar físicamente el dispositivo, esto debido a que nos va a proporcionar un margen de seguridad cerca del 90 % de que nuestro diseño físico va a funcionar como lo indica la simulación digital

Debemos tomar en cuenta que esta fase nos debe ayudar a depurar los errores de funcionamiento en nuestro diseño y jamás debe elevar dramáticamente el tiempo de diseño del producto final, sino por el contrario reducir ese tiempo, al evitar llegar hasta la programación física y retornar a corregir errores Para nuestro caso de aplicación hemos tomado en cuenta elementos que afectan el proceso de simulación

El tiempo requerido para simular depende de la herramienta utilizada, sin embargo la mayor parte depende de nuestro diseño, aspectos que afectan son

? Nuestro diseño emplea una señal base de 1 Hz, generada a partir de la frecuencia de operación de 8 MHz, de forma tal que la división es hecha por el instrumento Este aspecto de no ser manejado con cuidadosamente puede hacer crecer en forma exponencial el tiempo de simulación

? El nucleo debe ser simulado sin las interfaces de entrada y salida, las cuales solo aumentarían el tiempo de simulación inutilmente, ya que la estructura de estas interfaces puede ser común entre instrumentos

? El archivo de simulación sólo contiene las señales de entrada y de salida del núcleo, para minimizar los tiempos de simulación

Algunas estrategias de depurado son

? Iniciar el archivo de simulación con el tiempo mínimo requerido para verificar su funcionamiento

? Alterar la estructura de la frecuencia de operación del instrumento en relación con el tiempo de simulación utilizando un factor "F" que haga factible la duración de la simulación, el factor "F" debe aplicarse al finalizar la simulación para obtener resultados reales

? Iniciar el archivo de simulación solo con las entradas y salidas del "núcleo" del instrumento

? La causa de los errores en los resultados esperados se detecta insertando nodos alambrados del núcleo para ver el funcionamiento interno del mismo Esto evidentemente incrementa el tiempo de simulación

Simulación del modelado estructural

Dado que dibujamos en forma gráfica la estructura de la arquitectura del instrumento, resulta bastante complicado aplicar las estrategias de depurado como el escalamiento por un factor 'F' para minimizar el tiempo de simulación Esto no se puede hacer tan directamente porque no consiste en modificar un número, sino en modificar la estructura, como quitar etapas de conteo, para mantener el funcionamiento correcto con una frecuencia menor Esto significa que el proceso de depuración durante la simulación se hace más complicado cuando tenemos un diseño estructural En la Fig 3 13 se presenta la simulación del nucleo estructural del instrumento

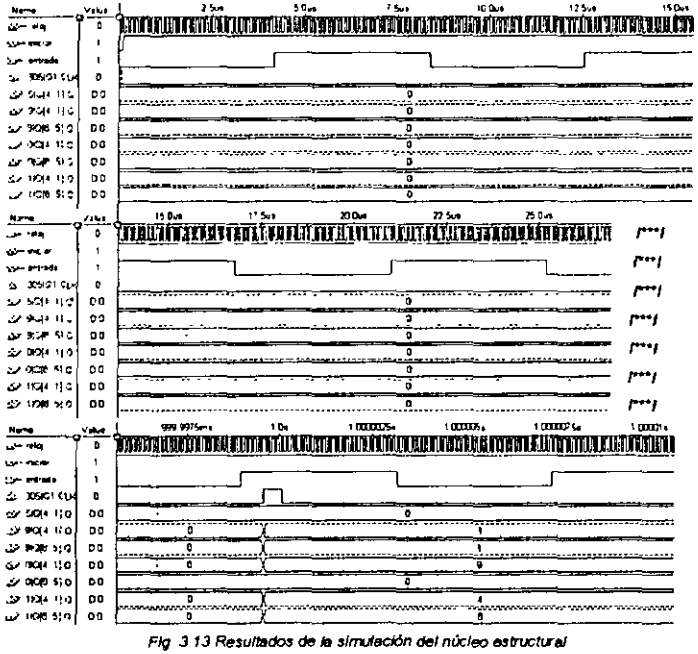


Fig 3 13 Resultados de la simulación del núcleo estructural

Las señales de entrada "reloj", "Iniciar" y "entrada" representan respectivamente la señal del oscilador que es de 8 MHz, la señal de reset para el instrumento y la señal TTL cuya frecuencia se desea medir y que ha sido previamente acondicionada La señal etiquetada "305|Q1 CLK" representa la señal de carga de la etapa de los circuitos de retención Los últimos siete grupos de señales agrupadas e ilustradas en el diagrama de tiempo de la Fig 3 13 muestran el contenido de los circuitos de retención en formato decimal La señal que deseamos medir está etiquetada como "entrada" y como se puede observar tiene un periodo de 8 4 ?s por tanto una frecuencia de 119047 61 Hz Al final de la simulación que corresponde con 1 segundo podemos observar que el

contenido de los circuitos de retención es "119048" que corresponde con el valor de la frecuencia de la señal a medir Por tanto concluimos que el instrumento funciona correctamente

Simulación del modelado por comportamiento

Dado que el instrumento está descrito en forma funcional por estructuras de alto nivel, resulta bastante sencillo aplicar las estrategias de depurado como es el escalamiento por un factor "F" para minimizar el tiempo de simulación Esto se puede hacer tan sencillo como modificar un simple numero, de manera que al recompilar el diseño descrito por comportamiento, el compilador genera automáticamente la estructura adecuada Esto significa que el proceso de depuración durante la simulación se hace más sencillo cuando tenemos un diseño por comportamiento, ya que es como estar manipulando un lenguaje de programación La Fig 3 14 ilustra la simulación del núcleo del instrumento

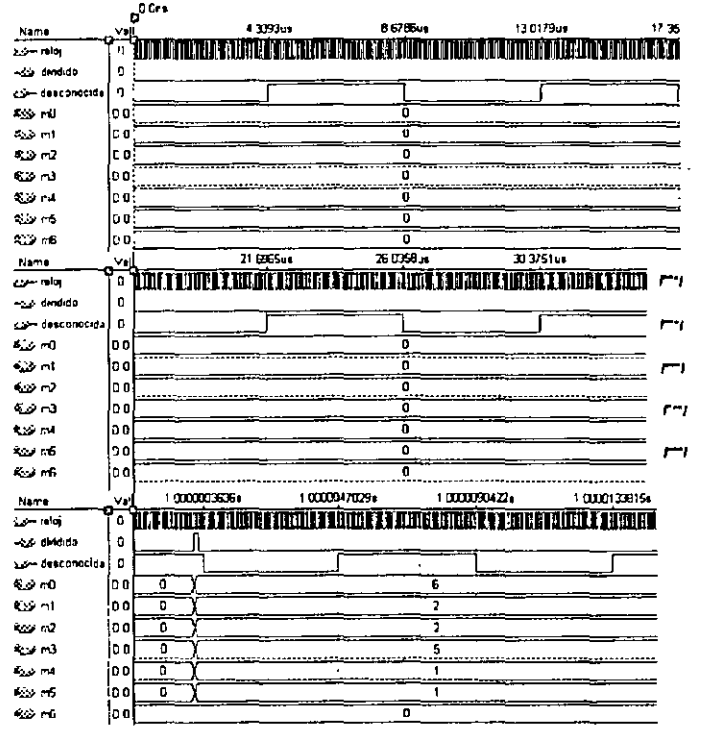


Fig. 3 14 Resultados de la simulación del núcleo funcional

Las señales de entrada "reloj", "dividido" y "desconocida" representan respectivamente la señal del oscilador que es de 8 MHz la señal base de 1 seg para el instrumento y la señal TTL cuya frecuencia se desea medir y que ha sido previamente acondicionada Los ultimos siete grupos de señales "m[0 6]" y agrupadas e ilustradas en el diagrama de tiempo de la Fig 3 14 muestran el contenido decimal correspondiente al valor de la frecuencia "desconocida La señal que deseamos medir está etiquetada como "desconocida" y como se puede observar tiene un periodo de 8 6786?s por tanto una frecuencia de 115226 Hz Al final de la simulación que corresponde con 1 segundo podemos observar que el valor calculado de la frecuencia en formato decimal es "115226" que corresponde con el valor de la frecuencia de la señal a medir Por tanto concluimos que el instrumento funciona correctamente

En la Tabla 3 3 se presenta una comparación de los tiempos de simulación para cada uno de los modelos En la simulación sin aplicar escalamientos observamos que el tiempo de simulación del modelo por comportamiento es 15 minutos mayor en relación al modelo estructural, esto es producto de una solución menos optimizada para el modelo por comportamiento Sin embargo es muy sencillo escalar la frecuencia de operación y hacer las modificaciones requeridas en la arquitectura para obtener tiempos de simulación menores en el modelo por comportamiento Estos escalamientos también se pueden hacer con el modelo estructural pero requiere modificaciones estructurales que consumen más tiempo, razón por la cual no se presentan en la tabla Observamos que reducimos el tiempo de simulación en el modelo por comportamiento hasta 11 minutos Resaltamos que para diseños cuyas simulaciones puedan llevar días completos esta es una alternativa muy atractiva para simular en horas

Frecuencia de simulación vs Modelo	Estructural	Comportamiento
Sin escalar, f = 8MHz	51 minutos	66 minutos
Escalando f = 4MHz	No disponible	41 minutos
Escalando f = 1MHz	No disponible	11 minutos

Tabla 3 3 Comparación de tiempos de simulación para el modelado estructural y por comportamiento

3.7 PROGRAMACIÓN DEL DISPOSITIVO

En el momento que el resultado de la simulación es satisfactorio, los archivos de programación que fueron generados en el proceso de compilación se pueden utilizar para programar físicamente el dispositivo En el caso de nuestro frecuencímetro hemos respetado la asignación de pines realizada por el compilador Al examinar el archivo de reporte observamos que se ha utilizado el 92% de los recursos del CPLD, de manera que nuestra metodología sugiere no modificar la asignación de pines, dado que esto va a ocasionar que el diseño no pueda ser implantado en el CPLD elegido Evidentemente esta determinación puede hacer más complicado el diseño del PCB, sin embargo, es preferible a tener que utilizar dos CPLD's o uno de mayor capacidad que sería desperdiciado

3.8 PRUEBAS Y RESULTADOS

Las pruebas definitivas sobre el dispositivo real y la posible depuración de errores la hicimos de las dos maneras planteadas en nuestra metodología

Prototipo Rápido

Programamos el dispositivo y verificamos su funcionamiento sobre una tarjeta protoboard, sobre la cual es posible grabar el dispositivo CPLD debido a sus características de ISP (In System Programming) Sobre el protoboard se alambro el CPLD que tiene programado el núcleo del instrumento así como la interfaz de salida La interfaz de entrada no existe, solo se requiere un modulo adicional de adecuamiento de señal como se describió anteriormente, pero que no es parte de la metodología Es importante señalar que se deben esperan obtener los resultados presentados por la simulación

Un prototipo de este tipo es muy fácil de hacer para verificar el funcionamiento real del instrumento Sin embargo es necesario tomar precauciones como la utilización de capacitores de desacople, fuentes bien reguladas, etc , tal y como lo sugiere el fabricante del CPLD, con el objetivo de minimizar problemas en el prototipo el cual está alambrado sobre un protoboard

En la Fig 3 15 ilustramos este prototipo funcionando y midiendo una frecuencia de 1000 Hz producida utilizando un generador digital de señales HP

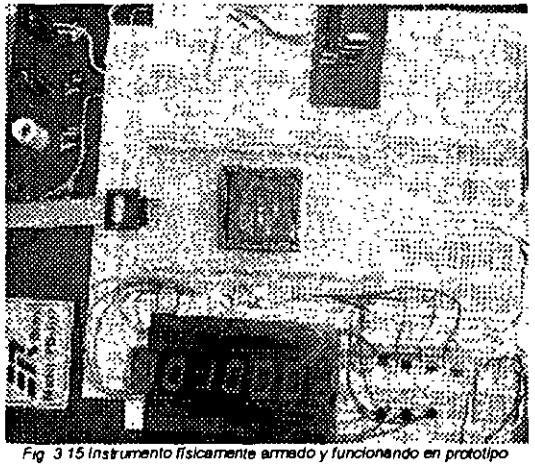


Fig 3 15 Instrumento físicamente armado y funcionando en prototipo

Prototipo PCB

Como continuación del prototipo rápido y sobre todo pensando en un producto final terminado es posible verificar el funcionamiento del instrumento sobre tarjetas de circuito impreso diseñadas especialmente para la aplicación

De esta manera utilizando el software para el diseño de circuitos impresos llamado PCAD 2001 se procedió a diseñar los impresos para el nucleo del instrumento y las interfaces de entrada / salida Los PCBs se muestran en las Figs 3 16, 3 17, 3 18, 3 19, 3 20 y 3 21 respectivamente

El costo de fabricar estos impresos es muy bajo, dado que el diseño del PCB del instrumento se realizo mediante un ruteado cuidadoso que permitió utilizar una sola cara a pesar del número de componentes utilizados Este impreso contiene

? Etapa de adecuación

? Núcleo del instrumento e interfaz de salida (en el CPLD)

? Fuente de alimentación

El impreso para el despliegue de los valores calculados unicamente contiene los displays de siete segmentos así como sus transistores de control Este impreso se tuvo que hacer en dos caras en virtud de la cantidad de lineas con intersección

En la Fig 3 22 se muestra una fotografía de los impresos armados representados en las figuras anteriormente citadas

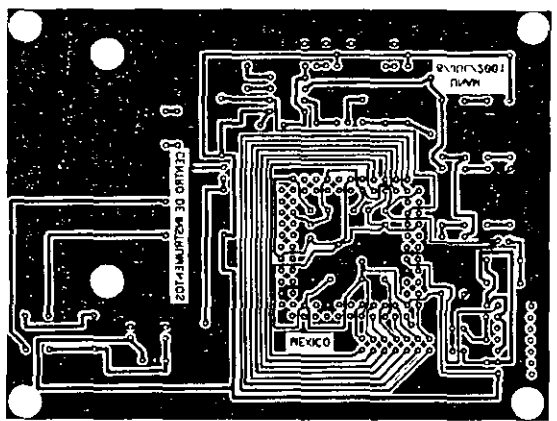


Fig 3 16 Cara inferior del impreso del instrumento

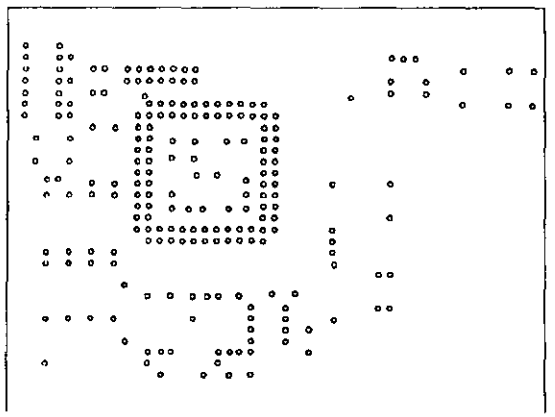


Fig 3 17 Cara superior del impreso del instrumento

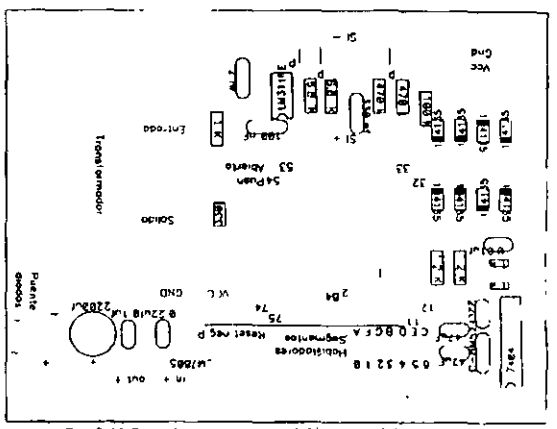


Fig 3 18 Cara de componentes del impreso del instrumento

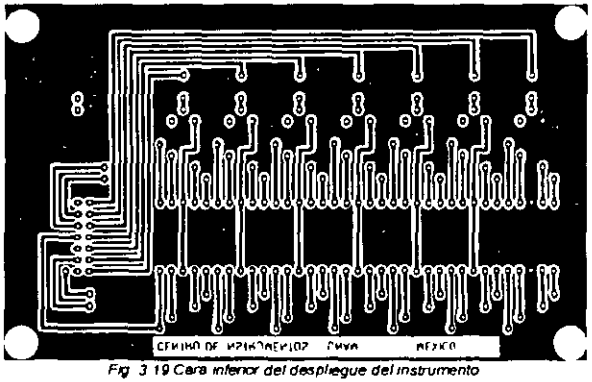


Fig 3 19 Cara inferior del despliegue del instrumento

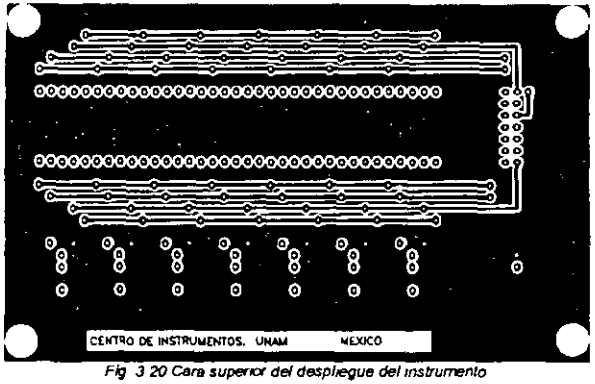


Fig 3 20 Cara superior del despliegue del instrumento

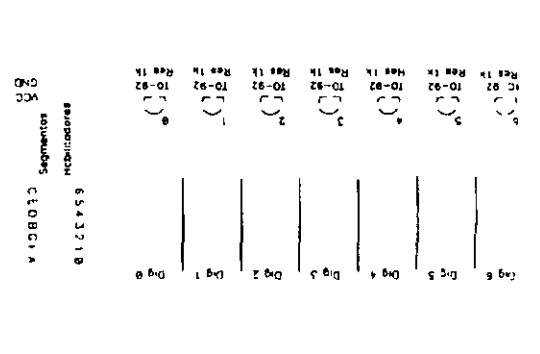


Fig 3 21 Cara de componentes del despliegue del instrumento

Resultados

En la Fig 3 15 se ilustra el instrumento físicamente armado, funcionando y midiendo una frecuencia de 10 KHz la cual es alimentada utilizando un generador de señales Se hicieron pruebas al instrumento midiendo frecuencias en el intervalo 1 Hz a 10 MHz con el fin de comparar la frecuencia calculada por el instrumento contra la frecuencia entregada por el generador de señales De esta forma llegamos a la Tabla 3 4 donde podemos concluir que el instrumento tiene un error porcentual con respecto al valor real de la frecuencia medida que es constante para toda la gama de frecuencias con valor límite de 0 028% Asumiendo que la frecuencia entregada por el generador de señales no tiene error lo cual no es cierto, podemos decir que la precisión y exactitud de nuestro instrumento es muy buena

Frecuencia obtenida por el generador de señales HP 33120 A (FHP)	Frecuencia medida por el instrumento diseñado (FID)	Error en número de cuentas entre la frecuencia generada y la frecuencia medida E=(FHP-FID)	Error porcentual con respecto a la frecuencia medida (100*E)/FHP
1	1	0	0 %
10	10	0	0 %
100	100	0	0 %
1000	1001	1	0.1 %
10000	10004	4	0 04 %
100000	100029	29	0 029 %
1000000	1000289	289	0 0289 %
10000000	10002888	2888	0 02888 %

Tabla 3 4 Resultados del instrumento

En la Tabla 3 5 presentamos características técnicas de operación de instrumento referentes a consumo de potencia, frecuencia máxima de operación, voltaje máximo de alimentación, voltajes de señal de entrada etc

Parámetro / Valor	Mínimo	Máximo
Voltaje de alimentación	4 75 V	5 25 V
Consumo de potencia	4 9 W	5 4 W
Corriente de alimentación	1 0 A	1 1 A
Frecuencias de medición	1 Hz	10 MHz
Tiempo de respuesta Escala 1 MHz Escala 10 MHz	 $10 - 10^{-7}$ seg $1 - 10^{-6}$ seg	 $10 + 10^{-6}$ seg $1 + 10^{-6}$ seg
Temperatura de operación	0 ºC	70 ºC

Tabla 3 5 Características técnicas de operación del instrumento

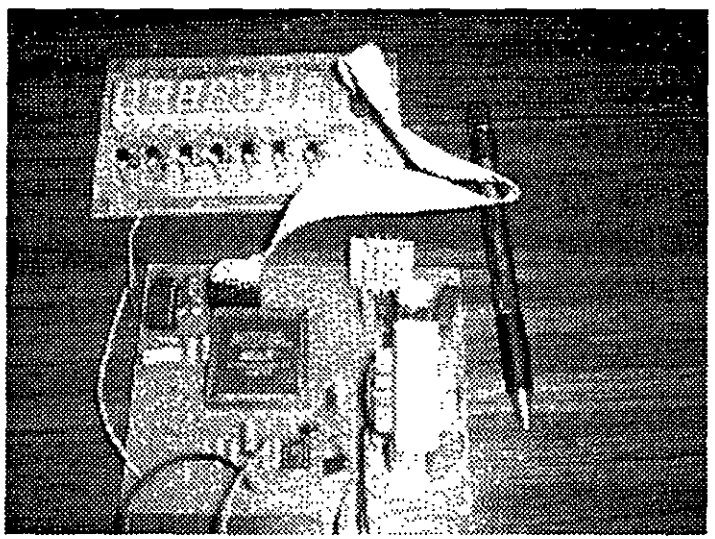


Fig 3 22 Impresos armados que conforman el instrumento

3.9 PRODUCTO FINAL

Posterior a la etapa de pruebas y resultados sobre el prototipo real en PCB es posible ensamblar el instrumento final tal y como será entregado como producto al usuario En la Fig 3 23 ilustramos varias vistas del instrumento, después del proceso de ingeniería del producto que incluye tanto el proceso de diseño del exterior del instrumento así como del embalaje correspondiente La ingeniería del producto se realizó en la sección de Ingeniería del Producto del CCADET

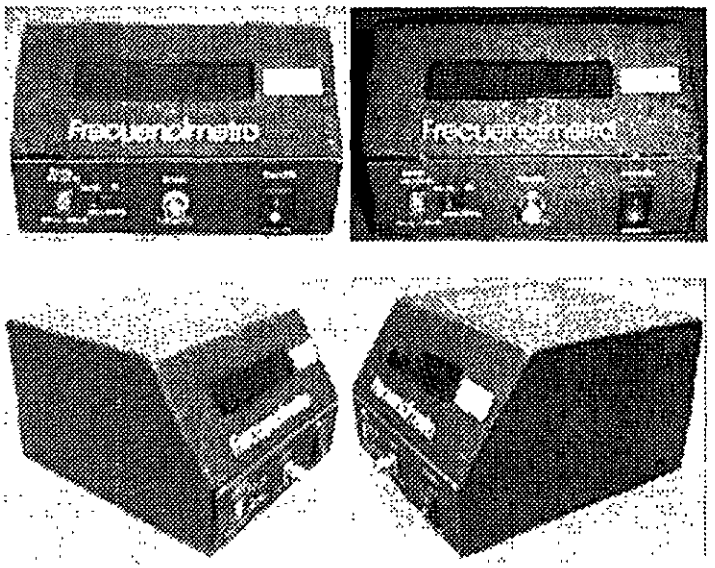


Fig 3 23 Diferentes vistas del producto final

CAPÍTULO 4

INTERFAZ DE SALIDA GRÁFICA VGA

CONTENIDO

4.1 IMPORTANCIA Y UTILIDAD.

En este capítulo presentamos un módulo denominado *interfaz de salida gráfica VGA*, la cual es una aportación de la metodología propuesta para ser utilizada en el diseño de cualquier instrumento que requiera de una salida gráfica para el despliegue

Un módulo de este tipo en la instrumentación digital es muy importante porque puede dar capacidades gráficas impresionantes al momento de presentar la información, como lo vamos a demostrar aplicándolo a un prototipo de osciloscopio Además considerando que un monitor VGA es muy comun podemos tener instrumentos completos que solo requieran compartir un monitor con cualquier computadora personal

La utilidad esencialmente va a depender del tipo de instrumento a diseñar, por ejemplo este tipo de interfaz aplicada a un osciloscopio es muy util por las capacidades de despliegue que proporcionará; sin embargo, sera de poca utilidad para un frecuencimetro como el diseñado en el Capítulo 3

En este capítulo vamos a presentar la interfaz genérica VGA y a demostrar su importancia y utilidad en el diseño de un osciloscopio digital prototipo muy preliminar, pero suficiente para mostrar la funcionalidad del modulo de interfaz gráfica

4.2 ESTRUCTURA A BLOQUES.

En la Fig 4 1 se muestra la estructura a bloques de la interfaz completa Consiste esencialmente de tres módulos El primero denominado *"controlador de video"* se encarga de generar las señales de barrido horizontal, vertical y recibir las señales de color RGB en los tiempos precisos, que se generan a partir de un oscilador de 25 175 MHz, en función del pixel que se desea iluminar. El segundo módulo es un *"controlador VRAM"* que se encarga de leer y escribir datos en una memoria SRAM cuyo objetivo es mantener un mapa en memoria del contenido desplegado en el monitor El tercer y último bloque está constituido por la base de tiempos utilizada para la sincronización de todos los módulos así como para la generación de las señales de tiempo requeridas

La filosofía bajo la cual trabaja la interfaz, consiste en que el controlador VRAM escribe la información en la memoria SRAM durante los tiempos muertos tanto de linea como de cuadro mediante la transformación de un sistema coordenado (X,Y) a un sistema de arreglo unidimensional compatible con la memoria SRAM La información es recibida por el módulo VRAM proveniente de lo que hemos denominado un convertidor A/D, pero en forma genérica proviene de cualquier fuente digital de información

Por otra parte, el resto del tiempo el controlador VRAM se encuentra haciendo la lectura de la SRAM generando las señales adecuadas para el *"controlador de video"*, desplegando de esta manera de forma continua el contenido de la memoria

Por lo tanto la memoria SRAM es el sitio de almacenamiento donde el instrumento en cuestión debe escribir su información a ser desplegada mediante su interfaz con el controlador VRAM, esto es, el instrumento se comunica con la interfaz a través del módulo VRAM

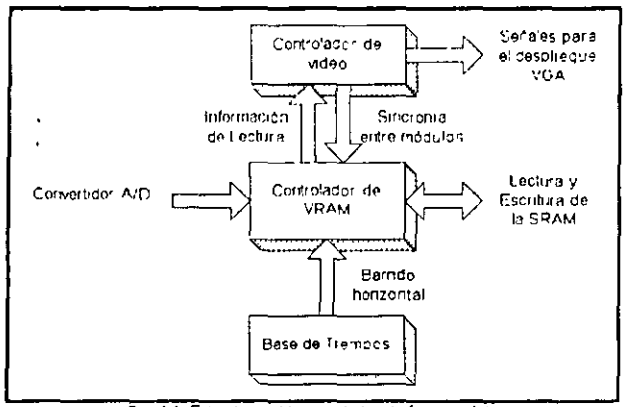


Fig 4 1 Estructura a bloques de la interfaz completa

4.3 CONTROLADOR DE VIDEO.

El módulo *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, como se observa en la Fig 4 1. Adicionalmente y como detalles de diseño en casos especiales se puede tocar directamente este módulo para optimizar ciertas funciones de despliegue constantes, como podría ser el cuadriculado típico de un osciloscopio digital, etiquetas de las escalas de tiempo etc

El módulo debe proveer al monitor VGA 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 con ligeras modificaciones

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 (aproximadamente 40 ns) 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 mínimo adecuado es el EPF10K20 que 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 a un cuadrado de 112 pixeles por lado Por esa razón, se decidió utilizar una memoria externa preferentemente del tipo estático

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) 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 distribuir sobre la pantalla el área disponible Entre varias alternativas se eligió un área de 401 x 501 pixeles, dividida en 80 cuadrados de 50 x 50 pixeles Además de la ventaja de ser un área estandarizada como en los osciloscopios digitales, 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 es la división en cuadros de 50 pixeles permite simplificar el diseño de las escalas horizontal y vertical de los instrumentos, 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 Esta área modelo se muestra en la Fig 4 2 con un cuadriculado típico de osciloscopio como ejemplo de distribución

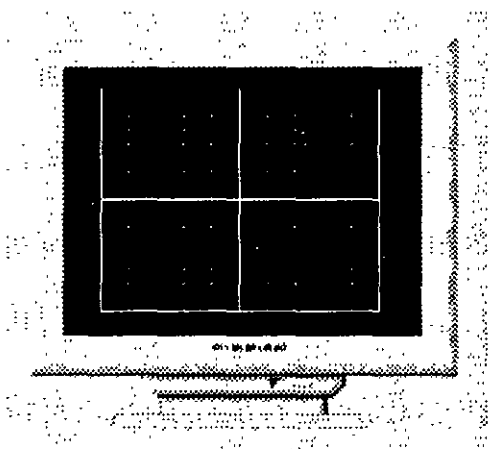


Fig 4 2 Área modelo

A manera de convención en este trabajo, 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 numero 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 reditua en una mayor facilidad para vincular las coordenadas de cualquier pixel con su respectiva información dentro del mapa de memoria

El módulo es denominado *control_video* el código VHDL que describe la entidad es el siguiente

```
--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 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
```

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

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* unicamente 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

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 como lo muestra el código VHDL siguiente

```
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);
```

Descripción de variables y constantes

- *H_cont* y *V_cont* 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.
- *Encendido* Consta de un solo bit que sirve para inicializar algunas de las demás variables internas
- *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
- *H_max* y *V_max* son constantes que establecen las cuentas máximas horizontal y vertical de los contadores *H_cont* y *V_cont* respectivamente
- *CentroH* y *CentroV* son las constantes que se emplean para centrar el área de trabajo dentro de la pantalla VGA

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*

```
Matriz <= Leclinea and Lecrenglon;     -- (1)
Pulsolec <= Reloj and Matriz;          -- (2)
Pulsescr <= not Matriz;                -- (3)
```

La sentencia (1) indica que la señal *Matriz* sólo se activa cuando las señales *Leclinea* y *Lecrenglon* se encuentran ambas en estado lógico '1' 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 renglón (2) 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 sentencia (3) 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

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 *Pulsecr* 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 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 instrumento. 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 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 variables 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, esto se ilustra en el código VHDL siguiente:

```
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. 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.

El proceso siguiente descrito en VHDL 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 dispositivo, 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. 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.

```
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 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, 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 numero 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 unicamente 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;
```


En resumen, el módulo controlador de video proporciona las señales requeridas por el monitor VGA que servirá como interfaz de despliegue de información para el instrumento en cuestión. Las 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*.

Los recursos de espacio de silicio utilizados por el módulo *Controlador de Video* dentro del dispositivo lógico programable EPF10K20, ocupa un 8% de la totalidad disponible.

4.4 CONTROLADOR VRAM.

Seleccionamos la memoria SRAM (Static Random Access Memory) 62256A con tecnología CMOS, la cual se ajustó a nuestros requerimientos. 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.

Un par de funciones indispensables para lograr el control total de la memoria de video consisten en almacenar o "escribir" la información recibida, que se interpreta como el par de coordenadas (x,y) del pixel que se debe iluminar en la pantalla y su transformación que corresponde a un lugar específico dentro del mapa de memoria contiguo de la SRAM, y por otro lado, el módulo debe ser capaz de "leer" la información almacenada en la SRAM desde la primera localidad de memoria hasta la que corresponde al ultimo 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 SRAM, 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 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* y un *Ciclo de Escritura*. En la Fig. 4.3 se ilustra el diagrama a bloques detallado que presenta las funciones que debe desarrollar el módulo VRAM. El ciclo de lectura accesa secuencialmente las direcciones de la SRAM, transfiriendo paquetes de bytes al buffer auxiliar desde donde usando un buffer primario se envían bit a bit hacia el módulo *Controlador de Video*. En el ciclo de escritura se recibe la información del pixel a iluminar en el formato coordenado (x,y), de forma que el decodificador de dirección localiza el correspondiente byte donde se encuentra el bit en cuestión, este byte es actualizado por el módulo denominado *Actualización* y transferido a un módulo de tres estados que en los tiempos de descanso actualizará la información de la SRAM. Podemos observar que los ciclos de escritura están restringidos a los tiempos en los cuales el *controlador de video* no está desplegando información.

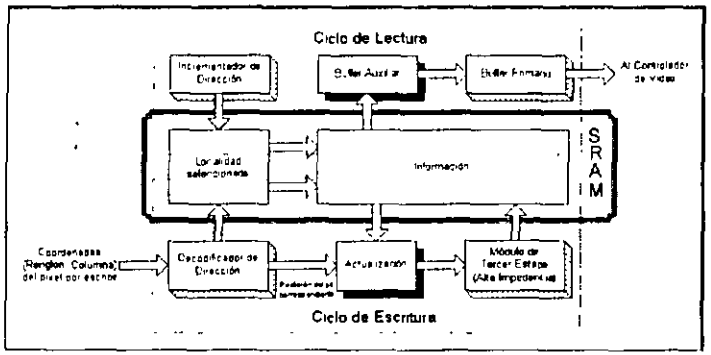


Fig 4 3 Diagrama de bloques del módulo Controlador de VRAM

Una de las primeras consideraciones que observamos para diseñar el Ciclo de Lectura fue tomar en cuenta la necesidad de programar un módulo que incremente 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 como se ilustra en la Fig. 4.4.

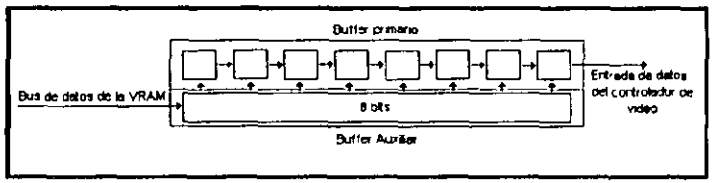


Fig. 4.4 Esquema de transferencia del buffer

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 proveniente 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 de 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 provenientes de la memoria.

Por otro lado, para diseñar el Ciclo de Escritura partimos de la premisa de que siempre contamos con información proveniente del instrumento que utiliza la interfaz. 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 ya que el bus del chip SRAM elegido utiliza un bus bidireccional, entonces 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 manejar alta impedancia. 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.

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 utilizadas un byte conformado 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 realizar el control de los tres procedimientos, como se ilustra en la Fig. 4.5. 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 SRAM el valor H00, a fin de que todos los píxeles 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 SRAM 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 VRAM verifica si el controlador de video se encuentra en un período "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 SRAM hasta que el período "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 SRAM al que pertenece, encender (o apagar) el bit correspondiente y escribir el byte de vuelta a la memoria.

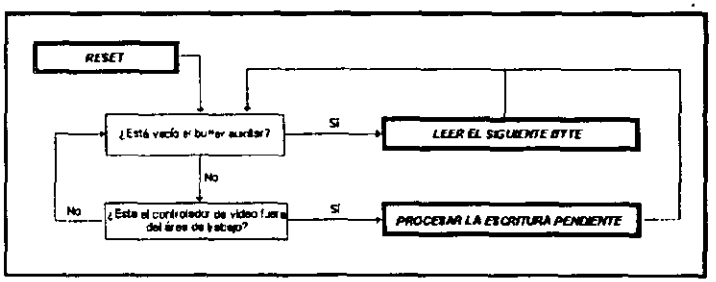


Fig. 4.5 Diagrama de flujo del Controlador VRAM

A continuación se hará una breve descripción de cada uno de los módulos presentando el código VHDL que lo describe.

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*.

Su función es muy simple, al encontrarse la señal *Conectar* en estado activo alto 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 del área de trabajo a una dirección del mapa de memoria es la siguiente:

$$Dirección = Renglón \times (512 / 8) + Ent(Columna / 8)$$

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 SRAM 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, esta última salida debe ser de 3 bits.

```
-- 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;
```

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.

```
-- 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;


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 unicamente 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*, proveniente precisamente del buffer auxiliar, para llenar nuevamente el 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;

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

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

        process(Pulsolec, Rst) begin
```



```
                if Rst = '1' then
                        Prim(0) <= '0';
                        Vacio <= '0';
                        Espera <= '1';
                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;
                        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;
```

ESTRUCTURA COMPLETA DEL MÓDULO CONTROLADOR VRAM

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. Se definen las variables internas de la arquitectura y seguidamente los componentes *triest*, *bufferpri*, *decodirec* e *incredirec*. El código completo del módulo se muestra a continuación.

```
-- 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;

architecture vramarq of control_vram is
        signal oe, we, Inhabilbuf, Actualbuf, Bandebuf, Enviaescr: std_logic;
        signal Estado: std_logic_vector(2 downto 0);
        signal Direcactual, Almacedirec, Direcsig: std_logic_vector(14 downto 0);
        signal Bufaux, Bytesal, 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);

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);

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

        process(Rst, Medio) begin
            If Rst = '1' then
                Estado <= "000";
                Direcactual <= "00000000000000";
                we <= '1';
                oe <= '1';
                Enviaescr <= '0';
                Inhabilbuf <= '1';
                Bandebuf <= '0';
                Bytenue <= "00000000";
            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;
                    when "010" =>   If Actuabuf = Bandebuf then
                                        Bandebuf <= not Bandebuf;
                                        oe <= '0';
                                        Estado <= "011";
                                    elsif Pulsescr = '1' then
                                        Almecedirec <= Direcactual;
                                        Direcactual <= Direcdeco;
                                        Almacepos <= Posdeco
```



```
                                        Estado <= "100";
                                        oe <= '0';
                                    end if;
                    when "011" =>   Bufaux <= sram_byte;
                                    Direcactual <= Direcsig
                                    Inhabilbuf <= '0';
                                    oe <= '1';
                                    Estado <= "010";
                    when "100" =>   Byteant <= sram_byte;
                                    Estado <= Estado + "001";
                    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";
                    when "110" =>   we <= '0';
                                    Bytenue <= Byteant;
                                    Enviaescr <= '1';
                                    Estado <= Estado + "001";
                    when others =>  we <= '1';
                                    Enviaescr <= '0';
                                    Direcactual <= Almacedirec;
                                    Estado <= "010";
                end case;
            end if;
        end process;
    end vramerq;
```

El código descrito anteriormente tiene por objetivo implementar la máquina de estados para el control de la lectura y escritura mostrada en la Fig 4.6

Ciclo Reset.

En el ciclo Reset, se produce 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 vacia 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.

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' (ver 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

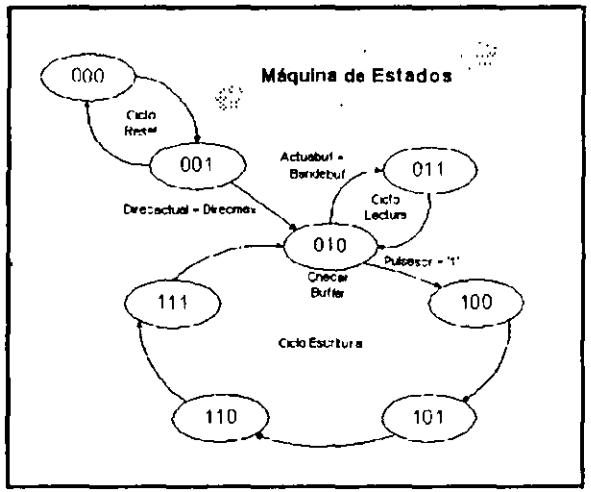


Fig 4 6 Máquina de Estados que muestra los ciclos Reset, Lectura y Escritura del módulo Controlador de VRAM

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)

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 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*

Ciclo de Escritura

Como ya se ha mencionado, éste ciclo que contiene cuatro estados solo 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 SRAM 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*

En el 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)

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

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 ultimo se provoca el ingreso al ciclo que revisa el buffer primario

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 SRAM 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

Los recursos consumidos por el módulo *Controlador de VRAM* dentro del dispositivo lógico programable EPF10K20, ocupan un 19% de la totalidad disponible

4.5 APLICACIÓN AL CASO DE UN OSCILOSCOPIO PROTOTIPO.

Como lo hemos venido comentando el objetivo de este Capitulo ha sido presentar la aportación de la metodología, objeto de esta tesis, en el ámbito de proporcionar una interfaz gráfica VGA genérica que pueda ser utilizada por diversos instrumentos Por tanto para mostrar la utilidad, en esta sección presentamos la aplicación de un osciloscopio prototipo que usa la interfaz diseñada

El diagrama a bloques del osciloscopio prototipo propuesto se ilustra en la Fig 4 7 Como se puede apreciar, el diseño 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 ultimo 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

- ? 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 el empleo de nuestra interfaz gráfica

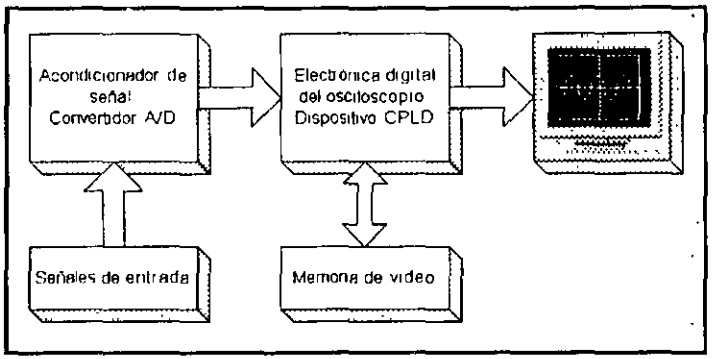


Fig 4 7 Diagrama a bloques del osciloscopio digital

El código VHDL que implementa este osciloscopio prototipo es el siguiente

```
--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 Esctiempo, Raf, 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;

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 Pulsoelec : 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, Pulsoelec, 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;

begin
        video   control_video port map (Reloj, Rojo, Verde, Azul, Horiz_sinc,
                Vert_sinc, Pulsoelec, Video, Selector, Pixcero, Pulsescr)

        vram    control_vram port map (VRrst, Medio, Pulsoelec, 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;

        process(Rst, Pixcero) begin
                if Rst = '1' then
                        Selector <= "0000";
                        Fincont <= '0';
                        Retardo <= conv_std_logic_vector(0,2);
                elsif Pixcero = '0' and Pixcero'event then
                        if Esctiempo = '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;


        process(Rst, Ref) begin
                if Rst = '1' then
                        Increren <= conv_std_logic_vector(0,9);
                elsif Ref = '1' and Refevent then
                        Increren <= Increren + "000000001";
                end if;
        end process
```



```
        process(Reloj) begin
                if Reloj = '1' and Reloj'event then
                        Medio <= not Medio;
                end if;
        end process
end arq;
```

Los recursos totales consumidos por el osciloscopio prototipo en el dispositivo lógico programable EPF10K20, 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

RESULTADOS

El prototipo que diseñamos se puede considerar como un instrumento de medición para señales 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 de tiempo se muestran en la Tabla 4 1

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

Tabla 4 1 Escalas de tiempo

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 4 Vpp, a una frecuencia de 14 7 Hz, como se muestra en la Fig 4 8

En las Figs 4 9 y 4 10 se 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 despliegue gráfico del osciloscopio sobre el monitor VGA trabaja de acuerdo a las expectativas de diseño y por tanto la interfaz gráfica puede ser explotada ampliamente por otro tipo de instrumentos

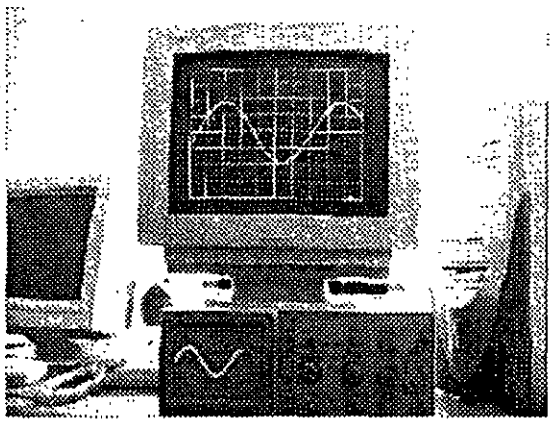


Fig 4 8 Señal sinusoidal de 4 Vpp a 14 7 Hz, registrada por un osciloscopio comercial y por el prototipo diseñado

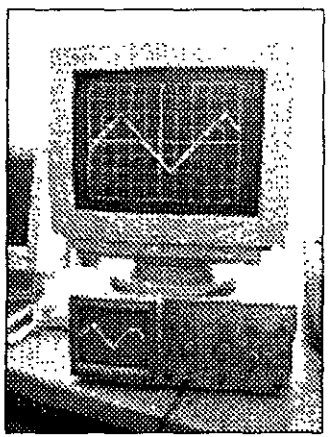


Fig 4 9 Señal triangular

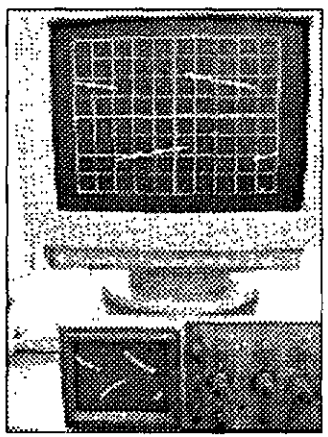


Fig 4 10 Señal Cuadrada

University Program
UP2 Development Kit

July 2003, v3.0 User Guide

Introduction

The University Program UP2 Development Kit was designed to meet the needs of universities teaching digital logic design with state-of-the-art development tools and programmable logic devices (PLDs). The package provides all of the necessary tools for creating and implementing digital logic designs, including the following features:

- MAX+PLUS® II University development software
- UP Education Board
  - An EPF10K70 device for the UP2 board in a 240-pin power quad flat pack (RQFP) package
  - An 84-pin plastic J-lead chip carrier (PLCC) package
- ByteBlaster™ II parallel port download cable

MAX+PLUS II University Software

The MAX+PLUS II University software contains many of the features of the commercial version of the MAX+PLUS II software including a completely integrated design flow and an intuitive graphical user interface. This software supports schematic capture and text-based hardware description language (HDL) design entry, including Verilog HDL, VHDL, and the Altera® Hardware Description Language (AHDL™). It also provides design programming, compilation, and verification support for all devices supported by the MAX+PLUS II BASELINE software including the EPM7128S, and EPF10K70 devices. The MAX+PLUS II University software can be freely distributed to students for installation on their personal computers and provides instant access to online help.

For information on how to install the MAX+PLUS II University software on your computer, see "Software Installation" on page 17.

UP2 Education Board

The UP2 Education Board is a stand-alone experiment board based on a FLEX® 10K device and includes a MAX® 7000 device. When used with the MAX+PLUS II University software, the board provides a superior platform for learning digital logic design using industry-standard development tools and PLDs.

The board is designed to meet the needs of instructors and students in a laboratory environment. The UP2 Education Board supports both look-up table (LUT)-based and product term-based architectures. The EPF10K70 device can be configured in-system with either the ByteBlaster II download cable or an EPC1 configuration device. Additional download cables can be purchased separately. The EPM7128S device can be programmed in-system with the ByteBlaster II download cable.

EPF10K70 Device

The EPF10K70 device is based on SRAM technology. It is available in a 240-pin RQFP package and has 3,744 logic elements (LEs) and nine embedded array blocks (EABs). Each LE consists of a four-input LUT, a programmable flipflop, and dedicated signal paths for carry-and-cascade functions. Each EAB provides 2,048 bits of memory which can be used to create RAM, ROM, or first-in first-out (FIFO) functions. EABs can also implement logic functions, such as multipliers, microcontrollers, state machines, and digital signal processing (DSP) functions. With 70,000 typical gates, the EPF10K70 device is ideal for intermediate to advanced digital design courses, including computer architecture, communications, and DSP applications.

For more information on FLEX 10K devices, see the *FLEX 10K Embedded Programmable Logic Family Data Sheet*.

EPM7128S Device

The EPM7128S device, a member of the high-density, high-performance MAX 7000S family, is based on erasable programmable read-only memory (EEPROM) elements. The EPM7128S device features a socket-mounted 84-pin plastic J-lead chip carrier (PLCC) package and has 128 macrocells. Each macrocell has a programmable-AND/fixed-OR array as well as a configurable register with independently-programmable clock, clock enable, clear, and preset functions. With a capacity of 2,500 gates and a simple architecture, the EPM7128S device is ideal for introductory designs as well as larger combinatorial and sequential logic functions.

For more information on MAX 7000 devices, go to the *MAX 7000 Programmable Logic Device Family Data Sheet*.

ByteBlaster II Parallel Port Download Cable

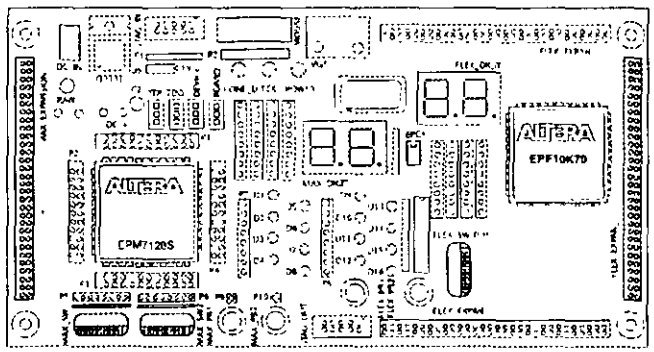
Designs can be easily and quickly downloaded into the UP2 Education Board using the ByteBlaster II download cable, which is a hardware interface to a standard parallel port. This cable sends programming or configuration data between the MAX+PLUS II University software and

the UP Education Boards. Because design changes are downloaded directly to the devices on the board, prototyping is easy and multiple design iterations can be accomplished in quick succession.

UP Education Board Description

The UP2 Education Board, shown in Figure 1, contains the features described in this section.

Figure 1. UP2 Education Board Block Diagram



DC_IN & RAW Power Input

The DC_IN power input accepts a 2.5-mm × 5.55-mm female connector. The acceptable DC input is 7 to 9 V at a minimum of 350 mA. The RAW power input consists of two holes for connecting an unregulated power source. The hole marked with a plus sign (+) is the positive input; the hole marked with a minus sign (–) is board-common.

Oscillator

The UP Education Board contains a 25.175-MHz crystal oscillator. The output of the oscillator drives a global clock input on the EPM7128S device (pin 83) and a global clock input on the FLEX 10K device (pin 91).

JTAG_IN Header

The 10-pin female plug on the ByteBlaster II download cable connects with the JTAG_IN 10-pin male header on the UP Education Board. The board provides power and ground to the ByteBlaster II download cable. Data is shifted into the devices via the TDI pin and shifted out of the devices via the TDO pin. Table 1 identifies the JTAG_IN pin names when the ByteBlaster II is operating in Joint Test Action Group (JTAG) mode.

Table 1. JTAG_IN 10-Pin Header Pin-Outs

Pin	JTAG Signal
1	TCK
2	GND
3	TDO
4	VCC
5	TMS
6	No Connect
7	No Connect
8	No Connect
9	TDI
10	GND

Jumpers

The UP Education Board has four three-pin jumpers (TDI, TDO, DEVICE, and BOARD) that set the JTAG configuration. The JTAG chain can be set for a variety of configurations (i.e., to program only the EPM7128S device, to configure only the FLEX 10K device, to configure and program both devices, or to connect multiple UP Education Boards together). Figure 2 shows the positions of the three connectors (C1, C2, and C3) on each of the four jumpers.

Figure 2 Position of C1, C2 & C3 Connectors

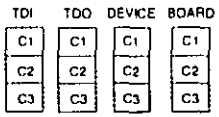


Table 2 defines the settings for each configuration

Table 2 JTAG Jumper Settings

Desired Action	TDI	TDO	DEVICE	BOARD
Program EPM7128S device only	C1 & C2	C1 & C2	C1 & C2	C1 & C2
Configure FLEX 10K device only	C2 & C3	C2 & C3	C1 & C2	C1 & C2
Program/configure both devices *(1)*	C2 & C3	C1 & C2	C2 & C3	C1 & C2
Connect multiple boards together *(2)*	C2 & C3	OPEN	C2 & C3	C2 & C3

Notes to Table 2:

(1) The first device in the JTAG chain is the FLEX 10K device, and the second device is the EPM7128S device.

(2) The first device in the JTAG chain is the FLEX 10K device, and the second device is the EPM7128S device. The last board in the chain must be set for a single board configuration (i.e., for programming only the EPM7128S device, configuring only the FLEX 10K device, or configuring/programming both devices). The last board cannot be set for connecting multiple boards together.

During configuration, the green CONF_D LED will turn off and the green TCK LED will modulate to indicate that data is transferring. After the device has successfully configured, the CONF_D LED will illuminate.

For information on how to program or configure the EPF10K70, or EPM7128S devices, see "Programming or Configuring Devices" on page 18.

EPM7128S Device

The UP2 Education Board provides the following resources for the EPM7128S device:

- Socket-mounted 84-pin PLCC package
- Signal pins that are accessible via female headers
- JTAG chain connection for the ByteBlaster II cable
- Two momentary push-button switches
- Two octal dual inline package (DIP) switches
- 16 LEDs
- Dual-digit seven-segment display
- On-board oscillator (25.175 MHz)
- Expansion port with 42 I/O pins and the dedicated global CLR, OE1, and OE2/GCLK2 pins

Pins from the EPM7128S device are not pre-assigned to switches and LEDs, but are instead connected to female headers. With direct access to the pins, students can concentrate on design fundamentals and learn about the programmability of I/O pins and PLDs. After successfully compiling and verifying a design with the MAX+PLUS II University software, students can easily connect the assigned I/O pins to the switches and LEDs using a common hook-up wire. Students can then download their design into the device and compare their design's simulation to the actual hardware implementation.

EPM7128S Prototyping Headers

The EPM7128S prototyping headers are female headers that surround the device and provide access to the device's signal pins. The 21 pins on each side of the 84-pin PLCC package connect to one of the 22-pin, dual-row 0.1-inch female headers. The pin numbers for the EPM7128S device are printed on the UP2 Education Board (an "X" indicates an unassigned pin). Table 3 lists the pin numbers for the four female headers: P1, P2, P3, and P4. The power, ground, and JTAG signal pins are not accessible through these female headers.

Table 3. Pin Numbers for Each Prototyping Header *Note (1)*

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

Note to Table 3:

(1) Inside refers to the row of female headers closest to the device. Outside refers to the row of female headers furthest from the device.

MAX_PB1 & MAX_PB2 Push-Buttons

MAX_PB1 and MAX_PB2 are two push-buttons that provide active-low signals and are pulled-up through 10-KΩ resistors. Connections to these signals are easily made by inserting one end of the hook-up wire into the push-button female header. The other end of the hook-up wire should be inserted into the appropriate female header assigned to the I/O pin of the EPM7128S device.

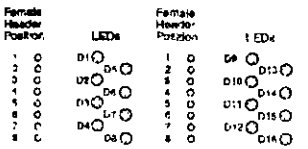
MAX_SW1 & MAX_SW2 Switches

MAX_SW1 and MAX_SW2 each contain eight switches that provide logic-level signals. These switches are pulled-up through 10-KΩ resistors. Connections to these signals are made by inserting one end of the hook-up wire into the female header aligned with the appropriate switch. Insert the other end of the hook-up wire into the appropriate female header assigned to the I/O pin of the EPM7128S device. The switch output is set to logic 1 when the switch is open and set to logic 0 when the switch is closed.

D1 through D16 LEDs

The UP Education Board contains 16 LEDs that are pulled-up with a 330-Ω resistor. An LED is illuminated when a logic 0 is applied to the female header associated with the LED. LEDs D1 through D8 are connected in the same sequence to the female headers (i.e., D1 is connected to position 1, and D2 is connected to position 2, etc.). LEDs D9 through D16 are connected in the same sequence to the female headers (i.e., D9 is connected to position 1, and D10 is connected to position 2, etc.). See Figure 3.

Figure 3. LED Positions



MAX_DIGIT Display

MAX_DIGIT is a dual-digit, seven-segment display connected directly to the EPM7128S device. Each LED segment of the display can be illuminated by driving the connected EPM7128S device I/O pin with a logic 0. Figure 4 shows the name of each segment.

Figure 4. Display Segment Name

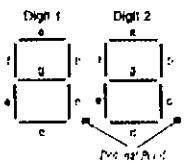


Table 4 lists the pin assignments for each segment

Table 4 MAX_DIGIT Segment I/O Connections

Display Segment	Pin for Digit 1	Pin for Digit 2
a	58	69
b	60	70
c	61	73
d	63	74
e	64	76
f	65	75
g	67	77
Decimal point	68	79

MAX_EXPANSION

MAX_EXPANSION is a dual row of 0.1-inch-spaced holes for accessing signal I/O pins and global signals on the EPM7128S device, power, and ground. Figure 5 shows the numbering convention for the holes.

Figure 5 MAX_EXPANSION Numbering Convention

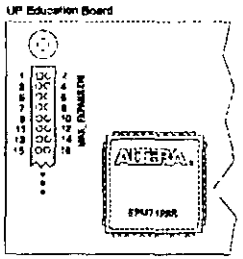


Table 5 lists the signal names and the EPM7128S device pins connected to each hole.

Table 5. MAX_EXPANSION Signal Names & Device Connections

Hole Number	Signal/Pin	Hole Number	Signal/Pin
1	RAW	2	GND
3	VCC	4	GND
5	VCC	6	GND
7	No Connect	8	No Connect
9	No Connect	10	No Connect
11	No Connect	12	GCLRn/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	25
31	24	32	27
33	29	34	28
35	31	36	30
37	33	38	34
39	35	40	36
41	37	42	40
43	39	44	41
45	44	46	46
47	45	48	48
49	50	50	49
51	52	52	51
53	54	54	55
55	56	56	57
57	VCC	58	GND
59	VCC	60	GND

FLEX 10K Device

The UP2 Education Board provides the following resources for the FLEX 10K device. The pins from the FLEX 10K device are pre-assigned to switches and LEDs on the board.

- JTAG chain connection for the ByteBlaster II cable
- Socket for an EPC1 configuration device
- Two momentary push button switches
- One octal DIP switch
- Dual-digit seven segment display
- On-board oscillator (25.175 MHz)
- VGA port
- Mouse port
- Three expansion ports, each with 42 I/O pins and seven global pins

FLEX_PB1 & FLEX_PB2 Push Buttons

FLEX_PB1 and FLEX_PB2 are two push buttons that provide active-low signals to two general-purpose I/O pins on the FLEX 10K device. FLEX_PB1 connects to pin 28, and FLEX_PB2 connects to pin 29. Each push button is pulled-up through a 10-KΩ resistor.

FLEX_SW1 Switches

FLEX_SW1 contains eight switches that provide logic-level signals to eight general-purpose I/O pins on the FLEX 10K device. An input pin is set to logic 1 when the switch is open and set to logic 0 when the switch is closed. Table 6 lists the pin assignment for each switch.

Table 6. FLEX_SW1 Pin Assignments

Switch	FLEX 10K 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

FLEX_DIGIT Display

FLEX_DIGIT is a dual-digit, seven-segment display connected directly to the FLEX 10K device. Each LED segment on the display can be illuminated by driving the connected FLEX 10K device I/O pin with a logic 0. See Figure 4 on page 8 for the name of each segment. Table 7 lists the pin assignment for each segment.

Table 7. FLEX_DIGIT Segment I/O Connections

Display Segment	Pin for Digit 1	Pin for Digit 2
a	6	17
b	7	18
c	8	19
d	9	20
e	11	21
f	12	23
g	13	24
Decimal point	14	25

VGA Interface

The VGA interface allows the FLEX 10K device to control an external video monitor. This interface is composed of a simple diode-resistor network and a 15-pin D-sub connector (labeled VGA), where the monitor can plug into the boards. The diode-resistor network and D-sub connector are designed to generate voltages that conform to the VGA standard.

Information about the color, row, and column indexing of the screen is sent from the FLEX 10K device to the monitor via five signals. Three VGA signals are red, green, and blue, while the other two signals are horizontal and vertical synchronization. Manipulating these signals allows images to be written to the monitor's screen.

See "VGA Driver Operation" on page 25 for details on how the VGA interface operates.

Table 8 lists the D-sub connector and the FLEX 10K device connections.

Table 8. D-Sub Connections

Signal	D-Sub Connector Pin	FLEX 10K Pin
RED	1	236
GREEN	2	237
BLUE	3	238
GND	6, 7, 8, 10, 11	–
HORIZ_SYNC	13	240
VERT_SYNC	14	239
No Connect	4, 5, 9, 15	–

Mouse Connector

The mouse interface, is a six-pin mini-DIN connector that allows the FLEX 10K device to receive data from a PS/2 mouse or a PS/2 keyboard. The board provides power and ground to the attached mouse or keyboard. The FLEX 10K device outputs the DATA_CLOCK signal to the mouse and inputs the data signal from the mouse. Table 9 lists the signal names and the mini-DIN and FLEX 10K pin connections.

See "Mouse Interface Operation" on page 28 for details on how the mouse interface operates.

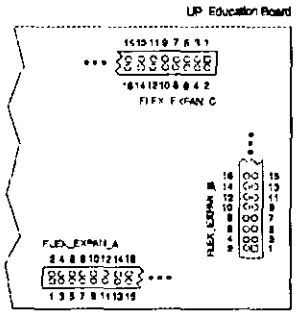
Table 9. Mouse Connections

Mouse Signal	Mini-DIN Pin	FLEX 10K Pin
MOUSE_CLK	1	30
MOUSE_DATA	3	31
VCC	5	–
GND	2	–

FLEX_EXPAN_A, FLEX_EXPAN_B & FLEX_EXPAN_C

FLEX_EXPAN_A, FLEX_EXPAN_B, and FLEX_EXPAN_C are dual rows of 0.1-inch spaced holes for accessing signal I/O pins and global signals on the FLEX 10K device, power, and ground. Figure 6 shows the numbering convention for these holes.

Figure 8. FLEX_EXPAN_A, FLEX_EXPAN_B & FLEX_EXPAN_C Numbering Convention



Tables 10 through 12 list the signal name and the FLEX 10K device pin connected to each hole.

Table 10. FLEX_EXPAN_A Signal Names & Device Connections

Hole Number	Signal/Pin	Hole Number	Signal/Pin
1	RAW	2	GND
3	VCC	4	GND
5	VCC	6	GND
7	No Connect	8	D11/90
9	D12/92	10	D13/210
11	D14/212	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

Table 11. FLEX_EXPAN_B Signal Names & Device Connections

Hole Number	Signal/Pin	Hole Number	Signal/Pin
1	RAW	2	GND
3	VCC	4	GND
5	VCC	6	GND
7	No Connect	8	D11/80
9	D12/92	10	D13/210
11	D14/212	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

Table 12 FLEX_EXPAN_C Signal Names & Device Connections

Hole Number	Signal/Pin	Hole Number	Signal/Pin
1	RAW	2	GND
3	VCC	4	GND
5	VCC	6	GND
7	No Connect	8	D11/90
9	D12/92	10	D13/210
11	D14/212	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	199
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

Software Installation

This section describes how to install the MAX+PLUS II University software for the Windows 98/2000 and Windows NT 4.0 operating systems.

After installation, students can register to obtain an authorization code via the Altera world-wide web site at the following URL: **http://www.altera.com/support/licensing/lic-university.html**.

For complete installation instructions, refer to the readme file on the *MAX+PLUS II University CD-ROM*, or see the *MAX+PLUS II Getting Started* manual.

Windows 98/2000 & Windows NT 4.0

Follow the below steps to install the MAX+PLUS II University software onto a PC

1. Insert the *MAX+PLUS II University CD-ROM* into the CD-ROM drive
2. Choose Run (Start menu)
3. Type *<CD-ROM drive>*`:\mp2_101se.exe` and click OK. The Setup Wizard will guide you through the installation procedure.

Programming or Configuring Devices

Programming or configuring the devices on the UP Education Boards requires setting the on-board jumpers and the JTAG programming options in the MAX+PLUS II software, and connecting the ByteBlaster II download cable to the PC's parallel port and to the JTAG_IN connector on the UP Education Boards. This section describes how to set these options.

- Program only the EPM7128S device
- Configure only the EPF10K70 device
- Configure/program both devices
- Connect multiple UP Education Boards together in a chain

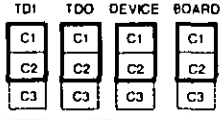
EPM7128S Programming

This section describes the procedures for programming only EPM7128S devices, (i.e., how to set the on-board jumpers, connect the ByteBlaster II download cable, and set options in the MAX+PLUS II software).

Setting the On-Board Jumpers for EPM7128S Programming

To program only the EPM7128S device in a JTAG chain, set the jumpers TDI, TDO, DEVICE, and BOARD as shown in Figure 7.

Figure 7. Jumper Settings for Programming Only the EPM7128S Device



Connecting the ByteBlaster II Download Cable for EPM7128S Programming

Attach the ByteBlaster II cable directly to the PC's parallel port and to the JTAG_IN connector on the board. For more information on setting up the ByteBlaster II cable, go to the *ByteBlaster II Parallel Port Download Cable Data Sheet*

Setting the JTAG Options in the MAX+PLUS II Software for EPM7128S Device Programming

The following steps describe how to use the MAX+PLUS II software to program the EPM7128S device in a JTAG chain.

For more information on how to use the MAX+PLUS II software, see the MAX+PLUS II Help.

1. Turn on the **Multi-Device JTAG Chain** command (JTAG menu) in the MAX+PLUS II Programmer to program a device. Follow this procedure even if you are only programming one device.
2. Choose **Multi-Device JTAG Chain Setup** (JTAG menu).
3. Select **EPM7128S** in the **Device Name** list in the **Multi-Device JTAG Chain Setup** dialog box.
4. Type the name of the programming file for the EPM7128S device in the **Programming File Name** box. You can use the **Select Programming File** button to browse a computer's directory structure to locate the appropriate programming file.
5. Click **Add** to add the device and associated programming file to the **Device Names & Programming File Names** box. The number to the left of the device name shows the order of the device in the JTAG chain. The device's associated programming file is displayed on the same line as the device name. If no programming file is associated with a device, "<none>" is displayed next to the device name.

6. Click **Detect JTAG Chain Info** to have the ByteBlaster II cable check the device count, JTAG ID code, and total instruction length of the JTAG chain. A message just above the **Detect JTAG Chain Info** button reports the information detected by the ByteBlaster II cable. This message must be manually verified to match the information in the **Device Names & Programming File Names** box.
7. Click **Save JCF**. In the **Save JCF** dialog box, type the name of the file in the **File Name** box and then select the desired directory in the **Directories** box to save the current settings to a JTAG Chain File (.jcf) for future use. Click **OK**.
8. Click **OK** to save changes.
9. Click **Program** in the MAX+PLUS II Programmer.

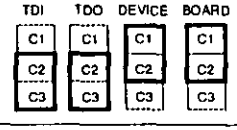
EPF10K70 Configuration

This section describes the procedures for configuring the EPF10K70 device (i.e., how to set the on-board jumpers, connect the ByteBlaster II download cable, and set options in the MAX+PLUS II software).

Setting the On-Board Jumpers for EPF10K70 Configuration

To configure the EPF10K70 device in a JTAG chain, set the jumpers TDI, TDO, DEVICE, and BOARD as shown in Figure 8.

Figure 8. Jumper Settings for Configuring Only the FLEX 10K Device



Connecting the ByteBlaster II Download Cable for the EPF10K70 Configuration

Attach the ByteBlaster II cable directly to the PC's parallel port and to the JTAG_IN connector on the UP Education Board. For more information on setting up the ByteBlaster II cable, see the *ByteBlaster II Parallel Port Download Cable Data Sheet*

Setting the JTAG Options in the MAX+PLUS II Software for EPF10K70 Configuration

The following steps describe how to use the MAX+PLUS II software to configure the EPF10K70 device in a JTAG chain.

For more information on how to configure a device, see the MAX+PLUS II Help.

1. Turn on the **Multi-Device JTAG Chain** command (JTAG menu) in the MAX+PLUS II Programmer to configure the EPF10K70 device. Follow this step even if you are only programming one device.
2. Choose **Multi-Device JTAG Chain Setup** (JTAG menu).
3. Select EPF10K70 in the **Device Name** list in the **Multi-Device JTAG Chain Setup** dialog box.
4. Type the name of the programming file for the EPF10K70 device in the **Programming File Name** box. You can also use the **Select Programming File** button to browse your computer's directory structure to locate the appropriate programming file.
5. Click **Add** to add the device and associated programming file to the **Device Names & Programming File Names** box. The number to the left of the device name shows the order of the device in the JTAG chain. The device's associated programming file is displayed on the same line as the device name. If no programming file is associated with a device, "<none>" is displayed next to the device name.
6. Click **Detect JTAG Chain Info** to have the ByteBlaster II cable check the device count, JTAG ID code, and total instruction length of the JTAG chain. A message just above the **Detect JTAG Chain Info** button reports the information detected by the ByteBlaster II cable. You must manually verify that this message matches the information in the **Device Names & Programming File Names** box.
7. Click **Save JCF** to save the current settings to a JCF for future use. Type the name of the file in the **File Name** box and then select the desired directory in the **Directories** box in the **Save JCF** dialog box. Click **OK**.
8. Click **OK** to save your changes.
9. Click **Configure** in the MAX+PLUS II Programmer.

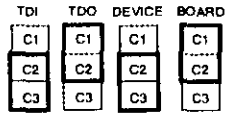
Configure/Program Both Devices

This section describes the procedures for configuring/programming both the FLEX 10K and EPM7128S devices in a JTAG chain, (i.e., how to set the on-board jumpers, connect the ByteBlaster II download cable, and set options in the MAX+PLUS II software).

Setting the On-Board Jumpers for Configuring/Programming Both Devices

To configure and program the FLEX 10K and EPM7128S devices in a multi-device JTAG chain, set the jumpers TDI, TDO, DEVICE, and BOARD as shown in Figure 9.

Figure 9. Jumper Settings for Configuring/Programming Both Devices



Connecting the ByteBlaster II Download Cable for Configuring & Programming Both Devices

Attach the ByteBlaster II cable directly to the PC's parallel port and to the JTAG_IN connector on the UP Education Board.

For more information on setting up the ByteBlaster II cable, see the *ByteBlaster II Parallel Port Download Cable Data Sheet*.

Setting the JTAG Options in the MAX+PLUS II Software for Configuring & Programming Both Devices

The following steps describe how to use the MAX+PLUS II software to configure and program both devices in a multi-device JTAG chain. For more information on how to program or configure a device, see the MAX+PLUS II Help.

1. Turn on the **Multi-Device JTAG Chain** command (JTAG menu).
2. Choose **Multi-Device JTAG Chain Setup** (JTAG menu).
3. Select the first target device name in the **Device Name** list in the **Multi-Device JTAG Chain Setup** dialog box.

4. Type the name of the programming file for the device listed in the **Device Name** box in the **Programming File Names** box. The **Select Programming File** button can also be used to browse your computer's directory structure to locate the appropriate programming file.

5. Click **Add** to add the device and associated programming file to the **Device Names & Programming File Names** box. The number to the left of the device name shows the device's order in the JTAG chain. The device's associated programming file is displayed on the same line as the device name. If no programming file is associated with a device, "<none>" is displayed next to the device name.

6. Repeat steps 3 through 5 to add information for each device in the JTAG chain.

7. Click **Detect JTAG Chain Info** to have the ByteBlaster II cable check the device count, JTAG ID code, and total instruction length of the multi-device JTAG chain. A message just above the **Detect JTAG Chain Info** button reports the information detected by the ByteBlaster II cable. You must manually verify that this message matches the information in the **Device Names & Programming File Names** box.

8. Click **Save JCF** to save the current settings to a JCF for future use. Type the name of the file in the **File Name** box and then select the desired directory in the **Directories** box. Click **OK**.

9. Click **OK** to save the changes.

10. Click **Configure** in the MAX+PLUS II Programmer to configure all FLEX 10K devices in the JTAG chain. Then, click **Program** to program all EPM7128S devices in the JTAG chain.

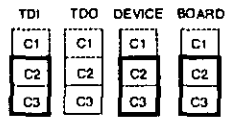
Connect Multiple UP Education Boards Together in a Chain

This section describes the procedures for connecting multiple UP Education Boards together (i.e., how to set the on-board jumpers, connect the ByteBlaster II download cable, and set options in the MAX+PLUS II software).

Setting the On-Board Jumpers for Connecting Multiple UP Education Boards Together

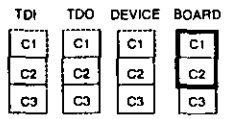
To configure/program EPM7128S and FLEX 10K devices on multiple UP Education Boards connected in a multi-device JTAG chain, set the jumpers TDI, TDO, DEVICE, and BOARD for all boards except the last board in the chain as shown in Figure 10.

Figure 10. Jumper Settings for All Boards Except the Last Board in the Chain



The last UP Education Board in the chain can configure and program one or both devices. However, the BOARD jumper must be set as shown in Figure 11.

Figure 11. Jumper Settings for the Last Board in the Chain



Connecting the ByteBlaster II Download Cable for Connecting Multiple UP Education Boards Together

Attach the ByteBlaster II cable directly to your PC's parallel port and to the JTAG_IN connector on the UP Education Board.

For more information on setting up the ByteBlaster II cable, see the *ByteBlaster II Parallel Port Download Cable Data Sheet*.

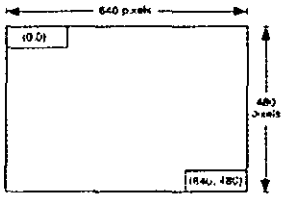
Setting the JTAG Options in the MAX+PLUS II Software for Connecting Multiple UP Education Boards

For information on how to set the JTAG options in the MAX+PLUS II software, see "Setting the JTAG Options in the MAX+PLUS II Software for Configuring & Programming Both Devices" on page 22

VGA Driver Operation

A standard VGA monitor consists of a grid of pixels that can be divided into rows and columns. A VGA monitor contains at least 480 rows, with 640 pixels per row, as shown in Figure 12. Each pixel can display various colors, depending on the state of the red, green, and blue signals.

Figure 12. VGA Monitor

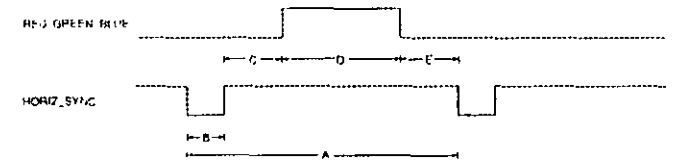


Each VGA monitor has an internal clock that determines when each pixel is updated. This clock operates at the VGA-specified frequency of 25.175 MHz. The monitor refreshes the screen in a prescribed manner that is partially controlled by the horizontal and vertical synchronization signals. The monitor starts each refresh cycle by updating the pixel in the top left-hand corner of the screen, which can be treated as the origin of an X–Y plane (see Figure 12). After the first pixel is refreshed, the monitor refreshes the remaining pixels in the row. When the monitor receives a pulse on the horizontal synchronization, it refreshes the next row of pixels. This process is repeated until the monitor reaches the bottom of the screen. When the monitor reaches the bottom of the screen, the vertical synchronization pulses, causing the monitor to begin refreshing pixels at the top of the screen (i.e., at [0,0]).

VGA Timing

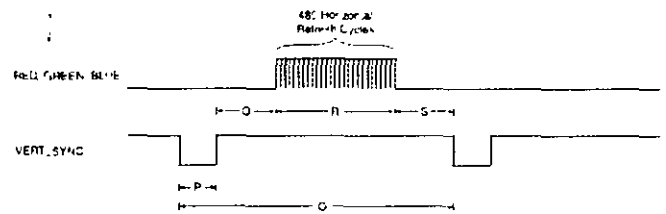
For the VGA monitor to work properly, it must receive data at specific times with specific pulses. Horizontal and vertical synchronization pulses must occur at specified times to synchronize the monitor while it is receiving color data. Figures 13 and 14 show the timing waveforms for the color information with respect to the horizontal and vertical synchronization signals.

Figure 13. Horizontal Refresh Cycle



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

Figure 14 Vertical Refresh Cycle



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

The frequency of operation and the number of pixels that the monitor must update determines the time required to update each pixel, and the time required to update the whole screen. The following equations roughly calculate the time required for the monitor to perform all of its functions:

$T_{pixel} = 1/f_{CLK} = 40$ ns

$T_{ROW} = A = B + C + D + E$
$= (T_{pixel} \times 640$ pixels$) +$ row + guard bands $= 31.77$ µs

$T_{screen} = O = P + Q + R + S$
$= (T_{ROW} \times 480$ rows$) +$ guard bands $= 16.6$ ms

Where: T_{pixel} = Time required to update a pixel
f_{CLK} = 25.175 MHz
T_{ROW} = Time required to update one row
T_{screen} = Time required to update the screen
B, C, E = Guard bands
P, Q, S = Guard bands

The monitor writes to the screen by sending red, green, blue, horizontal synchronization, and vertical synchronization signals when the screen is at the expected location. Once the timing of the horizontal and vertical synchronization signals is accurate, the monitor only needs to keep track of the current location, so it can send the correct color data to the pixel.

Mouse Interface Operation

You can connect a mouse to the UP Education Board via the 6-pin mini-DIN connector. The data is sent using a synchronous serial protocol, and the transmission is controlled by the CLK and DATA signals. During non-transmission, CLK is at logic 1 and DATA can be either logic 0 or logic 1.

Each transmission contains one start bit, eight data bits, odd parity, and one stop bit. Data transmission starts from the least significant bit (LSB), (i.e., the sequence of transmission is start bit, DATA0 through DATA7, parity, and stop bit). Start bits are logic 0, and stop bits are logic 1. Each clock period is 30 to 50 µsec; the data transition to the falling edge of the clock is 5 to 25 µsec. Table 13 shows the data packet format.

Table 13 Data Packet Format Note (1)

Packet Number	D7	D6	D5	D4	D3	D2	D1	D0
1	YV	XV	YS	XS	1	0	R	L
2	X7	X6	X5	X4	X3	X2	X1	X0
3	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0

Note to Table 13:
(1) Where: L = Left button state (1 = left mouse button is pressed down)
R = Right button state (1 = right mouse button is pressed down)
X0 – X7 = Movement in X direction
Y0 – Y7 = Movement in Y direction
XS, YS = Movement data sign (1 = negative)
XV, YV = Movement data overflow (1 = overflow has occurred)

The mouse operates on a Cartesian coordinate system (i.e., moving to the right is positive, moving to the left is negative, moving up is positive, and moving down is negative). The magnitude of the movement is a function of the mouse's rate of movement. The faster the mouse moves, the greater the magnitude.

Notes:

101 Innovation Drive
San Jose, CA 95134
(408) 544 7000
http://www.altera.com
University Program:
university@altera.com
Literature Services
(888) 3 ALTERA
lit_req@altera.com

3 0 EN UPG 0001