A la memoria de mi abuelo
quien fue más que un padre
Don Rodolfo Valeriano García †

AGRADECIMIENTOS

Profundamente a la Universidad Nacional Autónoma de México por su espíritu altruista único en México. Al Consejo Nacional de Ciencia y Tecnología por apoyar y confiar en el estudiante, mediante el otorgamiento de becas que le permiten continuar estudios de posgrado. Al Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, a todo su personal académico y administrativo que dan cede al posgrado en Ciencia e Ingeniería de la Computación, por sus enseñanzas y paciencia durante la estancia como estudiante. Al Centro de Ciencias Aplicadas y Desarrollo Tecnológico (CCADET) antes Centro de Instrumentos, por abrir sus puertas en el Departamento de Desarrollo Tecnológico en el Laboratorio de Electrónica donde se realizo la presente tesis. A mi director de tesis Dr. Felipe Lara Rosano por su apoyo incondicional en la realización de este trabajo. A mis sinodales M.I Arturo Haro Ruiz, Dr. Fernando Arámbula Cosío, Dr. Jesús Savage Carmona, M.I. Roberto Tovar Medina, por sus valiosos comentarios sobre el presente trabajo. A mis amigos Carlos Ojeda, Erika Martínez, Etna Cervantes, Gerardo Calva, José Castillo, Miguel Ángel Bañuelos, Rosendo Fuentes, Sandro Caballero, Sergio Quintana del grupo del Laboratorio de Electrónica et. al. del CCADET, por sus valiosos comentarios respecto al presente trabajo y por su apoyo moral. A mis amigos José Luis Juárez y Salvador Pérez Lomelí del grupo del Laboratorio de Ingeniería del Producto del Departamento de Ingeniería de Servicios del CCADET por su apoyo y paciencia en el diseño y construcción del prototipo. De corazón a Marisol Cortés Pérez por el tiempo que ha compartido conmigo en los momentos divertidos, felices, tristes y problemáticos que nos ha dado la vida durante más de dos años, tiempo que ha contribuido a acercarnos un poco más; así como por todo aquello que en común nos depare el destino. A la Facultad de Ingeniería y especialmente a Norma Elva Chávez por apoyarme moralmente siendo y demostrando ser una verdadera amiga en todo momento, en la recta final de este trabajo y en la publicación de material apoyado tanto por esta tesis como por otros trabajos en el área. A mi familia por apoyarme siempre en todos mis éxitos y errores, comprendiendo cada vez con más claridad el significado de la ley de la vida, bien dicho y entendido, por mi abuelo Don Rodolfo Valeriano. A Sofía Paredes, María Sofía y Gabriela Mendieta amigas de la FI por su participación en esta tesis. A todas aquellas personas e instituciones que por la emoción he omitido quiero dar en primer lugar una disculpa y en segundo lugar un profundo agradecimiento. A todos ustedes GRACIAS.

ÍNDICE

Tema **Pág.**

Introducción 1
Capítulo 1. La Tecnología de los Dispositivos Lógicos Programables Complejos CPLDs 4
1.1 Introducción 5
1.2 Memoria de solo lectura (ROM) 6
1.3 Arreglo Lógico Programable (PLA) 9
1.4 Lógica de Arreglo Programable (PAL) 11
1.5 Dispositivos Lógicos Programables (PLDs) 11
1.5.1 El caso CPLD MAX 7000S de ALTERA 14
1.5.2 El caso CPLD FLEX10K de ALTERA 20

Capítulo 2. Metodología Propuesta 34
2.1 Introducción 35
2.2 Metodología de las herramientas CAD-EDA 36
2.3 Metodología propuesta 39
2.3.1 Especificación y análisis del instrumento 41
2.3.2 Modelado del instrumento 44
2.3.2.1 Modelado del NÚCLEO por Comportamiento 48
2.3.2.2 Modelado de la interfaz de entrada por Comportamiento 56
2.3.2.3 Modelado de la interfaz de salida por Comportamiento 57
2.3.3 Elección del dispositivo CPLD 60
2.3.4 Compilación y síntesis 62
2.3.5 Simulación Digital 63
2.3.6 Programación del dispositivo 64
2.3.7 Pruebas y Depurado 65

Capítulo 3. Caso de aplicación Frecuencímetro Digital 66
3.1 Introducción 67
3.2 Especificación y Análisis del Frecuencímetro 68
3.3 Modelado del Frecuencímetro 71
3.3.1 Modelado Estructural del Núcleo del Frecuencímetro 71
3.3.2 Modelado Estructural de la Interfaz de Entrada del Frecuencímetro 77
3.3.3 Modelado Estructural de la Interfaz de Salida del Frecuencímetro 79
3.3.4 Modelado por Comportamiento del Núcleo del Frecuencímetro 81
3.3.5 Modelado por Comportamiento de la Interfaz de Entrada del Frecuencímetro 83
3.3.6 Modelado por Comportamiento de la Interfaz de Salida del Frecuencímetro 83
3.3.7 Escalas del Instrumento 85
3.4 Elección del Dispositivo CPLD 86
3.5 Compilación y Síntesis 87
3.6 Simulación Digital 89
3.7 Programación del Dispositivo 92
3.8 Pruebas y Resultados 93
3.9 Producto Final 99

Capítulo 4. Interfaz de salida gráfica VGA 100
4.1 Importancia y utilidad 101
4.2 Estructura a bloques 101
4.3 Controlador de video 102
4.4 Controlador VRAM 109
4.5 Aplicación al caso de un osciloscopio prototipo 120

Capítulo 5. Generador digital de tren de pulsos programable 125
5.1 Introducción 126
5.2 Diseño de un generador digital de tren de pulsos programable en frecuencia y ancho de pulso 126
5.3 Resultados 129

Capítulo 6. Conclusiones y perspectivas futuras 131
6.1 Ventajas de la Metodología 132
6.2 Aplicabilidad de la Metodología mas allá de la instrumentación 133
6.3 Desventajas de la metodología 133
6.4 Instrumentos propuestos para la aplicabilidad directa de la metodología 134
6.5 Trabajo futuro para enriquecer la metodología 134
6.6 Módulos aportados por el presente trabajo para la librería de funciones de la metodología 135
6.7 Material producto de este trabajo y de trabajo de terceros en la misma área 135
6.8 Ciclo de desarrollo en el diseño por comportamiento vs. diseño estructural 137
6.9 Costo del frecuencímetro prototipo 137

Referencias 138

INTRODUCCIÓN

Actualmente la tecnología de Lógica Programable es una alternativa real en el diseño de cualquier sistema digital. Hoy día el diseño digital plantea una complejidad creciente, especificaciones variables en el momento de desarrollo, necesidad de adaptabilidad, tiempos de desarrollo cada vez más cortos y búsqueda de costos menores. De igual manera se plantean ciertas exigencias como mayor confiabilidad y capacidades de depuración. Este tipo de solución permite los siguientes beneficios:

- Fácil adaptabilidad a cambios de diseño.
- Mayor rendimiento (desempeño).
- Mejor aprovechamiento de los recursos de ingeniería.
- Disminución de los costos de stock.

En nuestro país la lógica programable es especialmente útil, en primer lugar debido a que con una baja inversión inicial de $2000 dólares es posible contar con un sistema completo de desarrollo. En segundo lugar brinda la posibilidad de fabricar series reducidas al no requerir de máscaras y procesos básicos como en los ASICs (Application Specific Integrated Circuit) permitiendo el desarrollo de series mínimas e incluso unitarias. En tercer lugar la reutilización de los componentes permite cubrir con pocos productos el espectro completo de aplicaciones.

En la Fig. A podemos observar un gráfico comparativo de costos totales que brinda la Lógica Programable (LP) con respecto al volumen de partes fabricadas, en comparación con los ASIC y MPGA (Mask-Programmed Gate Arrays). Observamos que para pequeños volúmenes de fabricación es indiscutible que la LP es la mejor alternativa en relación a los costos; sin embargo también la LP es competitiva a medianos volúmenes de fabricación, siendo rebasada por las otras dos alternativas cuando se habla de enormes volúmenes de fabricación.

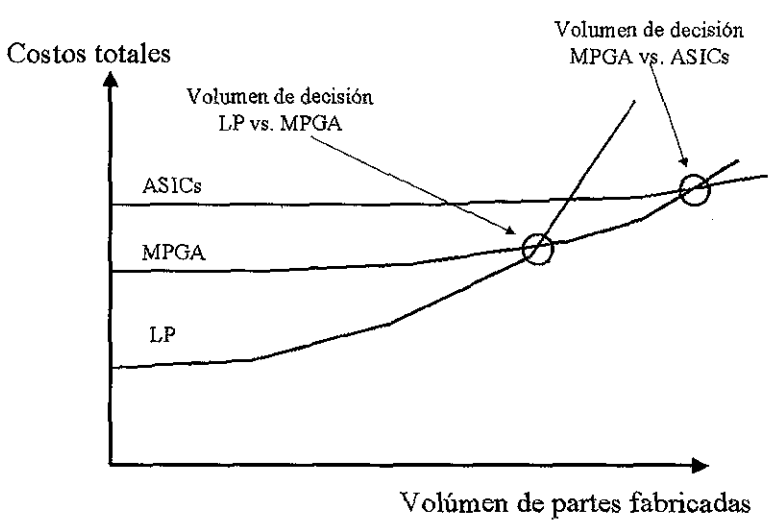


FIG. A. Comparación de Costos vs. Volumen de fabricación (Unidades)

Estas tecnologías ofrecen dispositivos programables de alta capacidad junto con herramientas de diseño digital asistido por computadora (CAD-EDA). Todo este conjunto trae los siguientes beneficios:

- *Diseño y Verificación más rápido en relación a ASICs y MPGAs.* Esto es porque la tecnología de LP puede ser diseñada y verificada rápidamente, mientras el mismo proceso requiere semanas con las otras alternativas. No se requieren de tiempos de retraso mientras se espera a que los prototipos sean manufacturados.

- *Los diseños cambian sin penalizaciones.* Debido a que los dispositivos son configurados por software y programados por el usuario, se logra que las modificaciones sean mucho menos riesgosas y puedan hacerse en cualquier momento, con un consumo de tiempo de minutos u horas dependiendo de la complejidad del diseño. Esto se refleja en el ahorro de costos.

- *Tiempo menor para la comercialización.* Cuando se diseña utilizando Lógica Programable el tiempo para poner en el mercado un producto final se puede medir en días o algunas pocas semanas, no en meses como en las otras alternativas.

Existe variedad en la tecnología de los Dispositivos Lógicos Programables, estas incluyen Arreglos Lógicos Programables (PLAs), Lógica de Arreglos Programables (PAL), Dispositivos Lógicos Programables Complejos (CPLDs) y Arreglos de Compuertas Programables por Campo (FPGAs).

De esta manera los PLDs son circuitos integrados cuya ultima funcionalidad es determinada por el diseñador; esto es, los fabricantes dejan los dispositivos en un estado de no programación. La cual es hecha cargando registros de almacenamiento interno, o provocando cambios físicos reversibles o no en partes selectas de los circuitos.

Como se dijo anteriormente los PLDs son una alternativa en el diseño de sistemas digitales complejos, esto abarca toda la gama del diseño digital. Las herramientas EDA-CAD incluidas por las empresas implican una metodología genérica para el diseño e implantación de diseños digitales. El hecho de tener estas herramientas y la metodología genérica si bien es de gran ayuda y acelera el proceso de diseño hasta la obtención del producto final, tiene aún un punto débil; esto es al ser tan general es poco orientada hacia algún tipo especial de diseños, por ejemplo diseño de microprocesadores, DSPs etc.

En el Laboratorio de Electrónica del Centro de Instrumentos de la Universidad Nacional Autónoma de México se diseñan instrumentos electrónicos como voltímetros, phímetros, fuentes de poder, multímetros, generadores de funciones etc. Muchos de los instrumentos diseñados son digitales; de modo tal que la presente tesis surge de la necesidad de contar con una metodología de diseño e implantación orientada a la instrumentación digital, que permita la utilización de la tecnología de los Dispositivos Lógicos Programable y sus ventajas para reducir tiempo de diseño e implantación de un instrumento final.

El desarrollo de este trabajo es muy importante para el laboratorio de electrónica porque sienta las bases de una metodología para el desarrollo de instrumentación digital utilizando la tecnología de LP. Los objetivos de esta metodología son:

- Reducir los tiempos desde la definición del problema hasta la creación del producto final.
- Plantea la máxima y provechosa utilización de los lenguajes de descripción de hardware (HDL) utilizando un estilo de descripción por comportamiento.
- Determinar los módulos genéricos que conforman cualquier instrumento digital.

Dos características de los PLDs que explota nuestra metodología son:

- *Programación en Sistema (In System Programming ISP).* Esto permite programar el diseño ya en el sistema final o equipo definitivo.
- *Reconfiguración en Circuito (In Circuit Reconfiguration ICR).* Propia de los dispositivos basados en SRAM, la reconfiguración puede ser en modo activo realizado por el mismo

dispositivo leyendo los datos de una EEPROM o en modo pasivo realizada por algún procesador externo.

Para demostrar la eficacia de la metodología se ha procedido al diseño de un frecuencímetro digital y se compara contra la versión digital implantada en el laboratorio de electrónica del Centro de Instrumentos utilizando diseño convencional con componentes discretos.

La metodología propuesta esta fundamentada en la experiencia adquirida en el diseño de sistemas digitales orientados a la instrumentación empleando lenguajes de descripción de hardware y pretende ser una guía general para el diseño de instrumentación digital y no para el diseño digital en general, como es planteado por las metodologías implícitas en las herramientas CAD-EDA.

En el capítulo 1 presentamos la tecnología de los dispositivos lógicos programables orientado fundamentalmente al estudio de los PLDs de la empresa Altera por varias razones:

- Se tiene experiencia en el diseño utilizando las herramientas CAD-EDA de la empresa Altera.
- Es más fácil diseñar utilizando CPLDs dado que el modelo de tiempo es más predecible y por tanto los diseños presentan menos problemas en cuanto a análisis temporal.
- La arquitectura interna de los CPLDs así como las herramientas de desarrollo son más simples en comparación con los FPGAs (Field Programmable Gate Array).

En el Capítulo 2 presentamos la metodología propuesta que engloba desde la etapa de especificación del problema hasta la obtención del producto final sobre circuito impreso (PCB). Se hace un énfasis especial en la etapa del modelado de la solución en alto nivel empleando únicamente lenguajes de descripción de hardware y haciendo siempre una descripción por comportamiento, con el objetivo en primer lugar de minimizar el tiempo de modelado, en segundo lugar de dar claridad al diseño y en tercer lugar facilitar el proceso de depuración.

En el Capítulo 3 presentamos un caso de aplicación con el cual ejemplificamos el desarrollo de la metodología propuesta. Se trata del diseño de un frecuencímetro que tiene un objetivo adicional que consiste en comparar el proceso de diseño y las ventajas y desventajas del producto final en relación a un diseño tradicional ya hecho en el Laboratorio de Electrónica.

Finalmente en el Capítulo 4 presentamos las conclusiones y una serie de perspectivas y trabajo futuro tanto para la aplicabilidad de la metodología así como para su enriquecimiento sucesivo.

CAPÍTULO 1

LA TECNOLOGÍA DE LOS DISPOSITIVOS LÓGICOS PROGRAMABLES COMPLEJOS CPLDs

CONTENIDO DEL CAPÍTULO

1.1 Introducción
1.2 Memoria de solo lectura (ROM)
1.3 Arreglo Lógico Programable (PLA)
1.4 Lógica de Arreglo Programable (PAL)
1.5 Dispositivos Lógicos Programables (PLD)
1.5.1 El caso CPLD MAX 7000S DE ALTERA
1.5.2 El caso CPLD FLEX10K DE ALTERA

1.1 INTRODUCCIÓN

Dentro de los dispositivos lógicos programables tenemos: memorias de solo lectura (ROM), arreglos lógicos programables (PLA), lógica de arreglos programable (PAL), dispositivos lógicos programables complejos (CPLD) y los arreglos de compuertas programables por campo (FPGA). En los PLDs las tecnologías de programación aplicadas son esencialmente las siguientes:

- La primera aplicación de las tecnologías de programación es para el control de conexiones. Una de las tecnologías esta basada en el uso de fusibles. Cada punto programable consiste en una conexión formada por un fusible. Cuando un voltaje considerablemente más grande que el voltaje de la fuente es aplicado a través del fusible, el valor de la corriente produce que se rompa la conexión fundiendo el fusible. Los dos estados de conexión existentes son "cerrado" y "abierto" representados por un fusible intacto y uno fundido respectivamente.

- La segunda tecnología de programación que también esta enfocada al control de conexiones es la programación de máscara, que es hecha por el fabricante del semiconductor durante los últimos pasos del proceso de fabricación. Las conexiones son realizadas o no realizadas en las capas de metal sirviendo como conductores en el chip. Dependiendo de la función deseada para el chip la estructura de estas capas es determinada por el proceso de fabricación. Sin embargo este procedimiento es costoso porque el vendedor hace un cargo extra especial al cliente por la máscara del dispositivo en particular. Por esta razón la programación de máscara es económica solo si una cantidad elevada de estos dispositivos con la misma configuración de máscara es ordenada.

- Una tercera tecnología de programación que controla conexiones es el "antifusible" como lo sugiere el nombre, el antifusible es justo lo opuesto al fusible. En contraste a un fusible el antifusible consiste de una área pequeña en la cual dos conductores están separados por un material que tiene una alta resistencia. El antifusible actúa como un camino abierto antes de la programación. Por la aplicación de un voltaje más grande que el voltaje normal de operación a través de los dos conductores, el material separador de estos dos conductores es derretido cambiando a un valor de resistencia. El material de resistencia baja se convierte en conductor proporcionando un camino cerrado.

Estas tres tecnologías de conexión son permanentes: los dispositivos no pueden ser reprogramados porque los cambios físicos irreversibles han ocurrido como un resultado de la programación previa. Debido a esto, si la programación es incorrecta el dispositivo debe ser desechado.

- La última tecnología de programación que puede ser aplicada al control de conexión es un bit de RAM estática (SRAM) manejando la compuerta de un transistor MOS en el punto de programación. Si el bit SRAM almacena un 1, el transistor es encendido y la conexión entre su fuente (source) y drenador (drain) forman un camino cerrado. Para el bit SRAM igual a 0, el transistor esta apagado y la conexión entre la fuente y el drenador es un camino abierto. Dado que el contenido del bit SRAM puede ser cambiado electrónicamente el dispositivo puede ser fácilmente reprogramado. Pero para almacenar el bit SRAM la fuente de energía suministrada debe estar disponible. Debido a esto, la tecnología basada SRAM es de tipo volátil, es decir, la programación lógica se pierde en la ausencia de energía suministrada.

La segunda aplicación de las tecnologías de programación es para la construcción de memorias RAM (look-up tables). En suma para controlar conexiones la tecnología SRAM es ideal para este tipo de tablas. Las entradas lógicas son entradas de dirección para leer la SRAM y las salidas

lógicas son para almacenar valores para la palabra direccionada que aparece en las salidas de datos de la SRAM. De esta forma la lógica puede ser implementada simplemente por el almacenamiento de la tabla de verdad en la SRAM, por lo tanto el término "look-up table".

1.2 MEMORIA DE SOLO LECTURA (ROM)

Una memoria de solo lectura (ROM) es esencialmente un dispositivo en el cual la información binaria es almacenada "permanentemente". La información debe ser especificada por el diseñador y es programada en la ROM para formar la interconexión requerida. Una vez que el patrón es establecido se queda en la ROM aun cuando la energía no este presente, es decir la ROM es no volátil.

El diagrama de bloques de un dispositivo ROM se muestra en la Fig. 1.1. Tiene "k" entradas y "n" salidas. Las entradas dan la dirección para la memoria y las salidas dan los bits de datos de la palabra de almacenamiento que ha sido seleccionada por la dirección. El número de palabras en una ROM es determinado por el hecho de que un valor "k" en la dirección de las líneas de entrada puede especificar 2^k palabras diferentes. Los chips ROM de circuitos integrados tienen uno o más entradas de habilitación que vienen con salidas de tres estados para facilitar la construcción de arreglos grandes de ROM.

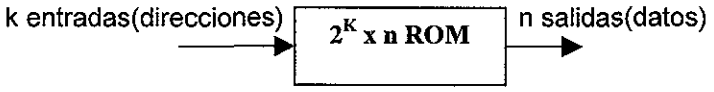


Fig. 1.1. Diagrama de Bloques de la ROM

Por ejemplo consideremos una ROM de 32x8 la unidad consiste de 32 palabras de 8 bits cada una. Hay 5 líneas de entrada que forman los números binarios desde 0 hasta 31 para las direcciones. En la Fig. 1.2 se muestra la construcción lógica interna de esta ROM. Contiene 5 entradas que son decodificadas a 32 distintas salidas, es decir, se tiene un decodificador de 5 a 32 líneas. Cada salida del decodificador representa una dirección de memoria.

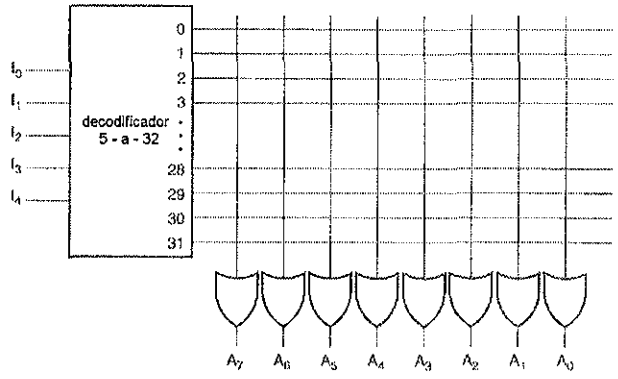


Figura 1.2. Lógica interna de una ROM de 32 x 8.

Las 32 salidas están conectadas a través de conexiones programables a cada una de las 8 compuertas OR. El diagrama usa la convención de arreglo lógico usado en los circuitos complejos. Cada compuerta OR debe ser considerada como de 32 entradas. Cada salida del decodificador

esta conectado a un fusible y a cada una de las entradas de la compuerta OR. Dado que se cuenta con 8 compuertas OR con 32 conexiones programables internas cada una, la ROM contiene 32x8 = 256 conexiones programables. En general una ROM de "2^k x n" tendrá un decodificador interno con " k - a - 2k " líneas y "n" compuertas OR. Cada compuerta OR tiene " 2k "entradas, las cuales están conectadas a través de las conexiones programables a cada una de las salidas del decodificador. El almacenamiento binario interno de una ROM es especificado por una tabla de verdad que muestra el contenido de una palabra en cada dirección. Por ejemplo, el contenido de una ROM de 32x8 puede ser especificado con la tabla de verdad que se muestra en la Tabla 1.1.

Entradas					Salidas							
I4	I3	I2	I1	I0	A7	A6	A5	A4	A3	A2	A1	A0
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
		.							.			
		.							.			
		.							.			
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

Tabla 1.1. Tabla de verdad de la ROM

Esta tabla de verdad muestra las 5 entradas en la parte superior, las cuales listan todas las 32 direcciones. Cada combinación de entrada especifica la dirección de una palabra de 8 bits que están en el lado derecho de la tabla. Sin embargo solo se observan las cuatro primeras y las cuatro últimas palabras en la ROM. La tabla completa debe incluir la lista de las 32 palabras.

El procedimiento que programa la ROM con conexiones que siguen la tabla de verdad especificada es sencillo. Por ejemplo si programamos la ROM de acuerdo a la Tabla 1.1 los resultados de la configuración se muestran en la Fig. 1.3.

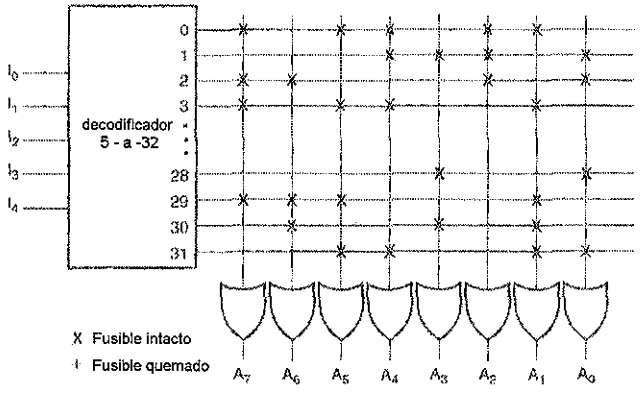


Figura 1.3. Programación de la ROM de acuerdo a la Tabla 1.1

Cada "0" listado en la Tabla 1.1 especifica un circuito abierto y un "1" especifica un circuito cerrado para ejemplificarlo tenemos en la tabla una palabra de 8 bits 10110010 para un almacenamiento permanente en la dirección de entrada 00011.Los cuatro ceros de la palabra están programados por una circuitería abierta entre la salida 3 del decodificador y las entradas de las compuertas OR asociadas con las salidas A6, A3, A2 y A0.

Los cuatro 1's en la palabra están marcados con una cruz en la Fig. 1.3 para designar un circuito cerrado. Cuando la entrada de la ROM es 00011, todas las salidas del decodificador son 0's excepto la salida 3 la cual es un "1" lógico. La señal que es equivalente a un "1" lógico en la salida 3 del decodificador se propaga a través de circuitos cerrados de las compuertas OR hacia las salidas A7, A5, A4 y A1. Las otras 4 salidas tienen un "0". El resultado es que la palabra 1011010 es aplicada a la salida de datos.

Cuatro son las tecnologías utilizadas para programar la ROMs. Si se utilizan fusibles, la ROM puede ser programada por el usuario, en este caso, la ROM es referida como una ROM programable o PROM. Si la ROM utiliza tecnología de compuertas flotantes borrables la ROM es referida como una ROM programable y borrable EPROM. Finalmente, sí la tecnología es borrable eléctricamente se llama EEPROM ó E2PROM.

Por otra parte en la Fig. 1.4 se muestran las configuraciones básicas de tres tipos de dispositivos lógicos programables PLDs, que serán detallados en secciones siguientes.

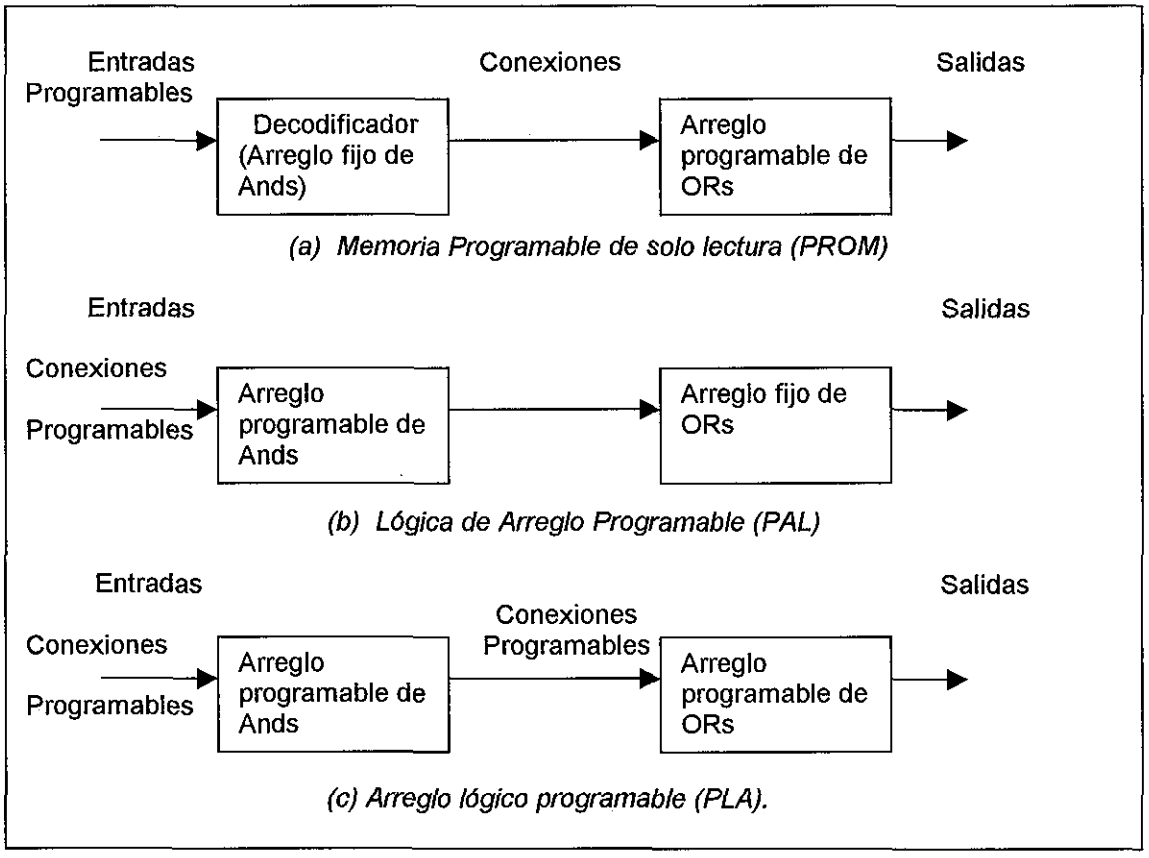


Fig. 1.4. Tres tipos de PLDs

1.3 ARREGLO LÓGICO PROGRAMABLE (PLA)

El PLA es similar en concepto a la PROM, excepto que el PLA no provee decodificación completa de las variables y no generan todos los mintérminos. El decodificador es reemplazado por un arreglo de compuertas AND que pueden ser programadas para generar términos de productos de las variables de entrada. Los términos de productos son entonces selectivamente conectados a compuertas OR para proveer la suma de productos requeridos para las funciones Boolenas.

La lógica interna de un PLA con 3 entradas y 2 salidas es mostrada en la Fig. 1.5. El diagrama usa un arreglo lógico de símbolos gráficos para circuitos complejos. Cada entrada va a través de un buffer y un inversor, representado en el diagrama por un símbolo gráfico compuesto que tiene tanto sus salidas verdaderas como sus complementos. Las conexiones programables van desde cada entrada y son complementadas a las entradas de cada compuerta AND, como se indica por las intersecciones entre las líneas verticales y horizontales. Las salidas de las compuertas AND tienen conexiones programables para las entradas de cada compuerta OR. Las salidas de las compuertas OR van a las compuertas XOR, donde las otras entradas pueden ser programadas para recibir una señal igual a un "1" lógico o un "0" lógico. La salida es invertida cuando la entrada de la compuerta XOR es conectada a 1 (porque X ⊕ 1 = X'). La salida no cambia cuando la entrada de la compuerta XOR es conectada a cero (porque X ⊕ 0 = X). Las funciones Boolenas en particular implementadas en la PLA de la Fig. 1.5 son:

$$F_1 = A\bar{B} + AC + \bar{A}B\bar{C}$$
$$F_2 = \overline{AC + BC}$$

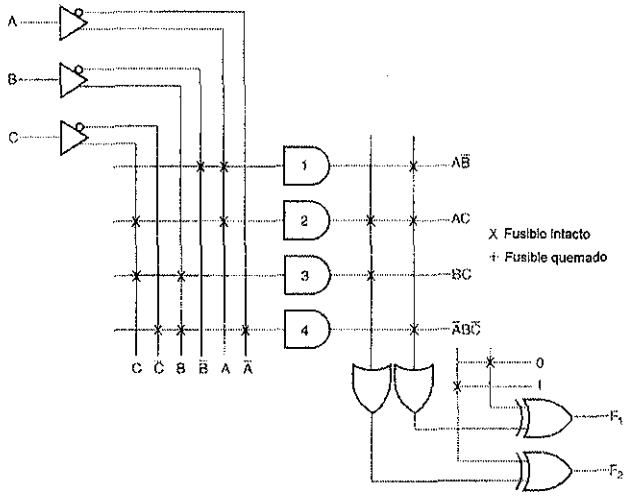


Fig. 1.5. PLA con tres entradas, cuatro términos producto y dos salidas

Los términos de productos generados en cada compuerta AND son listados por las salidas de las compuertas en el diagrama. El término producto es determinado por las entradas con conexiones de circuito cerrado. La salida de una compuerta OR proporciona la suma lógica de la selección de los términos productos. La salida puede ser complementada o dejarse en su forma verdadera dependiendo de la programación de las conexiones asociadas con las compuertas XOR.

El mapa de fusibles de un PLA puede ser especificado en forma tabular. Por ejemplo, la tabla de programación que especifica el PLA de la Fig. 1.5 es listada en la Tabla 1.2. La tabla tiene tres secciones, la primera contiene los números de los términos producto, la segunda especifica los

caminos requeridos entre las entradas y las compuertas AND, la tercera especifica los caminos entre las compuertas AND y OR. Para cada variable de salida se coloca una T (para verdadera) o una C(para complemento) para controlar las salidas de la compuerta XOR. Los términos productos listados a la izquierda no son parte de la tabla, estos están solo incluidos como referencia. Para cada término producto las entradas están marcadas con un "1", "0" ó "-". Si una variable en el término producto aparece en su forma verdadera, la variable de entrada correspondiente esta marcada con un "1". Si la variable en el término producto aparece complementada, la variable de entrada correspondiente aparece con un "0". Si la variable esta ausente en el término producto se marca con una raya.

Términos Producto		Entradas			Salidas	
		A	B	C	(T) F_1	(T) F_2
AB'	1	1	0	-	1	-
AC	2	1	-	1	1	1
BC	3	-	1	1	-	1
A'BC'	4	0	1	0	1	-

Tabla 1.2. Tabla de programación para el PLA de la Fig. 1.5

Los caminos entre las entradas y las compuertas AND son especificados debajo de la columna indicada como "Entradas" en la Tabla 1.2. Un "1" en la columna indica un circuito cerrado de la variable de entrada a la compuerta AND. Un "0" indica un circuito cerrado para el complemento de la variable a la entrada de la compuerta AND. Una raya especifica circuitos abiertos para ambas entradas normal y su complemento. Se asume que una terminal abierta en la entrada de una compuerta AND representa un 1.

Por otra parte los caminos entre las compuertas AND y OR se especifican en la columna que tiene el subtítulo de "Salidas". Las variables de salida son marcadas con 1's para aquellos términos productos que son incluidos en la función. Cada término de producto que tiene un 1 en la columna de salida requiere de un camino cerrado desde la salida de la compuerta AND hacia la entrada de la compuerta OR. Los términos producto marcados con una raya especifican un circuito abierto. Se asume que una terminal abierta en la entrada de una compuerta OR se comporta como un "0". Finalmente una T(true) en la salida indica que la otra terminal correspondiente de la compuerta XOR debe ser conectada un "0" y una C(complemento) especifica una conexión a un "1".

El tamaño de un PLA es especificado por el número de entradas, el número de términos de producto y el número de salidas. Un PLA típico tiene 16 entradas 48 términos de producto y 8 salidas. Para cada "n" entradas se tienen "k" términos producto y "m" salidas, la lógica interna del PLA consiste de "n" inversores, "k" compuertas AND, "m" compuertas OR y "m" compuertas XOR. Hay "2nk" conexiones programables entre las entradas y el arreglo de ANDs, "km" conexiones programables entre los arreglos de ANDs y arreglo de ORs, y "m" conexiones programables asociadas con las compuertas XOR.

En el diseño de un sistema digital con un PLA, no es necesario mostrar las conexiones internas de la unidad como se muestra en la Fig. 1.5. Todo lo que se necesita es una tabla de programación para el PLA que se programe para proporcionar la lógica requerida.

1.4 LÓGICA DE ARREGLO PROGRAMABLE (PAL)

El dispositivo PAL es un PLD con un arreglo de compuertas ORs fijas y un arreglo de compuertas ANDs programable. Debido a que solo las compuertas ANDs son programables, el dispositivo PAL es más fácil de programar que el PLA pero no tan flexible como este último. La Fig. 1.6 presenta la configuración lógica de un dispositivo típico PAL. El dispositivo particular que se muestra tiene 4 entradas y 4 salidas. Cada entrada tiene una compuerta buffer inversor, cada salida es generada por una compuerta OR fija, esto significa que hay 3 compuertas AND programables en cada sección. Cada AND tiene 10 conexiones de entrada programables, indicadas en el diagrama por la intersección de líneas verticales con cada línea horizontal. La línea horizontal simboliza la configuración de entrada múltiple de una compuerta AND. Una de las salidas mostrada esta conectada a una compuerta de buffer inversor y se realimenta hacia la entrada de la compuerta AND a través de las conexiones programables.

Los dispositivos comerciales PAL contiene más compuertas que la que se muestra en la Fig. 1.6. Un pequeño circuito integrado PAL puede tener 8 entradas, 8 salidas y 8 secciones que consisten de 8 arreglos AND-OR. Cada salida del dispositivo PAL es conducida por un buffer de 3 estados y sirve también como una entrada. Estas entradas/salidas pueden ser programadas para ser solo entradas, solo salidas o ser bidireccionales con una señal variable que maneja la señal habilitadora del buffer de 3 estados. Flip-flops son frecuentemente incluidos en un dispositivo PAL entre el arreglo y los buffers de 3 estados a las salidas. Desde que cada salida se alimenta como una entrada a través de la compuerta del buffer inversor hacia el arreglo de ANDs entonces un circuito secuencial puede ser fácilmente implementado.

1.5 DISPOSITIVOS LÓGICOS PROGRAMABLES (PLDs)

Los PLDs son dispositivos electrónicos digitales cuya funcionalidad puede ser programada por el usuario. El modo en que se implementa la programación depende de la arquitectura del dispositivo. En los dispositivos actuales se utilizan fundamentalmente dos tipos de estructuras programables.

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

Estructura PAL

Consiste en la programación de un número limitado de miniterminos agrupados en una OR lógica, como se ilustra en la Fig. 1.7.

Memoria RAM

Consiste en el almacenamiento en una memoria RAM de la tabla de verdad de una función combinacional, como se ilustra en la Fig. 1.8.

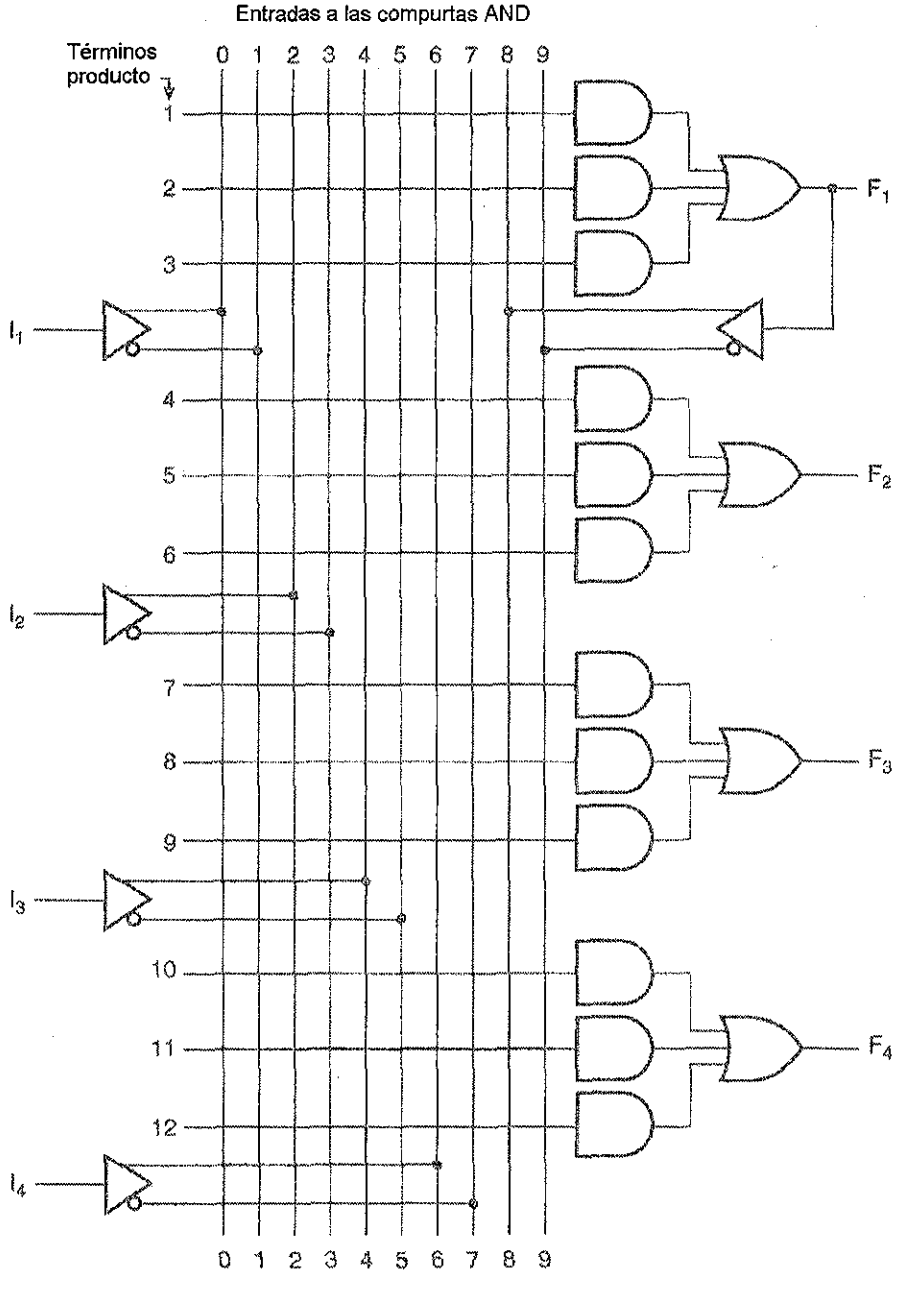


Fig. 1.6. PAL con cuatro entradas, cuatro salidas y tres AND-OR

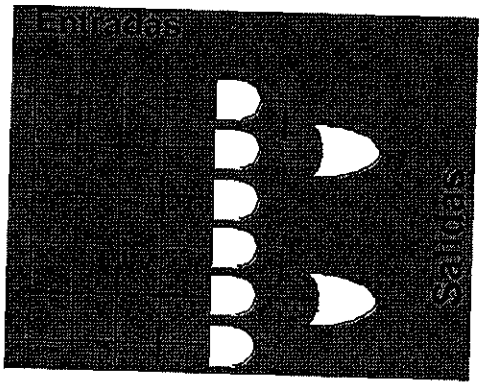


Fig. 1.7. Estructura básica de una PAL.

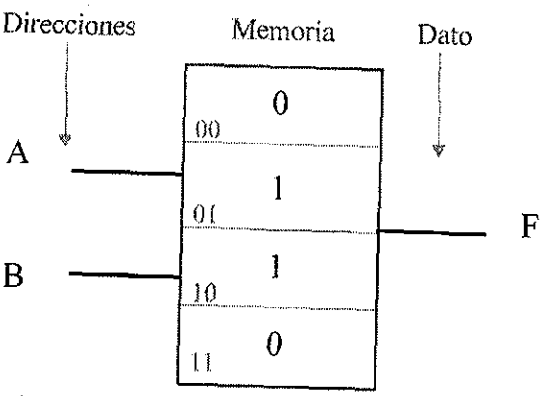


Fig. 1.8. Dispositivo programable con memoria RAM

Ventajas de los PLDs

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

Diferentes siglas para los PLDs

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

Clasificación de los Dispositivos Lógicos Programables.

El esquema de la Fig. 1.9 es un cuadro sinóptico de una clasificación típica de los dispositivos lógicos programables de acuerdo a la densidad de integración.

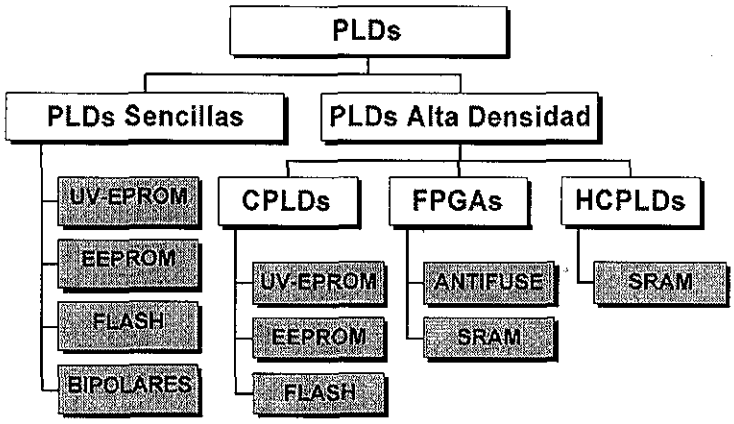


Fig. 1.9. Clasificación de los PLDs

Modos de Programación.

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

1.5.1 EL CASO CPLD MAX 7000S DE ALTERA.

DESCRIPCIÓN FUNCIONAL.

La arquitectura del CPLD MAX 7000S incluye los siguientes elementos:

- Bloques de arreglos lógicos.
- Macroceldas.
- Expansores de términos producto (compartidos y paralelos).
- Arreglos de interconexión programables.
- Bloques de control de entrada/salida (I/O).

La arquitectura MAX 7000S incluye cuatro entradas dedicadas que pueden ser usadas como entradas de propósito general o como señales de alta velocidad, están son las señales de control global (clock, clear y dos señales de salida de habilitación) para cada macrocelda y pin de I/O. La Fig. 1.10 muestra la arquitectura del dispositivo MAX 7000S.

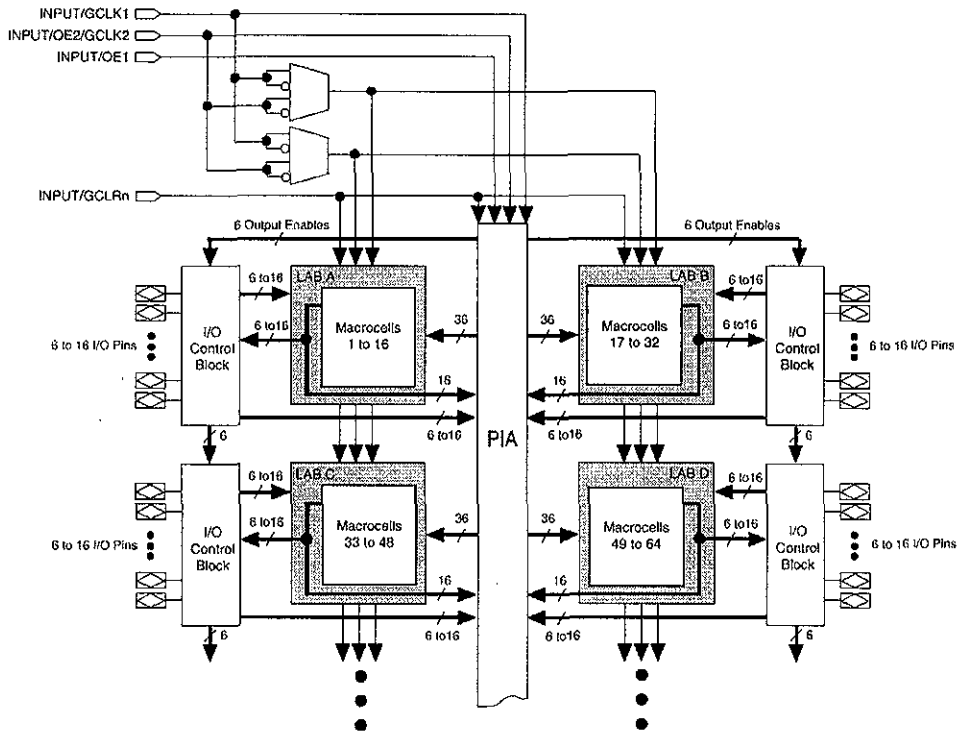


Fig. 1.10. Arquitectura del dispositivo MAX 7000S. Cortesía Altera

Bloques de Arreglos Lógicos.

La arquitectura del dispositivo MAX 7000S esta fundamentada en la conexión de arreglos de bloques lógicos (LABs) flexibles y de alto rendimiento. Los LABs consisten de arreglos de 16 macroceldas, como se muestra en la Fig. 1.10. Múltiples LABs son interconectados vía un arreglo de interconexión programable (PIA) es decir un bus global que es alimentado por todas las entradas dedicadas, pines de I/O y macroceldas.

Cada LAB es alimentado por las siguientes señales:

- 36 señales del PIA que son usadas como entradas lógicas generales.
- Controles globales que son usados par funciones de registro secundario.
- Caminos de entradas directas desde los pines de I/O a los registros que son usados para tiempos de acceso pequeños para los dispositivos MAX7000S.

Macroceldas.

La macrocelda del dispositivo MAX 7000S puede ser configurada individualmente para cualquiera de las operaciones lógicas secuencial o combinacional. Las macroceldas consisten de tres bloques funcionales: el arreglo lógico, la matriz de selección de los términos producto y el registro programable. La macrocelda del dispositivo MAX 7000S se ilustra en la Fig. 1.11.

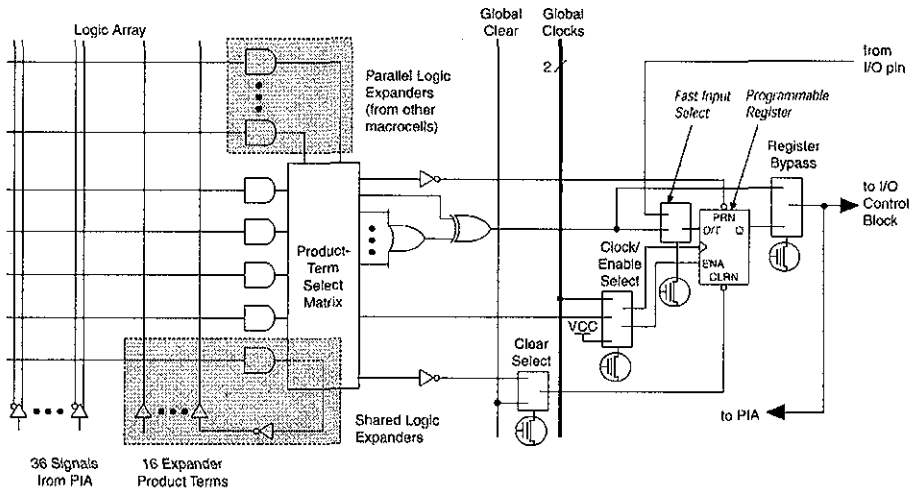


Fig. 1.11 Macrocelda del dispositivo MAX 7000S. Cortesía Altera.

La lógica combinacional es implementada en el arreglo lógico, el cuál provee cinco términos producto de hasta 36 variables por macrocelda. La matriz de selección de términos de producto distribuye estos términos para usarlos como entradas lógicas primarias (a las compuertas OR y XOR) para implementar funciones combinacionales, o bien como entradas secundarias a las líneas de control clear, preset, clock y habilitación de los registros de la macrocelda. Dos tipos de expansores de términos producto están disponibles para proporcionar más recursos lógicos a la macrocelda:

- *Expansores Compartidos.* Estos son términos producto invertidos que son realimentados al arreglo lógico.
- *Expansores Paralelos.* Estos son términos producto que son prestados por macroceldas adyacentes.

Para funciones secuenciales, cada flip-flop de la macrocelda puede ser individualmente programado para implementar operación D, T, JK o SR con el control de reloj programable. El flip-flop puede ser omitido para realizar una operación combinacional.

Cada registro programable puede ser sincronizado en tres diferentes modos:

- *Por una señal de reloj global.* Este modo logra el funcionamiento más rápido desde el flanco de reloj a la obtención de la salida.
- *Por una señal de reloj global y habilitada por un habilitador activo alto.* Este modo provee un habilitador en cada flip-flop logrando un funcionamiento rápido desde el flanco del reloj a la obtención de la salida.
- *Por un arreglo de reloj implementado con un término de producto.* En este modo, el flip-flop puede ser sincronizado por señales alambradas dentro de las macroceldas o de los pines de I/O.

En el dispositivo MAX7000S, dos señales globales de reloj están disponibles. Como se observa en la Fig. 1.11. Estas señales globales de reloj pueden ser "verdaderas" o "complementadas" de los pines globales de reloj, GCLK1 o GCLK2. Cada registro también soporta funciones asíncronas de preset y clear. La matriz de selección de términos producto almacena los términos producto para controlar esas operaciones. Aunque los términos producto que manejan el preset y clear del registro son activos altos, el control activo bajo puede ser obtenido invirtiendo la señal dentro del arreglo lógico. En adición, cada función de clear del registro puede ser manejado individualmente por un pin dedicado de clear global mediante activo bajo(GCLRn).

Todos los pines de I/O de los dispositivos MAX7000S, tienen una ruta de entrada más rápida a los registro de la macroceldas. Esta ruta dedicada permite a una señal pasar por alto el PIA y la lógica combinacional y ser manejada como una entrada "D" de un flip-flop con un tiempo de carga muy rápido(2.5 ns).

Expansores de términos producto

Aunque muchas funciones lógicas pueden ser implementadas con los cinco términos producto disponibles en cada macrocelda, existen funciones lógicas más complejas que requieren de términos producto adicionales. En este caso otra macrocelda puede ser usada para suministrar los recursos lógicos requeridos; sin embargo, la arquitectura del MAX7000S provee dos tipos de expansores para los términos producto llamados "compartido" y "paralelo". Estos proveen términos producto adicionales directamente a cualquier macrocelda en el mismo LAB. Esos expansores ayudan a garantizar que la lógica es sintetizada con los menores recursos lógicos para obtener la velocidad más rápida posible.

Expansores Compartidos.

Cada LAB tiene 16 expansores compartidos que pueden ser vistos como términos producto individuales no dedicados (uno de cada macrocelda) con salidas invertidas que se realimentan dentro del arreglo lógico. Cada expansor compartido puede ser usado por una o todas las macroceldas en el LAB para construir funciones lógicas complejas. Un pequeño retardo(t_{SEXP}) es producido cuando los expansores compartidos son usados. La Fig. 1.11 muestra cómo los expansores compartidos pueden alimentar a múltiples macroceldas.

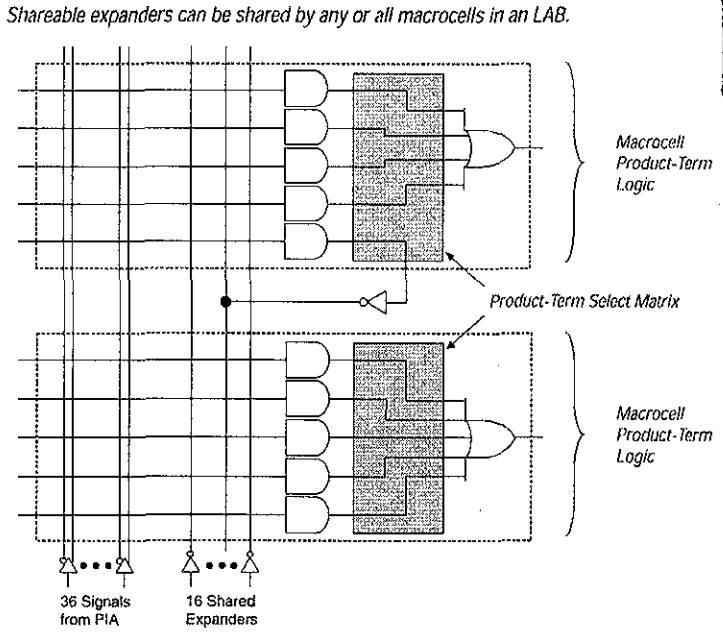


Fig. 1.11. Expansores Compartidos. Cortesía Altera.

Expansores Paralelos.

Los expansores paralelos son términos producto no usados que pueden ser utilizados en una macrocelda vecina para implementar funciones lógicas, complejas y rápidas. Los expansores paralelos permiten hasta 20 términos producto directamente alimentados desde la compuerta OR de las 15 macroceldas vecinas, mas los 5 términos de producto suministrados por la macrocelda

en uso. La Fig. 1.12 muestra cómo los expansores paralelos pueden ser tomados de una macrocleda vecina.

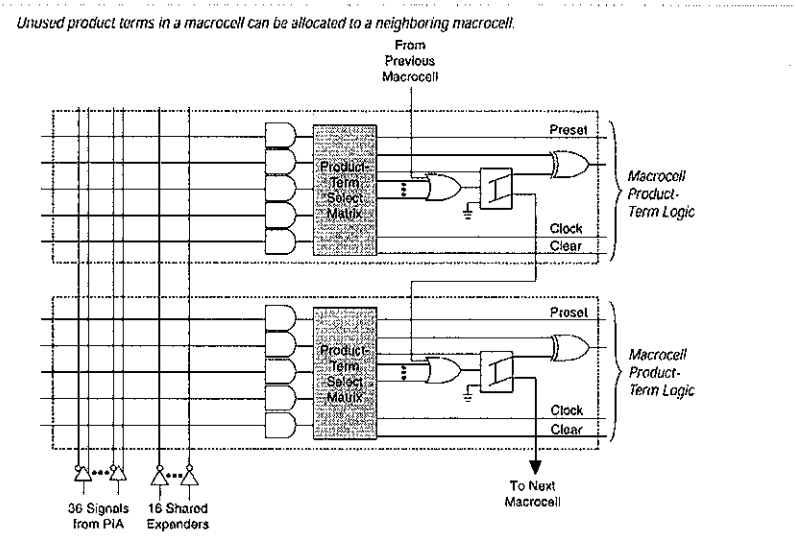


Fig. 1.12. Expansores Paralelos. Cortesía Altera.

Arreglo de Interconexión Programable.

La lógica es ruteada entre los LABs utilizando el arreglo de interconexión programable (PIA). Este bus global es una ruta programable que conecta cualquier señal de origen a cualquier destino en el dispositivo. Todas las entradas dedicadas, los pines de I/O y las salidas de las macroceldas alimentan el PIA, el cuál hace disponibles todas estas señales a través del dispositivo completo. Solo las señales requeridas por cada LAB se rutean desde el PIA hacia los LAB. La Fig. 1.13 muestra cómo las señales del PIA son ruteados dentro del LAB. Una celda EEPROM controla una entrada a una compuerta AND de dos entradas, la cuál selecciona una señal del PIA para manejarla dentro del LAB. Característica importante de este esquema de interconexión es el tiempo fijo de propagación.

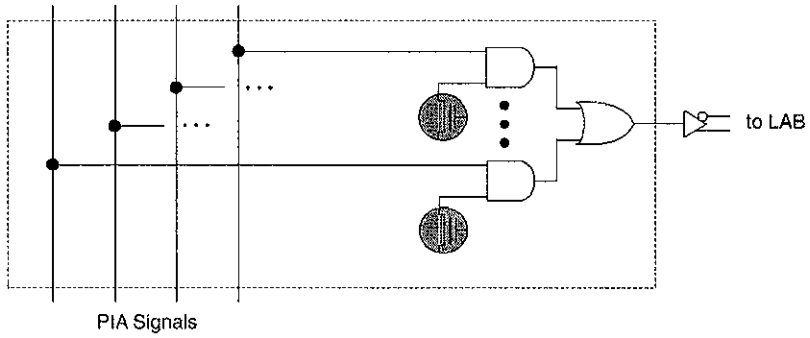


Fig. 1.13. Ruteo de las señales del PIA. Cortesía Altera

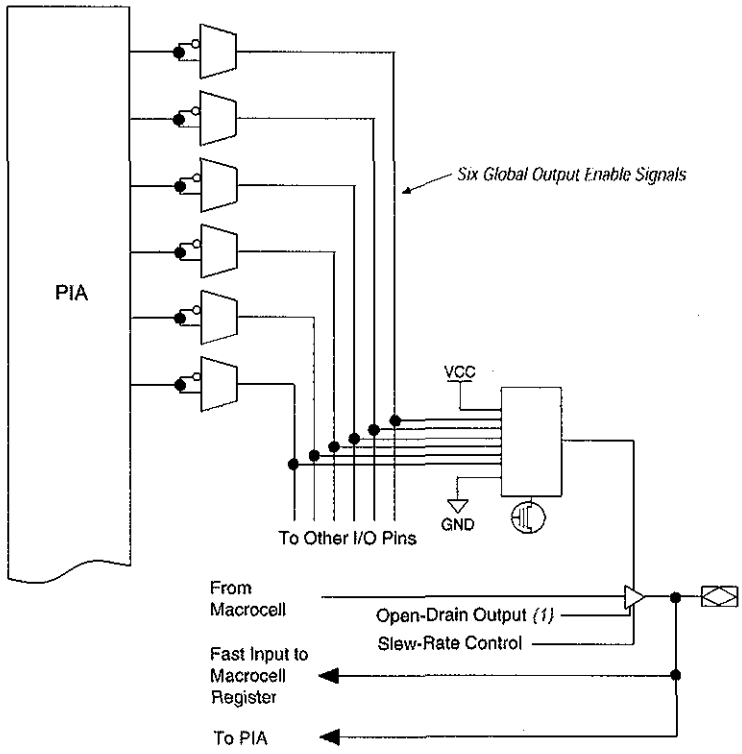
Bloques de Control de I/O.

Los bloques de control de I/O permiten que cada pin de I/O sea individualmente configurado para funcionar como entrada, salida o bidireccional. Todos los pines de I/O tienen buffers de tres estados que son individualmente controlados por una de las señales de habilitación de salida

global o directamente conectados a tierra o Vcc. La Fig. 1.14 muestra el bloque de control de I/O para la familia MAX 7000S. El bloque de control de I/O tienen 6 señales de habilitación de salida que son manejadas por las señales verdaderas o complementadas provenientes de del PIA (Arreglo de Interconexión Programable).

Cuando el control del buffer de tres estados es conectado a tierra, la salida está en alta impedancia y el pin de I/O puede ser usado como una entrada dedicada. Cuando el control del buffer de tres estados es conectado a Vcc la salida es habilitada.

La arquitectura del MAX 7000S provee una realimentación dual, en la cual la macrocelda y los pines de realimentación son independientes. Cuando un pin de I/O es configurado como una entrada, la macrocelda asociada puede ser usada como lógica alambrada.

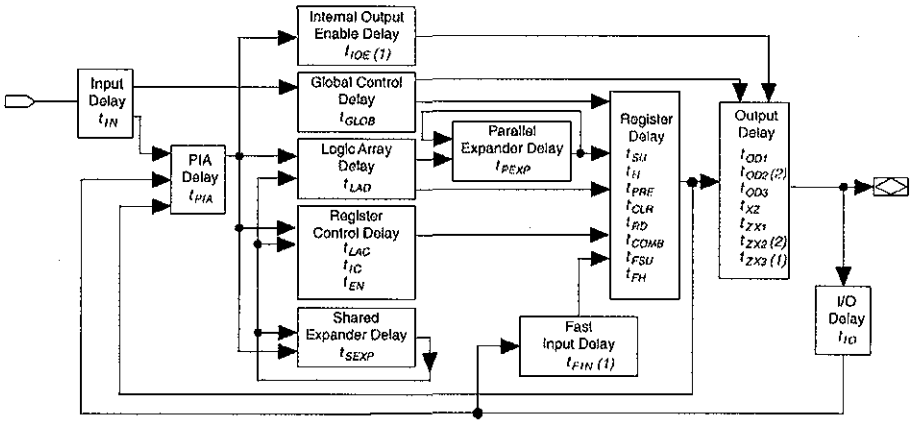


Note:
(1) The open-drain output option is available in MAX 7000S devices only.

Fig 1.14. Bloque de control de I/O. Cortesía Altera.

MODELO DE TIEMPO.

Los dispositivos MAX 7000S tienen retardos internos fijos que permiten al diseñador determinar el peor caso en tiempo de cualquier diseño. Las características de tiempo de la ruta seguida por una señal puede ser obtenida a partir de el modelo de tiempo y parámetros de un dispositivo particular. La Fig. 1.15 muestra el modelo de tiempo del MAX 7000S.



Notes:
(1) Only available in MAX 7000E and MAX 7000S devices.
(2) Not available in 44-pin devices.

Fig. 1.15. Modelo de tiempo del MAX 7000S. Cortesía Altera.

1.5.2 EL CASO CPLD FLEX10K DE ALTERA

DESCRIPCIÓN FUNCIONAL

Cada dispositivo FLEX10K contiene un arreglo embebido para implementar funciones de lógica especializada (memoria RAM), también contiene un arreglo lógico para implementar lógica general.

El arreglo embebido consiste de series de bloques de arreglos embebidos (EABs). Cuando se implementan funciones de memoria cada EAB provee 2,048 bits los cuales pueden ser usados para crear RAM, ROM, RAM de puerto dual, o funciones FIFO(first-in first-out). Cuando se implementa lógica cada EAB puede contribuir desde 100 a 600 compuertas para funciones lógicas complejas, como multiplicadores, microcontroladores, máquinas de estados y funciones DSP. Las EABs pueden ser usadas independientemente o múltiples EABs pueden ser combinadas para implementar funciones más grandes y complejas.

El arreglo lógico consiste de bloques de arreglos lógicos(LABs). Cada LAB contiene 8 elementos lógicos (LEs) y una interconexión local. Cada LE consiste de una tabla de 4 entradas (LUT "Look-up Table"), un flip-flop programable, y rutas de señales dedicadas para funciones de acarreo y cascada. Los 8 LEs pueden ser usados para crear bloques de tamaño mediano de lógica (contadores de 8 bits, decodificadores de direcciones o máquinas de estados) o combinarlos a través de los LABs para crear bloques lógicos más complejos. Cada LAB representa alrededor de 96 compuertas lógicas disponibles.

Las interconexiones de señales dentro del dispositivos FLEX 10K que conectan pin con pin de I/O del dispositivo son hechos mediante series de interconexión de renglón y columna de alta velocidad conocido como "FastTrack Interconnect". Estas interconexiones se propagan en forma horizontal y vertical dentro del dispositivo.

Cada pin de I/O es alimentado por un elemento de I/O (IOE) localizado al final de cada renglón y columna del "FastTrack Interconnect". Cada IOE contiene un buffer de I/O bidireccional y un flip-

flop que puede ser usado como un registro de entrada o salida para alimentar señales de entrada, salida o bidirecionales. Cuando se usan con un pin dedicado de reloj, estos registros proveen un funcionamiento de alto desempeño. Como entradas, ellos proveen tiempos de propagación tan bajos como 1.6 ns y tiempos de espera de 0 ns; como salidas, esos registros proveen tiempos de propagación tan bajos como 5.3 ns. IOEs provee una variedad de características como buffers de tres estados y salidas colector abierto.

En la Fig. 1.16 se muestra un diagrama de bloques de la arquitectura del FLEX 10K. Cada grupo de LEs es combinado dentro de un LAB; los LABs son organizados en columnas y renglones. Cada renglón también contiene un EAB. Los LABs y EABs son interconectados por el "FastTrack Interconnect". Los IOEs están localizados al final de cada renglón y columna del "FastTrack Interconnect".

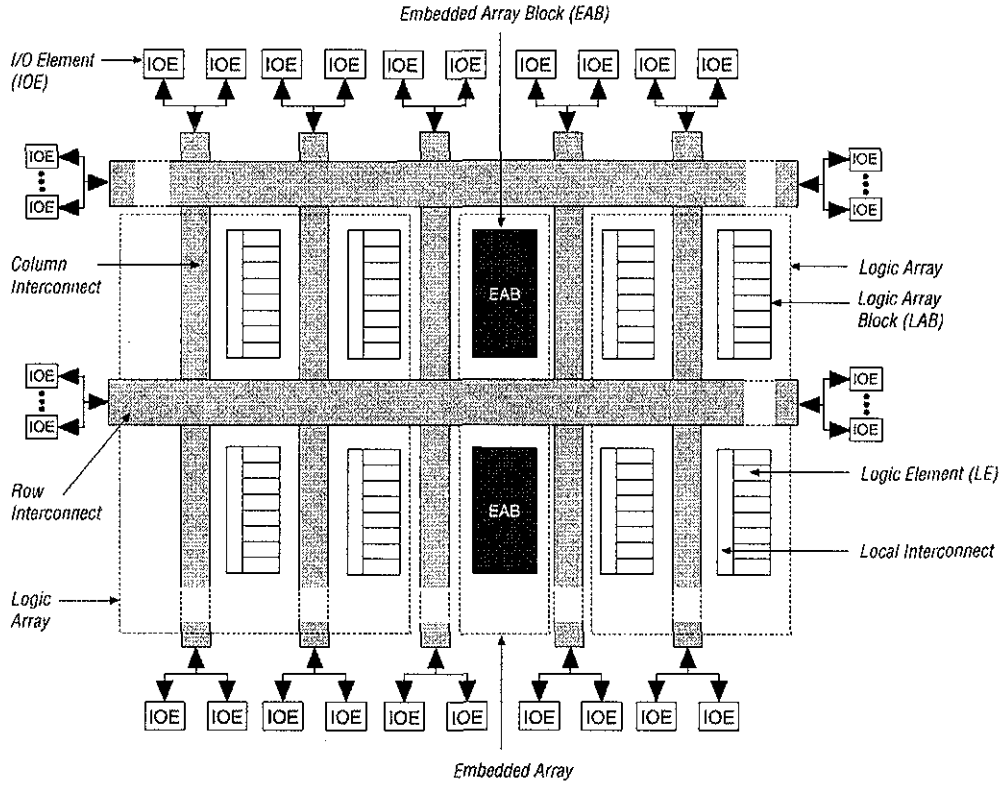


Fig. 1.16. Diagrama a bloques del FLEX 10K. Cortesía Altera.

Los dispositivos FLEX 10K proveen 6 entradas dedicadas que manejan las entradas de control de los flip-flops para asegurar una distribución eficiente a alta velocidad de las señales de control. Estas señales usan canales de ruteo dedicados que proveen retardos más cortos y menos desfasamientos que el "FastTrack Interconnect". Cuatro de las entradas dedicadas manejan cuatro señales globales. Esas cuatro señales globales pueden también ser manejadas por una lógica interna, proveyendo una solución ideal para un divisor de frecuencia o como una señal de clear asíncrona generada internamente que limpia muchos registros en el dispositivo.

Bloque de Arreglo Embebido.

Un EAB es un bloque flexible de RAM embebido con registros en los puertos de salida y entrada, y es usado para implementar megafunciones de arreglos de compuertas comunes. El EAB es también conveniente para implementar funciones como multiplicadores, vectores escalares y

circuitos de corrección de errores, por que es grande y flexible. Estas funciones pueden ser combinadas en aplicaciones como filtros digitales y microcontroladores.

Las funciones lógicas son implementadas mediante la programación del EAB con un patrón de solo lectura durante la configuración, creando una larga LUT. Con las LUTs, las funciones combinacionales son implementadas mediante la búsqueda de los resultados en vez de calcularlos. Esta implementación de las funciones combinacionales es mucho más rápida que usando algoritmos implementados en lógica general, una ventaja de funcionamiento que es resaltada por los tiempos de acceso cortos de los EABs.

Los EABs pueden ser usados para implementar RAM síncrona, la cuál es más fácil de usar que la RAM asíncrona. Un circuito que usa una RAM asíncrona debe generar una señal de habilitación de escritura (WE), la cual asegura que las señales de datos y de direcciones estén estables y soportar los tiempos relativos a la señal WE. En contraste, la RAM síncrona del EAB genera su propia señal WE y ésta se sincroniza con respecto al reloj global. Cuando se usa como una RAM, cada EAB puede ser configurada en cualquiera de los siguientes tamaños: 256 x 8, 512 x 4, 1024 x 2 o 2048 x 1. La Fig. 1.17 ilustra lo dicho anteriormente.

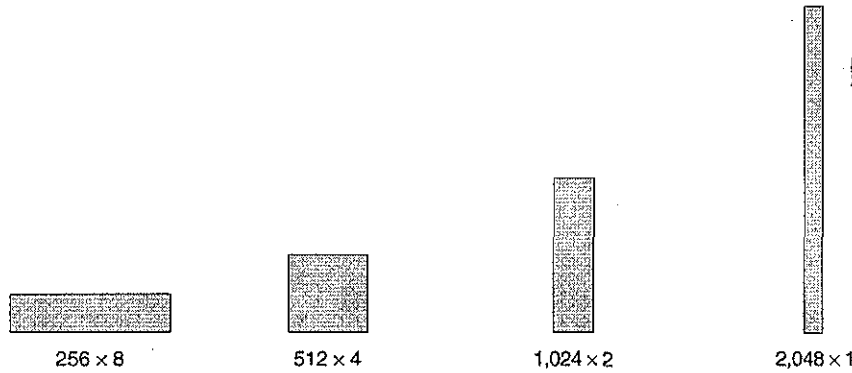


Fig. 1.17. Configuraciones posibles para los EABs. Cortesía Altera.

Bloques de RAM más grandes son creados mediante la combinación de múltiples EABs. Por ejemplo, dos bloques de RAM de 256x8 pueden ser combinados para formar un bloque de RAM de 256x16; dos bloques de RAM de 512x4 pueden ser combinados para formar un bloque de RAM de 512x8, como se ilustra en la Fig. 1.18.

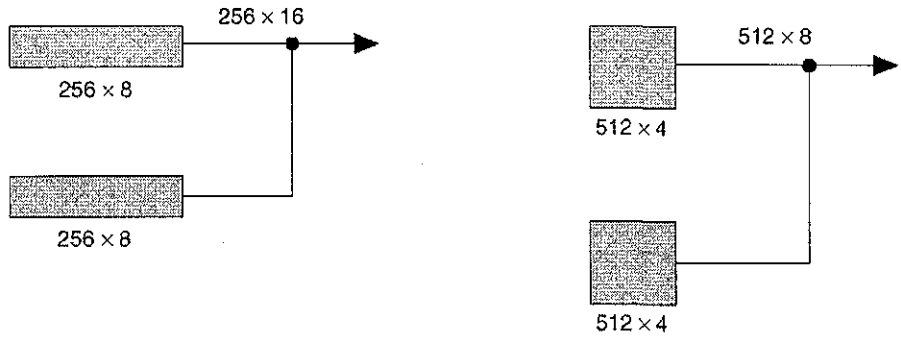


Fig. 1.18. Combinación de bloques EABs para formar bloques de RAM más grandes. Cortesía Altera.

Si es necesario, todos los EABs en un dispositivo pueden estar en cascada para formar un solo bloque de RAM. El EAB provee opciones flexibles para manejar y controlar señales de reloj. Diferentes relojes pueden ser usados para las entradas y salidas de los EABs. Registros pueden estar independientemente insertados en la entrada de datos, la salida del EAB, las entradas de

direcciones o de la señal WE. Las señales globales y la interconexión local del EAB puede manejar la señal WE. Las señales globales, pines de reloj dedicados, y la interconexión local del EAB pueden manejar las señales de reloj del EAB. Debido a que los LEs manejan la interconexión local del EAB, los LEs pueden controlar la señal WE o las señales de reloj del EAB.

Cada EAB es alimentado por una interconexión renglón y puede guiar las interconexiones de renglones y columnas. Cada salida del EAB puede manejar hasta dos canales de renglones y hasta dos canales columnas; el canal renglón sin usar puede ser manejado por otros LEs. Esta característica incrementa los recursos de ruteo disponibles para las salidas del EAB, como se observa en la Fig. 1.19.

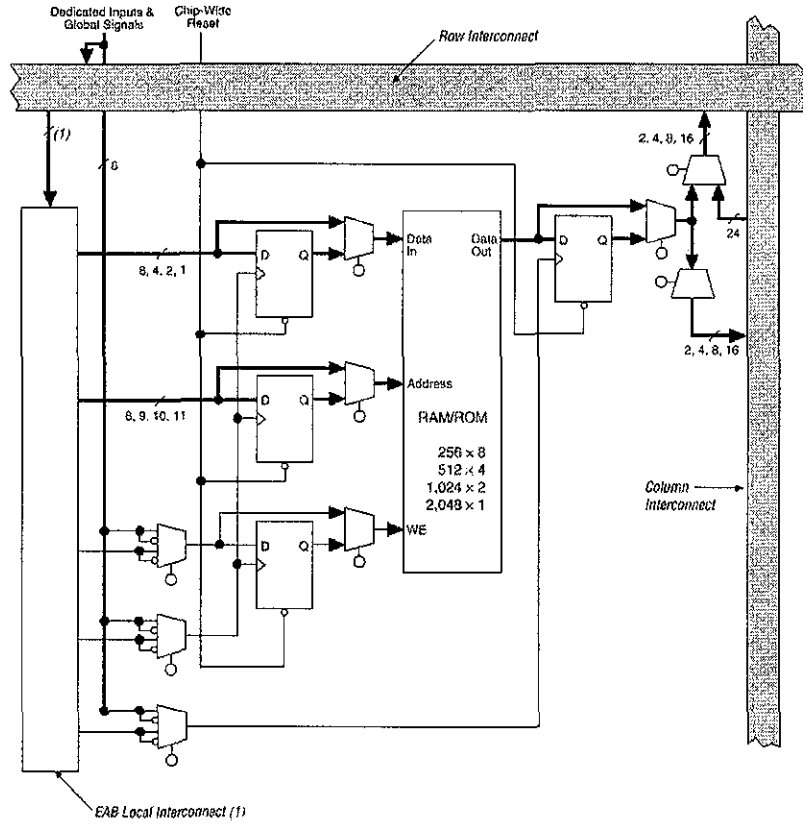


Fig. 1.19. Bloque de arreglos embebidos. Cortesía Altera.

Bloque de Arreglo Lógico.

Cada LAB consiste de 8 LEs, cadenas de acarreo y cascada, señales de control del LAB, y la interconexión local del LAB. El LAB provee la estructura de grano-fino que es característica de la arquitectura FLEX 10K, facilitando el ruteo eficiente con una utilización óptima del dispositivo y un alto desempeño. La Fig. 1.20 ilustra el LAB.

Cada LAB provee cuatro señales de control con inversor programable que pueden ser usadas en los ocho LEs. Dos de esas señales pueden ser utilizadas como relojes; las otras dos pueden ser usadas como señales de control clear/preset. Los relojes del LAB pueden ser manejados por los pines de entrada de los relojes dedicados, señales globales, señales de I/O, o señales internas vía la interconexión local del LAB. Las señales de control globales, preset y clear, del LAB pueden ser manejadas por las señales globales, señales de I/O, o señales internas vía la interconexión local

del LAB. Las señales de control globales son típicamente usadas para señales globales de reloj, clear o preset porque estas proveen un control asíncrono con un pequeño desfasamiento a través del dispositivo. Si lógica adicional es requerida en una señal de control, puede ser generada en uno o más LEs de cualquier LAB y manejada dentro de la interconexión local del LAB destino. Adicionalmente las señales de control globales pueden ser generadas de las salidas del LE.

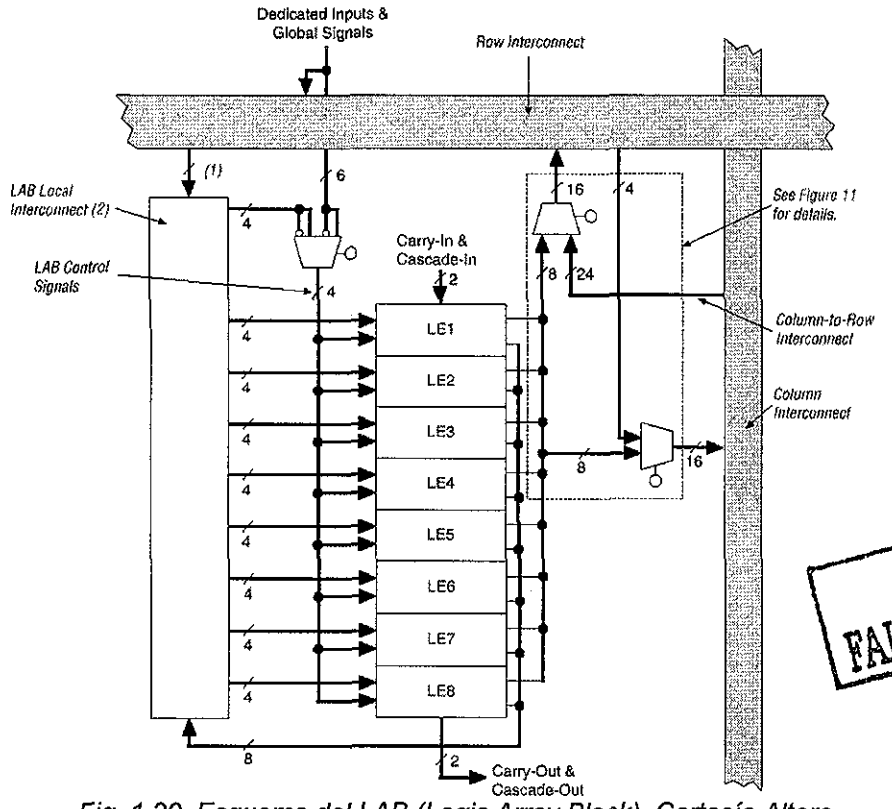


Fig. 1.20. Esquema del LAB (Logic Array Block). Cortesía Altera.

Elemento Lógico (LE).

El LE es la unidad más pequeña de lógica en la arquitectura del dispositivo FLEX 10K. Tiene un tamaño compacto que proporciona una utilización lógica eficiente. Cada LE contiene una LUT (look-up Table) de cuatro entradas, la cuál se comporta como un generador de funciones que puede rápidamente procesar cualquier función de cuatro variables. En adición, cada LE contiene un flip-flop programable con un habilitador síncrono, una línea de acarreo y una línea para conexión en cascada. Cada LE maneja tanto la interconexión local como la interconexión "FastTrack". La Fig. 1.21. ilustra la arquitectura del elemento lógico.

El flip-flop programable en el LE puede ser configurado para una operación D, T, JK o SR. Las señales de control: reloj, clear y preset en el flip-flop pueden ser manejados por señales globales, por los pines de I/O de propósito general, o por cualquier lógica interna. Para funciones combinacionales, los flip-flops son omitidos y la salida de la LUT maneja la salida del LE. El LE tiene dos salidas que manejan la interconexión; una maneja la interconexión local y la otra maneja la interconexión FastTrack tanto de renglón como de columna. Las dos salidas pueden ser controladas independientemente, por ejemplo, la LUT puede manejar una salida mientras el registro maneja la otra salida. Esta característica, llamada empaquetamiento de registros, pueden mejorar la utilización del LE por que el registro y la LUT pueden ser usados para funciones independientes.

La arquitectura de la FLEX 10K proporciona dos tipos de rutas de datos de alta velocidad que conectan LEs adyacentes sin el uso de rutas locales de interconexión, estas son las cadenas de acarreo y las cadenas de cascada. La cadena de acarreo soporta la implementación de contadores y sumadores de alta velocidad; la cadena en cascada permite la implementación de funciones con un mínimo retraso. Las cadenas en cascada y de acarreo conectan todos los LEs en un LAB y todos los LABs en el mismo renglón. El uso intensivo de las cadenas de acarreo y en cascada pueden reducir la flexibilidad de ruteo. Por lo tanto, el uso de estas cadenas puede ser limitado por las porciones de velocidad crítica del diseño.

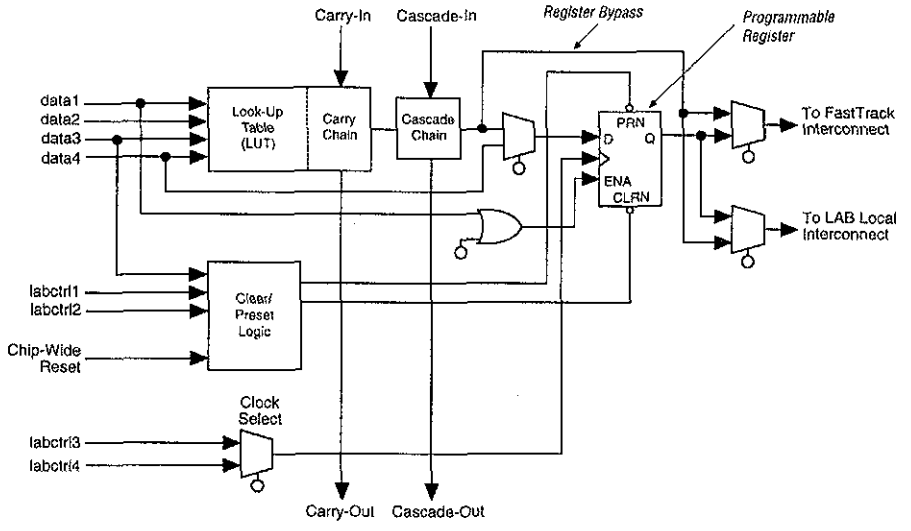


Fig. 1.21. Arquitectura del elemento lógico. Cortesía Altera.

Cadena de Acarreo.

La cadena de acarreo proporciona una función de acarreo hacia adelante muy rápida (0.2 ns) entre LEs. La señal de acarreo de entrada de los bits de menor orden se desplazan hacia los bits de mayor orden vía una cadena de acarreo y alimenta a la LUT y a la siguiente porción de la cadena de acarreo. Esta característica permite a la arquitectura de la FLEX 10K implementar contadores, sumadores de alta velocidad y comparadores de alta eficiencia. Las cadenas de acarreo más largas de 8 LEs son automáticamente implementadas por medio de enlazamientos de LABs contiguos.

La Fig. 1.22 muestra cómo un sumador completo de n-bits puede ser implementado en n+1 LEs con la cadena de acarreo. Una porción de la LUT genera la suma de dos bits usando las señales de entrada y la señal de carry-in, la suma es ruteada a la salida del LE utilizando la cadena de acarreo. El registro puede ser omitido por los sumadores simples o ser usado por una función acumuladora.

Cadena en Cascada.

Con una cadena en cascada, la arquitectura de la FLE 10K puede implementar funciones que tengan un alto fan-in. La LUT adyacente puede ser usada para procesar las porciones de la función en paralelo. La cadena en cascada puede usar una AND lógica o un a OR lógica para conectar las salidas de los LEs adyacentes. Cada LE adicional proporciona cuatro entradas más al ancho efectivo de una función, con un retraso de solo 0.7ns por LE.

Las cadenas en cascada más largas de 8 bits son implementadas automáticamente mediante el enlace de muchas LABs. La Fig. 1.23 muestra cómo la cadena en cascada puede conectar LEs adyacentes para formar funciones con un gran fan-in. Estos ejemplos muestran funciones de "4n" variables implementadas con "n" LEs. El retraso del LE es de tan solo 1.6ns; el retraso de la cadena en cascada es tan solo de 0.7ns. Usando cadena en cascada se requieren de solo 3.7ns para decodificar direcciones de 16 bits.

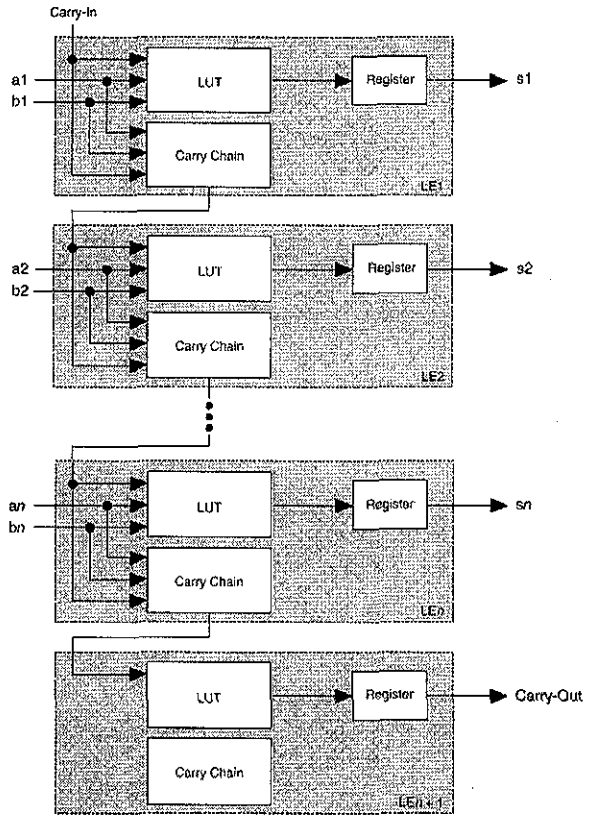


Fig. 1.22. Operación de la cadena de acarreo usando un full adder de n-bits. Cortesía Altera.

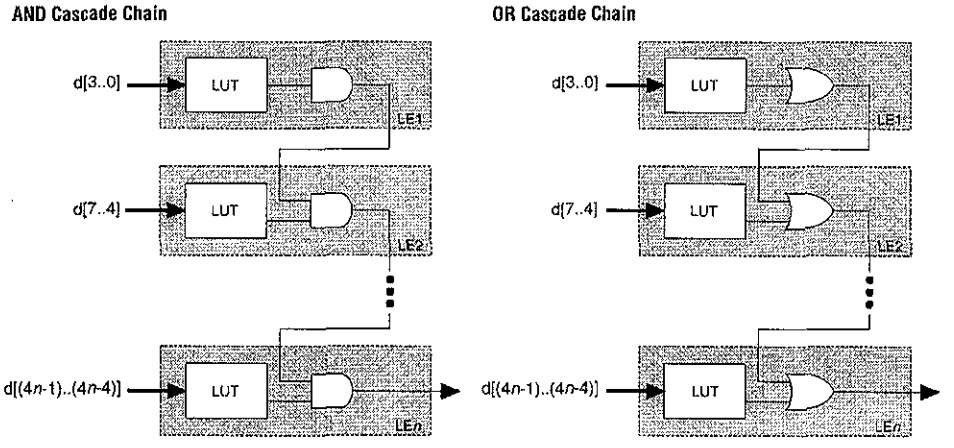


Fig. 1.23. Operación de la cadena de cascada para construir funciones de gran fan-in. Cortesía Altera.

Modos de Operación del LE.

El LE del dispositivo FLEX 10K puede operar en los siguientes cuatro modos:

Modo normal.
Modo Aritmético.
Modo Contador arriba/abajo.
Modo Contador borrable.

Cada uno de esos modos usa los recursos del LE de diferentes maneras. En cada modo existen siete entradas disponibles al LE que son las cuatro entradas de dato de la interconexión local del LAB, la realimentación del registro programable, el carry-in y cascada-in del previo LE; están directamente a diferentes destinos para implementar la función lógica deseada. Los modos de operación del LE se muestra en la Fig. 1.24.

Modo Normal.

El modo normal es conveniente para aplicaciones lógicas generales y funciones de decodificación de alto fan-in que pueden tomar ventaja de la cadena en cascada. En modo normal, las cuatro entradas de datos de la interconexión local del LAB y el carry-in son entradas a una LUT de 4 entradas. La salida de la LUT puede ser combinada con la señal cascada-in para formar una cadena de cascada a través de la señal cascada-out. Tanto el registro como la LUT pueden ser usados para manejar tanto la interconexión local como la interconexión FastTrack al mismo tiempo.

La LUT y el registro en el LE pueden ser usados independientemente, esta característica es conocida como empaquetamiento de registro. Para soportar esta característica de registro, el LE tiene dos salidas; una maneja la interconexión local y la otra maneja la interconexión FastTrack. La señal DATA4 puede manejar el registro directamente, permitiendo a la LUT procesar una función que es independiente de la señal registrada; una función de tres entradas puede ser procesada en la LUT, y una cuarta señal independiente puede ser registrada. Alternativamente, una función de cuatro entradas puede ser generada, y una de las entradas a esta función puede ser usada para manejar el registro. El registro en un LE empaquetado puede todavía usar el habilitador de las señales de reloj, clear y preset en el LE. En LE empaquetado, el registro puede manejar la interconexión FastTrack mientras la LUT maneja la interconexión local, o viceversa.

Modo Aritmético.

El Modo Aritmético ofrece dos LUTs de tres entradas cada una que son ideales para implementar sumadores, acumuladores y comparadores. Una LUT procesa una función de tres entradas, y la otra genera una salida de acarreo. Como se muestra en la Fig. 1.24, la primera LUT usa la señal carry-in y dos entradas de datos de la interconexión local del LAB para generar una salida combinacional o registrada. Por ejemplo, en un sumador, esta salida es la suma de tres señales: "a", "b" y "carry-in". La segunda LUT usa las mismas tres señales para generar una señal "carry-out" para crear una cadena de acarreo. El modo aritmético también soporta simultáneamente el uso de la cadena en cascada.

Modo Contador arriba/abajo.

Este modo ofrece un habilitador del contador, un habilitador del reloj, un control síncrono arriba/abajo y opciones de carga de datos. Esas señales de control son generadas por las entradas de datos de la interconexión local de LAB, la señal de carry-in y una realimentación de salida del registro programable. Se utilizan dos LUTs de tres entradas, una genera los datos del contador y la otra genera el bit de acarreo. Un multiplexor 2 a 1 proporciona carga asíncrona. Los datos pueden ser también cargados en un modo asíncrono con las señales de control, clear y preset, sin el uso de los recursos de la LUT. Esto se muestra en la Fig. 1.24.

Modo Contador Limpiable.

Este modo es similar al de contador arriba/abajo, a excepción de que este soporta un clear síncrono en lugar de un control arriba/abajo. La función clear es substituida por la señal cascada-in en el modo contador arriba/abajo. Se usan dos LUTs de tres entradas, una genera los datos del contador y la otra generan el bit de acarreo. La carga síncrona se implementa mediante un multiplexor de 2 a 1. La salida de este multiplexor es operada en forma AND con una señal de clear síncrona. Esto se muestra en la Fig. 1.24.

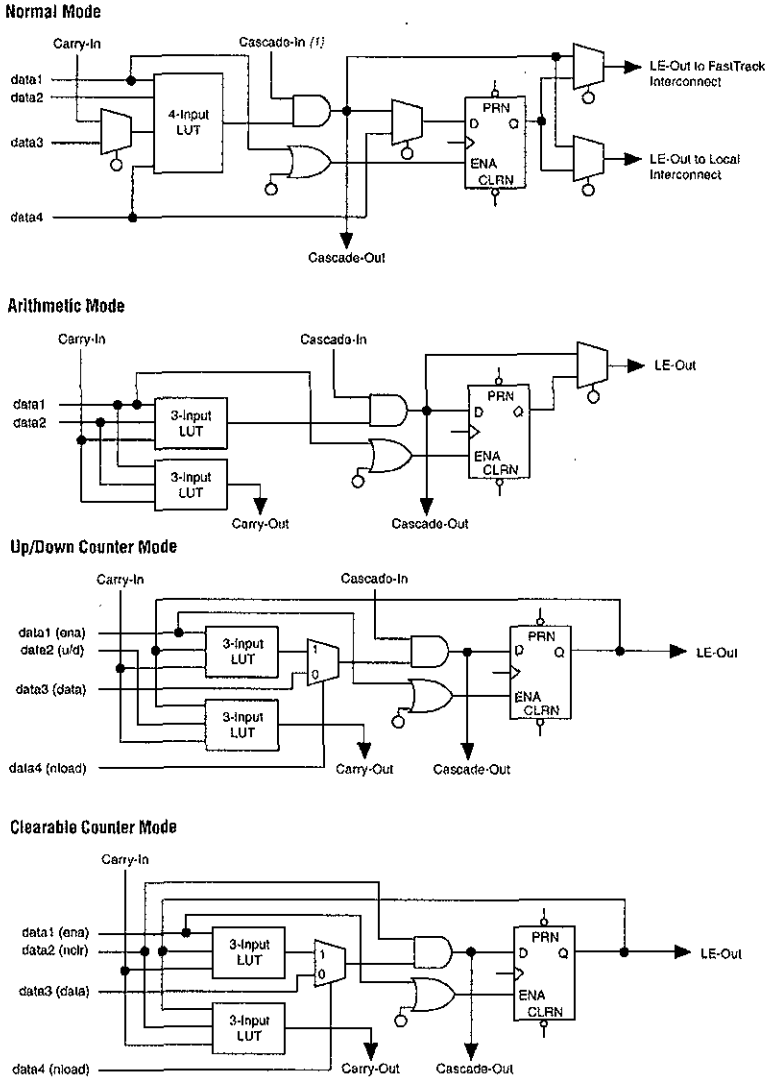


Fig. 1.24. Modos de operación de los Les. Cortesía Altera.

Interconexión FastTrack.

En la arquitectura de la FLEX 10K, las conexiones entre los LEs y los pines de I/O del dispositivo son proporcionadas por la Interconexión FastTrack, la cual es una serie de canales de ruteo horizontales y verticales continuos que cruzan el dispositivo. Esta estructura de ruteo global provee un desempeño predecible incluso en diseños complejos. Cada renglón de los LABs es comunicada

por una interconexión de renglón dedicada. La interconexión renglón puede manejar pines de I/O y alimentar a otros LABs en el dispositivo. La interconexión columna rutea señales entre renglones y puede manejar pines de I/O.

Cada columna de los LABs es comunicada por una interconexión de columna dedicada. La interconexión columna puede entonces manejar pines de I/O u otras interconexiones de renglón para rutear las señales a otros LABs en el dispositivo. Una señal de la interconexión columna, la cuál puede ser la salida de un LE o una entrada de un pin de I/O, debe ser ruteada a la interconexión renglón antes de que pueda entrar un LAB o EAB. Cada canal de renglón que es manejado por un IOE o un EAB puede manejar un canal columna específico. Toda esta estructura se ilustra en la Fig. 1.25.

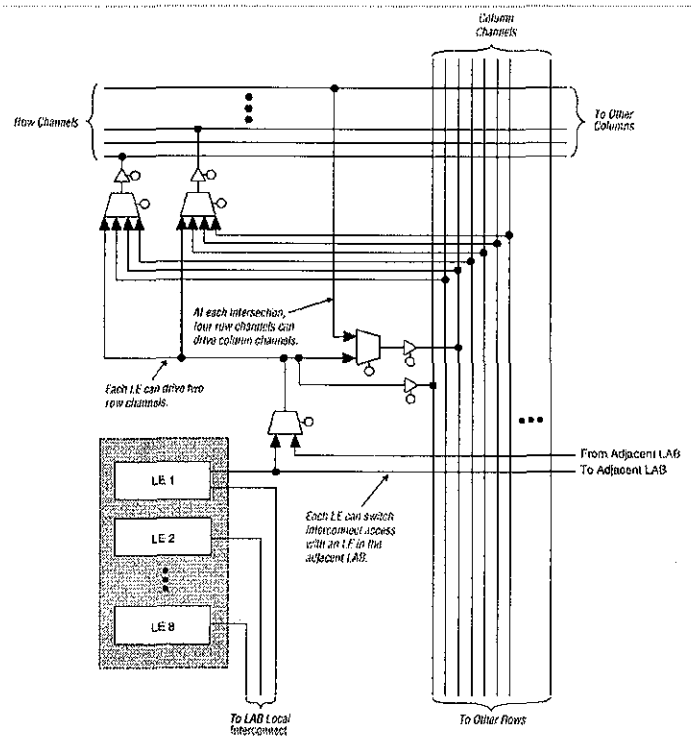


Fig. 1.25. Conección de los LABs hacia las interconexiones renglón y columna. Cortesía Altera.

Para mejorar el ruteo, la interconexión renglón está comprendida de una combinación de canales de longitud completa y de longitud media. Los canales de longitud completa conectan a todos los LABs en un renglón; los canales de longitud media conectan a los LABs en la mitad del renglón. El EAB puede ser manejado por los canales de longitud media en la mitad izquierda del renglón y por los canales de longitud completa. Adicionalmente para proveer una interconexión amplia de renglón que sea predecible, se provee de recursos de ruteo ampliados. Dos LABs vecinos pueden ser conectados usando un canal de medio renglón, salvando la otra mitad del canal para la otra mitad del renglón.

En adición a los pines de I/O de propósito general, los dispositivos FLEX 10K tienen seis pines de entrada dedicados que proveen una distribución de señal de bajo riesgo a través del dispositivo. Estas seis entradas pueden ser usadas para señales globales de control, de reloj, clear, preset, habilitador de salida y habilitador de reloj. Estas señales están disponibles como señales de control para todas las LABs y IOEs en el dispositivo. Las entradas dedicadas pueden también ser usadas como entradas de datos de propósito general por que pueden alimentar a la interconexión

local de cada LAB en el dispositivo. Sin embargo, el uso de entradas dedicadas como entradas de datos pueden introducir retrasos adicionales dentro de la red de señales de control. La Fig. 1.26 muestra la interconexión de LABs adyacentes y EABs con interconexiones renglón, columna e interconexión local, así como las cadenas en cascada y de acarreo asociadas.

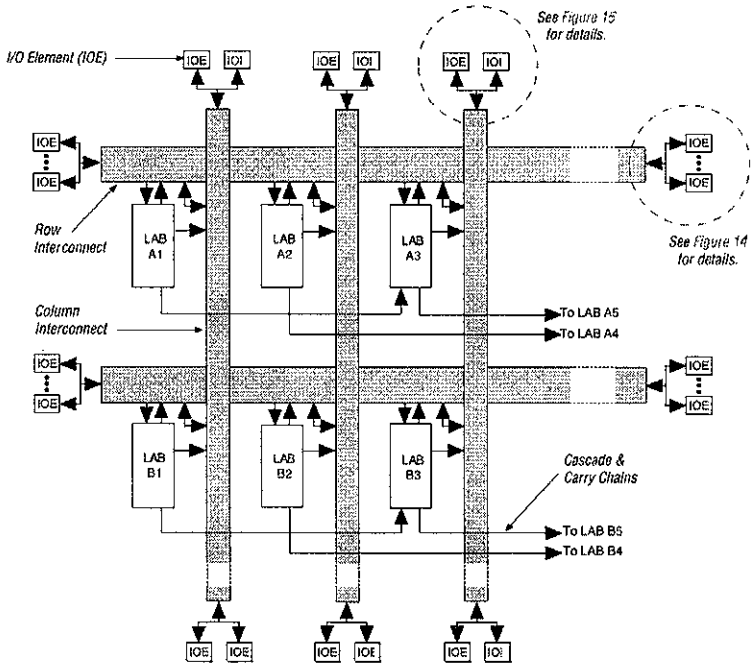


Fig. 1.26. Recursos de interconexión. Cortesía Altera.

Elemento I/O

Un elemento de I/O (IOE) contiene un buffer bidireccional de I/O y un registro que puede ser usado como una entrada registro para datos externos que requiere un rápido tiempo de carga, o como una salida del registro para datos que requiere alta velocidad. Los IOEs pueden ser usados como pines de entradas, salidas o bidireccionales. La Fig. 1.27 muestra el diagrama de bloques del IOE.

Cada IOE selecciona los habilitadores del reloj, clear, clock, y los controles de habilitación de salida de una red de señales de control de I/O llamadas bus de control periférico. El bus de control periférico usa caminos de alta velocidad para minimizar el retraso de la señal a través del dispositivo; este proporciona más de 12 señales de control que pueden ser asignadas como sigue:

- Hasta 8 señales de habilitación de salida.
- Hasta 6 señales de habilitación de reloj.
- Hasta 2 señales de reloj.
- Hasta 2 señales clear.

Conexiones renglón hacia IOE.

Cuando un IOE es usado como una señal de entrada, este puede manejar dos canales renglón separados. La señal es accesible para todos los LEs dentro de ese renglón. Cuando un IOE es usado como una salida, la señal es manejada por un multiplexor que selecciona una señal de los canales renglón. Hasta ocho IOEs conectados a cada lado de cada canal renglón son posibles. Esto es ilustrado en la Fig. 1.28.

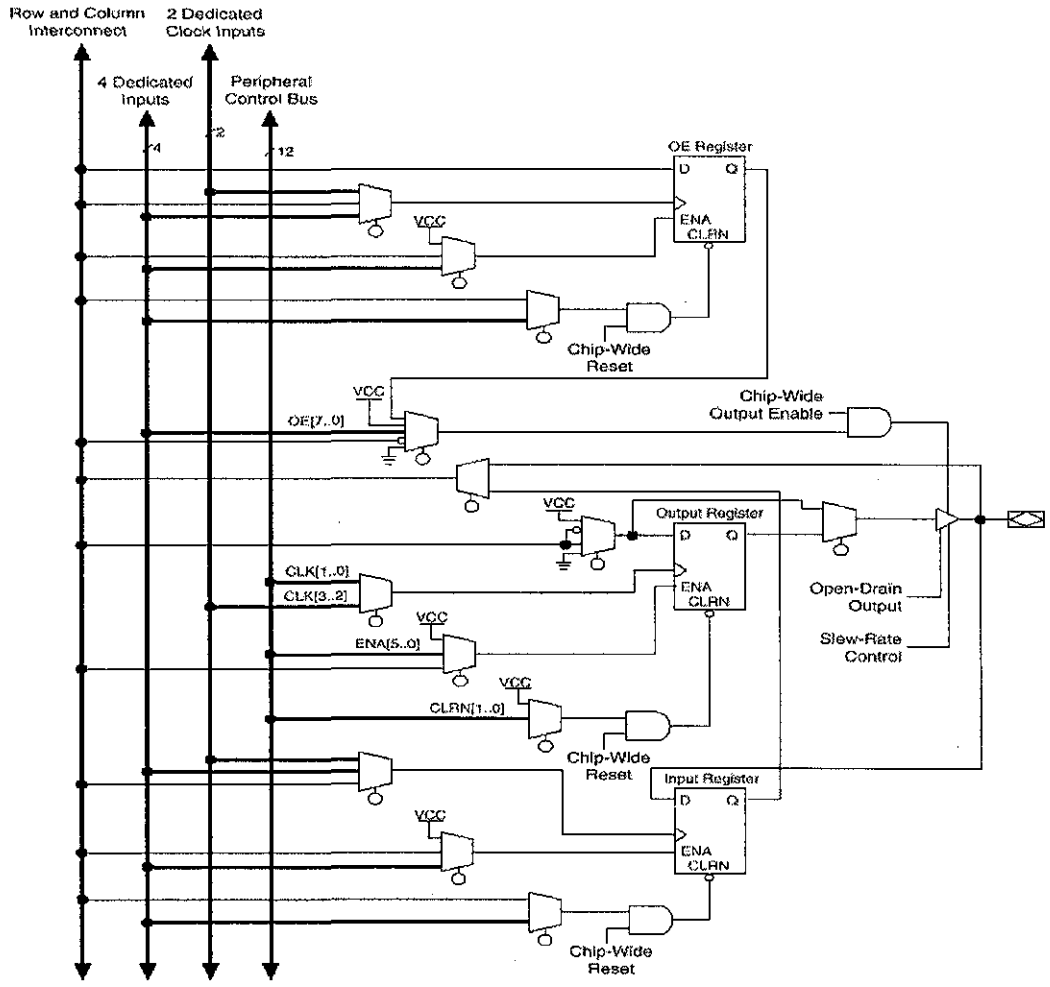


Fig. 1.27. Registros bidireccionales de I/O. Cortesía Altera.

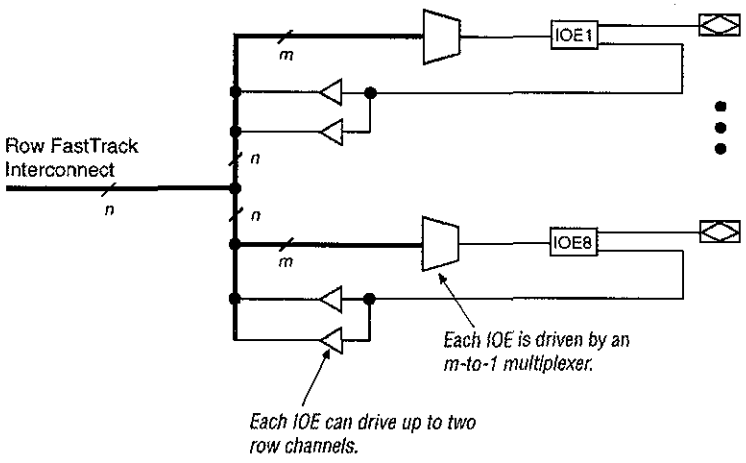


Fig. 1.28. Conección renglon hacia IOE. Cortesía Altera.

Conexiones Columna hacia IOE.

Cuando un IOE es usado como una entrada, este puede manejar más de dos canales columna separados. Cuando un IOE es usado como una salida, la señal es manejada por un multiplexor que selecciona una señal de los canales columna. Cada IOE puede ser manejado por canales columna vía un multiplexor. El conjunto de canales columna que cada IOE puede acceder es diferente para cada IOE. Esto se ilustra en la Fig. 1.29.

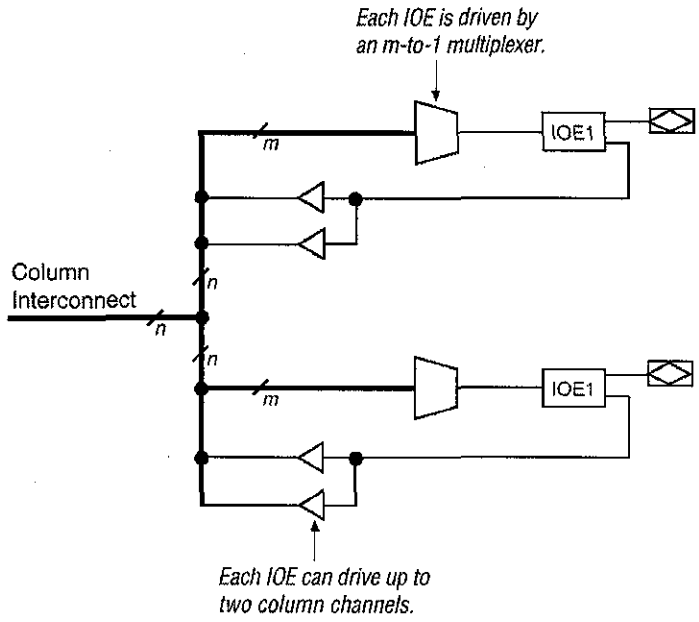


Fig. 1.29. Conección columna hacia IOE. Cortesía Altera.

MODELO DE TIEMPO.

Los continuos recursos de interconexión FastTrack de alto rendimiento aseguran un funcionamiento predecible y una simulación y análisis de tiempo precisos. El funcionamiento del dispositivo puede ser estimado por la ruta de la señal del origen a través de la interconexión hasta su destino. El funcionamiento registrado entre dos LEs en el mismo renglón puede ser calculado mediante la adición de los siguientes parámetros:

Tiempo de retardo del reloj a la salida del registro del LE (t_{CO}).
Tiempo de retardo de interconexión ($t_{SAMEROW}$).
Tiempo de retardo de la look-up Table del LE (t_{LUT}).
Tiempo de carga del registro del LE(t_{SU}).

El tiempo de retardo de ruteo depende de la colocación de los LEs origen y destino. Una ruta registrada más compleja puede incluir múltiples LEs combinacionales entre los LEs origen y destino. La Fig. 1.30 muestra el modelo de tiempo total, en el cuál se mapean las posibles rutas dentro de los varios elementos del dispositivo FLEX 10K.

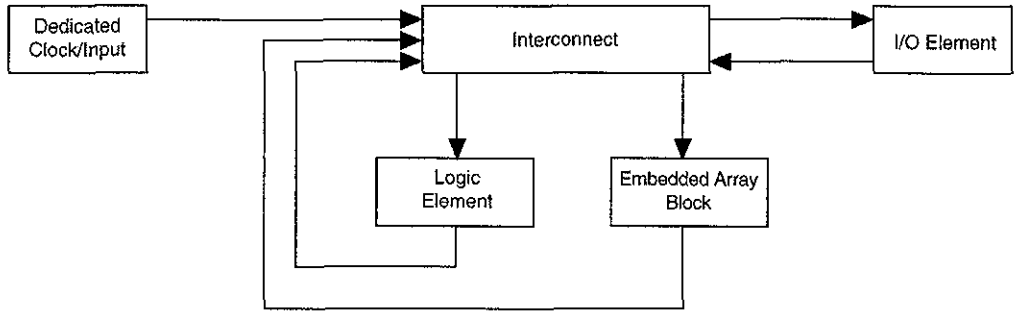


Fig. 1.30. Modelo de tiempo global del dispositivo FLEX10K. Cortesía Altera.

CAPÍTULO 2

METODOLOGÍA PROPUESTA

CONTENIDO DEL CAPÍTULO

2.1 Introducción
2.2 Metodología de las herramientas CAD-EDA
2.3 Metodología propuesta
2.3.1 Especificación y Análisis del instrumento
2.3.2 Modelado del instrumento
2.3.2.1 Modelado del NÚCLEO por Comportamiento
2.3.2.2 Modelado de la interfaz de entrada por Comportamiento
2.3.2.3 Modelado de la interfaz de salida por Comportamiento
2.3.3 Elección del dispositivo CPLD
2.3.4 Compilación y síntesis
2.3.5 Simulación Digital
2.3.6 Programación del dispositivo
2.3.7 Pruebas y Depurado

2.1 INTRODUCCIÓN

Las herramientas de software disponibles en el mercado para diseño utilizando dispositivos lógicos programables presentan una metodología típica y generalizada para diseñar cualquier sistema digital. Por supuesto no abordan el tema de especificación y análisis del problema por el enfoque tan genérico que utilizan.

La mayoría de estas herramientas soportan diseños descritos utilizando diferentes estrategias de modelado, pero nunca indican cual es la más o la menos apropiada, ya que esto da libertad al diseñador de utilizar la que mejor domine o incluso un híbrido de todas ellas.

Algunas otras herramientas que utilizan algún lenguaje de descripción de hardware para modelar el sistema digital, presentan diferentes esquemas de modelado, así como algunas ventajas y desventajas comparativas; sin embargo no aportan una guía que facilite la elección de la técnica más adecuada.

Una mala elección del método de modelado trae consigo un incremento considerable en el tiempo y esfuerzo requerido para completar el desarrollo del sistema, por tanto esta elección es muy importante en cualquier diseño.

Después de tener el sistema modelado, el resto del proceso involucra acciones como compilación, síntesis, simulación y programación del dispositivo que son bastante semejante en todas las herramientas y no representan problemas de elección ya que estas etapas son un proceso algorítmico con pocos grados de libertad y que no influye de manera dramática en el tiempo requerido que ocupa el proceso total.

La implantación del producto final en circuito impreso utilizando PCB es otra parte de proceso que no es abordada en las herramientas de software para el diseño con dispositivos lógicos programables; debido a que consideran que la implantación final ya es parte de otra metodología. Esto trae la desventaja de que estas herramientas dejan el diseño del sistema solo en la etapa de simulación, con la programación física del dispositivo o en una etapa de primer prototipo.

La generalidad de estas metodologías se debe a que están pensadas para atacar el diseño de cualquier sistema digital. Acotando adecuadamente el tipo de sistema digital a diseñar es posible moldear estas herramientas generando una metodología con una forma más adecuada para obtener los mejores rendimientos en términos económicos y en tiempo de diseño. Pensando de esta manera existe una metodología adecuada para cada tipo de sistema digital que se diseña.

En este trabajo presentamos una metodología útil para el diseño de instrumentos digitales de medición. Los objetivos esenciales de esta metodología son:

- Reducir el tiempo y esfuerzo desde la especificación del instrumento hasta su funcionamiento como producto final aprovechando la técnicas de modelado por comportamiento.

- Mostrar bloques funcionales genéricos que son comunes entre instrumentos, como pueden ser los módulos de interfaces de entrada y/o salida.

2.2 METODOLOGÍA DE LAS HERRAMIENTAS CAD-EDA

Podemos entender que una metodología en el ámbito del diseño de sistemas digitales es una secuencia de actividades bien definidas para pasar de una fase a otra en el diseño de algún sistema y cuyo objetivo es hacer metódico el desarrollo; ganando de esta manera tiempo y esfuerzo. Podemos entender la metodología también como una guía de desarrollo con un buen grado de abstracción. En nuestro ámbito una metodología de diseño está constituida por la siguiente terna:

Metodología de diseño = Conjunto de herramientas + Restricciones + Flujo de Diseño.

Conjunto de herramientas. Son las aplicaciones ofrecidas por el software de diseño utilizado, que van desde aplicaciones de captura del modelo pasando por simuladores y hasta herramientas de programación.

Restricciones. Son limitaciones impuestas tanto por la tecnología utilizada (CPLDs, FPGAs) como por las especificaciones del sistema a diseñar.

Flujo de diseño. Es el algoritmo que constituye la espina dorsal de la metodología, queda determinado en parte por la configuración del software de diseño utilizado, y esencialmente por la orientación dada por el diseñador con fundamento en su experiencia así como por las restricciones del sistema que estamos implementando.

Actualmente en el mercado existe varias herramientas CAD (Diseño Asistido por Computadora) para el Diseño Electrónico Automático (EDA); estas herramientas contienen un ambiente integrado que permiten el desarrollo de aproximadamente el 75 % del diseño frente a una computadora. Cada una de estas herramientas inherentemente tiene un Flujo de Diseño asociado y por tanto presenta una metodología de diseño bastante genérica. En la Fig. 2.1 ilustramos los elementos que involucra el uso de una herramienta CAD-EDA. Como se observa una herramienta de este tipo consta de un software que se ejecuta en una computadora personal y asiste en el diseño, modelado, simulación y depuración de la solución hasta llegar al prototipo final.

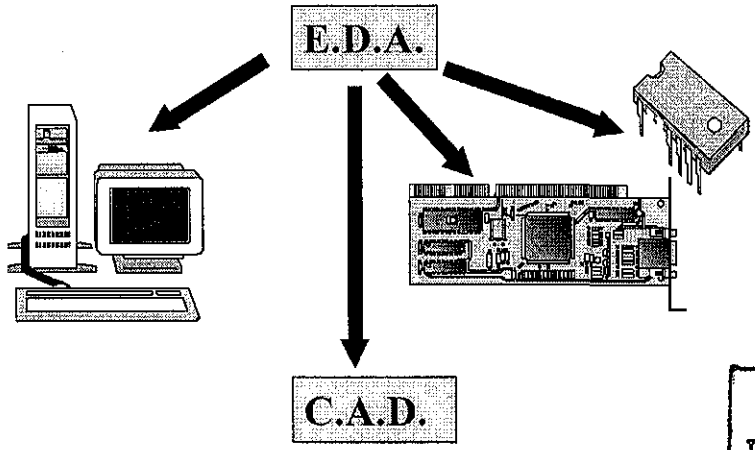


Fig. 2.1. Diseño electrónico automático

Cada una de estas herramientas EDA tienen asociado un flujo de diseño que es completamente dependiente de su construcción interna y que pretende ser lo más genérico posible. Un flujo típico se ilustra en la Fig. 2.2.

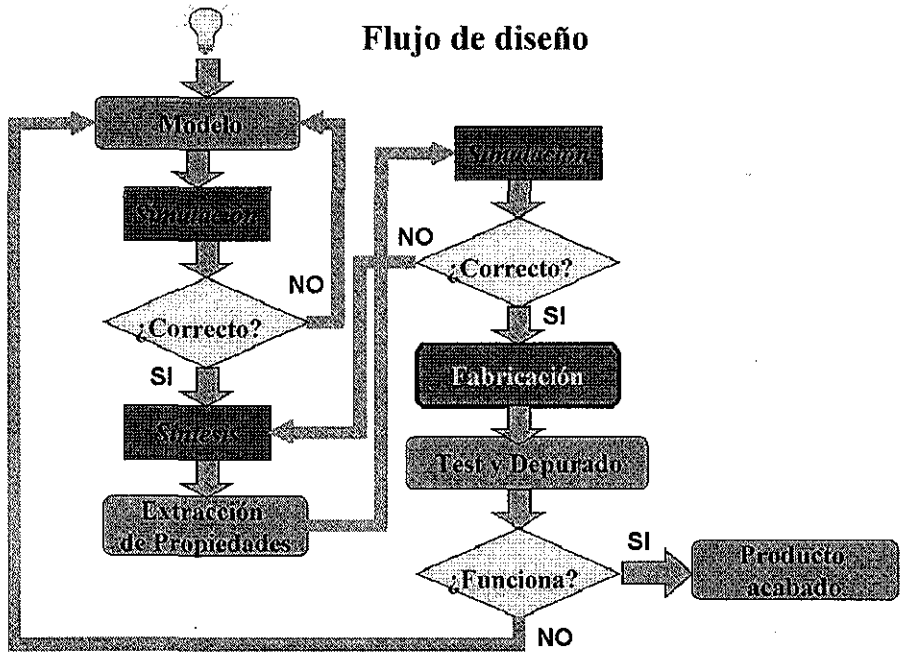


Fig. 2.2. Flujo de diseño típico de una herramienta EDA.

Pasos del flujo de diseño típico de una herramienta EDA.

Paso 1. Enunciar el problema con especificaciones completas, a partir de lo cual se procede a generar el modelo de la solución ideada por el diseñador. La forma de generar este modelo es algo no especificado explícitamente, solo se indican las alternativas de modelado por ejemplo *estructural*, por *comportamiento* o por *flujo de datos*. Una metodología proporcionada para simplificar el proceso de modelado en problemas complejos es la estrategia de diseño *arriba-abajo* ilustrada en la Fig. 2.3. Esencialmente consiste en dividir el problema en una jerarquía descendente de submódulos hasta el punto necesario en el cual cada uno de los módulos sea de complejidad manejable. Trae la ventaja adicional de que cada modulo puede ser atacado por personas diferentes.

Paso 2. Hacer la simulación funcional del modelo, esto significa comprobar que idealmente el modelo funciona. Tiene la desventaja de no considerar los efectos de propagación de cualquier electrónica digital real. Si el resultado de este proceso de simulación es diferente a los resultados esperados entonces se regresa a modificar el modelo. Si los resultados de la simulación son los esperados entonces se procede al siguiente paso.

Paso 3. Sintetizar el modelo obtenido, esto significa construir el modelo con los elementos básicos de la tecnología a utilizar (CPLD, FPGA ó ASIC), este proceso depende completamente del dispositivo físico.

Paso 4. Consiste en extraer las propiedades del modelo sintetizado para poder hacer una simulación digital del mismo.

Paso 5. Simulación del modelo sintetizado tomando ahora en cuenta efectos inherentes a cualquier circuito digital como son los tiempos de propagación. Si el resultado de este proceso de simulación no es correcto se regresa al proceso de síntesis para buscar alternativas y resolver los problemas

de tiempos, este proceso es común en dispositivos con un modelo de tiempo dependiente del diseño como en los FPGAs. Si el resultado de la simulación digital es el esperado se procede al siguiente paso.

Paso 6. Fabricación o programación del dispositivo.

Paso 7. Probar y depurar el circuito funcionando, si da los resultados esperados se ha terminado el producto, de lo contrario se procede a la revisión del modelo por que significa que los simuladores no fueron precisos al reproducir el efecto real del modelo en la física del dispositivo.

El flujo de diseño típico de una herramienta CAD-EDA descrito anteriormente optimiza el proceso de diseño de un sistema digital, ya que la mayor parte del trabajo es frente a la computadora y no es necesario obtener el prototipo final sino hasta que la simulación trabaja correctamente. Sin utilizar estas herramientas el ciclo se hace más largo e involucra en cada iteración del ciclo la prueba del prototipo, lo que hace más lento el proceso de diseño. Este proceso se ilustra en la Fig. 2.4.

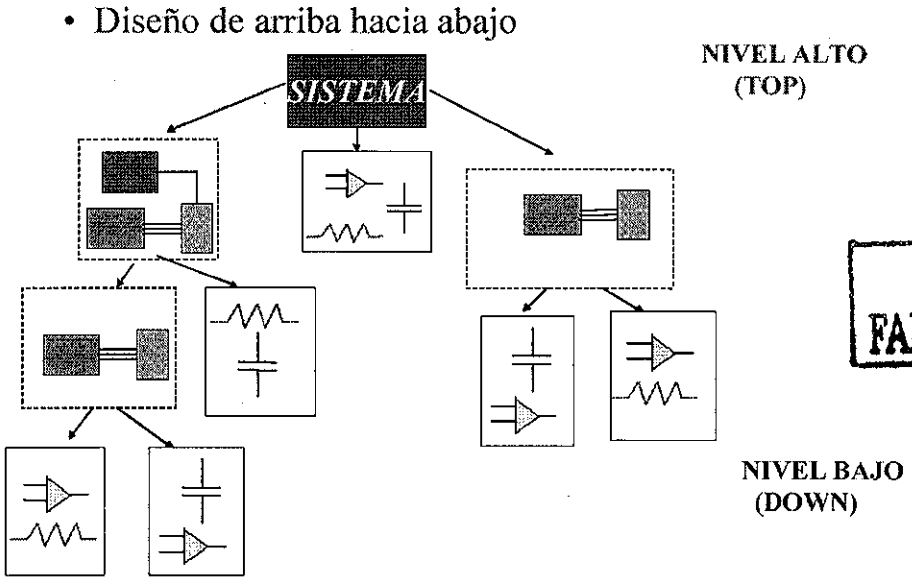


Fig. 2.3 Diseño de arriba - abajo

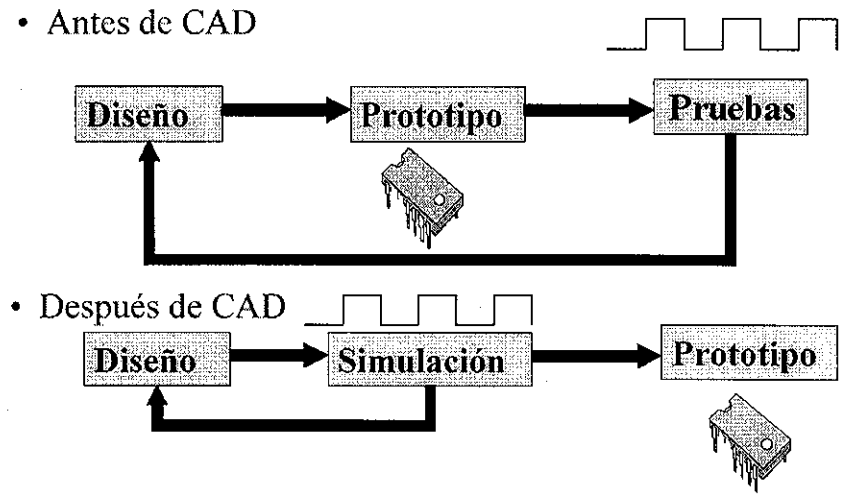


Fig. 2.4. Proceso de diseño sin herramientas CAD y con ellas.

2.3 METODOLOGÍA PROPUESTA

La metodología propuesta pretende ser útil para el diseño e implantación de instrumentos digitales de medición, plasmando la experiencia adquirida en este tipo de diseños. Nos hemos enfocado a la utilización de una herramienta CAD que brinda un ambiente de desarrollo completo de bajo costo sobre CPLDs. Esta herramienta es manufacturada junto con los dispositivos programables por la empresa ALTERA. La metodología también pretende aprovechar al máximo la utilización de los lenguajes de descripción de hardware estandarizados en la actualidad, para el proceso de modelado.

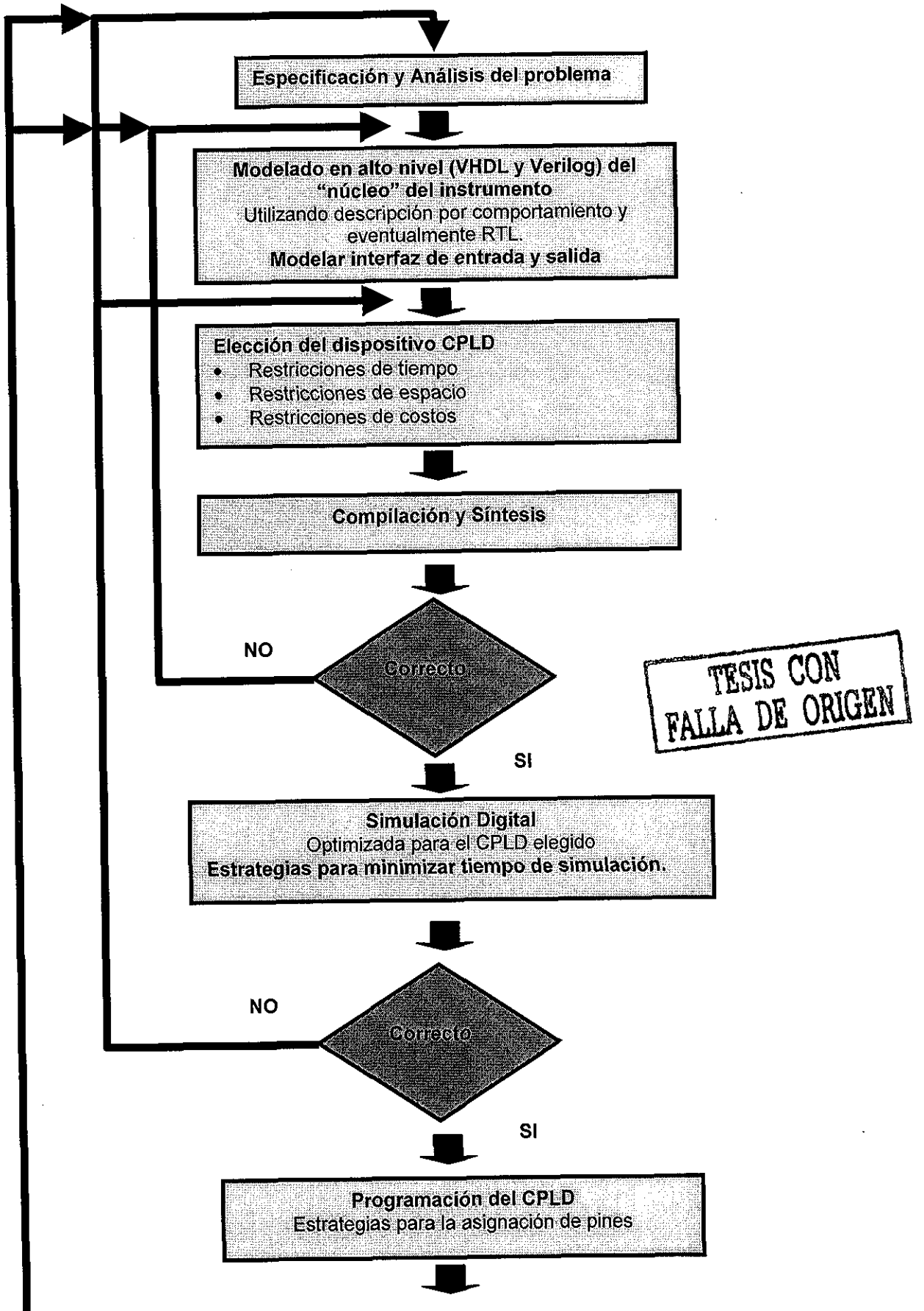
Hemos mencionado que una metodología de diseño está conformada por la terna citada en la sección 2.2. Esta terna aplicada a nuestra metodología propuesta se constituye por los siguientes elementos:

Conjunto de Herramientas. Max+Plus II herramienta CAD-EDA de la empresa Altera constituida por nueve aplicaciones: Display de Jerarquía, Editor Gráfico para captura del modelo, Editor de Símbolos, Editor de Texto para captura del modelo, Editor de Forma de onda, Editor de Planeación de *pines*, *Compilador – Sintetizador*, *Simulador*, *Analizador de Tiempos*, *Programador*.

Restricciones. En primer lugar constituidas por los dispositivos CPLDs utilizados en relación a espacio, velocidad, precio. En segundo lugar las propias restricciones y especificaciones del instrumento de medición a diseñar.

Flujo de diseño. Fundamentado en la optimización de tiempo, esfuerzo y costo del diseño e implantación. Este flujo de diseño presenta mucha de la experiencia adquirida en el diseño de instrumentos digitales de medición y constituye la parte medular de la metodología propuesta.

El Flujo de Diseño de la metodología propuesta es ilustrado en la Fig. 2.5 y detallado en secciones posteriores. Partimos de la especificación del problema procediendo a un análisis. Obtenemos el modelo de la solución utilizando descripción por comportamiento y eventualmente cuando se requiera descripción por flujo de datos, es decir en este punto modelamos el "núcleo" del instrumento; en forma separada también modelamos los módulos de interfaz de entrada y salida. Elegimos el dispositivo CPLD a utilizar considerando restricciones de tiempo, espacio y costos. Ejecutamos el proceso de compilación y síntesis de la herramienta Max+Plus II. Si el proceso de compilación genera errores retornamos al proceso de modelado. Continuamos con el proceso de simulación digital empleando ciertas estrategias para minimizar los tiempos de compilación. Si el resultado del proceso de simulación no es el esperado podemos saltar a elegir un CPLD con características de tiempo diferentes, cambiar el modelo ó incluso modificar las especificaciones de diseño y por tanto el análisis. Continuamos con el proceso de programación del CPLD, utilizando ciertas estrategias para la asignación de pines. Realizamos pruebas y depurado sobre el dispositivo CPLD programado, ya sea en un protoboard o en PCB; si el funcionamiento es correcto hemos finalizado el diseño e implantación, sino entonces podemos regresar a la etapa de modelado o hasta la especificación del problema y análisis del mismo.



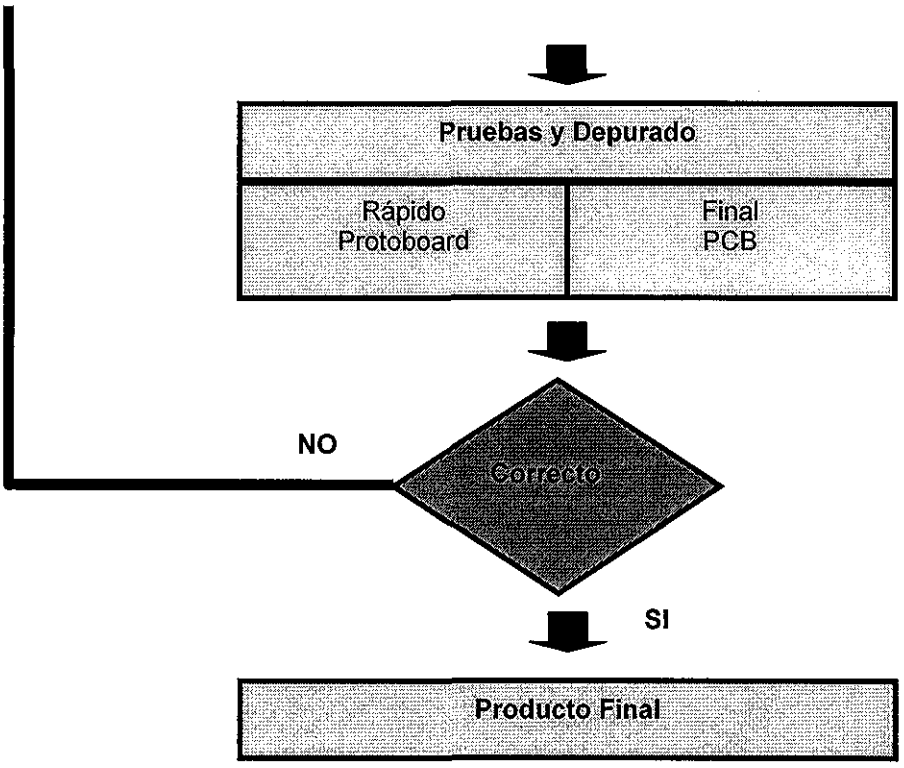


Fig. 2.5. Flujo de diseño propuesto en nuestra metodología.

2.3.1 Especificación y análisis del instrumento

Esta etapa de la metodología juega un papel muy importante, dado que estamos enfocándonos a la creación de instrumentos digitales de medición, entonces debemos contar con un conjunto de especificaciones reales al momento de ser solicitada su creación.

Es decir nuestra metodología parte de las especificaciones del instrumento a construir y por tanto antes de comenzar con cualquier etapa de modelado tenemos que evaluar la factibilidad de las especificaciones solicitadas, tomando en cuenta que vamos a implementar el "núcleo" del instrumento en dispositivos lógicos programables.

ESPECIFICACIÓN DEL INSTRUMENTO.

Los aspectos a tomar en cuenta al especificar el instrumento o evaluar las especificaciones dadas por un tercero son:

- **Velocidad de operación del instrumento (Ancho de Banda).**

Esta característica está en función de la variable ó variables físicas que se desean medir, podemos estar hablando de variables muy rápidas como en procesos electrónicos, o de variables muy lentas como en procesos térmicos, biológicos etc.

Este aspecto es de suma importancia debido a que todos los dispositivos digitales reales tienen intervalos de operación en relación a la frecuencia de las señales que manejan. A partir de este dato especificado para el instrumento es necesario consultar las hojas de datos técnicas para cada

una de las familias de dispositivos lógicos programables disponibles en el mercado. Por ejemplo la Familia MAX7000S de Altera puede operar a un límite máximo de 100 Mhz.

Es claro que esta característica varia de manera proporcional al costo del dispositivo, por tanto es muy importante quizá solicitar una re evaluación de las especificaciones del instrumento con el objetivo de evitar requerimientos que incrementan exponencialmente el costo del instrumento.

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

Es claro que los instrumentos reales de medición no son 100 % digitales, esto es el "núcleo" del instrumento puede ser 100 % digital, esto es el procesamiento de la información se realiza sobre señales digitalizadas. Sin embargo como el objetivo del instrumento de medición es medir variables reales, estas casi siempre consisten de señales analógicas, de esta manera es necesario evaluar la resolución de salida requerida por el instrumento, para así poder determinar la resolución requerida en las variables de entrada. En otras palabras esto nos determina el tipo, resolución y velocidad de los convertidores A/D requeridos para que el "núcleo" del instrumento a diseñar procese esa información.

De forma directamente proporcional al numero de variables de entrada y de la resolución requerida por cada una de ellas es el tamaño requerido por el dispositivo lógico en relación al número de pines de entrada y salida. Esto significa que altas resoluciones requerirán del uso de dispositivos de mayor tamaño. Por ejemplo el dispositivo más grande medido en número de pines de entrada / salida para la Familia MAX7000S de Altera es de 164 pines.

Esta características determinan el grado de adecuación de las señales de entrada para que el "núcleo" del instrumento pueda manipularlas. Lo que acabamos de decir es que en realidad un instrumento es un sistema *híbrido* que muy seguramente va a requerir de un módulo adicional analógico para el adecuamiento de las señales. Este módulo analógico no es tratado en la presente metodología. Sin embargo debe ser tomado en cuenta al diseñar, porque podríamos cometer el error de diseñar un "núcleo" que requiera de un sistema de adecuación de las señales muy complicado, o lo que es peor diseñar un "núcleo" muy sofisticado pero ideal porque no se le pueden proporcionar las variables de entrada de la manera que las requiere.

- **Precisión, exactitud y tolerancia del instrumento.**

En las especificaciones del instrumento también se debe hacer referencia a dos características básicas y conocidas en la instrumentación.

Debe tener especificada la precisión del instrumento que tiene que ver con la repetibilidad de la medición, un instrumento es muy preciso si sus lecturas son repetibles (poco cambiantes una de otra). Debe especificar la exactitud, que tiene que ver con la cercanía que proporciona el instrumento en su medición con respecto al valor real.

Las anteriores son características que se especifican en cualquier instrumento y por tanto imponen restricciones a todo el proceso de creación del instrumento. De acuerdo a las características de los dispositivos lógicos a utilizar es necesario evaluar si las especificaciones solicitadas son factibles de cumplirse con la herramienta a utilizar.

La mayoría de las veces el cumplimiento de las especificaciones mencionadas no depende totalmente de la tecnología a utilizar, sino más bien del diseño y modelo de solución. Por ejemplo esto nos puede establecer la necesidad de un diseño 100 % síncrono, asíncrono o el empleo de algoritmos más elaborados para el procesamiento de la información ó incluso de algoritmos especializados para corregir los problemas de precisión, exactitud y tolerancia requeridos.

ANÁLISIS DEL INSTRUMENTO

La etapa de análisis tiene la responsabilidad de separar el "núcleo" del instrumento del resto de la arquitectura con el objeto de trabajar la parte de la medición que caracteriza al instrumento en cuestión, obviando aspectos que pueden ser re utilizables por ser comunes entre instrumentos. Esencialmente consiste en :

- ***Tener claro el instrumento que queremos construir.*** Esto significa por un lado conocer y entender claramente todas las especificaciones funcionales del instrumento, y en segundo lugar dominar el principio y la teoría básica de la categoría del instrumento que estamos diseñando.

  En caso de estar diseñando un instrumento poco común o muy especializado en la medición que realiza, entonces tenemos que crear la teoría básica en que se fundamenta su operación para poder tener la capacidad de materializar esa teoría en un instrumento real.

  La característica anterior, generalmente plantea la necesidad de un trabajo en equipo, por una parte de todas aquellas personas involucradas con la conceptualización y creación de la teoría de operación del instrumento y por otra parte del equipo de ingenieros encargados de materializar la conceptualización.

- ***Definición del "núcleo" del instrumento.*** El "núcleo" del instrumento representa el corazón que será programado en el dispositivo lógico programable y que constituye la parte medular del instrumento digital. Es decir el "núcleo" es el modelo o algoritmo que va a procesar la información recibida para obtener la medición que realizará el instrumento y mostrar los resultados. El "núcleo" puede estar constituido por un algoritmo especificado en alto nivel y que finalmente debe ser implantado en hardware.

  Esta parte del análisis debe permitir determinar con toda claridad el "núcleo" del instrumento, y aislarlo de todo el resto que aunque forme parte del mismo y además sea digital no representa la esencia del instrumento. Es decir muchos instrumentos con funciones de medición muy diferentes pueden compartir cosas en común, como las interfaces de entrada y salida, por ejemplo.

- ***Definición de la interfaz de entrada.*** De todos los instrumentos de medición podemos extraer un denominador común conocido como interfaz de entrada a través de la cual la información de la variable o variables a medir es proporcionada al "núcleo" de la aplicación.

  Aspectos a considerar es el número de variables de entrada y la forma en que son sensadas y capturadas desde el mundo real. Es importante analizar esta etapa porque podemos tener muchas variables de entrada con alta resolución de manera tal que se requiera compartir recurso para minimizar costos, como por ejemplo compartir el convertidor A/D y minimizar el número de entradas al "núcleo" del instrumento mediante la multiplexión de las variables de entrada.

  La toma de estas decisiones afectan directamente el diseño del "núcleo" así como las especificaciones del instrumento. Por ejemplo una excesiva multiplexión de las variables de entrada ocasiona mayor tiempo de proceso y por tanto la posibilidad de no cumplir con ciertas especificaciones.

- ***Definición de la interfaz de salida.*** Otro denominador común en los instrumentos de medición digitales es el conocido como interfaz de salida, a través del cual la información y los resultados procesados por el "núcleo" son mostrados al usuario.

  Aspectos a considerar son el número de variables de salida y la forma en que estás son entregadas por el "núcleo" de la aplicación al mundo real. La importancia del análisis de esta

etapa radica en que el "núcleo" puede entregar muchas variables de salida de alta resolución (digitales por su puesto), de manera que es posible requerir la multiplexión de variables de salida para minimizar el numero de pines requeridos así como para compartir el tipo de decodificador utilizado para el despliegue. Pocas veces requerimos un convertidor D/A a la salida dado que estamos trabajando con instrumentos.

La toma de decisiones en esta interfaz de salida afectan directamente el diseño del "núcleo" y las especificaciones del instrumento, porque de la misma manera el abuso en la multiplexión de las variables de salida implica mayor tiempo de proceso y la posibilidad de no cumplir con las especificaciones iniciales del instrumento.

La especificación y análisis del instrumento es una etapa de la metodología a la cual se puede regresar varias veces desde algún otro punto, esto con el fin de re-especificar y re-analizar el instrumento en caso de ser necesario. Sin embargo el proceso de regresar a re-especificar es indicador de un mal análisis y malas especificaciones iniciales. Por tal motivo es conveniente dedicar el suficiente tiempo a esta etapa para determinar con exactitud lo que queremos hacer.

2.3.2 Modelado del Instrumento

Esta etapa pretende modelar tres partes fundamentales de cualquier instrumento digital:

- ***Núcleo del instrumento.*** Constituye el corazón de la aplicación, hablamos de un núcleo 100% digital, que será modelado para ser implementado en dispositivos lógicos programables. Es este módulo el que contiene el procesamiento digital o algoritmo que define el funcionamiento del instrumento. En esencia sus características debieron ser especificadas en la etapa de especificación y análisis del instrumento. Si el instrumento requiere de componentes analógicos para el preprocesamiento de entradas al *núcleo* estas deben ser implementadas de forma externa y no forman parte de la metodología propuesta.

- ***Interfaz de entrada.*** El instrumento va a medir variables de entrada de manera que la interfaz de entrada es la encargada de aceptar las variables de entrada y proporcionarlas en la forma que el "núcleo" las requiera. Esta etapa puede ser 100 % digital si es el caso de que las entradas al instrumento ya sean digitales en cuyo caso se implementará como un módulo completo dentro del dispositivo lógico programable. En otros casos la interfaz es híbrida siendo necesario un modulo analógico para la adecuación de la variable de entrada procedente del mundo real. En este caso nuestra metodología solo aborda la parte digital de la interfaz de forma genérica para cualquier instrumento, en caso de requerir un preprocesamiento este puede ser considerado como un módulo adicional ajeno tanto al "núcleo" del instrumento como de su interfaz y deberá ser diseñado separadamente.

- ***Interfaz de salida.*** Los resultados de las mediciones realizadas por el "núcleo" tienen por objetivo ser visualizadas como resultado del proceso, más que ser utilizadas como variables para controlar, de manera que requerimos de una interfaz que muestre los resultados, que por lo general es 100% digital y puede ser construida totalmente en el dispositivo lógico programable.

Para hacer el modelado de estas tres etapas existen diversas técnicas; nuestra metodología propone el empleo de lenguajes de descripción de hardware estandarizados es especial VHDL y Verilog, utilizando un estilo de descripción por comportamiento y eventualmente de flujo de datos que explicaremos más adelante. Las razones para utilizar esta forma de modelado son las siguientes:

- La descripción por comportamiento es una descripción de alto nivel, nos ahorra tiempo.
- La descripción por comportamiento da claridad al diseño.
- VHDL y Verilog, son lenguajes estandarizados por IEEE y empleando un subconjunto de ellos garantizamos portabilidad entre las diversas plataformas de dispositivos lógicos programables existentes en el mercado.
- El modelo hasta cierto punto es independiente del hardware utilizado.
- VHDL y Verilog permiten describir sistemas complejos utilizando descripción por comportamiento de una manera sencilla.

Ahora pasaremos a explicar las estrategias de modelado que pueden ser empleadas en las tres etapas de diseño de cualquier instrumento y pondremos énfasis en la elegida para nuestra metodología. El modelado de un sistema digital lo podemos hacer en tres dominios diferentes.

DOMINIOS DE MODELADO

Tenemos tres dominios de modelado para cualquier sistema digital, conocidos como:

- *Dominio Estructural.*
- *Dominio por Comportamiento.*
- *Dominio Físico.*

El diseño utilizando cada uno de los dominios citados anteriormente tiene a su vez diferentes grados de abstracción. Los cuales van de un nivel alto a un nivel bajo, entendemos una descripción en alto nivel a una forma más abstracta de descripción y compatible con la forma de representación de los humanos y entendemos por un nivel bajo a una forma de descripción más a detalle y menos compatible con la forma de representación humana. La Fig. 2.6. ilustra los tres dominios de modelado, a la figura se le conoce como diagrama de Gajski. En la Fig. 2.7. se ilustra con más detalle los diversos grados de abstracción de cada dominio que explicaremos a continuación.

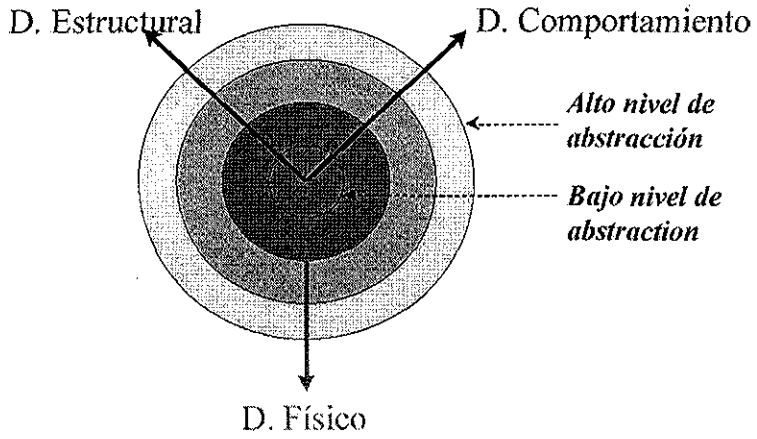


Fig. 2.6. Tres dominios de modelado. Estructural, Físico y por Comportamiento.

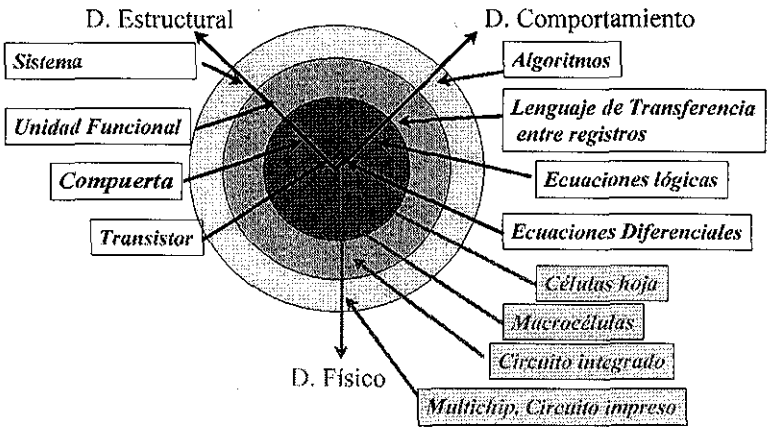


Fig. 2.7. Clasificación por grado de abstracción dentro de cada dominio.

DOMINIO ESTRUCTURAL

Este dominio del modelado consiste en la enumeración de los componentes del circuito así como sus interconexiones. Es decir estamos describiendo la arquitectura del circuito. La descripción puede ser en forma gráfica o mediante enumeración utilizando algún lenguaje de descripción de hardware.

Este tipo de descripción se puede hacer con diversos grados de abstracción, por lo general la descripción con mayor abstracción es más clara para el usuario y permite reducir tiempos de diseño, mientras que la descripción con menor grado de abstracción al ser más detallada puede resultar demasiado confusa para el usuario. Los diversos grados de abstracción que tenemos son los siguientes:

- **Transistor.** El diseño se describe utilizando el elemento estructural básico que es el transistor. Describir de esta manera es tedioso y no necesario para la mayoría de las aplicaciones.
- **Compuerta.** La descripción se realiza utilizando la unidad mínima lógica estructural que es la compuerta lógica. La descripción es menos tediosa que con transistores, sin embargo también es no necesaria en la mayoría de los diseños.
- **Unidad Funcional.** Se describe usando bloques que realizan funciones específicas, por ejemplo contadores, registros, multiplexores, decodificadores, etc. Esta es una forma común de mostrar la arquitectura de un diseño en el dominio estructural.
- **Sistema.** Se describen sistemas completos como un solo bloque, por ejemplo "núcleos" de microprocesadores, filtros digitales, etc. Esta forma se emplea en el diseño de sistemas complejos cuyos submódulos son también sistemas muy complejos.

En general en cualquiera de sus cuatro tipos de abstracción hablamos de describir la estructura, ya sea en forma gráfica o por enumeración de componentes y conexiones usando un HDL. Tiene la ventaja de presentar en forma visual una estructura jerárquica; sin embargo, en el diseño de un sistema complejo generar la arquitectura de esta manera implica el consumo de tiempo y esfuerzo considerables.

DOMINIO FÍSICO

En este dominio el modelado consiste en describir los elementos físicos constitutivos del sistema a partir de los diversos grados de abstracción que maneja la tecnología que estemos empleando; los grados de abstracción son los siguientes:

- **Células hoja.** Componentes físicos básicos e indivisibles a partir de los cuales es posible construir un sistema.
- **Macroceldas.** Componentes físicos que pueden implementar algún tipo de función digital, como por ejemplo alguna función lógica.
- **Circuito Integrado.** Componente físico que implementa algún tipo de diseño.
- **Multichip, Circuito Impreso.** Componentes físicos que pueden representar un sistema completo.

El modelado a este nivel es poco común y poco útil en virtud de que describimos únicamente componentes de la tecnología empleada.

DOMINIO POR COMPORTAMIENTO

En este dominio el modelado consiste en describir el comportamiento del circuito digital, utilizando algún lenguaje de descripción de hardware. Es muy importante destacar que no se describe estructura sino solo el comportamiento del sistema que se pretende modelar.

Esta descripción se puede hacer con distintos grados de abstracción, por lo general una descripción con mayor grado de abstracción, es mas clara para el usuario y describe mucho más cosas, permitiendo reducir notablemente tiempos de diseño. Una descripción con menos abstracción resulta más engorrosa porque describe el comportamiento de componentes cada vez más básicos. Los diferentes grados de abstracción de este dominio son:

- **Ecuaciones diferenciales.** Es un modelo matemático del comportamiento de la entidad modelada, por ejemplo de un *transistor.* Modelar a este nivel de detalle es muy engorroso y no necesario en la mayoría de las aplicaciones.
- **Ecuaciones Lógicas.** Es un modelo matemático utilizando álgebra booleana que describe el comportamiento de la entidad modelada. El grado de abstracción es suficiente para modelar con facilidad algunos sistemas digitales.
- **Lenguaje de Transferencia entre Registros.** Es una forma de modelar indicando el flujo de los datos en el diseño de un punto a otro mediante sentencias de alto nivel. Modelar de esta manera permite obviar detalles no necesarios de componentes digitales utilizados con mucha frecuencia en el diseño de sistemas digitales.
- **Algoritmos.** Es la forma más abstracta de modelar un sistema digital, porque estamos describiendo a través de un algoritmo como lo hacemos en los lenguajes de programación *como "C" o Pascal, el comportamiento y procedimiento realizado por el sistema digital que* pretendemos modelar. Este tipo de modelado permite optimizar tiempo de diseño y esfuerzo; sin embargo un mal algoritmos aunque fácil de generar puede producir hardware muy poco óptimo y la solución puede resultar costosa en espacio y por tanto en dinero.

Es cierto que se puede optimizar más utilizando un menor grado de abstracción tal como sucede con la generación de software en el cual es posible optimizar más usando lenguaje ensamblador que uno de alto nivel; sin embargo es más fácil programar en alto nivel.

De los tres dominios de modelado descritos anteriormente tiene un futuro prometedor el diseño por comportamiento, esto debido a que optimiza el tiempo de diseño de los sistemas digitales complejos, su objetivo es escribir el diseño como si estuviéramos programando software; de hecho el ideal es describir hardware utilizando lenguaje "C", por ejemplo.

2.3.2.1 Modelado del NÚCLEO por Comportamiento

La metodología propuesta plantea la explotación del modelado por comportamiento con el mayor grado de abstracción posible utilizando lenguajes de descripción de hardware VHDL o Verilog. La elección de este tipo de modelado radica en:

- El esfuerzo para diseños complejos disminuye notablemente.
- La claridad del diseño es mayor para el mantenimiento posterior.
- Explota las mejores características de los HDLs

Tiene una desventaja inherente y consiste en que este tipo de descripción puede perder de vista que estamos generando hardware y en consecuencia crear diseños de muy rápida creación pero de alto costo en espacio de semiconductor. Por tanto es necesario conocer los detalles finos de este dominio de diseño.

Presentaremos las estructuras básicas para el modelado por comportamiento disponibles en cualquier HDL y propuestas por nuestra metodología para modelar cualquier núcleo. Utilizamos pseudocódigo para estas estructuras sin particularizar en algún HDL específico, sin embargo todas ellas son posibles de ser descritas por los lenguajes VHDL o Verilog. Para detalles sobre implantaciones de estas estructuras usando los lenguajes citados recomendamos los manuales introductorios desarrollados por *Valeriano* et al. [1][2] como parte del desarrollo de la misma tesis y no incluidos como anexos por razones de espacio.

Es preciso aclarar que los HDLs también se pueden utilizar para describir modelado en el dominio estructural, que no es recomendado en esta metodología, por lo cual estamos enfocados en describir algoritmos única y exclusivamente.

CARACTERÍSTICAS DE LOS HDLs.

Ventajas

- Hacen frente a la creciente complejidad de los diseños, sobre todo al utilizar una descripción algorítmica por comportamiento.

- Existe una analogía directa con lo que podemos llamar lenguajes de descripción de *Software*, como se ilustra en la Fig. 2.8. Observamos que el equivalente de un HDL es comparable con un lenguaje de alto nivel cuando describimos software y ocupa el nivel de abstracción más elevado. En el nivel de abstracción menor por el lado del software hablamos de código máquina mientras que el equivalente en hardware es el nivel transistor. Es interesante observar que el nivel medio que corresponde, hablando del software, al uso del ensamblador por el lado de hardware es equivalente al uso de componentes básicos y sus interconexiones.

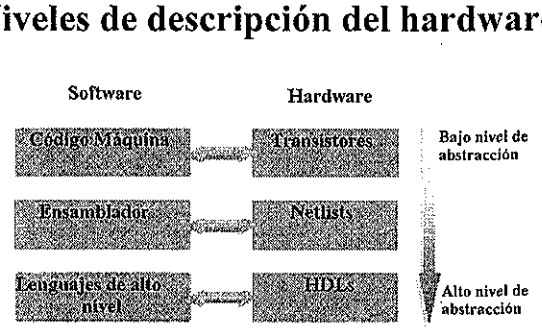


Fig. 2.8. Analogía entre Software y Hardware.

- Pueden describir el comportamiento de sistemas digitales. Esto significa que son capaces de no solo describir estructura, sino funcionalidad, logrando así la característica de alto nivel.
- Se utilizan en la fase de modelado.
- Requieren de simuladores para verificar el comportamiento descrito. El modelo descrito es factible de ser simulado sin tener el sistema físico armado.
- Son lo suficientemente descriptivos para que a partir de un modelo generado se pueda sintetizar el circuito a nivel de compuertas.
- El lenguaje es independiente de la tecnología. Esto significa que el diseño puede ser re utilizado por las herramientas de otros vendedores.
- La portabilidad es mayor, no hay tanta dependencia con relación al fabricante.
- Permiten descripciones por comportamiento, estructural y por flujo de datos, o la combinación de ellas.
- Sencillez de la descripción sobre todo a nivel de comportamiento. Por tanto se tiene ahorro de tiempo en los diseños.
- El HDL es un medio de comunicación entre diferentes herramientas CAD-EDA.
- Los lenguajes soportan jerarquías.
- Los lenguajes permiten generar marcos de prueba.
- Permite modelar sistemas síncronos y asíncronos.
- Es posible especificar directamente los algoritmos de las máquinas de estados.
- Son lenguajes bien conocidos. Estándares IEEE.

Desventajas

- Imponen un esfuerzo de aprendizaje dado que implica una nueva metodología.
- Se hace necesaria la adquisición de herramientas de simulación y síntesis.
- El diseño final no resulta tan optimizado como si lo realizara un humano a más bajo nivel.
- Se pierde control sobre el aspecto físico del diseño.
- No todos los constructores del lenguaje son soportados por las herramientas existentes en el mercado.
- Cada sintetizador comercial impone restricciones adicionales.

DESCRIPCIÓN POR COMPORTAMIENTO

Las descripciones por comportamiento son similares a un lenguaje de programación de software por su alto nivel de abstracción. No especifica la forma en que se deben conectar los elementos de un diseño sino solo describimos el comportamiento que tiene en conjunto.

Una descripción por comportamiento consiste de una serie de instrucciones, que ejecutadas secuencialmente, modelan el comportamiento del circuito. Nunca nos enfocamos al nivel de compuerta. El bloque fundamental es el de *proceso*.

Especificación de un proceso

No perdamos de vista que estamos describiendo hardware, lo cual nos permite tener piezas electrónicas que funcionan simultáneamente, eso significa que vamos a describir secuencialmente un hardware que funciona de forma paralela. Sin embargo para describir comportamientos secuenciales surge el concepto de proceso.

El *proceso* es el bloque básico *concurrente* de codificación secuencial, el cual contiene una serie de instrucciones secuenciales que permiten modelar el comportamiento del circuito; esto significa que el proceso es a su vez una solo instrucción concurrente, que genera un pieza de hardware que funciona de manera paralela con otras piezas de hardware.

La región de codificación del proceso solo puede contener instrucciones de codificación secuenciales, muy parecidas a los lenguajes de alto nivel de descripción de software. Estas ideas se ilustran en la Fig. 2.9.

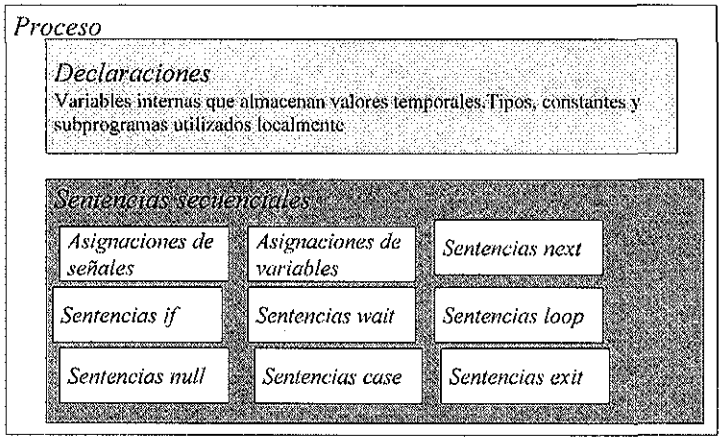


Fig. 2.9. Idea del proceso como elemento concurrente.

En esencia un diseño puede estar constituido de varios procesos operando concurrentemente, es decir en forma paralela e internamente cada proceso opera en forma secuencial. Aunque los procesos entre si son paralelos no necesariamente todos están procesando algo, es decir no todos están funcionando. Una característica especial de los procesos es que son disparados por algo conocido como *lista sensitiva*. La lista sensitiva son aquellas señales del sistema que deseamos activen el proceso cuando sufran algún cambio. En Pseudocódigo un proceso es representado como:

```
proceso (lista sensitiva)
   -- declaraciones internas
inicio
   -- instrucciones secuenciales
fin
```

Cuando la lista sensitiva del proceso no existe o especifica todas las señales del sistema, entonces estamos modelando *lógica combinacional*, lo que significa que el proceso siempre está operando. Cuando la especificación de la lista sensitiva es parcial o especifica la necesidad de *flancos* en alguna señal estamos modelando *lógica secuencial*.

El funcionamiento del proceso es similar al funcionamiento de un microprocesador, que funciona únicamente con interrupciones. Las señales dentro de la lista sensitiva son las interrupciones y el código dentro del proceso es la rutina de servicio. Cuando se termina de ejecutar el proceso este queda inactivo hasta que alguna de las señales de la lista cambie. Como podemos observar en la Fig. 2.9 un proceso puede tener un conjunto de variables internas que almacenan valores temporales. El resto son sentencias secuenciales que pueden ser de los siguientes tipos:

SI – ENTONCES – SINO.

Sintaxis:

```
Si (condición_cumple) Entonces
  -- sentencias
Sino
  -- sentencias
finsi
```

La interpretación de esta sentencia condicional es parecida a su equivalente en lenguaje "C", prueba una condición si esta se cumple hace algo y si no se cumple hace algo diferente.

Ejemplo:

```
Si (bandera) Entonces
  salida <= entrada1;
Sino
  salida <=entrada2;
finsi
```

Es claro que estamos definiendo un multiplexor de dos entradas, una salida y una línea de selección.

CASO – CUANDO – ES

Sintaxis:

```
Caso (expresión) Es
  Cuando alternativa1:  --sentencias
  Cuando alternativa2:  --sentencias
  ...
  Cuando alternativaN:  --sentencias
  Cuando OTROS:         --sentencias
Fincaso
```

Ejemplo:

```
Caso ({A,B}) Es
  Cuando 00: salida<=0;
  Cuando 01: salida<=1;
  Cuando 10: salida<=1;
  Cuando 11: salida<=1;
  Cuando OTROS: salida <=1;
Fincaso
```

Entendiendo que "A" y "B" son señales de un bit y "{}" es el operador concatenación, es claro que la sentencia esta describiendo una tabla de verdad o una función booleana que en este caso es la de la operación OR.

DESDE – HASTA

Sintaxis.

Desde identificador_inicio hasta identificador_final
--sentencias
Findesde

Ejemplo:

```
Desde i=0 hasta 15
  Si (reset(i) = '1') entonces
    Salida(i) <= 0;
  Finsi
Findesde
```

En este caso el ciclo se repite 16 veces y en cada una de ellas verifica el estado de las señales de "reset", las que estén verificadas en '1' ocasionan que la salida correspondiente se ponga en cero. Este código nos puede servir por ejemplo para saber que dispositivos re inicializar en función de las variables de entrada.

ASIGNACIÓN DE SEÑALES Y DE VARIABLES

Es importante distinguir los conceptos de señales y de variables cuando utilizamos un lenguaje de descripción de hardware y describimos un comportamiento utilizando un proceso.

Variable. Las variables solo pueden existir dentro de la estructura secuencial de un proceso y se interpretan de la misma manera que las variables en un lenguaje de programación de software. Una variable es una área de almacenamiento que puede alterar su valor en cualquier parte del proceso. Esto es las asignaciones y modificaciones que se hacen a las variables suceden en el instante que se ejecuta la instrucción. La asignación a una variable se representa con el signo ":=".

Por ejemplo:

```
Temporal := 20;
Contador := "11010010";
```

Señales. Las señales son las entradas y salidas del sistema que estamos modelando, por esta razón el alcance de las señales es tanto dentro como fuera de los procesos. La asignación de valores a las señales cuando se efectúa dentro del proceso no tienen lugar en el momento de la asignación, sino hasta el final del proceso. Es decir el valor asignado a una señal dentro de un proceso no puede ser utilizado en la siguiente sentencia del proceso. Se utiliza para la asignación el símbolo "<=".

Por ejemplo:

```
Datos_salida <= "10010001";
Datos_salida <= contador;
```

EJEMPLO DE UNA DESCRIPCION POR COMPORTAMIENTO

En seguida mostramos la descripción de un contador binario de cuatro bits usando un modelado por comportamiento y por tanto utilizando un proceso.

```
Proceso(reset,reloj)
  Variable Qint : vector(3 a 0);
  Inicio
    Si reset = '1' entonces
      Qint := "0000";
    Sino
      Si flanco_subida(reloj) entonces
        Q <= Qint;
        Qint := Qint + 1;
      Finsi
    Finsi
  Fin
Finproceso
```

Es claro que el proceso es sensible a cambios en "reset" y "reloj", las cuales son señales porque no están definidas dentro del proceso. Internamente al proceso definimos la variables "Qint" que llevará la cuenta del contador. La señal "reset" pone el contenido del contador en "0000".

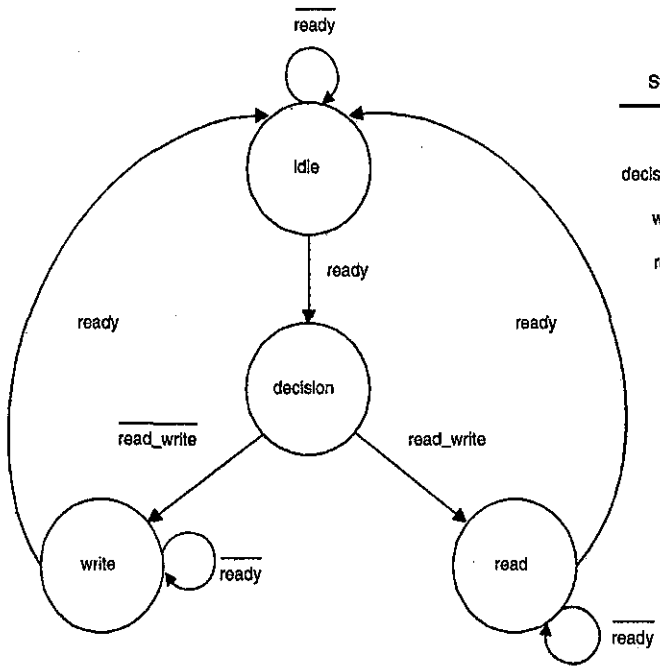
El proceso describe lógica secuencial porque sincroniza la cuenta del contador con la señal de reloj utilizando la función *flanco_subida* que está definida en cualquier HDL, de manera que solo en los flancos de subida se asigna el valor de la variable "Qint" a la señal "Q" que tuvo que ser definida fuera del proceso como señal, además se incrementa el valor de "Qint". De esta forma el proceso se invoca continuamente por la señal de "reloj", pero solo en los flancos de subida se incrementa el contador.

MAQUINA DE ESTADOS POR COMPORTAMIENTO USANDO DOS PROCESOS

En el modelado del "núcleo" del instrumento es muy útil el empleo de máquinas de estados dado que es una forma de describir algoritmos en hardware. De hecho la llamada carta ASM describe el comportamiento de un hardware solo que en forma gráfica.

Para definir una máquina de estados de esta manera necesitamos un par de procesos, que como hemos dicho trabajan en paralelo. Uno de los procesos es utilizado para definir el valor de las salidas así como las transiciones entre los estados, es decir calcula el estado siguiente en función del estado presente y del valor de las entradas. El segundo proceso solo sincroniza la carta y dicta *cuando el siguiente estado se convierte en el estado presente.*

Para aclarar esta idea presentamos el diseño de un controlador de RAM muy sencillo, como se ilustra en la Fig. 2.10. El controlador es usado para habilitar y des-habilitar el comando de escritura "we" y el de salida "oe" de un buffer de memoria durante peticiones de lectura/escritura. Las señales "ready" y "read_write" son salidas de un microprocesador y entradas a nuestro controlador. "we" y "oe" son salidas del controlador. Una nueva petición comienza con la habilitación de la señal "ready". Un ciclo de reloj después de iniciada la petición, el valor de "read_write" determina si la petición es de lectura o escritura. Si "read_write" es habilitada entonces se trata de un ciclo de lectura; de otro modo se trata de un ciclo de escritura. Un ciclo es completado al habilitar la señal de "ready" después de la cual una nueva petición puede ser aceptada.



State	Outputs oe	we
Idle	0	0
decision	0	0
write	0	1
read	1	0

Fig. 2.10. Diagrama de estados para el controlador de memoria.

El código que describe el comportamiento de esta máquina de estados es muy sencillo como se ilustra enseguida:

```
interfaz
  read_write, ready, clk   : entrada bit;
  oe, we                   : salida bit;
fininterfaz

hardware
  tipo estado es (idle, decision, read, write);
  señales present_state, next_state : estado;
inicio
  proceso(present_state, read_write, ready)
    caso present_state es
      cuando idle :
        oe <= '0'; we <= '0';
        si ready = '1' entonces
          next_state <= decision;
        sino
          next_state <= idle;
        finsi;
      cuando decision :
        oe <= '0'; we <= '0';
        si read_write = '1' entonces
          next_state <= read;
        sino
          next_state <= write;
        finsi;
      cuando read :
        oe <= '1'; we <= '0';
        si ready = '1' entonces
          next_state <= idle;
```

```
        sino
          next_state <= read;
        finsi;
      cuando write :
        oe <= '0'; we <= '1';
        si ready = '1' entonces
          next_state <= idle;
        sino
          next_state <= write;
        finsi;
    fincaso;
  finproceso;

    proceso(clk)
      si (flanco_subida(clk)) entonces
        present_state <= next_state;
      finsi;
    finproceso;
finhardware;
```

La interfaz del controlador consiste de las señales de entrada *read_write, ready, clk*, y de las de salida *oe, we*, de un bit cada una. Al inicio del hardware definimos un tipo llamado "estado" que es una enumeración de cuatro valores "idle, decision, read, write" los cuales son nombres que nos representan cada uno de los cuatro estados disponibles en la maquina de estados de la Fig. 2.10. Después definimos dos señales de uso global llamadas "present_state y next_state" que son del tipo que acabamos de crear y que nos van a servir para representar el estado actual y el estado siguiente.

El primer proceso es sensible al estado presente (present_state) así como a las señales de lectura escritura (read_write) y listo (ready). Este proceso lo que hace es probar utilizando la sentencia *caso* el valor del "present_state", el numero de casos corresponde con el número de estados, por tanto el constructor *caso* tendrá tantas opciones como numero de estados tenga la máquina de estados.

En cada opción contemplada por el *"caso"* dos son las acciones a realizar. La primera consiste en establecer el valor de las salidas "oe" y "we" para ese estado, debido a que se trata de una máquina de tipo Moore, estos valores son directamente tomados del diagrama de estados de la Fig. 2.10. La segunda acción para cada caso consiste en definir las transiciones del estado actual hacia el estado siguiente, en otras palabras calculamos cual es el estado siguiente en función de las entradas que sean importantes en el estado actual (ready o read_write); esta prueba se hace utilizando el constructor *"si entonces"* y anidándolos en caso de tener que probar varias entradas. De manera que el resultado de esta prueba es asignar a "next_state" el estado siguiente que puede ser alguno de los cuatro estados posibles definidos por "estado" según corresponda.

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

Esto significa que la transición del estado presente al estado siguiente está sincronizada con respecto a la señal de reloj del sistema y aunque se calcula en el proceso uno, no se asigna sino sincronizadamente con la señal de reloj en el proceso dos.

2.3.2.2 Modelado de la interfaz de entrada por Comportamiento

La interfaz de entrada es la encargada de aceptar las variables de entrada y proporcionarlas en la forma que el "núcleo" las requiera. El núcleo del instrumento es 100 % digital de manera que la interfaz a modelar puede ser de dos tipos:

- **Modelar el 100% de la interfaz.** Esto es posible cuando las variables a medir son proporcionadas en forma digital a nuestro instrumento, ya sea porque la variable a medir está en esa forma o porque existe una etapa previa de digitalización invisible para nuestro instrumento.

- **Modelar solo la parte digital de la interfaz.** Requerimos una interfaz híbrida cuando las variables de entrada a nuestro instrumento sean naturalmente analógicas, es este caso, y dependiendo del tipo y magnitud de la variable podemos requerir de un adecuamiento o preprocesamiento analógico para posteriormente ser convertidas a forma digital utilizando un convertidor analógico a digital y ahora poder diseñar la parte digital de la interfaz.

En esta metodología, nos enfocamos a diseñar la parte únicamente digital de la interfaz que es común a los dos puntos anteriores, es decir partiendo del hecho de que las variables a medir ya han sido preprocesadas y están en forma digital. Siendo este el caso tenemos dos posibilidades para proporcionar las entradas al núcleo:

Multiplexadas. Este tipo de interfaz, canaliza un numero cualquiera de entradas con una resolución determinada a una sola entrada dirigida al núcleo del instrumento. En este caso todas las magnitudes a medir entran al núcleo por un solo canal con una resolución determinada.

Ventajas:

- Reduce el tamaño del bus de datos de entrada al núcleo del instrumento.
- Reduce el área utilizada en el dispositivo lógico programable, debido al procesamiento serial de la información.

Desventajas:

- Afecta la velocidad de procesamiento, dado que toda la información de las variables a medir se transfiere por un solo canal hacia el núcleo. Por tanto reduce el ancho de banda disponible.
- Requiere una lógica adicional para el control de la velocidad de multiplexión.
- Puede generar un cuello de botella para núcleos que requieren procesamiento rápido.

Sin Multiplexar. En este tipo de interfaz, cada uno de las entradas a medir se ingresa directamente al núcleo del instrumento por canales diferentes utilizando una resolución posiblemente distinta para cada una de ellas.

Ventajas:

- Aumenta el ancho de banda disponible, ya que hay un canal para cada entrada.
- Permite satisfacer las demandas de entrada de núcleos muy rápidos.

Desventajas

- Aumenta el área utilizada en el dispositivo lógico programable, dado que el tamaño del bus de datos se incrementa.
- Requiere duplicidad de hardware con igual funcionalidad para procesar en paralelo las señales de entrada.

Recomendamos la utilización de una interfaz de entrada multiplexada siempre y cuando no represente un cuello de botella para la velocidad de procesamiento del núcleo. El esquema de esta interfaz se ilustra en la Fig. 2.11. Observamos que el multiplexor proporciona un solo camino para cada una de las "m" entradas de "n" bits cada una. Cada entrada es ingresada al núcleo del instrumento de manera controlada por la lógica de control. Observe que la interfaz es síncrona porque esta referenciada por el reloj del sistema global. La lógica de control es esencialmente un contador sincronizado con respecto al núcleo del instrumento que elige consecutivamente las señales de entrada.

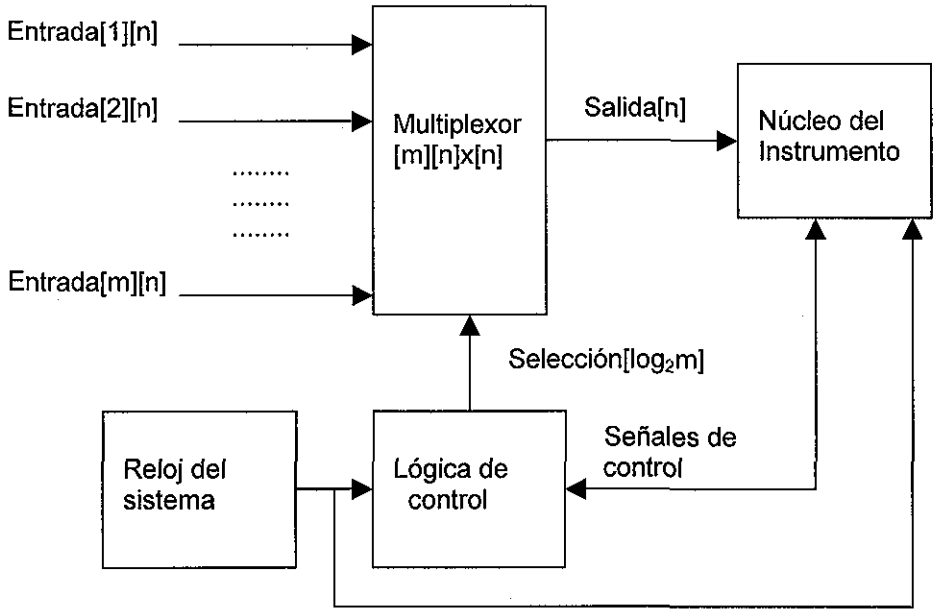


Fig. 2.11. Diagrama a bloques de la interfaz de entrada multiplexada.

2.3.2.3 Modelado de la interfaz de salida por Comportamiento

Los resultados de las mediciones realizadas por el "núcleo" tienen por objetivo ser visualizadas como resultado del proceso más que ser utilizadas como variables de control, de manera que requerimos de una interfaz que muestre los resultados, que por lo general es 100% digital y cuya lógica puede ser construida totalmente en el dispositivo lógico programable.

En esta metodología proponemos tres formas para la interfaz de salida, todas ellas modeladas por comportamiento. La complejidad de cada una de ellas varía sustancialmente y la elección de cual utilizar está en función de la complejidad o cantidad de datos a desplegar. Las tres propuestas son las siguientes:

- Displays de siete segmentos.
- Display de cristal líquido.
- Monitor VGA (640*480).

DISPLAY DE SIETE SEGMENTOS.

Esta opción es muy útil cuando requerimos una visualización decimal de la medición realizada por el núcleo del instrumento. Esencialmente el núcleo del instrumento debe entregar los resultados de la medición en código BCD. La interfaz de salida toma esta información y le aplica un conversión de código BCD a siete segmentos enviando estos resultados a displays de siete segmentos. Esta interfaz de salida puede ser de dos tipos.

Multiplexada.

Este tipo de interfaz, canaliza un numero cualquiera de salidas BCD provenientes del núcleo a una sola salida dirigida por un solo BUS hacia un decodificador BCD a siete segmentos y de ahí utilizando un solo BUS hacia los displays de siete segmentos.

Ventajas:

- Reduce el número de pines de salida requeridos para mostrar los resultados de la medición.
- Reduce el área física del dispositivo lógico programable, así como el área utilizada internamente.
- Facilita la fabricación del circuito impreso final PCB.

Desventajas:

- Requiere una lógica adicional para el control de la velocidad de multiplexión.
- Una multiplexión muy rápida puede producir problemas de ruido y reducir el ancho de banda del instrumento.

Sin Multiplexar.

En este tipo de interfaz, cada dígito de la medición calculado por el núcleo se alimenta a un decodificador de código BCD a siete segmentos, después del cual se envía la información a displays de siete segmentos.

Ventajas:

- Permite reducir problemas de ruido.
- No requiere lógica de control.

Desventajas

- Aumenta el número de pines de salida requeridos para mostrar los resultados de la medición.
- Aumenta el área física del dispositivo lógico programable.
- Dificulta la fabricación del impreso final PCB.

Recomendamos la utilización de una interfaz de salida multiplexada. El esquema de esta interfaz se ilustra en la Fig. 2.12. Observamos que el núcleo del instrumento debe entregar salidas codificadas en BCD a un multiplexor de "m" entradas BCD. Este multiplexor genera un solo bus que transfiere uno a uno cada dígito codificado a una frecuencia múltiplo de sesenta según el numero de displays necesarios. Esta acción permite compartir un solo decodificador BCD a 7 segmentos por todas las salidas. El contador a la frecuencia elegida se encarga de seleccionar consecutivamente cada dígito, el cual se decodifica y habilita el correspondiente display de siete segmentos usando un conjunto de transistores trabajando en corte y saturación.

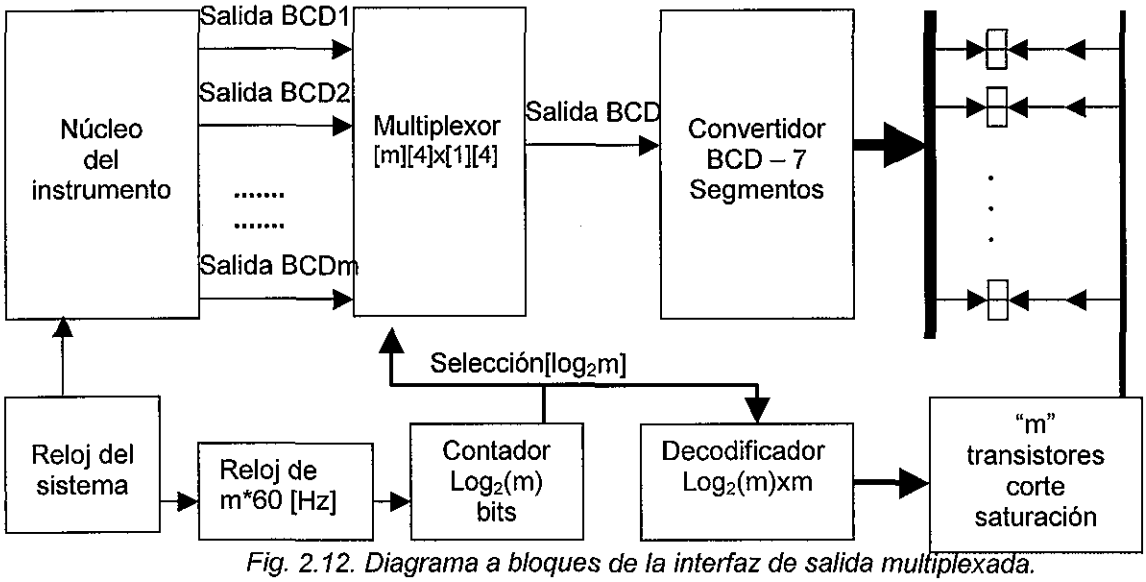


Fig. 2.12. Diagrama a bloques de la interfaz de salida multiplexada.

DISPLAY DE CRISTAL LÍQUIDO NUMÉRICO O GRÁFICO.

Esta opción es muy útil cuando requerimos una visualización decimal de la medición realizada por el núcleo del instrumento. Esencialmente el núcleo del instrumento debe entregar los resultados en la forma requerida por el display. En el mercado existen muchos tipos de displays de cristal líquido, con determinado número de dígitos así como indicadores adicionales.

Por supuesto la elección de un display de este tipo implica el uso de una lógica adicional para el manejo del mismo. Este tipo de displays operan con procesos multiplexados con el objeto de reducir el número de pines requeridos por el sistema.

MONITOR VGA (640*480)

Este tipo de interfaz utiliza un monitor VGA estándar para la visualización de la información en formato gráfico. Es una forma de interfaz de salida bastante sofisticada pero muy útil por la cantidad de información que puede ser mostrada. La información a mostrar puede ser cualquier elemento gráfico. De hecho puede mostrar no solamente caracteres alfanuméricos usando una ROM con códigos ASCII, sino que también es posible mostrar elementos gráficos.

Una interfaz de salida de este tipo implica la necesidad de una lógica de control adicional a la del "núcleo" del instrumento y puede requerir mayor espacio en el dispositivo lógico programable que el mismo "núcleo".

Sin embargo esta interfaz es sumamente útil y da a ciertos instrumentos una capacidad de despliegue impresionante.

Ventajas

- Permite desplegar cualquier tipo de información.
- Permite visualizar elementos gráficos.
- Requiere pocos pines de entrada y salida.

Desventajas

- Requiere de bastante espacio en el dispositivo lógico programable para implementar la lógica de control.
- Puede llegar a ser el elemento más costoso del instrumento.

En la Fig. 2.13 mostramos el diagrama a bloques de la interfaz utilizando como despliegue un monitor VGA.

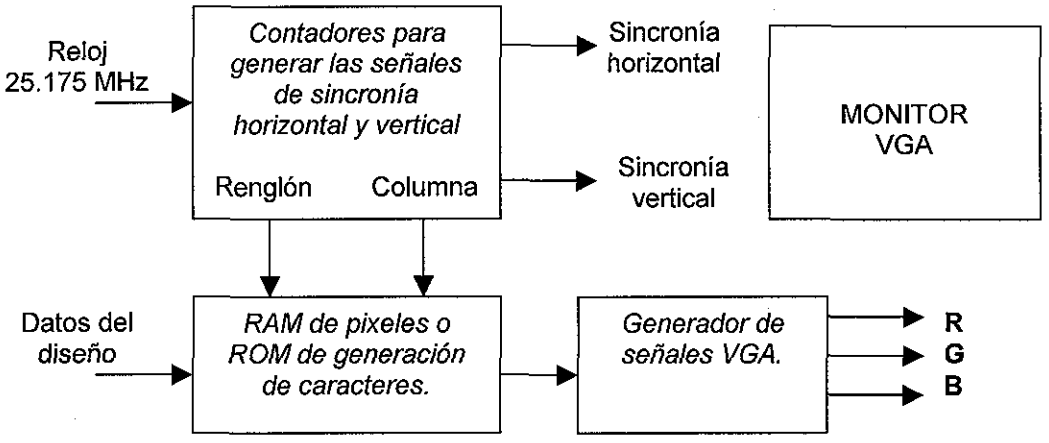


Fig. 2.13. Diagrama a bloques de la interfaz de salida VGA.

Se dispone de un bloque que partiendo de la señal base de 25.175 MHz genera las señales de sincronía horizontal y vertical, así como las señales de barrido de una memoria mediante renglón-columna. Los datos a desplegar pueden ser escritos en cualquier momento en la memoria por la aplicación que desee utilizar la interfaz. La información de la memoria se utiliza para generar las señales RGB, que junto con las señales de sincronía permiten el despliegue de información en un monitor VGA. La memoria de datos puede ser una RAM que contiene el mapa de bits de cada pixel del monitor de 640x480 para despliegue gráfico, o una ROM para la generación de caracteres en caso de despliegue de caracteres ASCII (texto). En el Capítulo 4, presentaremos la aportación específica de la presente metodología con respecto al diseño genérico de una interfaz de salida VGA muy útil en la instrumentación.

2.3.3 Elección del dispositivo CPLD

Esta etapa de la metodología no es crítica, sin embargo impacta considerablemente algunos aspectos finales, siendo el más importante el costo. Actualmente las empresas que fabrican dispositivos lógicos programables cuentan con un gran número de dispositivos, cada uno de ellos con diferentes costos y características.

Al elegir el dispositivo para nuestra aplicación debemos de considerar las siguientes tres características esenciales.

- **Restricciones de tiempo**

Todos los dispositivos lógicos programables tienen especificaciones de tiempo, esto es una medida de la velocidad máxima de operación del dispositivo. Esto significa que debemos evaluar el límite máximo de velocidad de operación requerido en el instrumento para descartar aquellos dispositivos que no cumplan con el requisito.

En el caso de los CPLDs tienen la ventaja de que su modelo de tiempo es bastante determinístico y puede ser conocido con bastante exactitud y no depende tanto de la forma final en que el diseño es implementado en el dispositivo por el algoritmo de ruteo.

En el caso de los FPGAs su modelo de tiempo no es tan determinístico y depende finalmente de la forma en que el algoritmo de ruteo programa el diseño dentro del dispositivo.

Nuestra metodología recomienda elegir dispositivos que por lo menos estén sobrados con un 20 % en cuanto a su limite de operación con respecto a la especificación máxima de nuestro diseño, esto con la finalidad de posibles expansiones futuras.

- **Restricciones de espacio**

La gran diversidad de dispositivos disponibles en el mercado, incluso por una misma empresa, se debe a la capacidad de espacio del dispositivo. Existen desde aquellos en los que podemos programar circuitos combinacionales y secuenciales simples, hasta aquellos que pueden contener el diseño de un microprocesador completo y complejo.

El costo de estos dispositivos es directamente proporcional a la capacidad del mismo, la unidad comúnmente utilizada para referirnos a la capacidad es por el número de compuertas que pueden ser programadas dentro del dispositivo, o por el número de bits RAM que pueden ser almacenados en el mismo.

Nuestra metodología recomienda que se elija un dispositivo el cual después de ser programado con el diseño tenga el 20% de espacio libre para posibles modificaciones, correcciones o ampliaciones de versión.

Muchas restricciones de espacio impuestas por las especificaciones de diseño se pueden satisfacer aplicando optimización a ciertas etapas. Por ejemplo una interfaz de salida multiplexada utilizando displays de siete segmentos ocupará menos espacio que la versión no multiplexada, y además ocupará menos espacio si en lugar de multiplexores ocupamos buses tres estados.

- **Restricciones de costos**

Cuando la especificación de diseño pone un costo tope, es necesario ocasionalmente en lugar de utilizar un solo dispositivo de más capacidad, emplear dos o tres dispositivos más pequeños cuyo costo total sea inferior a uno de mayor capacidad.

También se da el caso de cuando la elección de un dispositivo de mayor capacidad eleva el costo no por el dispositivo sino por ejemplo por el costo de generar el PCB, que pudiera requerir tecnología de montaje superficial por ejemplo.

Nuestra metodología recomienda siempre dejar por lo menos un 5% por debajo del costo tope impuesto en la especificación para tener un pequeño colchón en caso de requerir modificaciones de última hora.

2.3.4 Compilación y Síntesis

Esta etapa de la metodología depende completamente de la herramienta CAD-EDA que estemos utilizando en el proceso de diseño de nuestro instrumento. En el mercado existen varias herramientas de software, por ejemplo las de Altera, Xilinx, etc.

Estas herramientas además de aceptar diferentes entradas de diseño como descripción gráfica o funcional utilizando algún lenguaje de descripción de hardware; también permiten el paso siguiente que es la compilación y síntesis del diseño, que en esencia es lo siguiente:

Compilación. Recibe la descripción del diseño ya sea en formato gráfico o de lenguaje de descripción de hardware, analizando la sintaxis y semántica para determinar si el diseño representado está libre de errores y representa circuitos digitales factibles de implementar en dispositivos lógicos programables.

Síntesis. Este proceso tiene por objetivo generar el circuito diseñado de forma adecuada para que sea posible implementarlo en la tecnología de dispositivos lógicos programables soportado por la herramienta. En otras palabras este proceso traduce el diseño ya compilado a una representación utilizando los bloques funcionales básicos disponibles en la tecnología.

Esta etapa puede ser poco manipulada por el usuario, algunas herramientas brindan alternativas que pueden controlar las opciones de compilación y síntesis. Por ejemplo en la herramienta Max+Plus II de Altera es posible indicar que se ponga más atención a una optimización del diseño en velocidad y no en espacio de silicio o viceversa. Que se utilicen o no ciertas características especiales de la tecnología como salidas con colector abierto etc.

En la Fig. 2.14 se ilustra la ventana típica de la herramienta Max+Plus II que describe este proceso de Compilación y Síntesis.

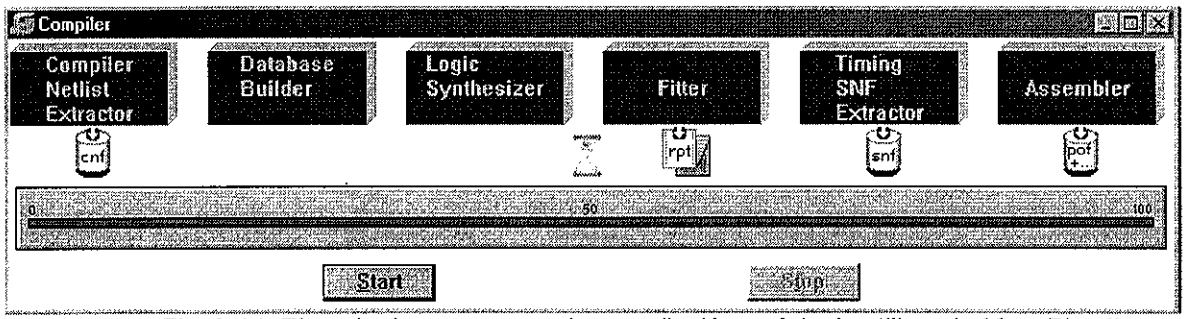


Fig. 2.14. Ejemplo de un proceso de compilación y síntesis utilizando Max+Plus II.

Durante el proceso de compilación y síntesis se pueden generar muchos errores cuyas causas pueden ser variadas, por ejemplo, nuestro diseño puede incluir características que violen restricciones eléctricas, como querer asignar un "1" y un "0" lógicos al mismo tiempo a la misma salida. Algunos otros errores pueden ser solo de sintaxis. Los errores más complicados pueden ser aquellos de diseño que pasen el proceso de compilación y síntesis y que no serán detectados sino hasta una etapa posterior.

La mayoría de las herramientas proporcionan una breve descripción de los errores encontrados; mientras el proceso de compilación y síntesis no resulte satisfactorio debemos regresar a la etapa de modelado en alto nivel para corregirlos, lo cual implica modificar consecutivamente el modelo hasta que no resulten errores; para poder pasar a la siguiente etapa.

2.3.5 Simulación Digital

Esta etapa de la metodología depende esencialmente de dos partes fundamentales. La primera es dependiente directamente de la herramienta de simulación proporcionada por el fabricante. La segunda depende de las estrategias del diseñador para optimizar el proceso de simulación fundamentalmente en tiempo.

Primera parte. Es importante distinguir entre simulación funcional y simulación digital. En la simulación funcional solo se verifica el funcionamiento del modelo tomando en cuenta el comportamiento ideal de todos los dispositivos digitales utilizados. En la simulación digital además de considerar los funcionamientos de todos y cada uno de los dispositivos digitales utilizados también se toma en cuenta, por ejemplo, los tiempos de propagación de todos los componentes digitales, así como restricciones propias de la tecnología a utilizar, de esta manera es posible obtener una simulación muy parecida al funcionamiento real del dispositivo final.

La mayoría de las herramientas del mercado ligadas a una tecnología específica como las de Altera y Xilinx proporcionan simuladores funcionales que operan directamente sobre sus dispositivos lógicos programables, de hecho es posible elegir el dispositivo antes de la compilación, síntesis y simulación de manera que se consideren todas las restricciones específicas del mismo.

Segunda parte. Depende esencialmente de las estrategias convenientes a utilizar para optimizar el proceso de simulación, sobretodo cuando se requieren tiempos de simulación grandes en relación a la frecuencia base del sistema diseñado. Algunas ideas proporcionadas por esta metodología son las siguientes:

- En el caso de requerir tiempos de simulación grandes en relación a la frecuencia de operación base del dispositivo, es conveniente escalar la frecuencia base del dispositivo. Por ejemplo suponga que deseamos verificar tres segundos de operación de un instrumento llamado frecuencímetro y la frecuencia base de operación es de 8MHz. Esta relación entre tiempo de simulación y frecuencia base va a ocasionar tiempos de simulación demasiado grandes e *innecesarios. En estos casos se recomienda escalar la frecuencia base de 8MHz tanto como* sea necesario para reducir el tiempo de simulación a unidades razonables. El resultado final del instrumento por supuesto deberá de ser afectado por el mismo factor.

- La primera simulación solo debe contener las entradas y salidas del núcleo del instrumento, es decir debemos centrar el esfuerzo en verificar el funcionamiento del núcleo, aislando las interfaces de entrada y salida; dado que estas últimas son comunes a la mayoría de los instrumento y únicamente ocasionarían mayores tiempos de simulación.

- Si los resultados de la simulación no son los esperados se deben utilizar estrategias de depuración que consisten en visualizar como resultado de las simulaciones no solamente las entradas y salidas del "núcleo", sino también señales intermedias que nos permitan determinar la causa de los errores.

- Los problemas de asíncronismos detectados en el proceso de simulación deben tratar de ser corregidos aunque no se afecten los resultados finales, esto debido a que un diseño con muchos asincronismos hace completamente dependiente al diseño del dispositivo específico en que se este implementando.

El proceso de simulación termina cuando los resultados coinciden con los resultados esperados para un marco de prueba lo suficientemente representativo de los casos reales tratados por el instrumento final. En caso de que los resultados no sean los esperados es necesario realizar estrategias de depuración como las indicadas anteriormente. Este proceso nos puede regresar a tener que elegir otro tipo de dispositivo en el caso de que el elegido no sea el idóneo para los

requerimientos. También es posible que tengamos que regresar a modificar el modelo de alto nivel o incluso hasta cambiar las especificaciones así como el análisis del problema.

2.3.6 Programación del dispositivo

Esta etapa de la metodología se adapta a la herramienta que estemos utilizando así como a la tecnología de dispositivos lógicos programables empleada. La mayoría de las herramientas como resultado del proceso de compilación y síntesis entregan archivos de programación que son utilizados directamente por la herramienta para programar o configurar el dispositivo.

Adicionalmente se incluye una herramienta que brinda mucha versatilidad en el diseño y que consiste en decidir de manera automática (por el software) o de manera manual (por el usuario) la asignación final de los pines de entrada / salida de nuestro instrumento en relación con el dispositivo programable.

Algunas estrategias sugeridas por la presente metodología para optimizar la asignación de pines son las siguientes:

- Durante todas las etapas anteriores y en esencia hasta que los resultados del proceso de simulación sean los esperados no se debe hacer una asignación manual de pines. Es decir se debe permitir que la misma herramienta haga su propia asignación. Esta recomendación se debe a que la herramienta decide la ubicación de los pines de manera tal de optimizar y aprovechar el espacio interno del dispositivo. Si manualmente forzamos la asignación podemos ocasionar una utilización adicional de recursos y la posibilidad de que el diseño no pueda ser sintetizado por problemas de espacio.

- Cuando los resultados del proceso de simulación son satisfactorios podemos comenzar a planear la reasignación manual de pines de entrada / salida. Esta decisión se debe elegir solo cuando consideremos que la asignación actual complica innecesariamente tanto en tiempo como en costo el diseño del circuito impreso (PCB) del prototipo.

- La reasignación de pines debe comenzar por colocar en forma adyacente todas aquellas señales que son un bus o que se podrían organizar como tal. Es recomendable hacer estas modificaciones por pasos y recompilar y resintetizar el diseño para poder determinar si los cambios hechos permiten que el diseño sea aún factible de ser implementado en el dispositivo.

En la Fig. 2.15 mostramos una ventana típica de programación utilizando la herramienta Max+Plus II de Altera.

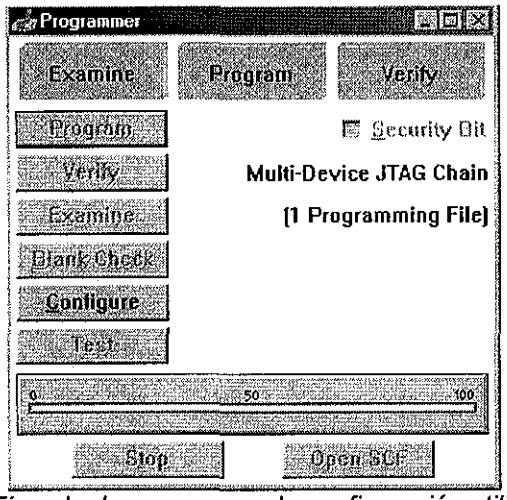


Fig. 2.15. Ejemplo de un proceso de configuración utilizando Max+Plus II.

2.3.7 Pruebas y Depurado

Esta etapa de la metodología tiene por objetivo hacer pruebas y depurado sobre el dispositivo físico real ya programado. Este proceso puede hacerse de dos maneras.

- **Prototipo rápido.** Consiste en utilizar el dispositivo ya programado y verificar su funcionalidad ya sea sobre un protoboard o sobre alguna tarjeta de desarrollo vendida por las mismas empresas que fabrican los dispositivos programables así como las herramientas de software. Esta es una alternativa rápida y eficiente. Cuando utilizamos protoboard y trabajamos con diseños que operan a alta frecuencia es necesario tomar en cuenta las restricciones y perturbaciones producidas por las capacitancias parásitas.

- **Prototipo PCB.** Consiste en fabricar un circuito impreso para colocar nuestro dispositivo *programables con nuestro diseño. Esta forma tiene la ventaja de generar un prototipo final que* es técnicamente ya el producto final. La desventaja es que se trata de un proceso más lento que implica el diseño y fabricación del PCB.

Las pruebas definitivas se realizan ya sobre el prototipo real y se espera un funcionamiento igual que el obtenido por la etapa de simulación. La mayoría de las veces los resultados de esta etapa serán correctos. En caso de no serlo es necesario aplicar un proceso de depuración, algunas estrategias son:

- Determinar si el funcionamiento no esperado se debe a una condición vulnerable del diseño como el caso de asincronismos, o a problemas de ruido inducidos a nuestro sistema por fuentes externas.

- Considerar todos los problemas de rebote introducidos por cualquier interruptor conectado como entrada a nuestro sistema.

- Considerar los problemas de ruido inducido por interruptores en la periferia de nuestro sistema.

- Desacoplar adecuadamente todos los dispositivos digitales incluido nuestro dispositivo lógico programables

Si el resultado de esta etapa no proporciona los resultados obtenidos con los recomendaciones indicadas anteriormente será necesario regresar a la etapa de modelado o incluso a la re especificación y análisis del problema.

Todas las etapas anteriores pretenden ser una guía útil y enfocada al diseño de instrumentos digitales utilizando dispositivos lógicos programables; con fundamento en la experiencia adquirida en el diseño de este tipo de sistemas utilizando la citada tecnología.

CAPÍTULO 3

CASO DE APLICACIÓN FRECUENCÍMETRO DIGITAL

CONTENIDO DEL CAPÍTULO

3.1 Introducción
3.2 Especificación y Análisis del Frecuencímetro.
3.3 Modelado del Frecuencímetro.
3.3.1 Modelado Estructural del Núcleo del Frecuencímetro.
3.3.2 Modelado Estructural de la Interfaz de Entrada del Frecuencímetro.
3.3.3 Modelado Estructural de la Interfaz de Salida del Frecuencímetro.
3.3.4 Modelado por Comportamiento del Núcleo del Frecuencímetro.
3.3.5 Modelado por Comportamiento de la Interfaz de Entrada del Frecuencímetro.
3.3.6 Modelado por Comportamiento de la Interfaz de Salida del Frecuencímetro.
3.3.7 Escalas del Instrumento.
3.4 Elección del Dispositivo CPLD.
3.5 Compilación y Síntesis.
3.6 Simulación Digital.
3.7 Programación del Dispositivo.
3.8 Pruebas y Resultados.
3.9 Producto Final.

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 limite critico 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 limites 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 común 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.

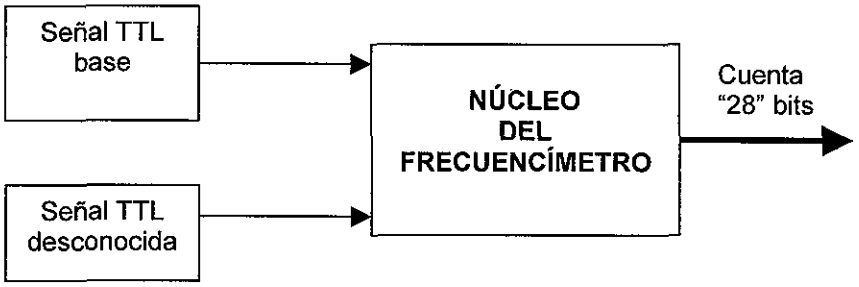


Fig. 3.1. Núcleo del frecuencímetro.

- **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 núcleo 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 único 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 específica 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 núcleo porque entonces estamos requiriendo que la cuenta entregada por el núcleo sea en código BCD, es decir cuatro bits por cada dígito de la cuenta y dado que la especificación solicita una frecuencia máxima medible de 10 MHZ, requerimos de siete dígitos 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 así como las interfaces de entrada y salida que fueron analizadas en la sección anterior. Recuerde que nuestra metodología 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 núcleo 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 frecuencímetro 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 frecuencímetro. 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 así 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.

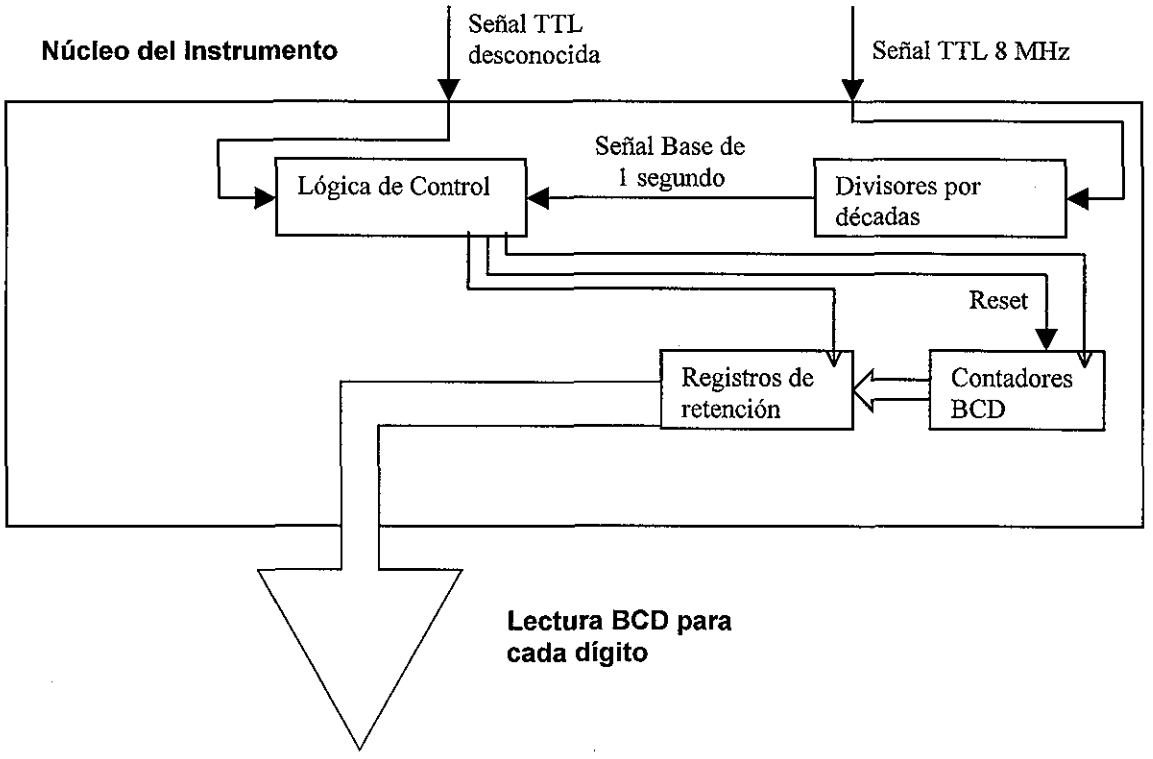


Fig. 3.2. Diagrama a bloques del núcleo estructural del frecuencímetro

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 μs. 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 μs. Un problema encontrado al implantar en forma estructural esta arquitectura resulta en que el contador Jonson no existe en las librerias 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.

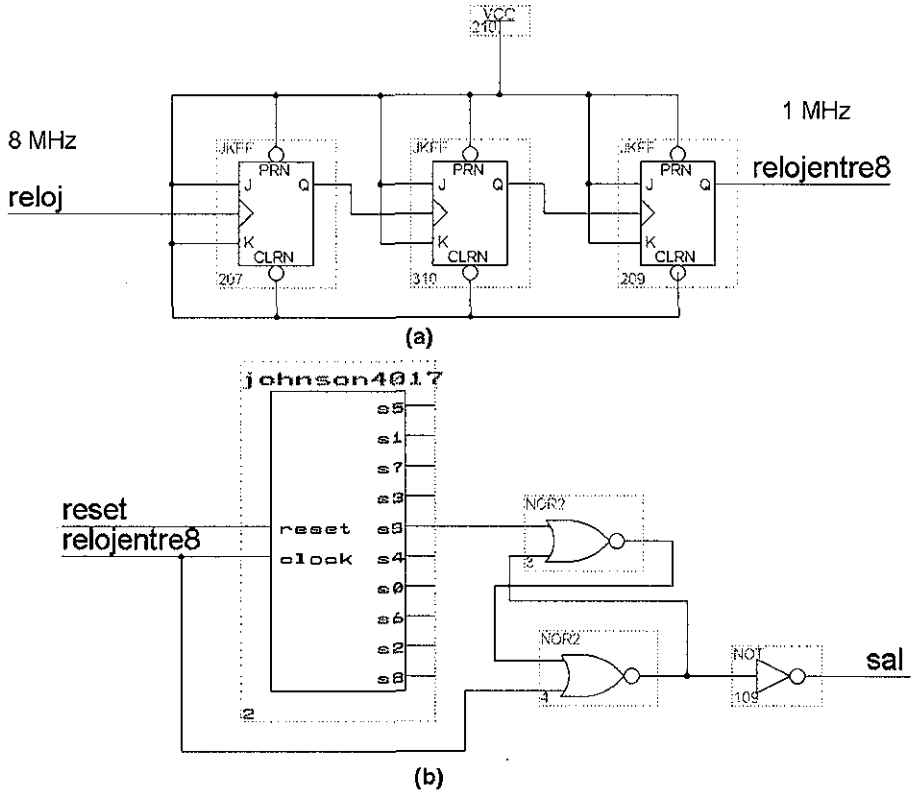


Fig. 3.3 Divisor por décadas. (a) Divisor por 8. (b) Divisor por 10.

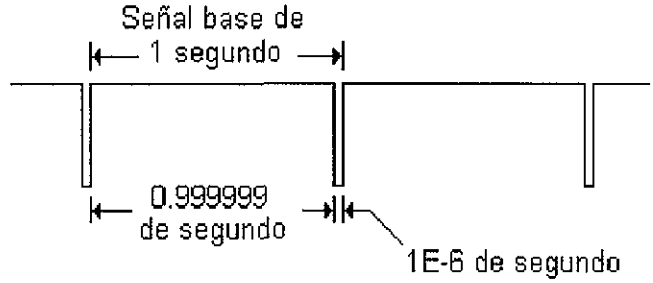


Fig. 3.4 Forma de la señal base de tiempo de 1 segundo.

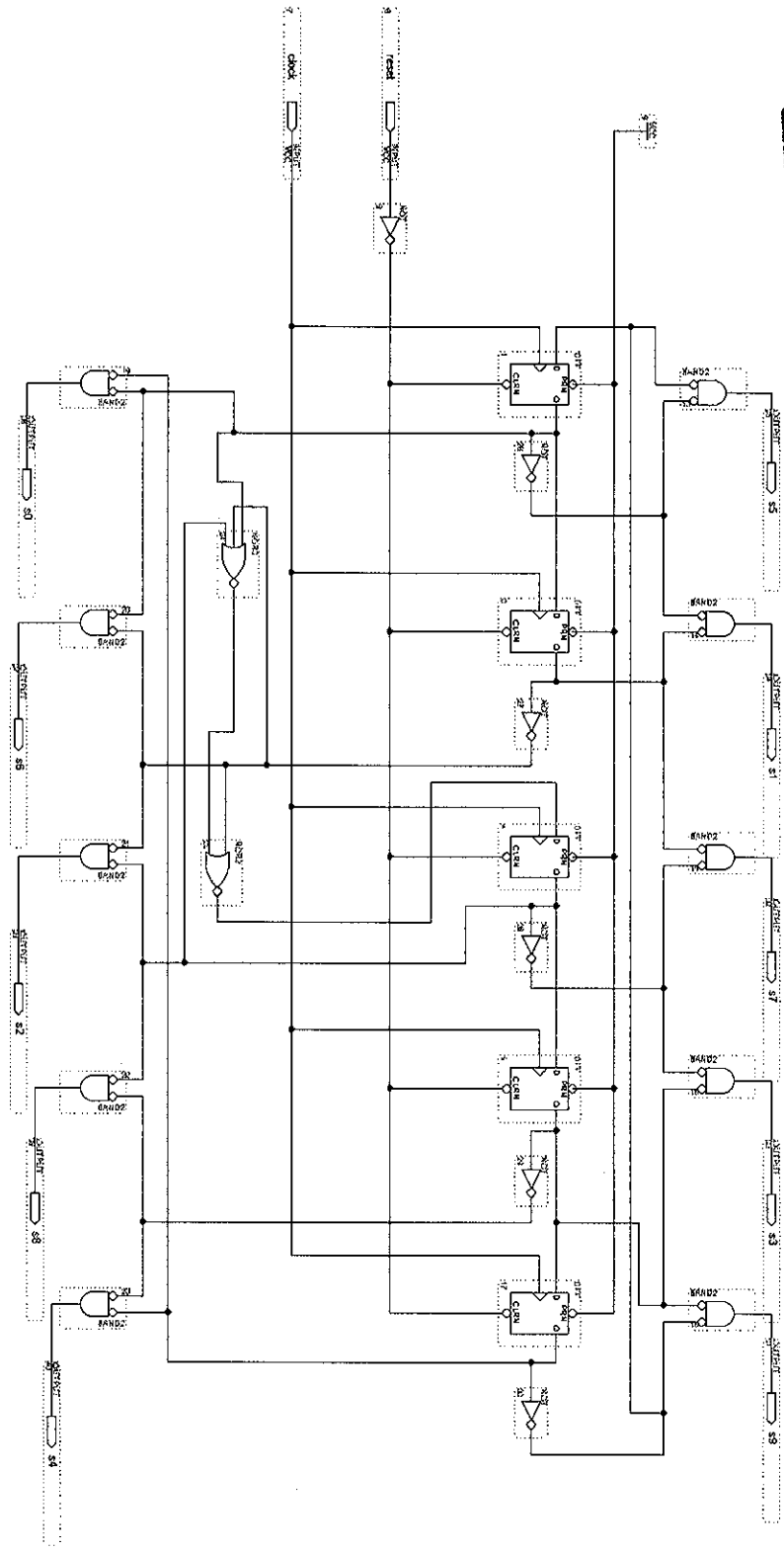


Fig. 3.5 Implementación del contador Johnson.

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 número 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.

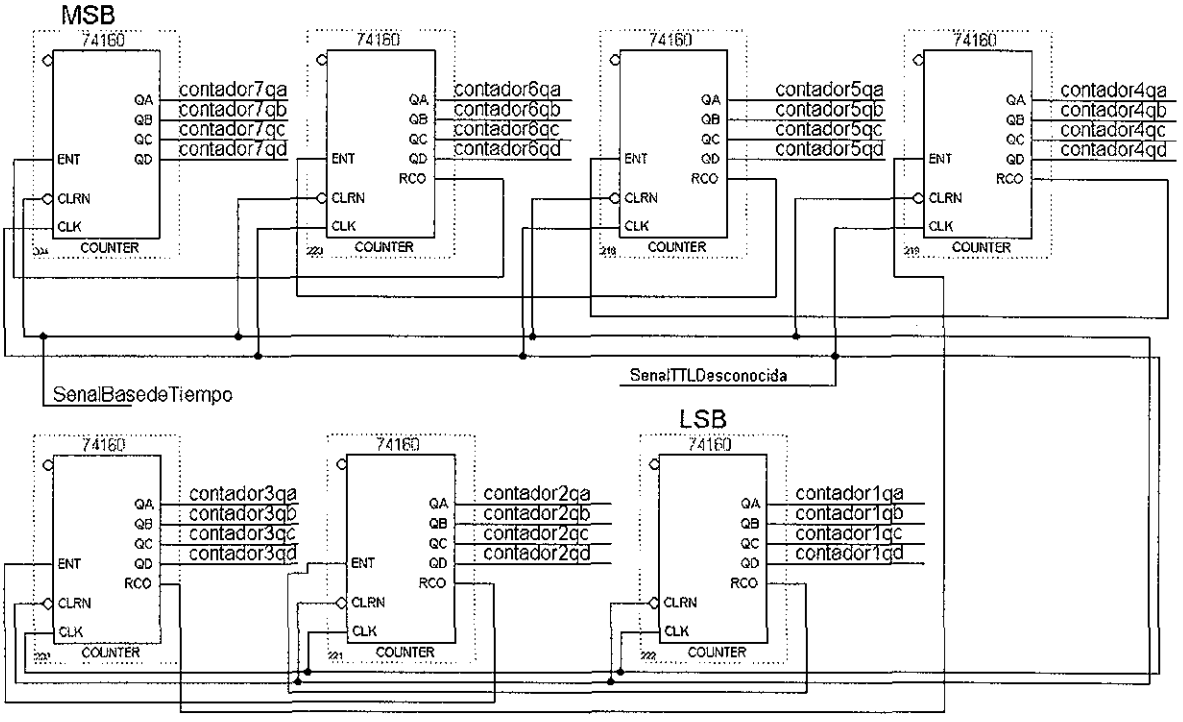


Fig. 3.6 Conteo de pulsos.

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 asíncrono. 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.

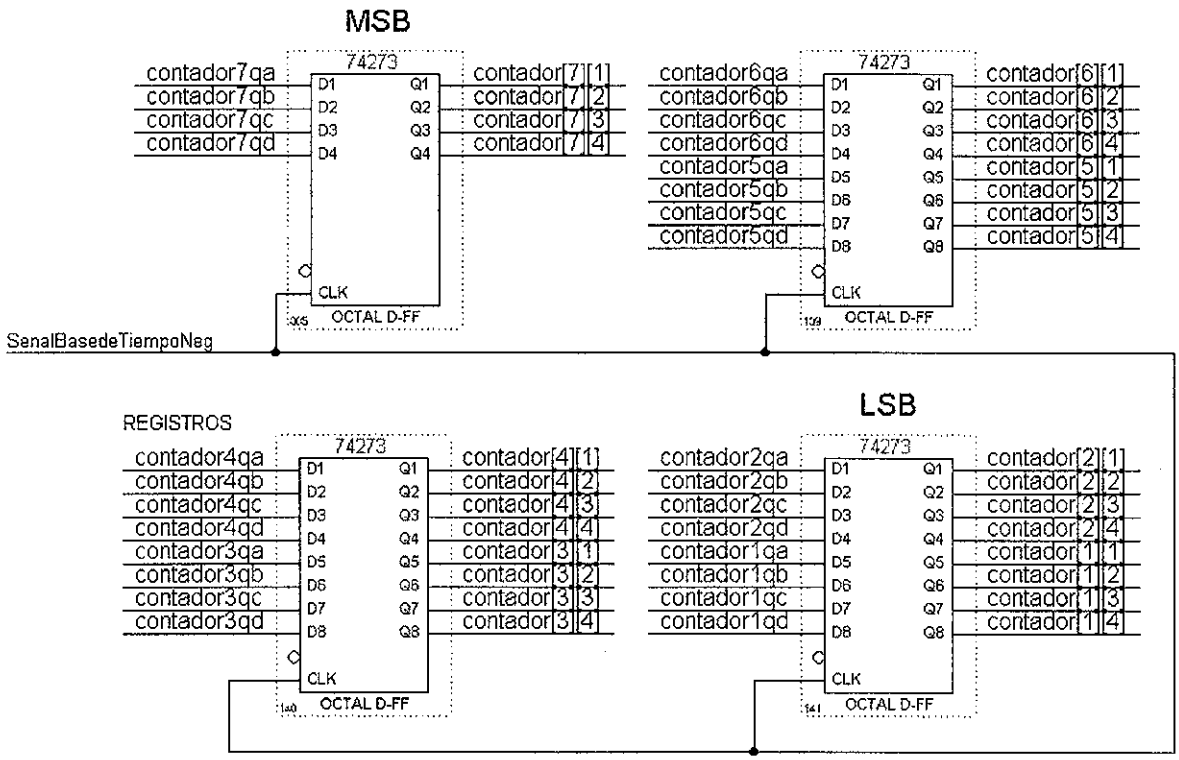


Fig. 3.7 Registros de retención.

Como podemos observar en la Fig. 3.7 estamos utilizando Flip-Flop "D" típicos de la serie 74, en específico 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.

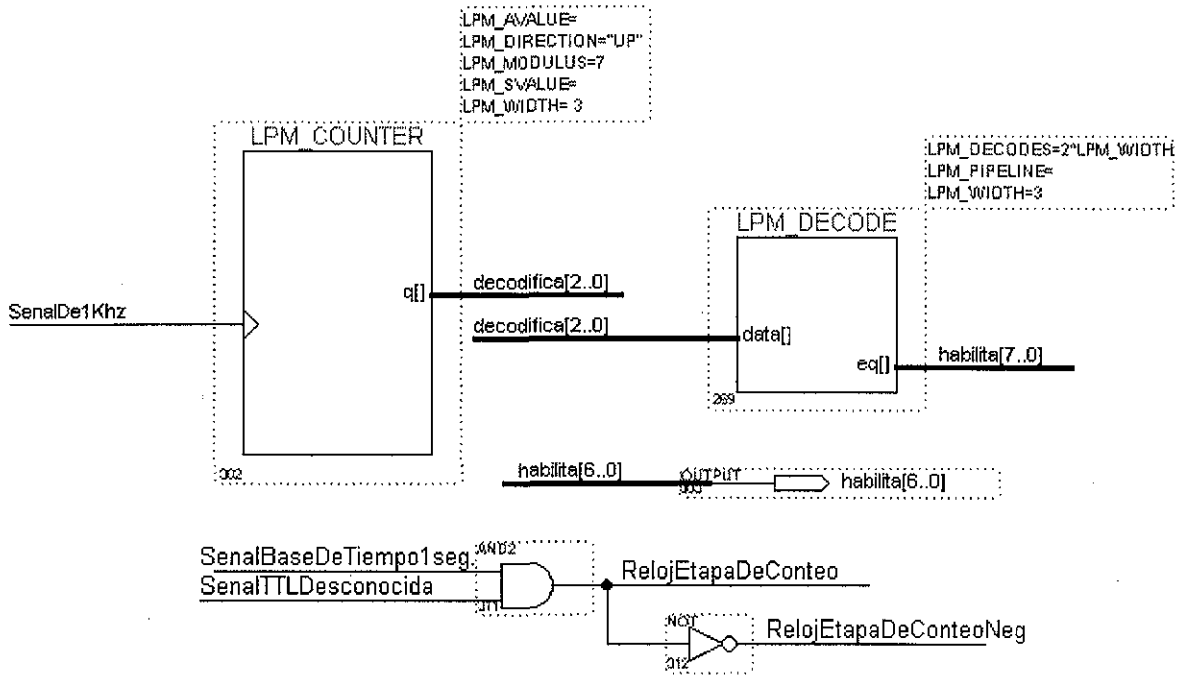


Fig. 3.8 Lógica de Control.

3.3.2 Modelado estructural de la interfaz de entrada del frecuencímetro.

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 ningún esquema de multiplexión a la entrada. Sin embargo estas dos señales que recibe el núcleo 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 frecuencímetro 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 47μF 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.

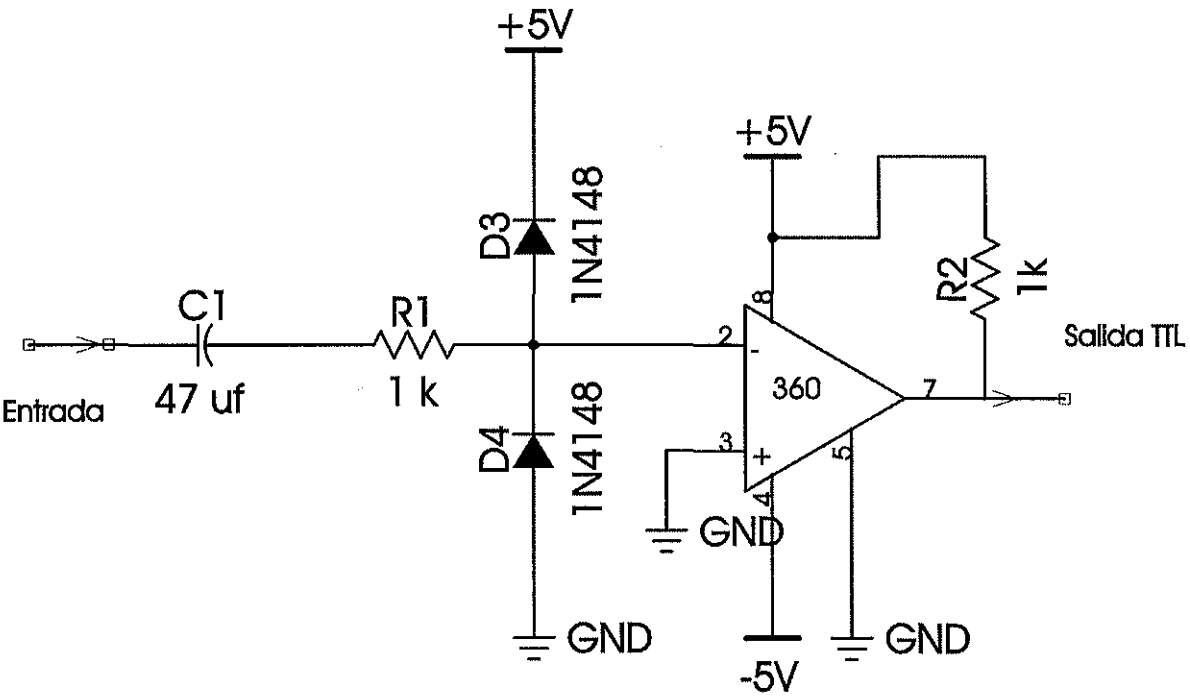
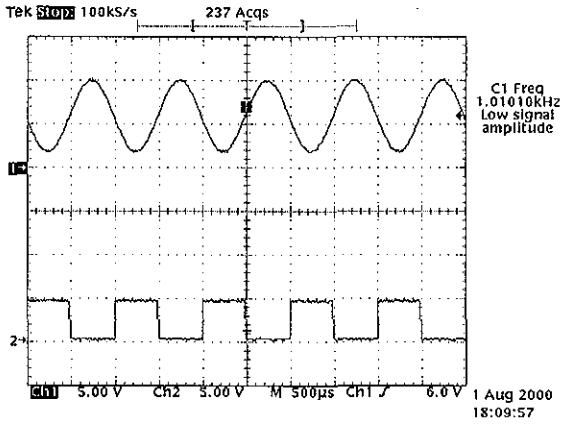
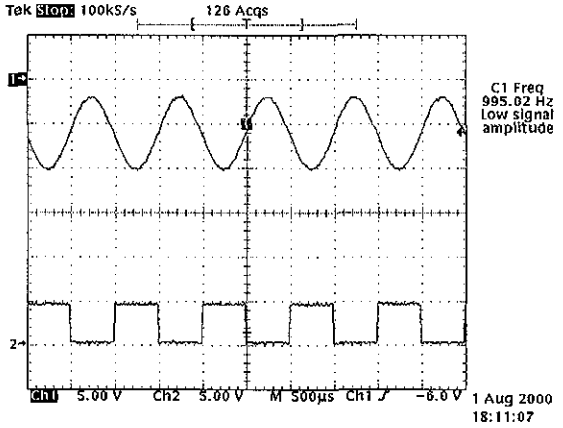


Fig. 3.9 Acondicionador de la señal de entrada a niveles TTL.

En la Fig. 3.10 se muestran los oscilogramas 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 común a todos los displays los cuales son controlados por las líneas de selección generadas en el módulo de lógica de control ubicada en el núcleo de la aplicación.

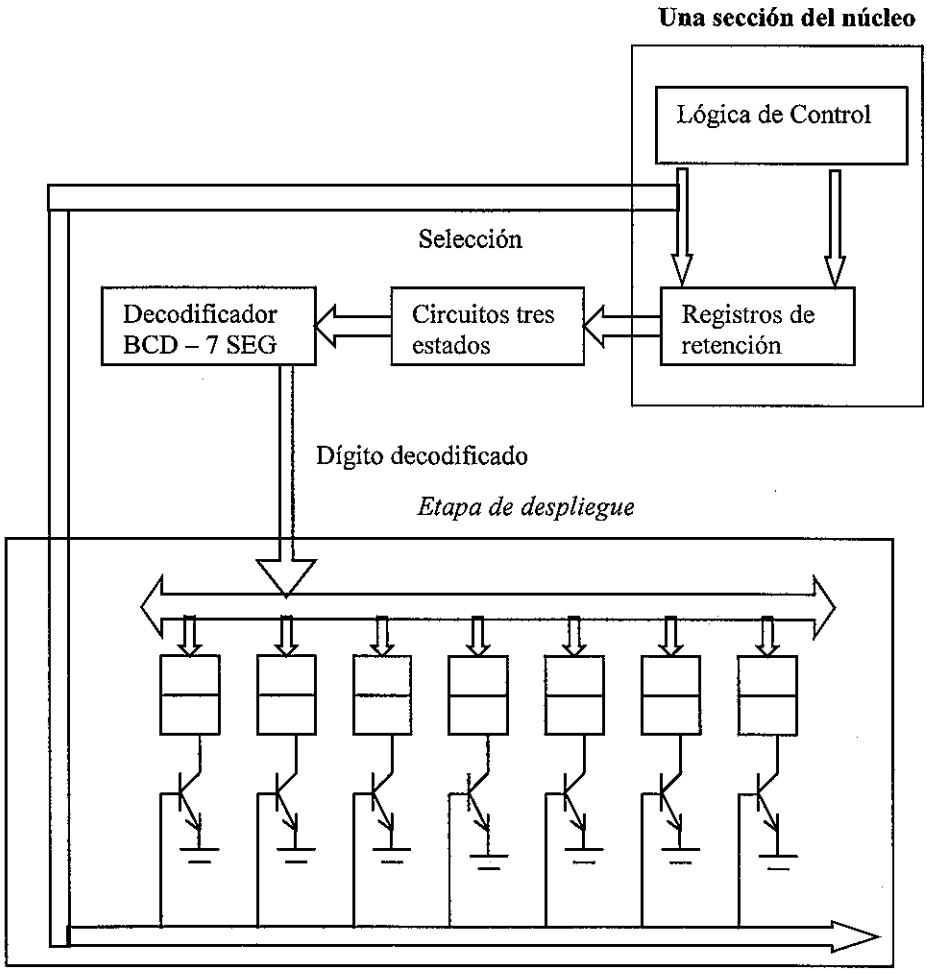


Fig. 3.11. Diagrama a bloques de la interfaz de salida.

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 frecuencímetro es necesario que los siete dígitos 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 dígito BCD convertido a 7 Segmentos. Es conveniente referirse a la Fig. 3.8 para recordar la frecuencia de multiplexión.

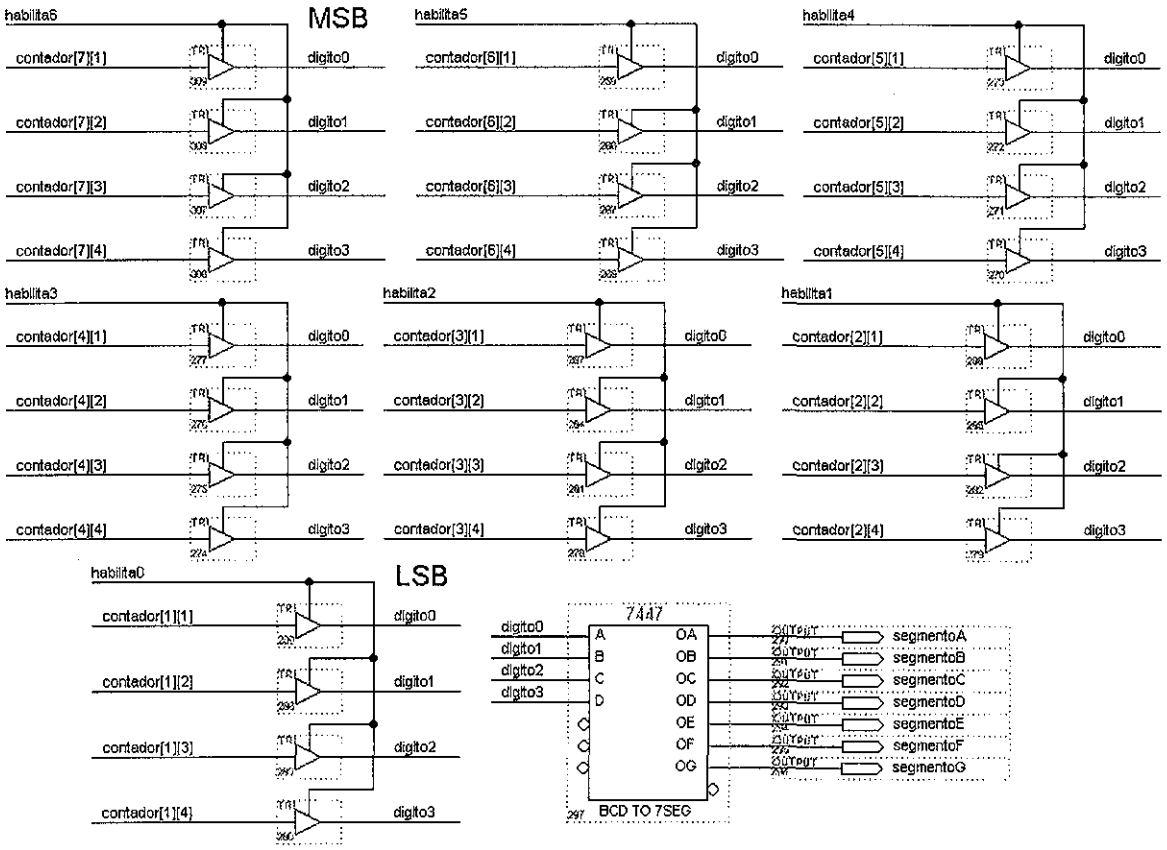


Fig. 3.12 Multiplexado del decodificador BCD – 7 Segmentos.

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 núcleo a través de las salidas "m[6..0]" .

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

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 núcleo 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 frecuencímetro.

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 ningún 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 número 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 capitulo 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 (frecuencímetro) 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 ves 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 frecuencímetro 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 frecuencimetro

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 paro 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 criticas 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 núcleo debe ser simulado sin las interfaces de entrada y salida, las cuales solo aumentarían el tiempo de simulación inútilmente, 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 núcleo estructural del instrumento.

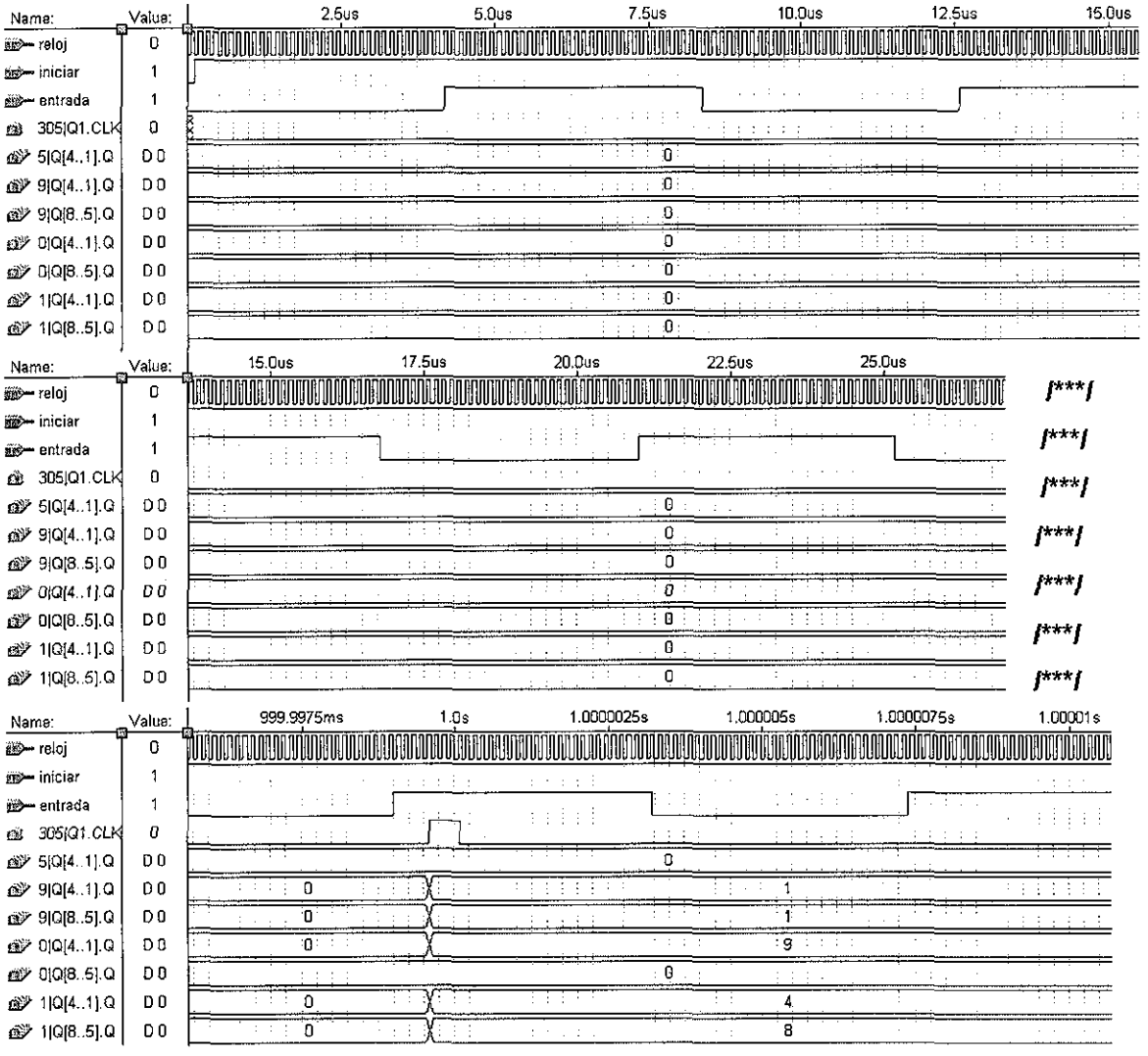


Fig. 3.13 Resultados de la simulación del núcleo estructural.

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 μs 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.

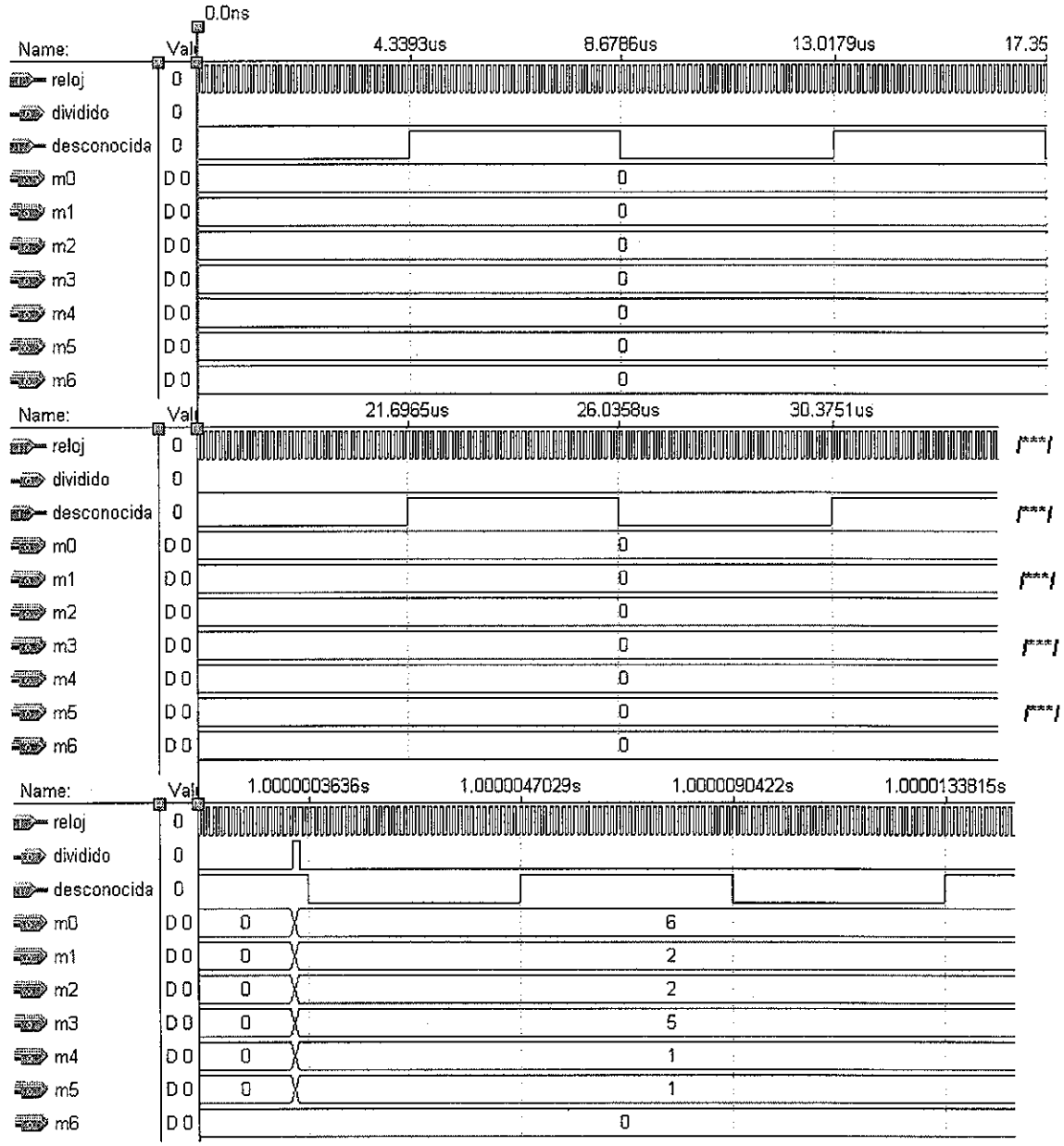


Fig. 3.14 Resultados de la simulación del núcleo funcional.

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 últimos 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 μs por tanto una frecuencia de 115225 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 seria 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.

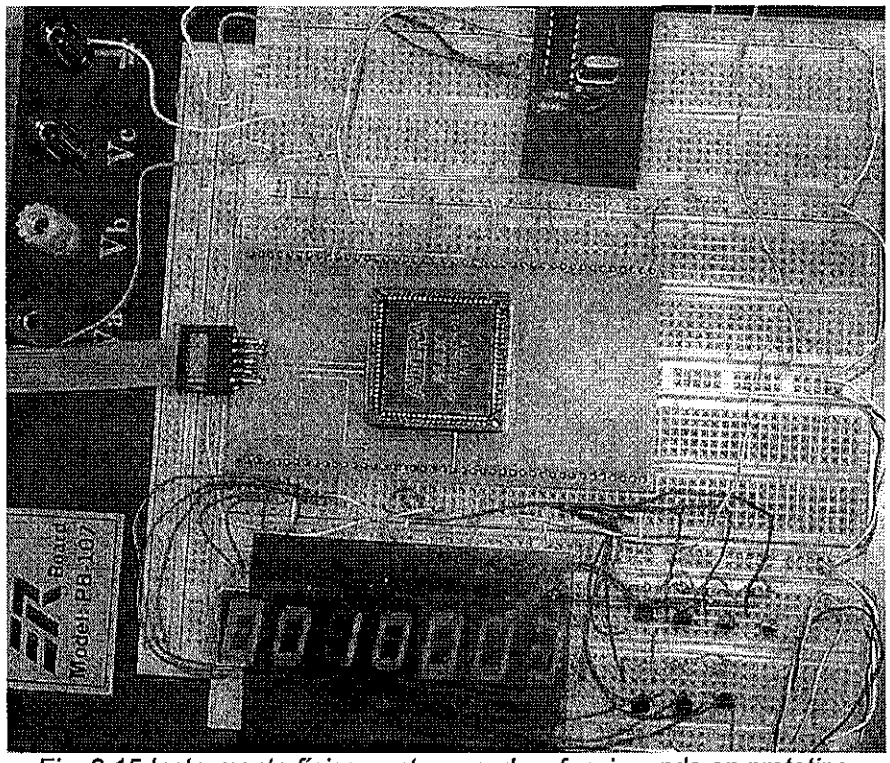


Fig. 3.15 Instrumento físicamente armado y funcionando en prototipo.

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 núcleo 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 únicamente 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 líneas con intersección.

En la Fig. 3.22 se muestra una fotografía de los impresos armados representados en las figuras anteriormente citadas.

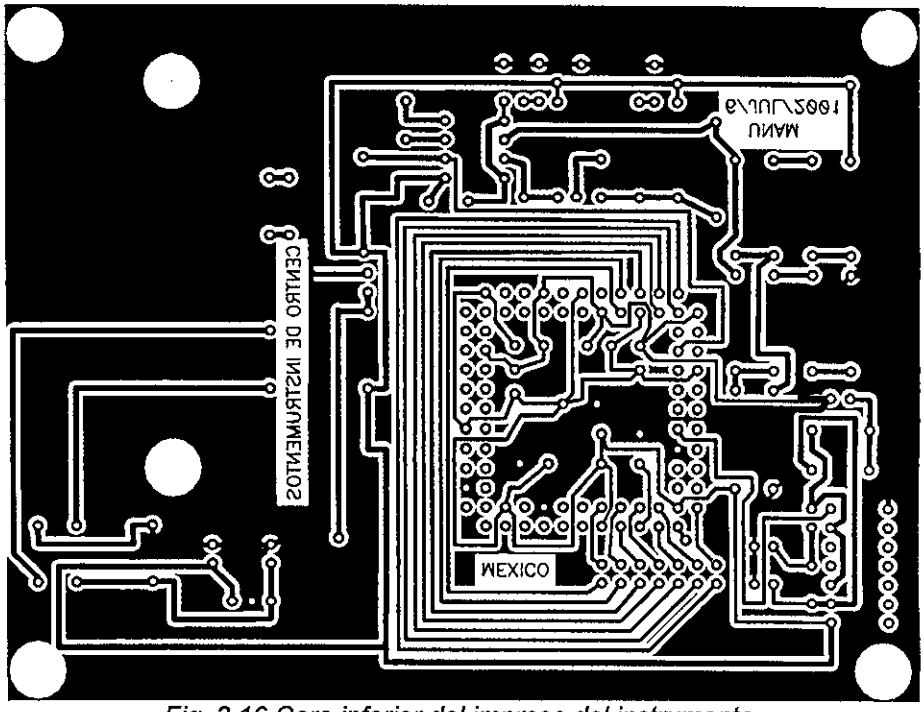


Fig. 3.16 Cara inferior del impreso del instrumento.

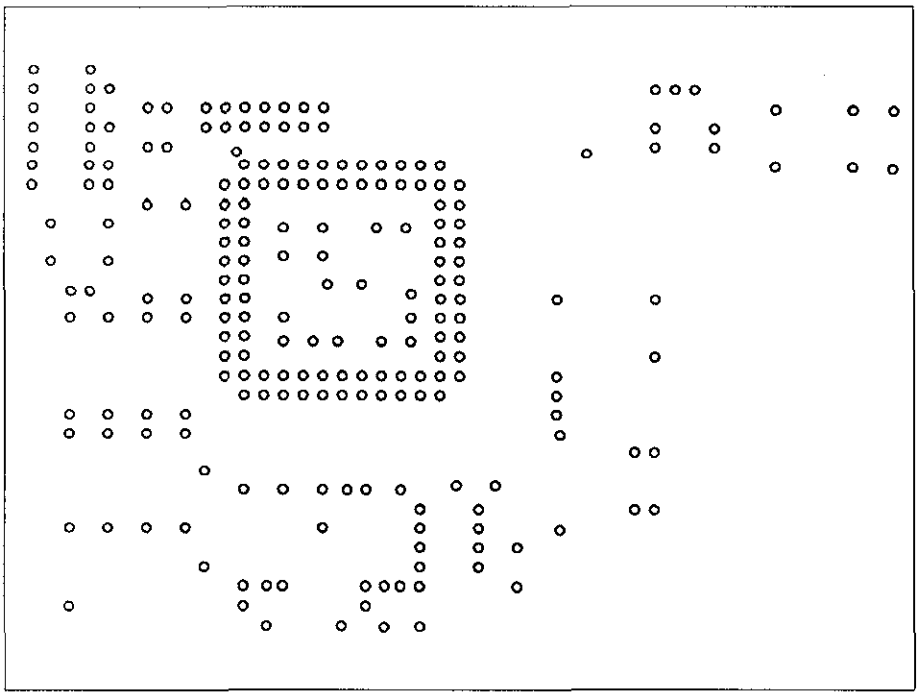


Fig. 3.17 Cara superior del impreso del instrumento.

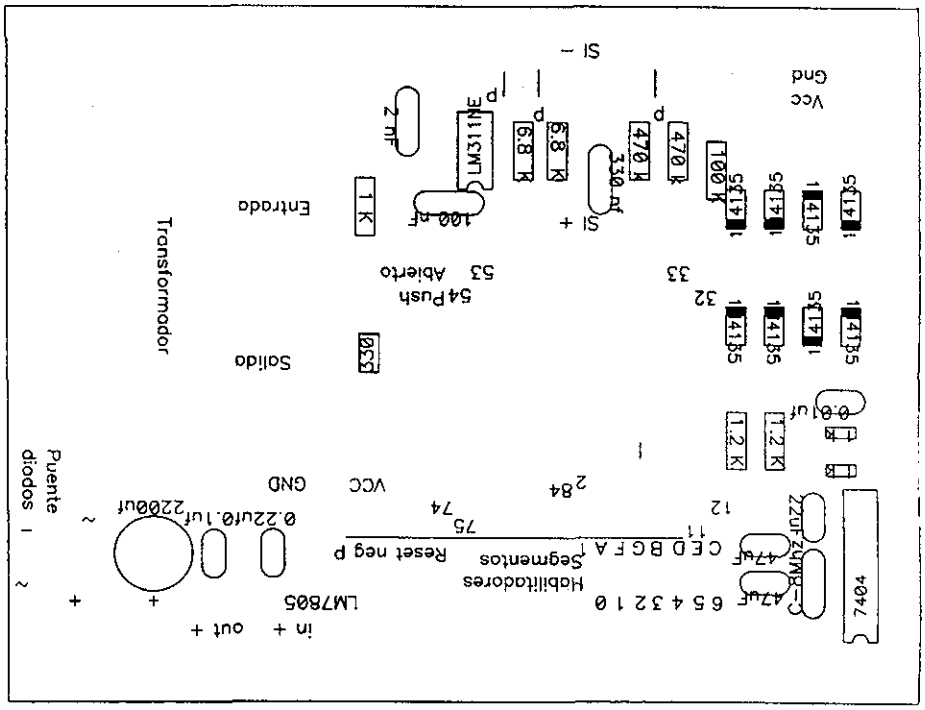


Fig. 3.18 Cara de componentes del impreso del instrumento.

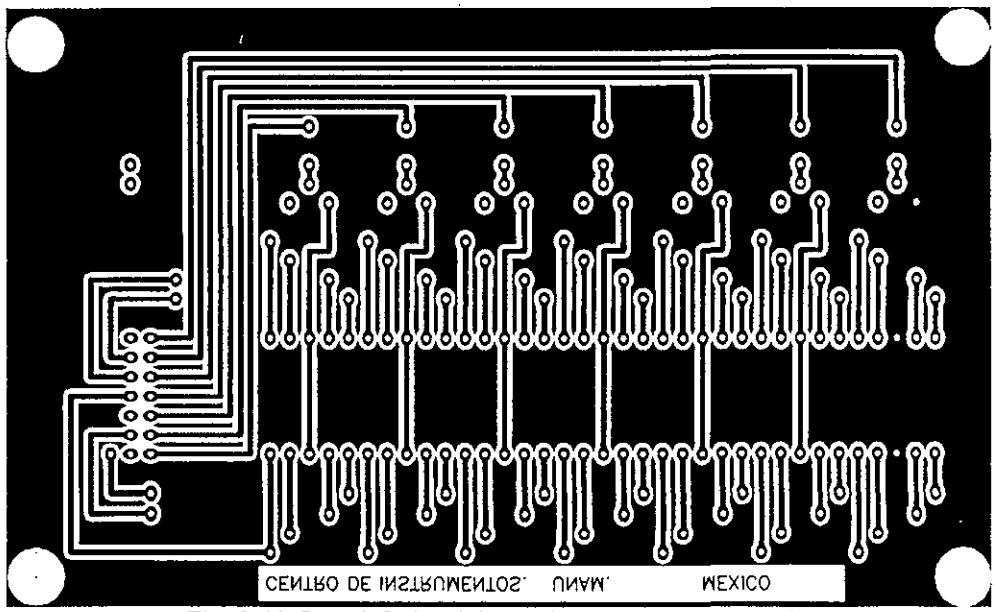


Fig. 3.19 Cara inferior del despliegue del instrumento.

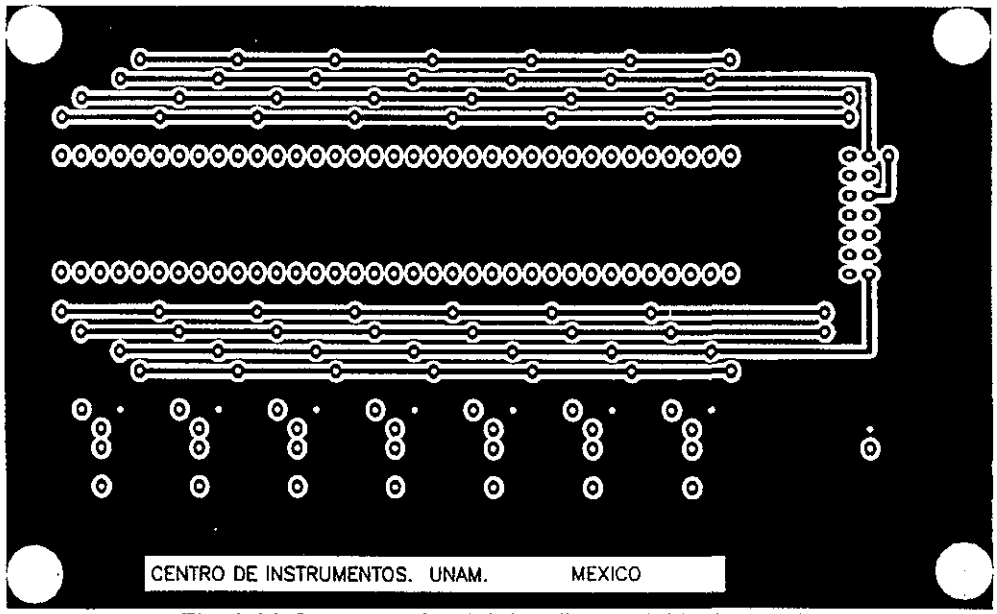


Fig. 3.20 Cara superior del despliegue del instrumento.

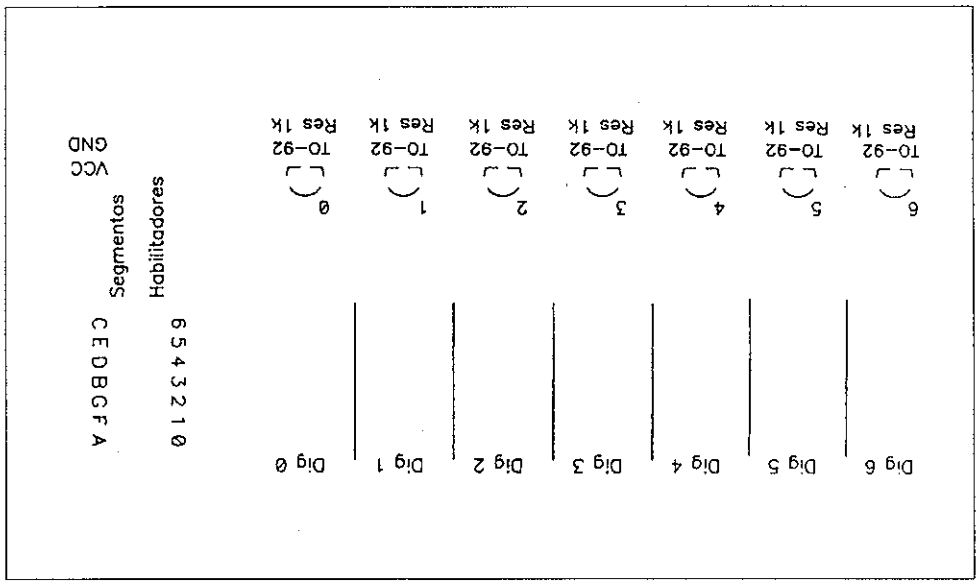


Fig. 3.21 Cara de componentes del despliegue del instrumento.

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	 $10 - 10^{-6}$ seg. $1 - 10^{-6}$ seg.	 $10 + 10^{-6}$ seg. $1 + 10^{-6}$ seg.
Temperatura de operación	0 °C	70 °C

Tabla 3.5 Características técnicas de operación del instrumento.

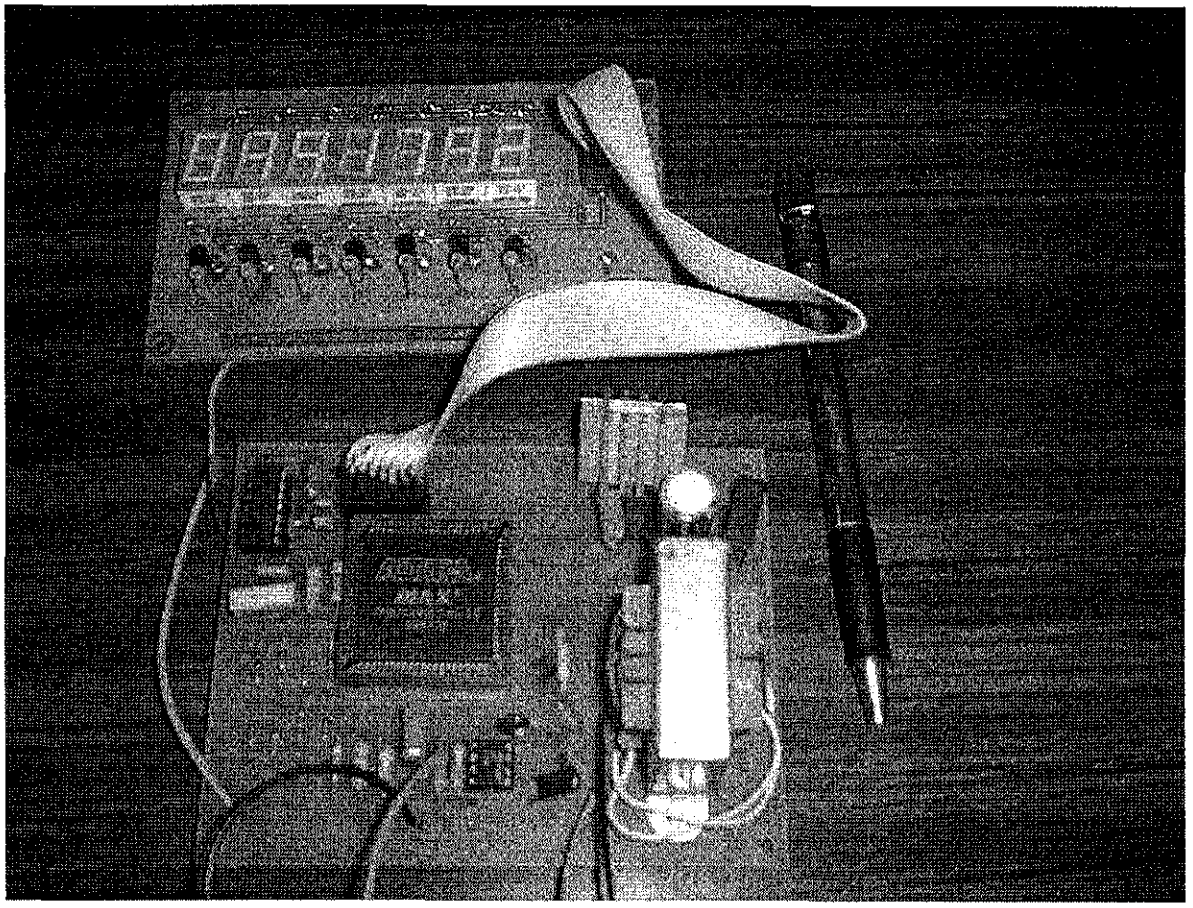


Fig. 3.22 Impresos armados que conforman el instrumento

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.
4.2 Estructura a bloques.
4.3 Controlador de video.
4.4 Controlador VRAM.
4.5 Aplicación al caso de un osciloscopio prototipo.

4.1 IMPORTANCIA Y UTILIDAD.

En este capitulo 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 común 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 útil por las capacidades de despliegue que proporcionará; sin embargo, seria de poca utilidad para un frecuencímetro como el diseñado en el Capitulo 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 píxel 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 línea 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.

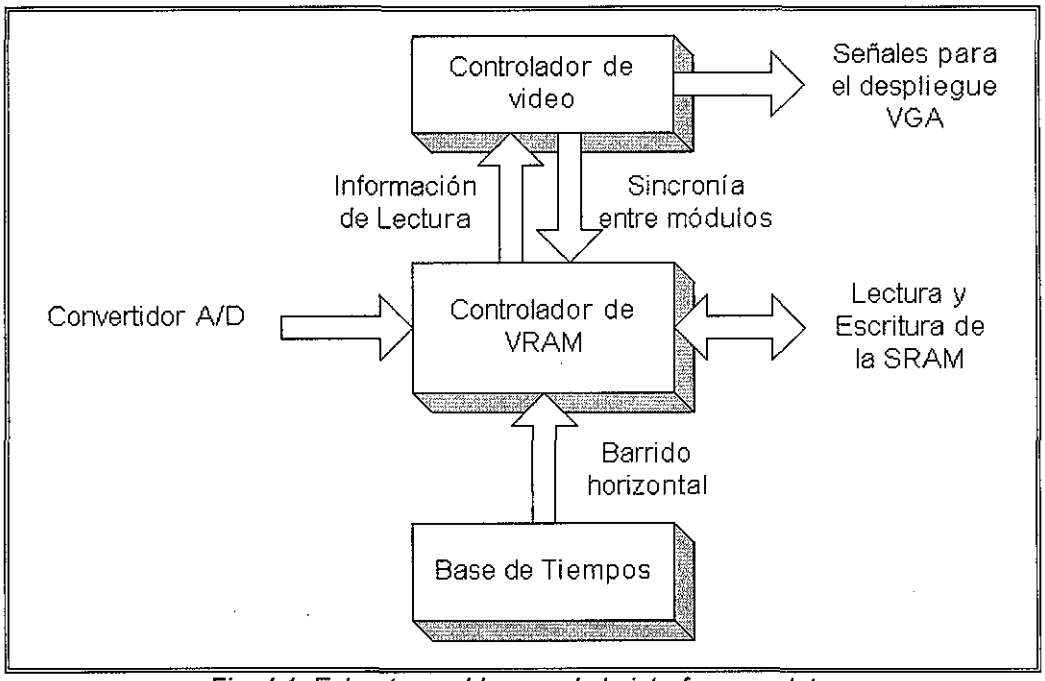


Fig. 4.1. Estructura a bloques de la interfaz completa.

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.

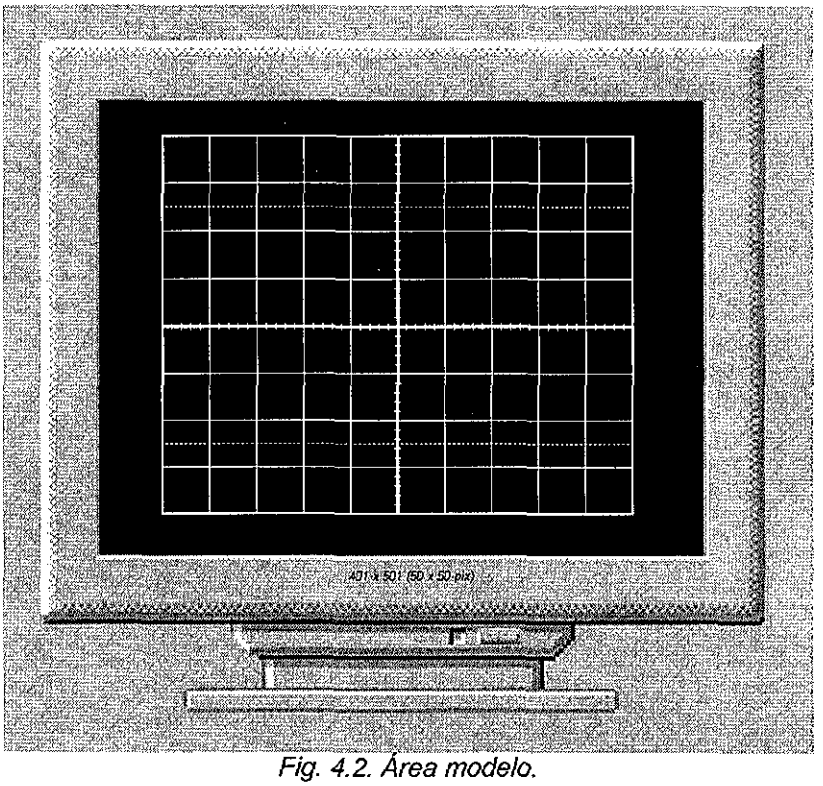


Fig. 4.2. Área modelo.

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 número de columnas compatible con potencia de 2. El hecho de que ambos conceptos coexistan implica que en la memoria de video se almacenan más datos de los que se desplegarán en la pantalla, pero redituá 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* únicamente cuando el cañón electrónico apunta a un pixel dentro del área de trabajo.
- *Pixcero*. Destinada también al submódulo "bufferpri". Sincroniza el inicio de lecturas en la memoria de video con el inicio del barrido izquierda-derecha en el primer renglón del área de trabajo por parte del cañón electrónico del monitor.
- *Pulsescr*. Señal empleada por el módulo *Controlador de VRAM*, que indica los intervalos de tiempo en los que el cañón electrónico del monitor se encuentra fuera del área de trabajo de la pantalla.

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 *Pulsescr* es una de ellas y tiene la función de indicar al *Controlador de VRAM* que no se requiere más información de la memoria de video. La importancia de ésta señal está dada por la definición de un tiempo que denominaremos *"de descanso"* a lo largo de éste capítulo, que no es mas que el tiempo de espera entre renglones 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 píxel, 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 número 699, esto es, unos pocos cientos de nanosegundos antes de que el cañón electrónico comience a trazar el primer pixel del renglón siguiente. Pero cuando *V_cont* se encuentra en el conteo máximo se reinicia en el renglón '0'.

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

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

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

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

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

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

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 último pixel del área de trabajo del monitor, situado en su esquina inferior derecha. Con lo anterior queremos establecer que la matriz de pixeles que forman una imagen en el monitor y que es trazada de izquierda a derecha renglón por renglón, mantiene una correspondencia uno-a-uno con el conjunto de bits que conforman las primeras 25664 localidades de la memoria de video y que deben ser "leídas" desde el bit menos significativo hasta el más significativo localidad por localidad. Para expresarlo más claramente, podemos decir que la información de los primeros 8 pixeles que deben ser actualizados por el cañón electrónico del monitor está almacenada en la primera localidad de la 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 píxel 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.

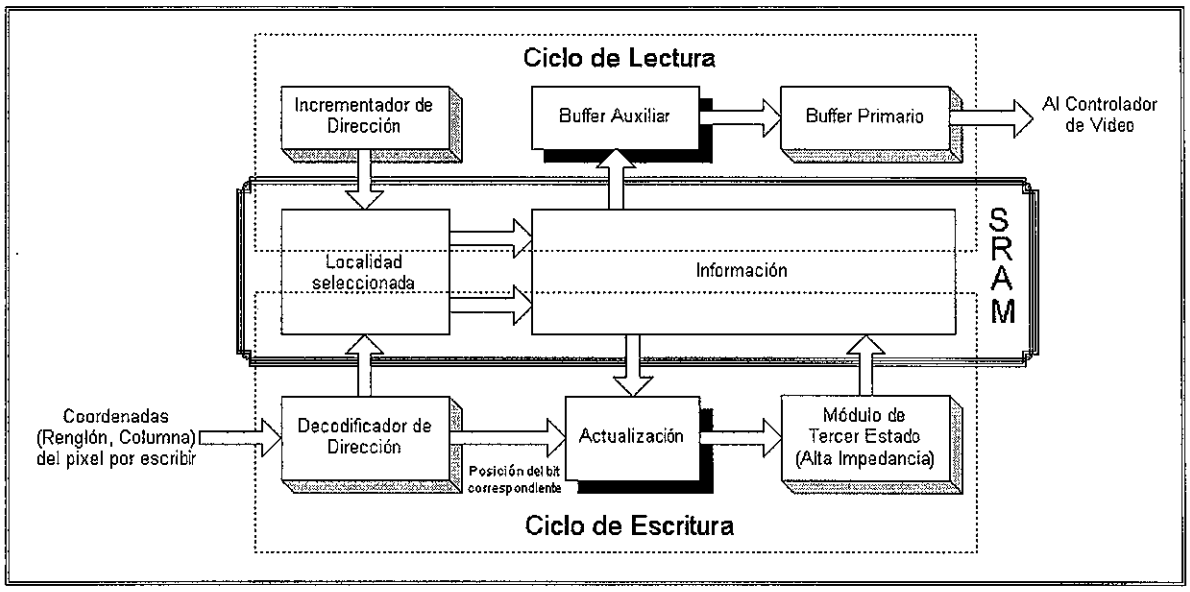


Fig. 4.3. Diagrama de bloques del módulo Controlador de VRAM

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.

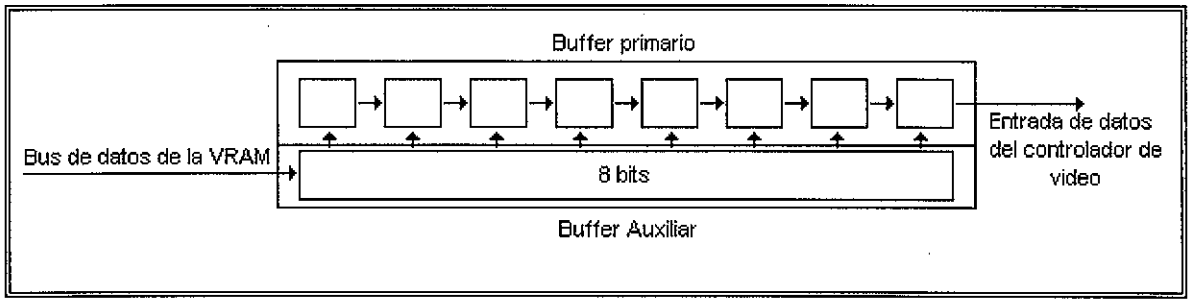


Fig. 4.4 Esquema de transferencia del buffer.

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 pixeles se encuentren en estado "apagado"; tras lo cual se da inicio al ciclo de Lectura. Durante el ciclo de Lectura, el controlador extrae la información almacenada en el byte de la 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 periodo "de descanso", es decir, cuando el cañón electrónico no apunta a ningún pixel dentro del área de trabajo. De ser así, el controlador entra al ciclo de Escritura, almacenando información en la SRAM hasta que el periodo "de descanso" llega a su fin. Cabe hacer notar que, si queremos actualizar la información de un solo pixel, debemos "leer" la información completa del byte de la SRAM al que pertenece, encender (o apagar) el bit correspondiente y escribir el byte de vuelta a la memoria.

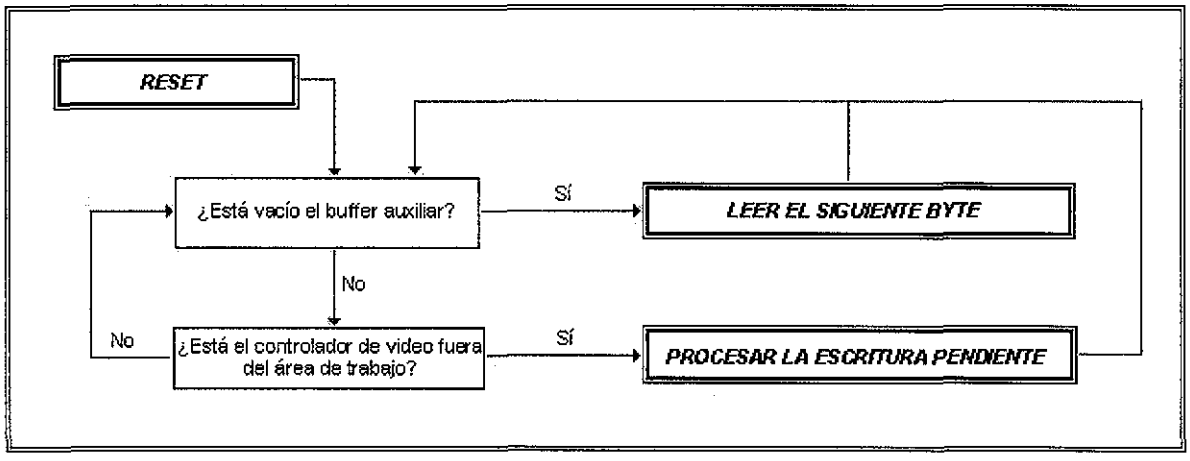


Fig. 4.5 Diagrama de flujo del Controlador VRAM.

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 C*ontrolador de Video* la información sobre el pixel que se está actualizando en pantalla. Esa información se encuentra en lo que llamamos buffer primario, implementado en éste componente como variable interna de la arquitectura con el nombre de *Prim*. El controlador de video proporciona una señal de reloj (*Pulsolec*) a 25.175 MHz, pero únicamente cuando el cañón electrónico del monitor envía su haz de electrones a un pixel correspondiente a un bit dentro de la matriz de datos del mapa de memoria. De este modo, por cada renglón se reciben 512 pulsos a la entrada *Pulsolec*, los que sirven para que éste componente le devuelva la información del buffer primario bit por bit a esa frecuencia, hasta quedar vacío. Cuando esto sucede, simplemente se toma la información presente en la entrada denominada *Auxiliar*, 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, Actuabuf, Bandebuf, Enviaescr: std_logic;
        signal Estado: std_logic_vector(2 downto 0);
        signal Direcactual, Almacedirec, Direcsig: std_logic_vector(14 downto 0);
        signal Bufaux, Byteant, Bytenue: std_logic_vector(7 downto 0);
        signal Posdeco, Almacepos: std_logic_vector(2 downto 0);
        signal Direcdeco: std_logic_vector(14 downto 0);
        component triest
```

```
            port(Entrada: in std_logic_vector(7 downto 0);
                Conectar: in std_logic;
                Salida: out std_logic_vector(7 downto 0));
        end component;
        component bufferpri
            port(Rst, Pulsolec, Pixcero: in std_logic;
                Auxiliar: in std_logic_vector(7 downto 0);
                Video, Actualiza: out std_logic);
        end component;
        component decodirec
            port(Renglon: in std_logic_vector(8 downto 0);
                Columna: in std_logic_vector(8 downto 0);
                Direccion: out std_logic_vector(14 downto 0);
                Posicion: out std_logic_vector(2 downto 0));
        end component;
        component incredirec
            port(Actual: in std_logic_vector(14 downto 0);
                Siguiente: out std_logic_vector(14 downto 0));
        end component;
        constant Direcmax: std_logic_vector(14 downto 0):= conv_std_logic_vector(25663, 15);

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 <= "000000000000000";
                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
                                        Almacedirec <= 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 vramarq;
```

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 vacía la localidad de memoria H00. Al siguiente pulso de reloj se inhabilita la escritura, se desconecta la salida y se incrementa la dirección. Este ciclo se realiza para las 25664 direcciones empleadas, durante un tiempo de 4.078 ms (25664 / 12.5875 MHz x 2 periodos). Una vez alcanzada la dirección máxima, se procede al estado que revisa el buffer.

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.

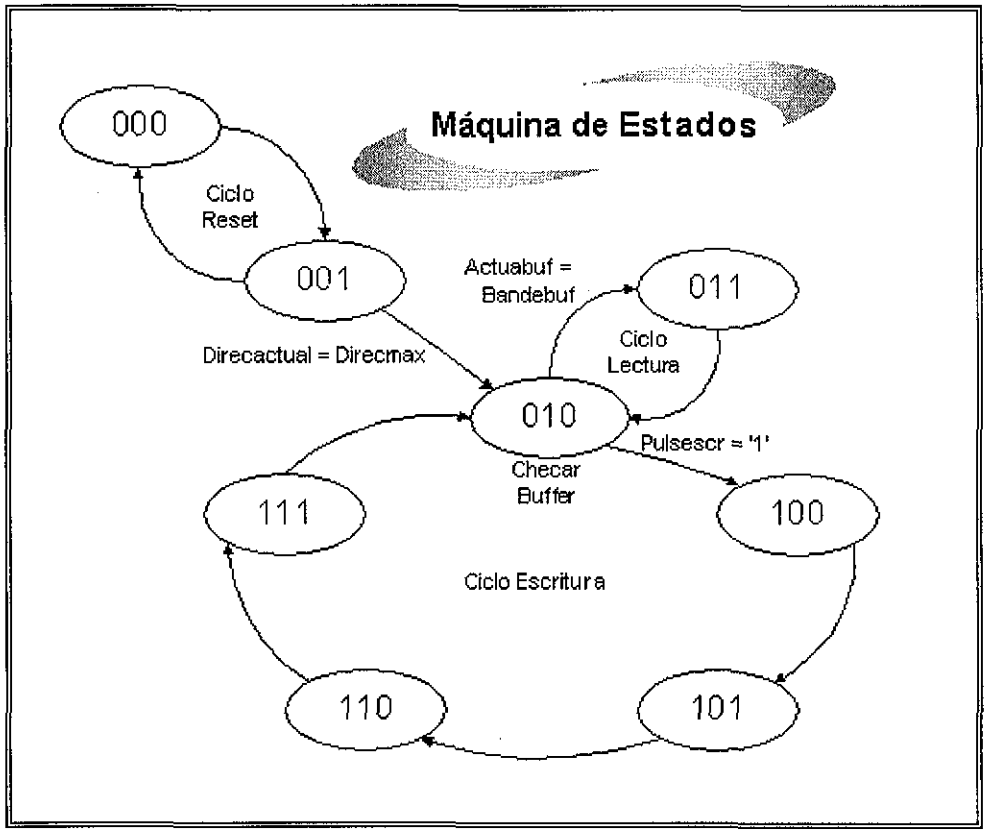


Fig. 4.6 Máquina de Estados que muestra los ciclos Reset, Lectura y Escritura del módulo Controlador de VRAM.

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 último 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 Capítulo 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 último también forma parte de este mismo bloque y es necesario para proporcionar información al bloque de electrónica digital en un formato que éste pueda manipular.

- 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.

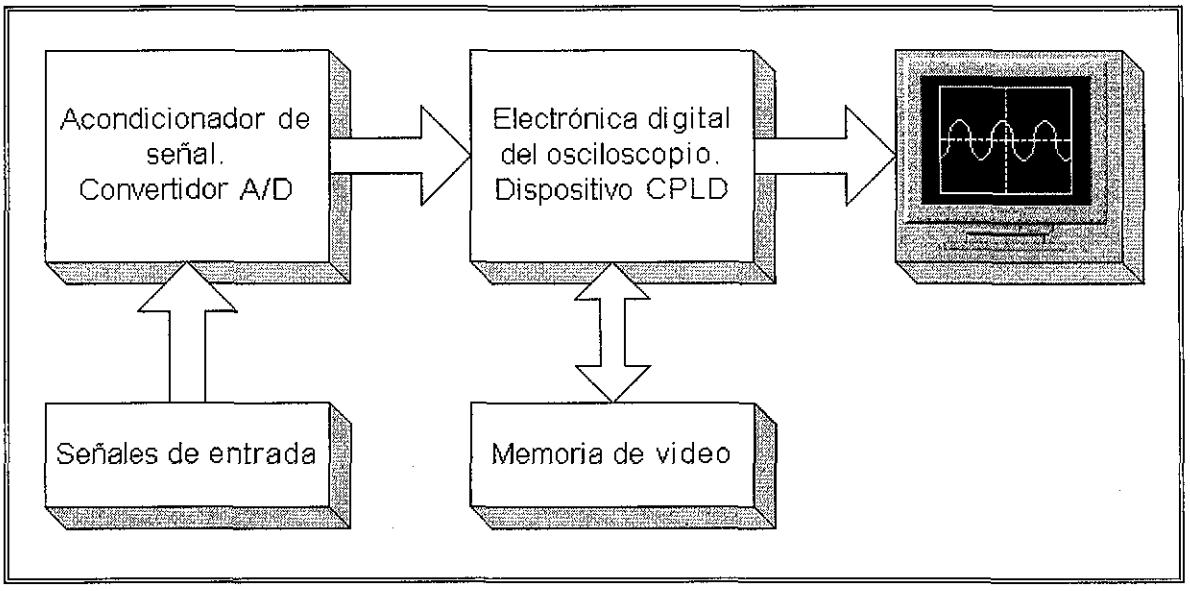


Fig. 4.7 Diagrama a bloques del osciloscopio digital.

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 Esctiemp, Ref, Congela: in std_logic;
            signal Reloj, Rst: in std_logic;
            sram_we, sram_oe: out std_logic;
            sram_direc: out std_logic_vector(14 downto 0);
            signal Rojo, Verde, Azul : out std_logic;
            signal Horiz_sinc, Vert_sinc : out std_logic);
            sram_byte: inout std_logic_vector(7 downto 0);
end osciloscopio;

architecture arq of osciloscopio is
                signal Medio: std_logic;
                signal VRrst, VRnivelpix: std_logic;
                signal VRren, Increren: std_logic_vector(8 downto 0);
                signal Limpia: std_logic;
                signal Selector: std_logic_vector(3 downto 0);
                signal Retardo: std_logic_vector(1 downto 0);
                signal Fincont: std_logic;

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

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

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

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

        vram:   control_vram port map (VRrst, Medio, Pulsolec, Pulsescr, Pixcero, VRnivelpix,
                VRren, Col, Video, sram_we, sram_oe, sram_direc, sram_byte);
        osc:    tiempos port map (Reloj, Rst, Congela, Selector, Limpia, Col);

        VRrst <= Limpia or Rst;
        VRren <= not ((Conver + Increren) + conv_std_logic_vector(183,9));
        VRnivelpix <= '1';

        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 Esctiemp = '0' then
                                if Fincont = '0' then
                                        if Retardo = conv_std_logic_vector(3,2) then
                                                Retardo <= conv_std_logic_vector(0,2);
                                                Fincont <= '1';
                                                if Selector = "1010" then
                                                        Selector <= "0000";
                                                else
                                                        Selector <= Selector + "0001";
                                                end if;
                                        else
                                                Retardo <= Retardo + '1';
                                        end if;
                                end if;
                        else
                                Retardo <= conv_std_logic_vector(0,2);
                                Fincont <= '0';
                        end if;
                end if;
        end process;


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

```
        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.

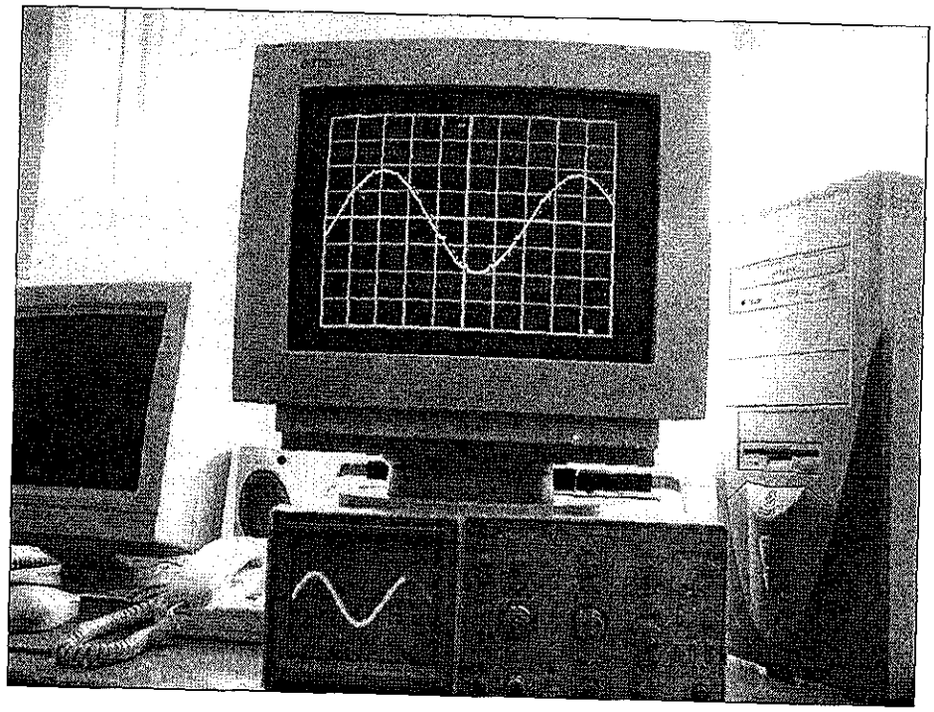


Fig. 4.8 Señal sinusoidal de 4 Vpp a 14.7 Hz, registrada por un osciloscopio comercial y por el prototipo diseñado.

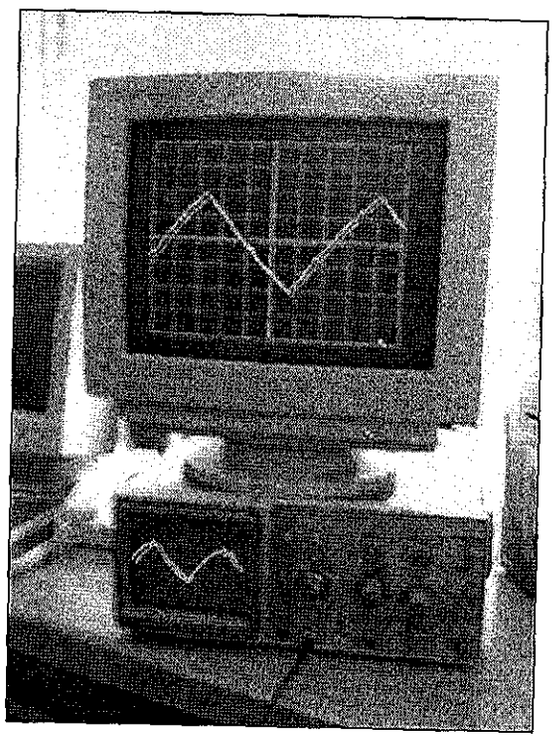


Fig. 4.9 Señal triangular

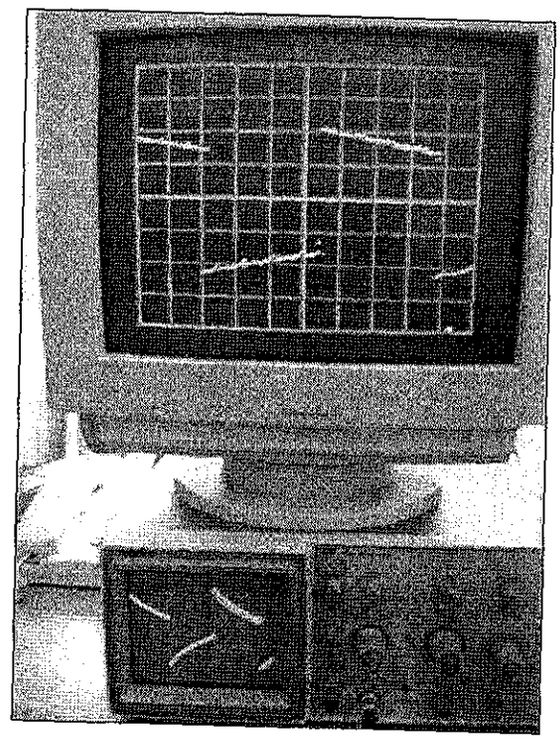


Fig. 4.10 Señal Cuadrada

CAPÍTULO 5

GENERADOR DIGITAL DE TREN DE PULSOS PROGRAMABLE

CONTENIDO

5.1 Introducción.
5.2 Diseño de un generador digital de tren de pulsos programable en frecuencia y ancho de pulso.
5.3 Resultados.

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".

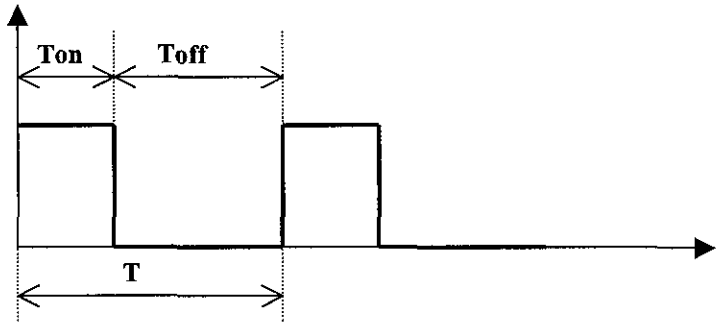


Fig. 5.1 Parámetros programables del tren de pulsos.

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.

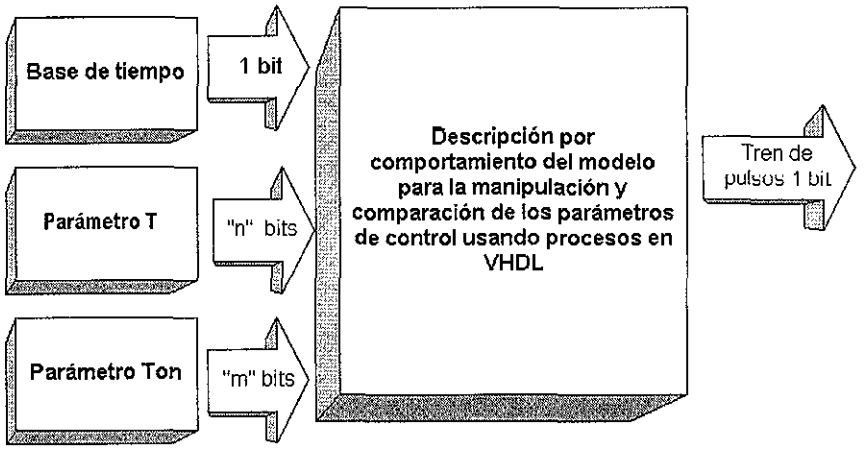


Fig. 5.2. Diagrama a bloques del generador de tren de pulsos.

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.

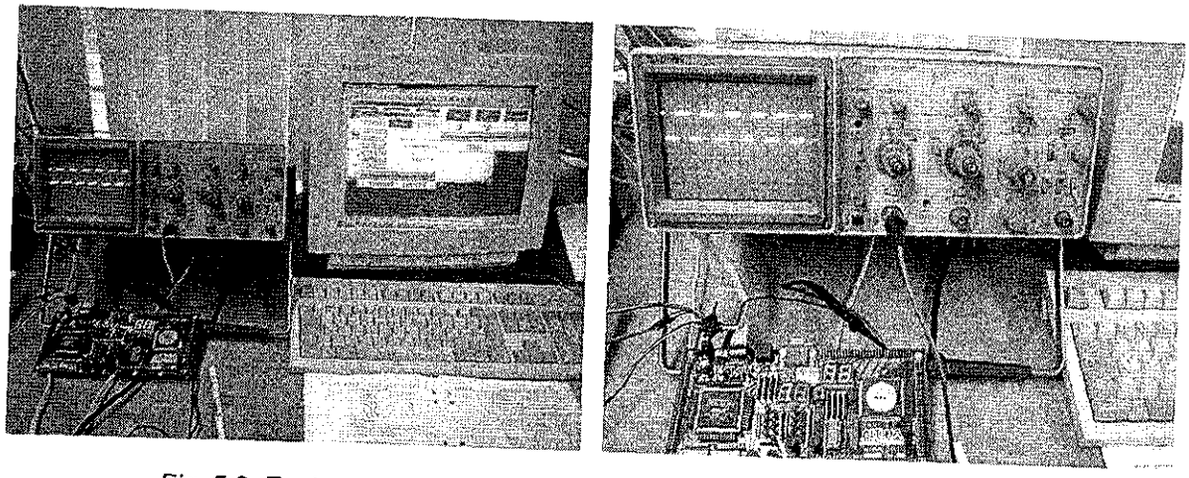


Fig. 5.3. Equipo completo después de programar la UP1 con el generador

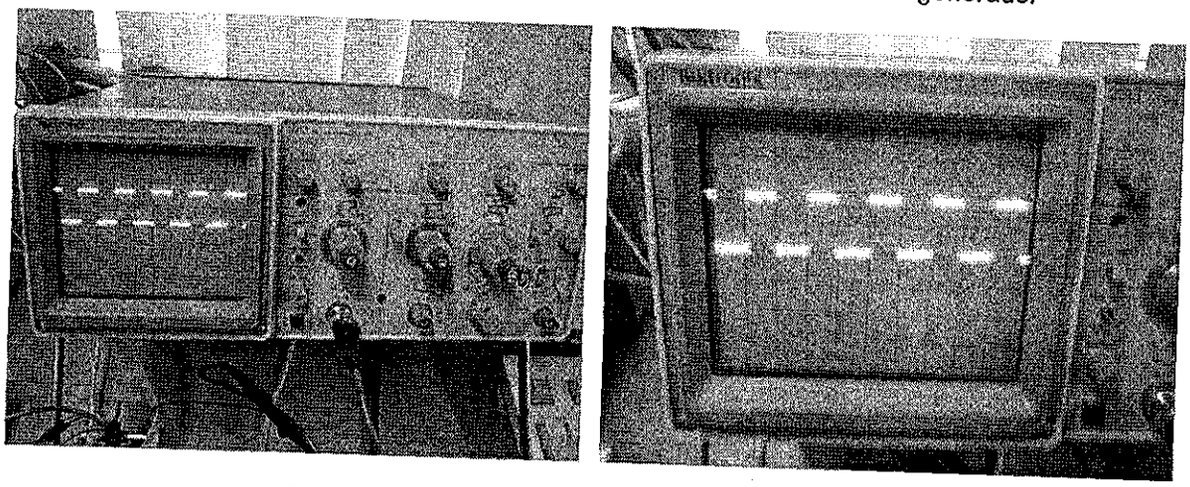


Fig. 5.4. Tren de pulsos con ciclo de trabajo del 50 %

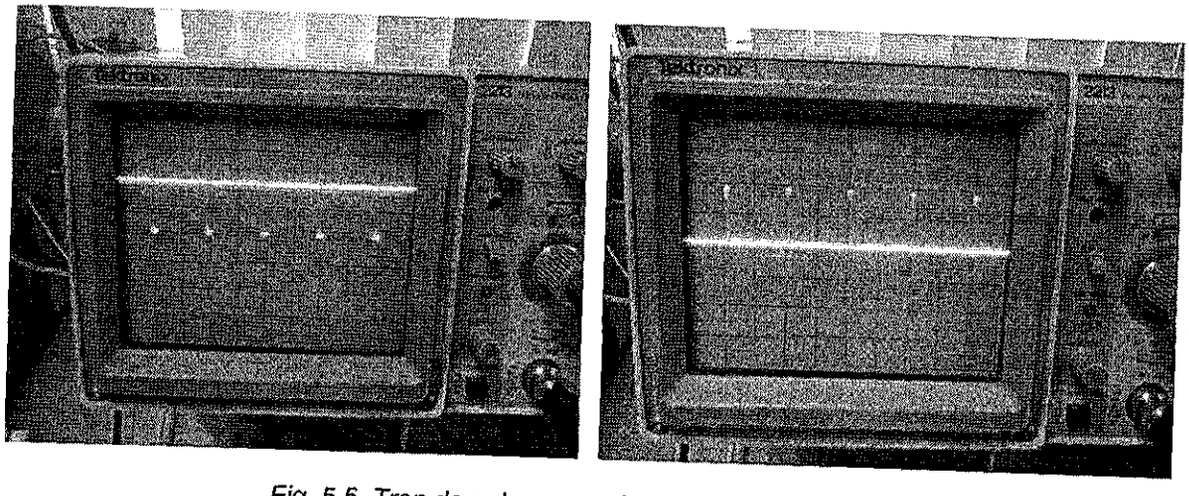


Fig. 5.5. Tren de pulsos con ciclos de trabajo extremos

CAPÍTULO 6

CONCLUSIONES Y PERSPECTIVAS FUTURAS

CONTENIDO

6.1 Ventajas de la Metodología.
6.2 Aplicabilidad de la Metodología mas allá de la instrumentación.
6.3 Desventajas de la metodología.
6.4 Instrumentos propuestos para la aplicabilidad directa de la metodología.
6.5 Trabajo futuro para enriquecer la metodología.
6.6 Módulos aportados por el presente trabajo para la librería de funciones de la metodología.
6.7 Material producto de este trabajo y de trabajos de terceros en la misma área.
6.8 Ciclo de desarrollo entre el diseño por comportamiento y el diseño estructural.
6.9 Costo del prototipo frecuencímetro.

6.1 VENTAJAS DE LA METODOLOGÍA.

Como lo hemos venido exponiendo a lo largo de este trabajo, el objetivo de la metodología es reducir el ciclo de desarrollo e implementación de instrumentación digital utilizando la tecnología de los dispositivos lógicos programables complejos (CPLDs).

La tecnología de los dispositivos lógicos programables es aplicable al diseño de cualquier dispositivo digital, es decir en áreas como DSP, Control Digital, Instrumentación, Comunicaciones, etc. Esta es la razón por la cual una metodología de uso universal es poco eficiente y es preferible tener una metodología orientada específicamente a cada una de las áreas de aplicación.

La metodología que proponemos tiene la siguientes ventajas que son claramente extraíbles durante el desarrollo del trabajo:

- Está orientada a la implantación de instrumentación digital utilizando Dispositivos Lógicos Programables Complejos.

- Subdivide cualquier tipo de instrumento digital en tres partes, la primera llamada "núcleo" el cual en si mismo es el instrumento de medición, la segunda llamada "interfaz de entrada" y la tercera la "interfaz de salida".

- Las interfaces de entrada y de salida pueden ser de diferentes tipos, por ejemplo multiplexadas o no multiplexadas etc., sin embargo pueden ser las mismas entre diferentes instrumentos. Esto implica que los diseños de las interfaces se pueden reutilizar.

- La metodología por tanto plantea como parte medular el desarrollo del "núcleo" que define por así decirlo el nombre del instrumento.

- Las descripciones del hardware de los tres grandes componentes de un instrumento digital se plantean utilizando descripción por comportamiento. Esto permite minimizar el tiempo de diseño, así como facilitar el proceso de depuración hasta lograr el funcionamiento esperado.

- La simulación inherente a esta tecnología es optimizada al utilizar descripción por comportamiento, debido a que los cambios necesarios implican modificación de código fuente como en los lenguajes de programación.

- La descripción del comportamiento se propone utilizando lenguajes de descripción de hardware estandarizados como VHDL o Verilog HDL, por que esto permite que un diseño pueda ser migrado con mayor facilidad a otras plataformas diferentes a ALTERA, como pueden ser XILINX etc.

- El empleo de descripción por comportamiento minimiza el ciclo de desarrollo de un instrumento.

6.2 APLICABILIDAD DE LA METODOLOGÍA MAS ALLÁ DE LA INSTRUMENTACIÓN.

Hemos dicho con especial énfasis que esta metodología está orientada a la instrumentación digital; sin embargo, existen elementos importantes que pueden ser considerados para el diseño de cualquier sistema digital como son:

- Descripción por comportamiento.
- Elección de un tipo de dispositivo CPLD.
- Estrategias de simulación.
- Re utilización de código en especial para interfaces de entrada y salida.

Nosotros sugerimos que esta metodología con variaciones pertinentes al área de aplicación específica puede ser explotada para obtener beneficios en cuanto a tiempo de desarrollo y costos.

6.3 DESVENTAJAS DE LA METODOLOGÍA.

La metodología aquí desarrollada tiene ciertos aspectos que la ponen en desventaja al desarrollar instrumentación digital. Desventajas aparentes al inicio, se dan sobre todo para usuarios con poca experiencia en la utilización de los dispositivos lógicos programables. Esto es debido a los siguientes aspectos:

- La descripción por comportamiento en algún lenguaje HDL, plantea la necesidad de conocer la sintaxis y aplicación de algún HDL. Esto en un principio puede resultar poco atractivo para un usuario que aún teniendo experiencia en instrumentación, no conozca los lenguajes de descripción de hardware; debido al tiempo que será necesario invertirle para su aprendizaje.

- La inercia de los usuarios al cambio y a aprender una nueva tecnología; sin embargo es importante hacer notar que es una inversión que da frutos muy rápidamente.

- Los usuarios prefieren una descripción gráfica por que esto no requiere de una inversión inicial en cuanto al aprendizaje.

- Una descripción por comportamiento es de alto nivel y puede impedir hacer optimizaciones que sean factibles trabajando a nivel estructural; sin embargo una experiencia considerable en el lenguaje utilizado y la forma en que se sintetiza en la tecnología CPLD puede casi no requerir de optimizaciones adicionales.

- El software utilizado por los ambientes de desarrollo, así como la tecnología de los CPLDs está en constante actualización, lo cual implica una revisión constante de cualquier metodología para explotar al máximo los recursos tecnológicos.

6.4 INSTRUMENTOS PROPUESTOS PARA LA APLICABILIDAD DIRECTA DE LA METODOLOGÍA.

Como parte de la continuación directa de este trabajo y de las necesidades en cuanto a instrumentación digital requerida en proyectos desarrollados en el Centro de Instrumentos de la UNAM, tenemos a bien proponer los siguientes instrumentos con una breve justificación.

Osciloscopio en tarjeta PCI de bajo costo para utilizarse con una PC convencional. Este instrumento pretende utilizar una PC como un osciloscopio de buen ancho de banda y bajo costo proporcionando las características más importantes de un osciloscopio convencional. Las ventajas son vistas a nivel didáctico y de equipamiento en los laboratorios, ya que por un costo razonable dispondríamos de un osciloscopio de calidad en gran número de computadoras, por ejemplo, de nuestra Facultad de Ingeniería.

Medidor de presión sanguínea de bajo costo. Medidor digital de presión sanguínea de muy fácil utilización y de bajo costo que pueda ser adquirido con facilidad por centros de salud y por pacientes que requieran monitorear constantemente este signo vital. Cabe hacer notar que existe un avance en esta propuesta [4].

Cronómetro digital. Instrumento para aplicarse en la realización de experimentos de física que requieran la medición de tiempos entre eventos disparados por diversos sensores. Cabe hacer notar que existe un avance en esta propuesta [5].

Medidor digital LCR. Instrumento de uso didáctico para los estudiantes que permita medir con confianza los parámetros de la impedancia.

Multímetro digital. Instrumento didáctico y de bajo costo que permita medir corriente, voltaje, resistencia, frecuencia, capacitancia y temperatura.

Generador digital de funciones. Instrumento didáctico y de bajo costo que permita generar las formas de onda más comunes y utilizadas en las practicas de laboratorio.

6.5 TRABAJO FUTURO PARA ENRIQUECER LA METODOLOGÍA.

Una metodología de este tipo es bastante ambiciosa y susceptible de modificaciones y trabajo adicional de manera constante para lograr que sea de aplicabilidad exitosa en el área de la instrumentación. Dentro de los aspectos de trabajo futuro que inmediatamente se sugieren son los siguientes:

- Crear una librería de módulos descritos funcionalmente y parametrizables para algunas de las posibles interfaces de entrada y salida para la instrumentación. El objetivo es que puedan ser incrustados directamente en el código del núcleo del instrumento.

- Diseñar estrategias de optimización al utilizar descripción por comportamiento que permitan explotar algunas de las virtudes del diseño estructural y en general de los diseños de bajo nivel.

- Diseñar una herramienta de software que defina un protocolo de comunicación entre las interfaces de entrada / salida con respecto al núcleo, la cual permita unir automáticamente el diseño del núcleo creado por el usuario.

- Diseñar una librería que incluya módulos descritos funcionalmente que contengan funciones especializadas útiles en la instrumentación digital como son divisores de frecuencia que generen bases de tiempo, algoritmos de corrección de errores, algoritmos de operaciones matemáticas especiales etc.

6.6 MÓDULOS APORTADOS POR EL PRESENTE TRABAJO PARA LA LIBRERÍA DE FUNCIONES DE LA METODOLOGÍA.

La metodología que ha sido presentada en este trabajo, además de recomendar el método apropiado para el diseño de instrumentación digital, aporta una librería de funciones útiles en el diseño de instrumentación; actualmente esta librería está conformada por dos funciones muy útiles:

- *Interfaz de salida gráfica VGA.* Esta función permite brindar capacidades de despliegue grafico VGA a cualquier instrumento que lo requiera para mostrar los resultados de una medición, como fue el caso del osciloscopio prototipo mostrada en el Capítulo 4. Este módulo es valioso ya que el diseñador de un instrumento no requiere invertir tiempo en el diseño de esta interfaz, por el contrario solo requiere dedicarse al diseño del núcleo del instrumento en cuestión.

- *Generador digital de tren de pulsos programable.* Esta función permite generar un tren de pulsos que puede ser programado digitalmente tanto en frecuencia como en ancho de pulso. Los instrumento requieren ya sea de una base de tiempos o de señales TTL con ciertas características en frecuencia y ciclo de trabajo para realizar sus propios procesos, de forma que este módulo es también muy útil; incluso este tipo de función puede ser utilizada para realizar ciertas acciones de control si es el caso del instrumentos que se este diseñando.

Las anteriores son solamente dos de las funciones aportadas por el presente trabajo, pero como se mencionó en la sección anterior, esta librería es susceptible de crecer aportando nuevas funciones, de forma tal de enriquecer la metodología.

6.7 MATERIAL PRODUCTO DE ESTE TRABAJO EN COLABORACIÓN CON TERCEROS EN LA MISMA ÁREA.

PRIMERO.

Parte del esfuerzo realizado en el desarrollo de esta metodología, así como de la participación de terceros con experiencia en el uso de los dispositivos lógicos programables ha dado lugar a la publicación de cuatro manuales en la Facultad de Ingeniería cuyas referencias se dan en [1], [2], [6] y [7] respectivamente. Los títulos de los manuales son:

- **Manual de consulta entorno de diseño MAX + PLUS II.**
- **Lenguaje de Descripción de Hardware Verilog.**
- **Lenguaje de Descripción de Hardware VHDL.**
- **Prácticas de Laboratorio en el Entorno de Diseño MAX+PLUS II.**

SEGUNDO.

La aportación de este trabajo ha dado lugar a varios artículos uno de ellos internacional, los cuales mencionamos a continuación y cuyas referencias se encuentran en [46], [47] y [48] respectivamente.

- "***Development of a Design Methodology for Digital Measurement Instrumentation Using Complex Programmable Logic Devices***". IMEKO TC7 Symposium 2002 Cracow. Junio 2002. Internacional.
- ***"Desarrollo de una Metodología para el Diseño e Implantación de Instrumentos Utilizando Dispositivos Lógicos Programables Complejos (CPLDs)".*** SOMI XVII Congreso de Instrumentación. Octubre 2002. Mérida Yuc. México. Nacional.
- ***"Osciloscopio digital didáctico, utilizando dispositivos lógicos programables complejos (CPLDs) con despliegue VGA".*** SOMI XVII. Congreso de Instrumentación. Octubre 2002. Mérida Yuc. México. Nacional.

TERCERO.

Se propusieron dos cursos a DGAPA (Dirección General de Asuntos del Personal Académico) en el marco del **Programa de Actualización docente para Profesores de Licenciatura**. Los cursos fueron aceptados y están programados de la siguiente manera:

- *Aplicaciones de Dispositivos Lógicos Programables.* Del 4 al 15 de marzo del 2002. Duración 40 hrs. Ponentes: Ing. Norma Elva Chávez Rodríguez; Ing. Jorge Valeriano Assem.
- *VHDL en Aplicaciones con CPLDs.* Del 2 al 13 de septiembre del 2002. Duración 40 hrs. Ponentes: Ing. Norma Elva Chávez Rodríguez; Ing. Jorge Valeriano Assem.

CUARTO.

Está en proceso un manual que presente de manera sistemática la metodología propuesta así como su aplicación practica en la instrumentación y que pueda ser publicado en la Facultad de Ingeniería de la UNAM y de utilidad para los alumnos de todas las materias afines como son Diseño Digital, Diseño de Sistemas Digitales, Organización de Computadoras, Microcomputadoras, Medición e Instrumentación, etc.

6.8 CICLO DE DESARROLLO ENTRE EL DISEÑO POR COMPORTAMIENTO Y EL DISEÑO ESTRUCTURAL.

Nuestra metodología propone el empleo de descripción por comportamiento como base para optimizar el ciclo de desarrollo de cualquier instrumento. Esto debido esencialmente por la similitud con relación a la programación con un lenguaje de alto nivel como "C", de forma tal que se le deja al compilador la tarea de sintetizar los componentes descritos en alto nivel. Evidentemente esto reduce el ciclo de desarrollo.

Sin embargo existirán secciones donde será necesario diseñar ciertos módulos de forma estructural para optimizar el espacio de silicio o el rendimiento en velocidad lo que es análogo a la programación en ensamblador.

Al describir por comportamiento podemos reducir el ciclo de desarrollo hasta en un 75% como lo reporta la Tabla 3.2, además de poder manipular con mayor facilidad el proceso de depuración y pruebas como lo reporta la Tabla 3.3.

6.9 COSTO DEL PROTOTIPO FRECUENCÍMETRO.

En la Tabla 4.1 presentamos el desglose de los costos requeridos para la fabricación de un prototipo.

INSUMO	COSTO (Pesos)
Un CPLD epm7128slc84-15	$160.00
Un LM360	$ 43.00
Dos 1N4148	$ 4.00
Un regulador 7805	$ 4.00
Resistencias y capacitores varios	$ 10.00
Transformador para la fuente	$ 30.00
2 Impresos	$ 60.00
7 Displays 7 seg.	$ 21.00
Cristal 8 MHz	$ 4.00
Bases diversas	$ 50.00
Caja y embalajes	$ 100.00
7 Transistores 2N4401 npn	$ 7.00
TOTAL	**$493.00**

Tabla 4.1. Desglose de costos

Como se observa en la Tabla 4.1 el costo total del prototipo tal y como es entregado al usuario (ver Fig. 3.23) tiene un costo aproximado de $ 493.00 pesos. Este costo es posible reducirlo principalmente porque una producción al mayoreo reduce significativamente los costos como es el caso de los impresos, en segundo lugar el costo del CPLD siempre va a la baja, anteriormente (un año atrás) tenia un costo de $200.00 pesos.

A los costos de insumos es necesario agregar los costos de ensamble, prueba de operación y empaque. En forma cuantitativa se estima se requiere de 3 Hrs. hombre por equipo, teniendo disponible las partes y el sitio de trabajo.

REFERENCIAS BIBLIOGRÁFICAS

[1] Valeriano Assem Jorge, Chávez Rodríguez Norma Elva, Haro Ruiz Arturo. *"Lenguaje de Descripción de Hardware Verilog"*, 49 Págs. Facultad de Ingeniería UNAM, División de Ingeniería Eléctrica, Departamento de Ingeniería en Computación e Ingeniería Electrónica. Junio 2001.

[2] Valeriano Assem Jorge, Chávez Rodríguez Norma Elva. *"Lenguaje de Descripción de Hardware VHDL"*, 69 Págs. Facultad de Ingeniería UNAM, División de Ingeniería Eléctrica, Departamento de Ingeniería en Computación e Ingeniería Electrónica. Agosto 2001.

[3] Altera Corporation. *Data Book.* (San Jose California1999).

[4] Fuentes R., Valeriano J. *"Monitor Digital de Presión Sanguínea"*. Memorias SOMI XVI Congreso de Instrumentación. (Querétaro, Querétaro, México, Octubre 2001).

[5] Fuentes R., Valeriano J. *"Implantación Digital de un Cronómetro Múltiple en CPLDs"*. Memorias SOMI XV Congreso de Instrumentación. (Guadalajara, Jalisco, México, Octubre 200).

[6] Chávez Rodríguez Norma Elva, Valeriano Assem Jorge, Haro Ruiz Arturo. *"Manual de consulta entorno de diseño MAX + PLUS II"*, 66 Págs. Facultad de Ingeniería UNAM, División de Ingeniería Eléctrica, Departamento de Ingeniería en Computación e Ingeniería Electrónica. Febrero 2001.

[7] Chávez Rodríguez Norma Elva, Valeriano Assem Jorge. *"Prácticas de Laboratorio en el Entorno de Diseño MAX+PLUS II"*, 29 Págs. Facultad de Ingeniería UNAM, División de Ingeniería Eléctrica, Departamento de Ingeniería en Computación e Ingeniería Electrónica. Agosto 2001.

[8] J.F. Wakerly. *Digital Design Principles & Practices.* (Ed. Prentice Hall. New Jersey), (2000).

[9] M. M. Mano, C. R. Kime. *Logic and Computer Design Fundamentals.* (Ed. Prentice Hall. New Jersey), (2000).

[10] S. Quintana, C. Ojeda, J.L. Pérez. *Diseño y Construcción de un Frecuencímetro. "Memorias SOMI XIV Congreso de Instrumentación"* (Tonantzintla, Puebla, México. Octubre, 1999).

[11] Altera Corporation. "*MAX+PLUS II Graphic & Symbol Editors*". (San Jose California1997).

[12] Altera Corporation. "*MAX+PLUS II Compiler*". (San Jose California1997).

[13] Altera Corporation. "*MAX+PLUS II Simulator, Timing Analizer & Waveform Editor*". (San Jose California1997).

[14] Altera Corporation. "*MAX+PLUS II Text Editor & AHDL*". (San Jose California1997).

[15] V. Vujieié, I. Zupunski, S. Milovancev. "*General Method for Quantization Error Predetermination in Digital Measurement System*". Fakultet tehnickih nauka u Novom Sadu. Yugoslavia.

[16] Cesare Alippi, et.al. *"A Composite System Design Methodology for Instrumentation and Embedded System"*. IEEE. 2000.

[17] Joseph Cerra. *"HDL Methodology Offers Fast Design Cycle and Vendor Independence"*. Actel Corporation. Abril, 1996.

[18] H.J. Herpel, M. Held, M. Glesner. *"A design Methodology for the Conceptual Design of Application Specific Digital Processors in Mechatronic Systems"*. IEEE. 1994.

[19] Li Shang, Niraj K. Jha. *"High-Level Power Modeling of CPLDs and FPGAs"*. IEEE. 2001.

[20] D. H. Cropley. *"A Graphical Modelling Method for the Representational Form of Measurement Theory"*. IEEE. 1997.

[21] Eduardo Becerra, et.al. *"Improving the Dependability of Embedded System Using Configurable"*. XIV International Symposium on Computer and Information Sciences, Ege University, Izmir, Turkey. Octubre, 1999.

[22] Bernardo Kastrup, Orlando Moreira. *"A Novel Approach to Minimising the Logic of Combinatorial Multiplexing Circuits in Product-Term-Based Hardware"*. IEEE. 2000.

[23] Kenneth Yan . *"Practical Logic Synthesis for CPLDs and FPGAs with PLA-Style Logic Blocks"*. IEEE. 2001.

[24] Jay Sturges. *"A Quantitative Approach to Bechmarking Programmable Logic Architectures"*. IEEE. 1993.

[25] Subbarao V. Wunnava, Alberto Rodreguiez. *"Scope of Digital Systems Increase with in System Reprogramability"*. IEEE. 2000.

[26] Joannis Papanuskas, et.al. *"A Uniform Design Methodology for Application Specific Digital Integrated Circuits in Automotive Applications"*. IEEE. 1995.

[27] Synthesis Methodology Guide. Actel. Synopsys.

[28] Lee H. Harrison. *"Certification Issues for Complex Digital Hardware"*. IEEE. 1994.

[29] Yee L. Low, et.al. *"Design Methodology for Chip-on-Chip Applications"*. IEEE. 1997.

[30] Bernardo Elayda, Ramine Roane. *"Synopsys/Xilinx High Density Design Methodology Using FPGA Compiler"*. Xilinx Synopsys. 1998.

[31] Xilinx. *"Embedded Instrumentation Using XC9500 CPLDs"*. 1997.

[32] Daniel D. Gajski, et.al. *"System Design Methodologies: Aiming at the 100 h Design Cycle"*. IEEE. 1996.

[33] Stephen Brown, Jonathan Rose. *"FPGA and CPLD Architectures: A Tutorial. IEEE Design & Test of Computers"*. IEEE1996.

[34] Fernando Pardo Carpio. *"VHDL Lenguaje para descripción y modelado de circuitos"*. Universitat de Valencia. 1997.

[35] Jonathan Rose, et.al. *"Architecture of Field –Programmable Gate Arrays"*. IEEE 1993.

[36] James O. Hamblen. *"Rapid Prototyping Using Field-Programmable Logic Devices"*. IEEE 2000.

[37] Andrej Zemva, et.al. *"Educational Programmable Hardware for Prototyping Digital Circuits"*. 1998.

[38] Keith Dimond, Kam Pang. *"Mapping VHDL Descriptions of Digital Systems to FPGAs"*. Electronic Engineering Laboratories. University of Kent at Canterbury.

[39] John Villaseñor, Brad Hutchings. *"The Flexibility of Configurable Computing".* IEEE Signal Processing Magazine. September, 1998.

[40] Inés del Campo, José Manuel Tarela. *"Consequences of the Digitization on the Performance of a Fuzzy Logic Controller".* IEEE Transactions on Fuzzy Systems. Febrero, 1999.

[41] Morris Chang. *"Teaching Top-Down Design Using VHDL and CPLD".* FIE '96 Proceedings. 1996.

[42] William S. Carter. *"The Future of programmable Logic and Its Impact on Digital System Design".* IEEE. 1994.

[43] Stephen Brown. *"FPGA Architectural Research: A Survey".* IEEE. 1996.

[44]Stephen D. Brown. *"An Overview of Technology, Architecture and CAD Tools for Programmable Logic Devices".* IEEE Custom Integrated circuits Conference.1994.

[45] Jose A. Torres. *"State of the Art in FPGA, Today and Tomorrow".* Mentor Graphics. Octubre, 2000.

[46] J. Valeriano, F. Lara, N.E Chávez. *"Development of a Design Methodology for Digital Measurement Instrumentation Using Complex Programmable Logic Devices".* Sympozjum TC7 Politechnika Krakowska Wydział Inżynierii Elektrycznej i Komputerowej Instytut Metrologii Elektrycznej E1 Ul. Warszawska 24, 31-155 Kraków, Poland. June 2002.

[47] J. Valeriano, F. Lara, N.E. Chávez. *"Desarrollo de una Metodología para el Diseño e Implantación de Instrumentos Utilizando Dispositivos Lógicos Programables Complejos (CPLDs)".* SOMI XVII. Congreso de Instrumentación. Octubre 2002. Mérida Yuc. México.

[48] J. Valeriano, R. Fuentes, G. Calva. *"Osciloscopio digital didáctico, utilizando dispositivos lógicos programables complejos (CPLDs) con despliegue VGA".* SOMI XVII. Congreso de Instrumentación. Octubre 2002. Mérida Yuc. México.