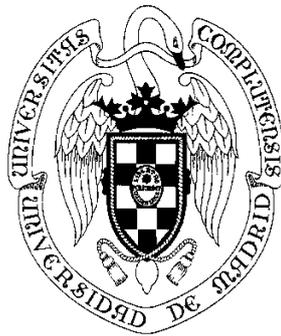


DISEÑO DE CIRCUITOS INTEGRADOS EL DESIGN FRAMEWORK II DE CADENCE



JUAN LANCHARES DÁVILA

**ESCUELA SUPERIOR DE INFORMÁTICA
DEPARTAMENTO DE ARQUITECTURA DE COMPUTADORES Y AUTOMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID**

1. ESTILOS DE DISEÑO

El diseño físico es un proceso muy complejo, y aún descomponiendo su realización en diferentes pasos, el desarrollo de cada uno de estas subtarear es muy dura, computacionalmente hablando. Sin embargo las necesidades del mercado demandan diseños realizados cada vez en menos tiempo y con un rendimiento de la oblea mayor, (entendiendo por rendimiento de la oblea el % de circuitos de funcionamiento correcto que se pueden obtener de una oblea de silicio).

Para intentar cumplir estas necesidades del diseño, han ido apareciendo a lo largo de los años diferentes estilos de diseño, con ligaduras de mayor o menor importancia que facilitan el diseño físico, generalmente a costa del ahorro de área o de los rendimientos o consumos de potencia.

De manera general los estilos de diseño se pueden clasificar en estilos full-custom o estilos semicustom. La principal característica del diseño full-custom es que los diferentes bloques que forman el diseño se pueden colocar en cualquier parte del area del circuito con la única condición que no se solapen entre sí.

En cuanto a los estilos del semicustom, algunas partes del diseño están prediseñadas y localizadas en partes determinadas del área del silicio.

1.1 EL DISEÑO FULL-CUSTOM

El diseño full custom es aquel en el que no existe ninguna restricción a la hora de realizar la ubicación ni el rutado de los diferentes módulos que lo componen. En este sentido se puede decir que no se utilizan elementos prefabricados y ni prediseñados por el fabricante. Esto significa que el diseñador debe poseer grandes conocimientos de microelectrónica.

En este estilo se debe diseñar todo el circuito de principio a fin, ayudado , como no podía ser de otra forma, por herramientas automáticas que facilitan la tarea. Pero esta ayuda no quita que se requiera un esfuerzo importante para llevar el proyecto a buen término.

El diseñador debe indicar exactamente donde quiere

Se divide el circuito en subcircuitos siguiendo algún tipo de criterio como el funcional. A estos subcircuitos se les llama bloques funcionales. Estos bloques funcionales pueden tener cualquier tamaño

La principal característica de este tipo de diseño es la ausencia total de ligaduras lo que permite diseños muy compactos

como principal inconveniente está la dificultad del proceso de automatización,. Esta es una de las razones de que se utilice cuando el diseño final debe ser de área mínima y además no tiene demasiada importancia el tiempo de diseño.

el espacio no ocupado por bloques se utiliza para el rutado. Inicialmente los bloques se sitúan con el objetivo de minimizar el área pero no se debe olvidar el area necesaria para realizar el rutado. Generalmente se utilizan varias capas de metal,

En un estilo de diseño jerárquico, un bloque a su vez puede estar compuesto de otros bloques, que a su vez pueden usar como estilo de diseño el full-custom u otro cualquiera, como

Como principal ventaja esta la gran flexibilidad a la hora de diseñar lo que permita alcanzar los óptimos de área, rendimiento, o potencia. Como principal desventaja se encuentra el enorme esfuerzo de desarrollo lo que hace que los tiempos de mercado sean elevados. Otra desventaja es que no se puede asegurar que el comportamiento eléctrico, que el diseñador había supuesto a alguno de los módulos sea el correcto.

Solo justificable cuando los costes pueden ser amortizados con un gran volumen de producción los microprocesadores y las memorias semiconductoras.

Cuando los bloques custom pueden ser reutilizados muchas veces por ejemplo mediante librerías de celdas

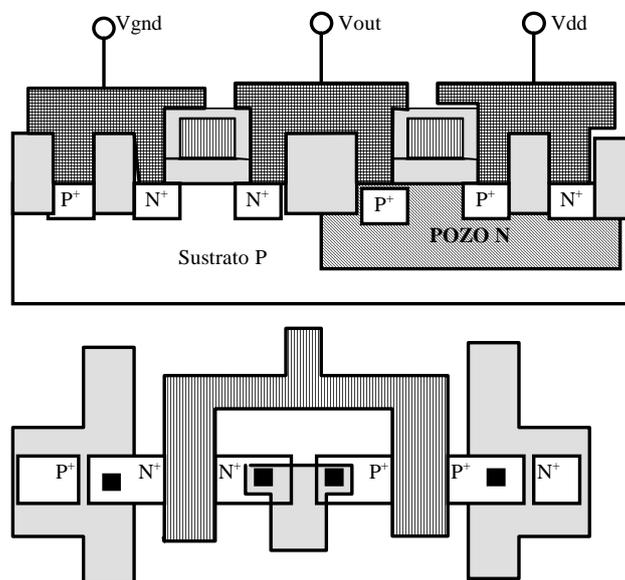
Cuando el coste no es demasiado importante

Debido al crecimiento y desarrollo de las herramientas de diseño automático el rango de diseños custom se reduce de año en año. Incluso algunos procesadores como el Alpha de DEC diseñan grandes porciones del mismo mediante estilo semicustom. Solo las unidades críticas como los operadores de coma flotante y de enteros utilizan este estilo.

Aunque las herramientas de diseño para estilo full-custom, no son numerosas si existen algunas de gran utilizadas. como el editor de layout

El editor de layout es la primera herramienta de trabajo que tuvieron los diseñadores y su objetivo es la generación de la representación física del diseño.

Puesto que el diseño físico ocupa una parte importante del tiempo total de diseño de una celda o nuevo componente este tipo de herramientas está en permanente desarrollo



- Las reglas de diseño son el punto de conexión entre el diseñador de C.I. y el ingeniero de procesos durante la fase de fabricación.

- El principal objetivo de estas reglas de diseño es obtener un circuito con un rendimiento de producción óptimo (circuitos válidos/circuitos no válidos) en un área lo menor posible sin comprometer la fiabilidad del circuito.
- Representan el mejor compromiso entre:
 - Rentabilidad de la fabricación
 - Performance del circuito

Las reglas más conservadoras nos llevan a circuitos que funcionan mejor pero más lentos y que ocupan mayor área.

Las reglas más agresivas tienen mayor probabilidad de generar mejoras en el "performance", pero estas mejoras pueden dañar la rentabilidad.

Las reglas de diseño especifican al diseñador ligaduras geométricas y topológicas que deben cumplir los patrones utilizados en el proceso de fabricación.

Estas ligaduras no son leyes rígidas que se deban cumplir inexorablemente para que los circuitos funcionen correctamente, sino más bien son recomendaciones del fabricante que aseguran una alta probabilidad de una fabricación correcta.

Se pueden encontrar diseños que violan las reglas y viceversa.

Existen dos conjuntos bien diferenciados de reglas de diseño:

Anchura mínima de las líneas

Distancia entre layers

Una anchura demasiado pequeña lleva consigo una discontinuidad en las líneas lo que puede provocar cortocircuitos.

Si los layers están demasiado cercanos se pueden fundir o interactuar el uno con el otro, cortocircuito entre dos nodos de circuitos diferentes

Hay dos aproximaciones para describir las reglas de diseño:

-Reglas "micron"

-Reglas basadas en λ .

Las reglas micron dan las anchuras y distancias entre layers en micras μ . La forma en que se trabaja en la industria.

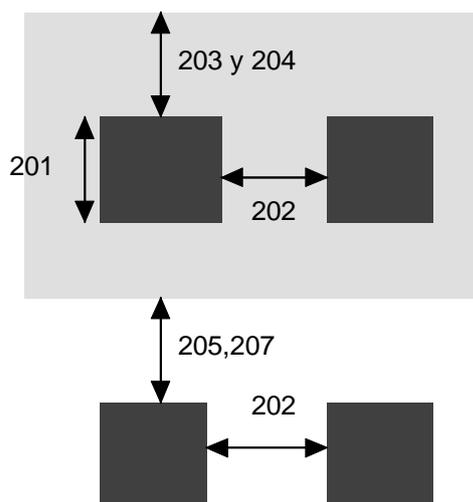
LAMBDA

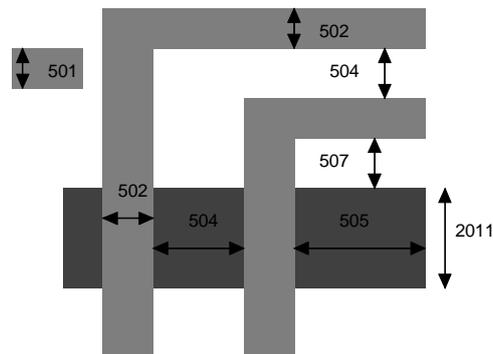
- Es un factor de corrección.
- Fue introducido por Mead-Conway.

- En teoría permite trabajar con diseños independientemente del avance tecnológico.
- Un diseño que utilizase reglas lambda en su descripción serviría para diferentes tecnologías:
- Las reglas lambda se han utilizado con éxito en diseños: 4-1.5 μ .
- No dan buenos resultados para distancias inferiores a las micras.
- En definitiva estas reglas permiten un cierto estado de escalamiento entre procesos diferentes, en este caso seria suficiente reducir el valor de Lambda.
- La experiencia demuestra que las disminuciones no son uniformes.

ENREJILLADO:

- Las herramientas CAD trabajan con enrejillados de dimensiones mínimas en términos de las cuales hay que expresar las reglas de diseño
- para procesos 1.25 m-2 m enrejillados 0.2 m - 0.25 m.
- Por ultimo, alguno de los sistemas de fabricación de mascarar tienen problemas de exactitud digital (de 16 BITS de precisión).





herramienta de extracción eléctrica que obtiene a partir del layout el esquemático del circuito, incluyendo los tamaños de los canales y las interconexiones. El circuito extraído puede utilizarse para comprobar que el layout implementa el diseño deseado.

Además el circuito extraído contiene información precisa sobre capacidades parásitas de hilos y difusiones y resistencias, lo que permite una simulación más y un análisis más preciso

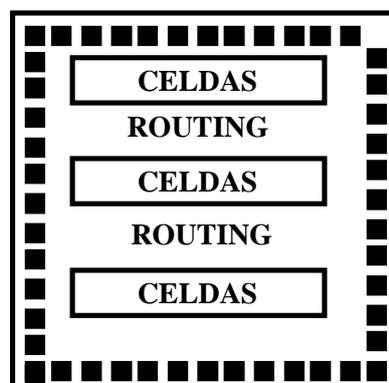
1.2 EL DISEÑO SEMICUSTOM

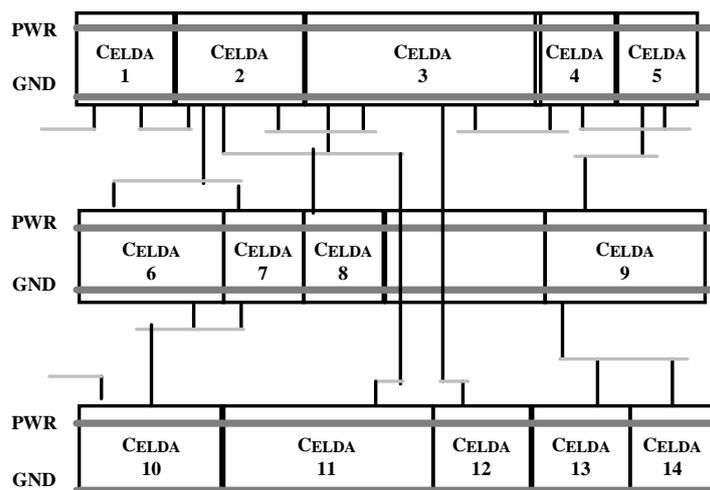
El diseño semicustom es aquel en el que existen ciertas restricciones a la hora de ubicar y rutar los módulos diseñados. Según sean este tipo de restricciones el estilo semicustom se puede subdividir en una serie de subestilos como son:

- Basado en celdas estándar
- celdas compiladas
- generadores de módulos

1.2.1 BIBLIOTECAS DE CELDAS ESTÁNDARES

- En muchas ocasiones la flexibilidad y grandes prestaciones que se consiguen con el full custom no son necesarias.
- restringe la geometría permitida al circuito y fija una topología de layout específica. Gracias a estas ligaduras se pueden diseñar eficientemente herramientas automáticas, lo acelera enormemente el tiempo de diseño. Como contrapartida los rendimientos y la densidades que se consiguen son menores.
 - Utiliza como elemento básico de diseño un conjunto de celdas que proporciona el fabricante. Cada una de estas celdas implementa una funcionalidad muy básica, como pueden ser puertas OR, AND, biestables, etc. La funcionalidad y características eléctricas de estas celdas están testadas analizadas y probadas por el fabricante para que funcionen correctamente bajo gran número de supuestos.
 - Estas celdas tienen forma rectangular y son todas de la misma altura.
 - todas tienen la toma de alimentación y de tierra en la misma posición que corren horizontalmente a través de las celdas. Y tienen las entradas y salida en las caras superior e inferior
 - Las celdas se colocan en filas y el espacio entre ellas se llama canal. En este estilo se debe fabricar todo el chip.
 - Cada celda puede tener la anchura que necesite para implementar su funcionalidad, por compleja que esta sea.





Como se ve en la figura un chip se compone de filas de celdas estándares intercaladas con zonas de rutado, siendo las filas de celdas estándares de la misma anchura, mientras que en las zonas de rutado tienen una anchura que depende de la densidad de rutado en esa zona.

Esta estructura necesita de líneas de rutado verticales que se pueden implementar mediante un layer de metal y añadiendo celdas de throughput, que son celdas que se dedican exclusivamente a dejar pasar rutados verticales.

Dado que los layout de las celdas estándar están prediseñados por el fabricante, el proceso de diseño se reduce a trasladar la especificaciones a una red de puertas de la biblioteca de celdas estándares. A este paso se le denomina correspondencia con la tecnología (technology mapping). A continuación hay que decidir en qué lugar del chip se colocan estas celdas, teniendo como objetivo la minimización del área-intentando que los canales de rutado sean lo más estrechos posibles.. A esta fase se la denomina placement.

Por último se ejecuta la fase de rutado, que consiste en realizar las conexiones entre las celdas estándar.

Dada la gran regularidad de las celdas estándares, cada uno de estos pasos se puede realizar mediante herramientas de diseño automático.

Dado que el proceso de fabricación de las celdas estándar y del fullcustom es idéntico, los costes y los tiempos de fabricación son los mismos.

El diseñador del full-custom puede realizar importantes optimizaciones de área, consumo o tiempo, que el diseñador de celdas estándares, no puede conseguir. Como contrapartida las celdas estándar pueden ser diseñadas mucho más rápidamente.

No debemos olvidar que el diseño de las celdas estándar, es un diseño Full-custom, es decir es un diseño que consume gran cantidad de tiempo, su ventaja es la enorme reutilización que se realiza de ellas. Esta última característica fuerza a que sean celdas muy robustas, para que funcionen dentro de un amplio rango de condiciones.

Dado que el fanin de cada puerta no se conoce hasta que el diseño está realizado, es práctica común asegurar que cada puerta es capaz de trabajar con capacidades de carga elevadas.. Esto simplifica el diseño pero tiene un importante impacto sobre el área y el consumo de potencia.

La información que el fabricante aporta de las celdas estándar, incluye su funcionalidad, su consumo , su fanout, tiempos de subida, tiempos de bajada. Etc.

Las celdas estándar se están utilizando principalmente en los diseños de lógica aleatoria, máquinas de estados finitos, porque se adaptan muy bien a los esquemas multinivel. Además son perfectas para la utilización de herramientas de síntesis lógica.

La síntesis lógica permite tomar como entrada descripciones de sistema mediante lenguajes de alto nivel y obtiene como salida una lista de celdas estándar minimizando el retardo y el área.

En la actualidad el estilo de celdas estándar es el más utilizado en el diseño de Circuitos de aplicación específica e incluso se empieza a utilizar regularmente en el diseño de microprocesadores.

1.2.1.1 Celdas compiladas

Las bibliotecas de celdas estándar tienen la desventaja de ser discretas, es decir de tener el número de opciones limitadas. Cuando se busca como objetivo el rendimiento del sistema son atractivas las celdas con los tamaños optimizados.

Se han generado un conjunto de herramientas para generar layouts según se van necesitando, dando el tamaño del transistor

1.2.1.2 generadores de módulos

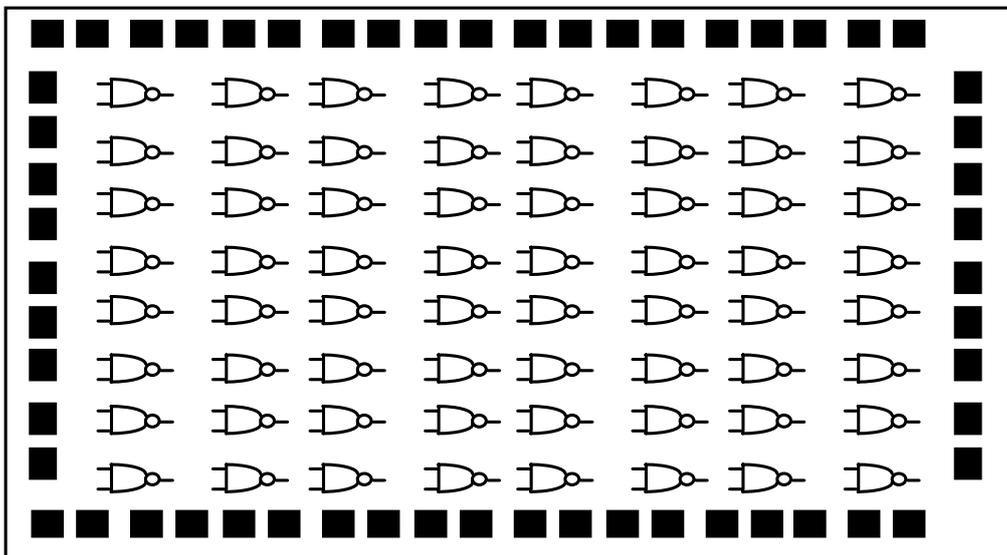
Las bibliotecas de celdas son útiles para lógica aleatoria pero son ineficientes para estructuras regulares como desplazadores, sumadores multiplicadores, caminos de datos PLAs o memorias. En todos estos módulos es importante la reducción de las capacidades internas de los nodos, pero esto es difícil de

conseguir utilizando celdas estándar. Además, las celdas estándar ignoran la regularidad de estos módulos.

Existen generadores de macroceldas y compiladores de camino de datos

1.2.2 GATE ARRAYS

- Es una simplificación del estilo anterior, sólo que en este caso todas las celdas son iguales.
- En este estilo el chip se prefabrica con un array de puertas idénticas. Estas celdas están separadas por canales verticales y horizontales. El diseño inicial se debe modificar hasta convertirlo en una red de puertas idénticas que se pueda implementar en el chip.
- Como paso final se debe acabar la fabricación realizando el rutado que une las puertas para implementar la funcionalidad deseada



- **FPGAS.** Es un chip ya fabricado formado por arrays de bloque lógicos programables (CLB), en los que se pueden grabar funciones combinacionales, bloques de entrada salida (IOB), que relacionan la lógica con los pines de entrada salida y bloques de interconexión que conectan entre si los bloques lógicos. Con este estilo el diseño no envía a fabricar sino que se implementa programando los CLB, IOB y los bloques de interconexión.

La principal ventaja del estilo semicustom es que utiliza partes ya diseñadas o prefabricadas, con lo que el diseñador puede asegurar con alto porcentaje de probabilidades de acierto su comportamiento eléctrico. Además La utilización de estas partes prediseñadas o prefabricadas ahorra tiempo y esfuerzo de diseño de diseño, y permite diseñadores con menores conocimientos de microelectronica, pero más versados en temas de diseño lógico o estructura de computadores. Su principal desventaja es que las cotas de optimización son inferiores a las que se alcanzan en el estilo full custom.

1.3 ELECCIÓN DEL ESTILO DE DISEÑO

v ESTILO FULL-CUSTOM VERSUS ESTILO SEMICUSTOM

La viabilidad de un diseño microelectrónico depende de muchos factores en conflicto como pueden ser el rendimiento en términos de velocidad, el consumo de potencia, el coste y el volumen de producción.

Por ejemplo para que un procesador debe tener un buen rendimiento y un bajo coste para que tenga un rendimiento de mercado adecuado. Conseguir ambos objetivo simultáneamente es sólo posible con volúmenes de producción a gran escala.

Existen otras aplicaciones como el radar o los sistemas espaciales en los que el volumen de producción es pequeño, pero el coste de las partes electrónicas es solo una pequeña parte del total.

Por último, la gran mayoría de los diseños que se realizan solo tiene como objetivo el máximo ahorro de área y el menor tiempo de mercado, para que salgan rentables y competitivos.

Implícitamente hemos podido observar que el coste de un diseño depende de dos factores :

- el coste de diseño,

- el coste de producción por parte, que depende de la complejidad del proceso, area del diseño y rendimiento del proceso. Vamos a explicar esto con un poco más de profundidad para entender el motivo por el que es tan importante el ahorro de área para los costes de un diseño.

La elección depende del tipo de producto que se esté diseñando. Si el producto es un diseño muy complejo, que se espera producir durante mucho tiempo y en grandes cantidades, la elección serie sin duda el estilo full custom, ya que las ventajas y beneficios finales en optimización de área, rendimiento o consumo superan con creces el esfuerzo de diseño.

En cambio, si el producto que se desea diseñar es un ASIC (Application Specific Integrated Circuit), el estilo de diseño que se elige es el semicustom. Los ASIC son circuitos en los que prima el tiempo de diseño sobre el grado de optimización, es decir diseños que deben estar rápidamente en el mercado, cuya tirada va a ser muy limitada y con un tiempo de vida muy corto porque se rediseñan o modifican muy a menudo.

1.4 ARQUITECTURAS FPGA

1.4.1 ¿QUÉ ES UNA FPGA?

La arquitectura de una FPGA consiste en una matriz o array de bloques lógicos que se pueden programar. Es muy similar a la MPGA (Mask Programmable Gate Array). Las FPGA's tienen tres componentes principales: bloques lógicos configurables, bloques de entrada - salida y bloques de conexión [Sangiovanni93].

Los bloques lógicos configurables (CLB'S) son los encargados de implementar toda la circuitería lógica del diseño. Están distribuidos en forma de Matriz en el circuito y serán nuestra principal referencia a la hora de hacer el proceso de partición.

Por otro lado están los bloques de entrada y salida (IOB's) que como su propio nombre indica, son los encargados de conectar la parte del circuito implementada en la FPGA con el mundo exterior. Este "mundo" exterior puede ser directamente la aplicación para la que esté diseñada o como en nuestro caso en el que son necesarias varias FPGA para implementar un circuito, el resto de las FPGA.

Por último están los bloques (switchboxes) y líneas de interconexión que son los elementos de los que dispone el diseñador para hacer el rutado del circuito. En ciertos casos en los que la ocupación de los CLB no es total, estos se pueden utilizar también para llevar a cabo esta tarea.

Los bloques lógicos de una FPGA puede ir desde algo tan simple como una puerta lógica hasta algo tan complejo como un Microprocesador [Murgai95]. Esto le permite implementar multitud de circuitos tanto combinacionales como secuenciales.

A parte de por la estructura y composición de los bloques lógicos, las FPGA se diferencian también por sus estructuras de rutado y por la tecnología

de programación de sus conexiones. Las arquitecturas de rutado de una FPGA puede ser tan simple como una línea de conexión directa entre dos bloques o tan compleja como un multiprocesador (perfect shuffle). Por su parte las tecnologías de programación más utilizadas son la SRAM, los antifusibles y las memorias EPROM [Trimberger94].

El proceso de diseño para implementar una FPGA es básicamente el mismo que para un Gate Array. La entrada puede ser tanto un esquemático como una descripción en un lenguaje de descripción de hardware. El fabricante suministra un software que convierte la descripción del diseño en el programa de la FPGA. El código resultante se puede cargar inmediatamente en el dispositivo y probar el diseño, lo que proporciona una manera muy sencilla de corregir fallos en un diseño.

En la figura 5-1 se muestra la estructura interna de una FPGA. En ella están señalados los bloques lógicos, los IOB's y las matrices de interconexión [Trimberger93].

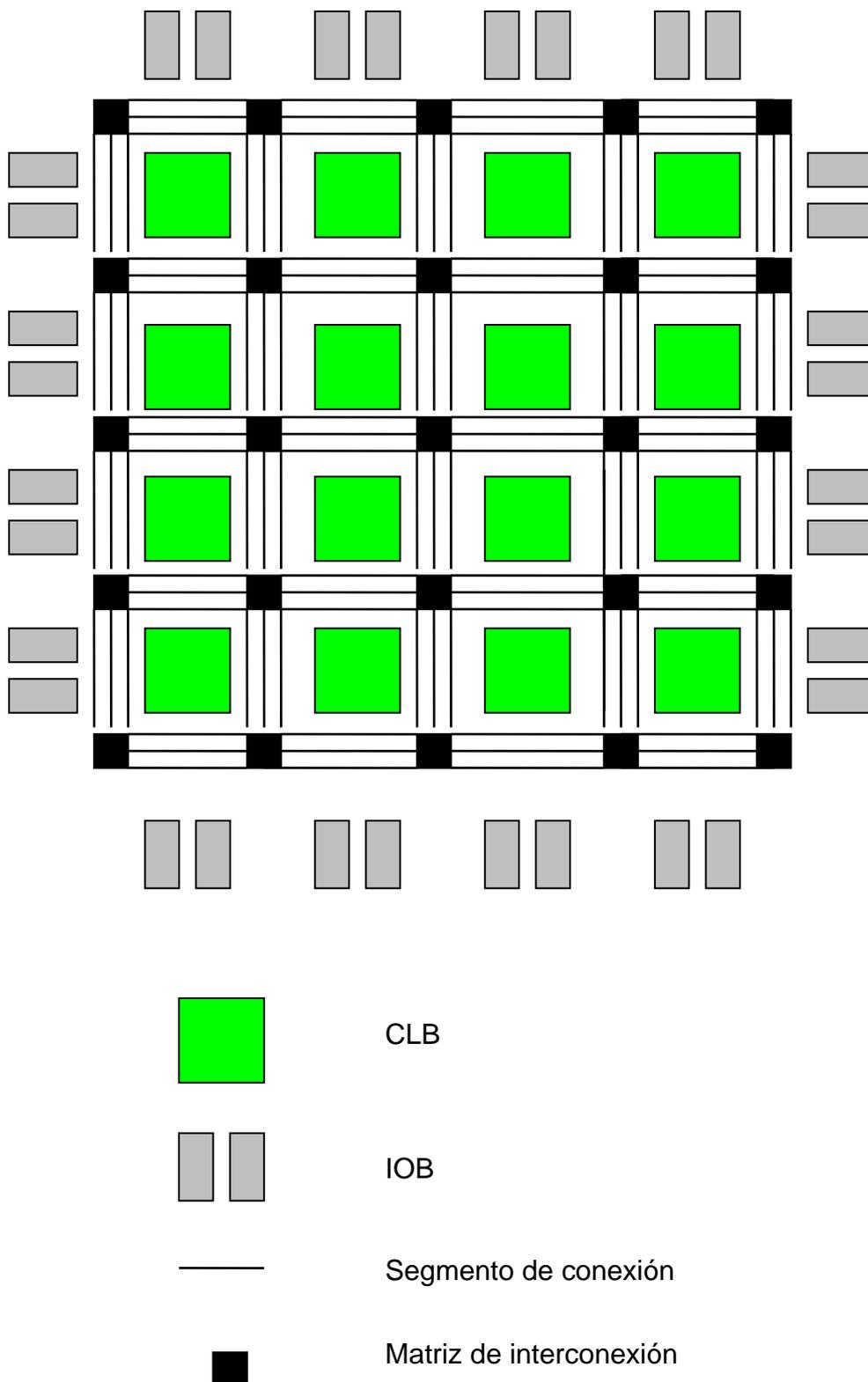


Figura 5-1: Estructura general de una FPGA

1.4.2 TECNOLOGÍAS FPGA

Para programar a la FPGA's se utilizan diversas tecnologías pero las más importantes son [Rose93] [Rosado95] [Hwang94]:

- SRAM
- Antifusible
- Puerta flotante

1.4.2.1 FPGA's basadas en SRAM

Las FPGA's que se programan mediante SRAM, utilizan celdas RAM estáticas para controlar la puerta de paso o los multiplexores. Esta tecnología la utilizan los circuitos fabricados por Xilinx, Plessey, Algotronic, Concurrent Logic y Toshiba.

En la figuras 5-2 y 5-3 podemos ver dos ejemplos de programación de la FPGA. Si cuando cargamos la SRAM ponemos un uno lógico la puerta de paso estará abierta y se comporta como un interruptor cerrado. Cuando tenemos un cero en la memoria la puerta estará configurada como un interruptor apagado.

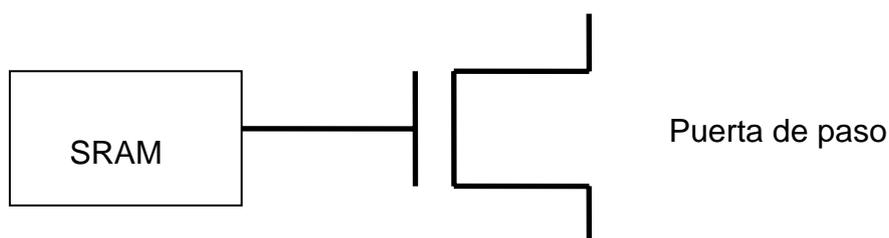


Figura 5-2 Programación con SRAM de una puerta de paso

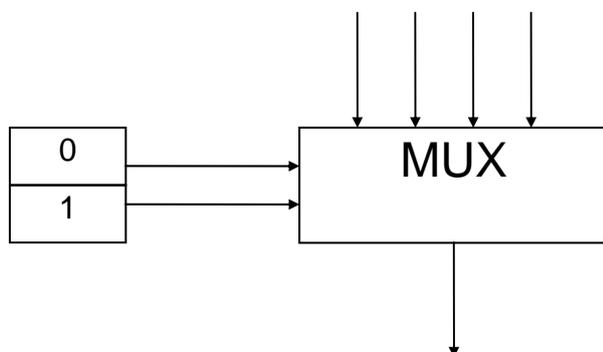


Figura 5-3: Programación con SRAM de un Multiplexor.

Como la SRAM es volátil, la FPGA se debe cargar y configurar en el momento de encender el chip, lo que hace necesaria la existencia de una memoria externa que suministre los bits programados por el usuario. Actualmente con las herramientas de desarrollo existentes esto no representa un problema insalvable si no se dispone de dicha memoria, pues el software permite realizar esta operación en poco tiempo en la mayoría de los casos.

La principal desventaja de la SRAM es que necesita una gran cantidad de área para su implementación, pero presenta dos ventajas fundamentales como son que es muy rápida de reprogramar y que sólo requiere tecnología estándar de circuitos integrados para su desarrollo.

1.4.2.2 FPGA´s programables por Antifusible

Un antifusible es un dispositivo de 2 terminales que cuando no está programado presenta una alta resistencia entre sus terminales. Cuando se le aplica un voltaje de entre 11 y 20 voltios, el antifusible se funde y crea una unión permanente de baja resistencia. Actualmente se utilizan básicamente dos tipos de antifusible:

- Los de ONO (Oxígeno-Nitrogeno-Oxígeno)
- Los amorfos, fabricados con silicio amorfo entre dos capas de metal o de polisilicio.

Este tipo de tecnologías lo utilizan las casas Actel, Quicklogic y Crosspoint. Su mayor ventaja es el poco espacio que ocupan y la baja

resistencia que se produce al realizar la programación. Además la capacidad parásita de un antifusible amorfo sin programar es significativamente más pequeña que en otras tecnologías de programación.

1.4.2.3 Tecnología de puerta flotante

Están basadas en EPROM's (Erasable Programmable ROM) borrables por rayos ultravioleta. Esta tecnología la utilizan las casas Altera y Plus Logic. La forma de programarlas es decrementar el voltaje umbral de los transistores que de lo contrario quedan permanentemente apagados.

La mayor ventaja de esta tecnología es que se pueden borrar y volver a programar muy fácilmente. Además no es necesaria la utilización de una memoria externa para alimentarlas.

Presenta algunos inconvenientes que las hacen poco adecuadas para nuestros propósitos. En primer lugar el flujo de diseño es más largo que en las otras tecnologías descritas. Otras dos desventajas son su alta resistencia de encendido y el alto consumo estático, debido principalmente a la resistencia de los transistores utilizados para el pull-up.

1.4.3 ARQUITECTURAS DE LOS BLOQUES LÓGICOS

Los bloques lógicos de las FPGA comercialmente más utilizadas son:

- Pares de transistores
- Pequeñas puertas básicas como NAND's
- Multiplexores
- Look _Up Tables (LUT's)
- Estructuras AND / OR de alto fan-in.

La clasificación de los bloques lógicos se realiza por su granularidad [Gajski94a]. La granularidad se puede definir en varias direcciones, por ejemplo como el número de funciones booleanas que se pueden implementar con un bloque lógico o el número de puertas NAND de dos entradas al que es equivalente. Otras formas de clasificación lo hacen por el número total de transistores necesarios, el área total necesaria para implementar el bloque o el número de entradas y salidas. Una forma más homogénea de hacer la clasificación es la que proponen Rose et al, que consiste en agruparlos por bloques lógicos de granularidad fina y gruesa [Rose93].

1.4.3.1 Bloques lógicos de granularidad fina

Las más importantes son las que se citan a continuación, precedidas por el nombre de la casa comercial que fabrica el dispositivo:

- Crosspoint: utiliza pares de transistores como módulo básico
- Plessey y Toshiba: NAND de 2 entradas
- Concurrent Logic: Una AND de 2 entradas y una OR de 2 entradas
- Algotronix: conjunto configurable de MUX

La mayor ventaja que ofrecen los bloques lógicos de grano fino es que se utilizan la mayoría de los bloques disponibles. La principal desventaja es que requieren un alto número de segmentos de línea y de interruptores programables. Como resultado las FPGA que utilizan este tipo de bloque lógico son en general más lenta y menos densas que las que utilizan bloques de granularidad gruesa.

1.4.3.2 Bloques Lógicos de grano grueso

En general los bloques lógicos de granularidad gruesa están formados por multiplexores y LUT's (Look Up Tables). Los LUT's son bloques que pueden implementar cualquier función lógica de tantas entradas como su orden indique. Por ejemplo un LUT de orden 6 puede implementar cualquier función

lógica de 6 literales. Es decir puede tomar 2^6 valores distintos. En la figura 5-4 podemos ver un ejemplo de LUT de 6 entradas o de orden 6:

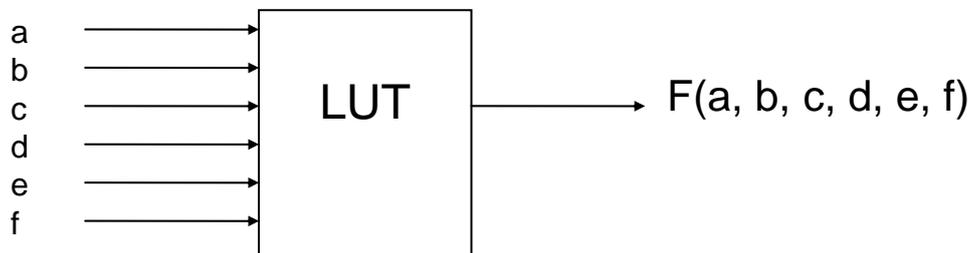


Figura 5-4 : Un LUT de 6 entradas puede implementar 2^6 funciones lógicas distintas.

Los bloques lógicos de granularidad gruesa más utilizados son los que implementan Actel, Quiclogic y Xilinx.

Actel: Utiliza multiplexores y varía dependiendo de la familia de dispositivos. Por ejemplo la ACT-1 incorpora un bloque lógico compuesto por 3 multiplexores y una puerta lógica. Tiene un total de 8 entradas y una salida, lo que hace que pueda implementar 702 funciones lógicas distintas. En la ACT-2 Actel fabrica 3 multiplexores y una puerta AND, lo que hace que pueda implementar 766 funciones lógicas distintas.

Por su parte Quicklogic utiliza un multiplexor de 4 a 1. Los sistemas basados en multiplexores suministran un alto grado de funcionalidad utilizando un número relativamente pequeño de transistores. Aunque también es cierto que necesitan un mayor número de entradas.

La base de los bloques lógicos de Xilinx es una SRAM funcionando como LUT's. La tabla de verdad de una función lógica de K entradas queda almacenada en una SRAM de $2^K \times 1$. La ventaja de las LUT's es que tienen una alta funcionalidad y versatilidad, aunque dependiendo del diseño pueden quedar bastantes sin utilizar. Como contrapartida estos bloques que han quedado libres se pueden utilizar para hacer el rutado del circuito. Más adelante dentro de este capítulo se explican los bloques lógicos de Xilinx con más detalle. En la tabla 5-1 podemos ver un resumen de las principales características de los dispositivos disponibles más importantes

Tabla 5-1: Comparación de los distintos tipos de FPGA's

1.4.4 VENTAJAS E INCONVENIENTES

Algunas de las ventajas de las FPGA ya se han citado anteriormente. A continuación exponemos algunas de las más importantes [Brown96] [Brown95] [Micro95] .

- El tiempo de programación y puesta en el mercado se reduce considerablemente
- Son programables por el usuario. Esto aparte de dar mayor libertad al diseñador permite una reducción de productos en stock, ya que se pueden utilizar para diferentes aplicaciones
- Algunos tipos son reprogramables, lo que las hace especialmente indicadas para procesos de prototipado en muchos diseños y permite la corrección de errores

- El proceso de diseño es muy simple y asequible.
- Existe una amplia gama de dispositivos que cubren las necesidades de usuarios de todo tipo.
- No necesita procesos de fabricación con máscaras
- Actualmente pueden implementar hasta circuitos de 25000 puertas equivalentes

Los inconvenientes de las FPGA's son debidos principalmente a su flexibilidad, lo que las hace que en ocasiones sean inapropiadas:

- En primer lugar es más cara que su equivalente programable por máscara, esto es debido a que al tener que dejar los canales de rutado ya delimitados ocupa una mayor área y en una oblea se pueden fabricar menos.
- Es un dispositivo más lento que otros sistemas de propósito específico, debido principalmente a los transistores y matrices de interconexión que utiliza. Para hacernos una idea aproximada los mecanismos de interconexión de una FPGA introducen aproximadamente entre el 30 y el 50 % del retardo total del circuito.
- En ocasiones se desaprovecha parte de la lógica para poder realizar el rutado completo del sistema.

1.4.5 HISTORIA

El nacimiento de los dispositivos lógicos programables es bastante reciente. Los primeros dispositivos no aparecieron hasta el año 1975. El motivo del desarrollo de estos circuitos se encuentra a su vez en la proliferación de los sistemas empotrados. Al popularizarse los sistemas digitales gobernados por microprocesadores, tanto los diseñadores como los científicos se dieron cuenta del gran problema que presentaban los circuitos implementados mediante tecnología SSI. Los principales inconvenientes que surgieron eran el tamaño, coste y funcionamiento de los mismos [Ukiley93] [Oldfield95].

En 1975 aparece como una primera solución un dispositivo fabricado por la firma Signetics, el 82s100 FPLA, que como sus siglas indican era un array lógico programable. Estaba formado por 48 términos productos, con posibilidad para 16 entradas y ocho salidas. Dos años más tarde aparece la primera FPLA de la compañía Monolithic Memories que mejoraba el consumo de potencia al cablear las OR de los arrays lógicos.

Sin embargo, el cambio más importante se produce cuando en 1985 Xilinx presenta su serie 2000 [Xilinx94]. La aportación fundamental consiste en que los bloques lógicos que componen el circuito integrado, además de poderse programar, están compuestos por algo más que puertas AND y OR, lo que hace que el abanico de aplicaciones se abra espectacularmente. Estos dispositivos se explican más adelante en este mismo capítulo. Paralelamente Actel lanza al mercado las FPGA programables por antifusible.

A partir de ese momento el interés y el desarrollo de las FPGA ha evolucionado muy rápidamente y en el año 1987 Xilinx ya comercializa la serie 3000, que completa con la aparición en 1991 con la fabricación de la serie 4000.

Este breve recorrido histórico nos permite comprobar la novedad de estos circuitos. Como consecuencia de la misma, las herramientas para ayudar al diseñador en el proceso de implementación de hardware mediante FPGA's no están desarrolladas hasta el punto deseado. Además debido al avance de la tecnología se hacen necesarias que sean posibles de manejar especificaciones más complejas. Estos dos puntos son una motivación más del desarrollo de este trabajo. En la figura 5-5 vemos un esquema de la evolución de las FPGA's desde 1975.

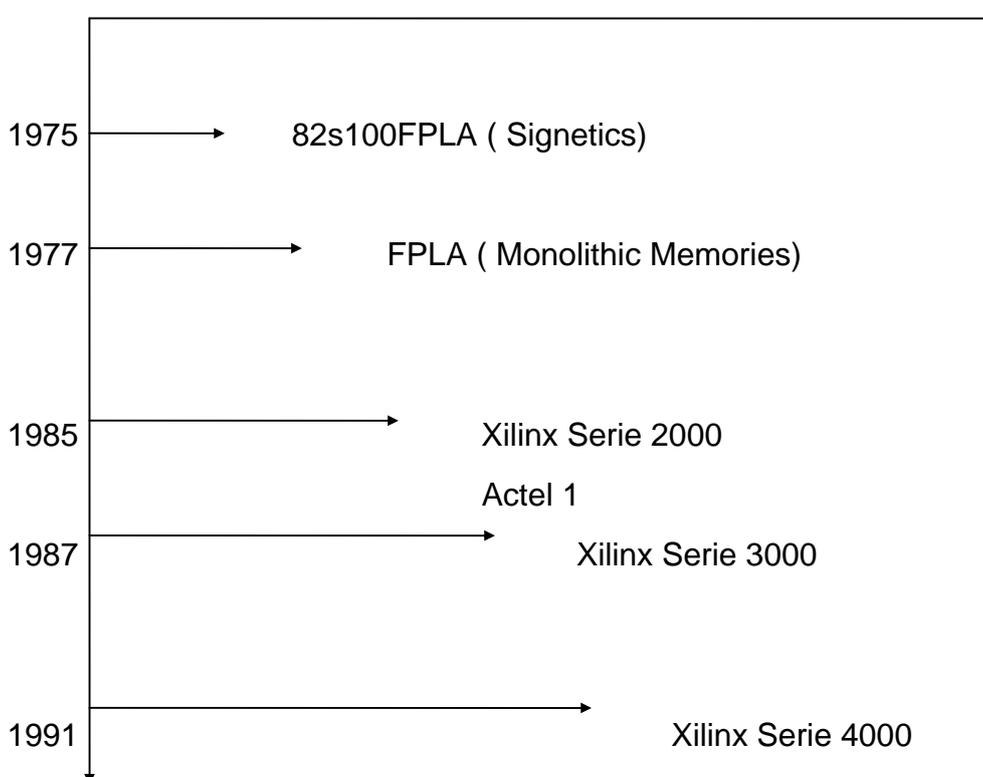


Figura 5-5 : Evolución histórica de los dispositivos lógicos programables

De los estilos de diseño expuestos anteriormente nosotros hemos escogido las FPGA's. Los motivos de esta elección son entre otros su perfecta adaptación para el problema de los sistemas empotrados y su arquitectura, especialmente adaptable a nuestro algoritmo. A continuación veremos más

extensamente las propiedades de las FPGA más importantes y en particular las de Xilinx, que son las elegidas para solucionar nuestro problema. En este caso se ha escogido la FPGA de Xilinx de la serie 3000 y 4000 por una serie de razones. La primera es que existe un punto en el que el producto (n° de LUT's * Área de un LUT) es mínimo y este punto se da para los LUT's de 4 entradas [Brown95] que son precisamente los que incorporan la serie 4000 de Xilinx. La serie 3000 es la predecesora de la 4000 y además nos permite comparar resultados con otras herramientas y algoritmos de partición. Otro motivo es que presentan una alta flexibilidad para el rutado que es una fase obligada después de la partición del circuito y que tendremos en cuenta a la hora de la implementación.

1.5 FPGA'S DE XILINX

La arquitectura básica de cualquier tipo de FPGA se puede comprender mejor si se identifican previamente sus componentes. Las FPGA de Xilinx que son las que estudiaremos en este capítulo, se componen principalmente de los siguientes bloques:

- Bloques Lógicos Configurables o CLB's
- Bloques de entrada y salida o IOB's
- Bloques y líneas de conexión

En los siguientes apartados veremos una descripción más a fondo de estos elementos para las tres series más sencillas de Xilinx:

- Serie 2000
- Serie 3000
- Serie 4000

Aunque los resultados comparados se orientan a la serie 3000, el algoritmo es totalmente válido para las otras dos familias y por ello se exponen sus características principales. Además para hacer una primera adaptación del algoritmo se utilizaron los valores de la serie 2000. Esta serie tiene 2 tipos de circuitos, lo que permite afrontar el problema como uno de bipartición.

1.5.1 SERIE 2000

Actualmente la serie 2000 de Xilinx esta prácticamente en desuso. Sin embargo exponemos aquí las principales características de estos circuitos ya que los bloques lógicos y de entrada - salida de la serie 2000 son muy simples. Esto hace que sean una buena primera aproximación. Al estudiarlos brevemente tendremos también una idea de la evolución que han sufrido estos dispositivos hasta llegar a los circuitos actuales, cuando veamos en este mismo capítulo la serie 4000. En la tabla 5-2 y las figuras 5-6 y 5-7 podemos ver las características de la serie 2000 y la configuración de sus CLB y Bloques de entrada - salida.

Dispositivo	Vcc	Nº CLB's	Nº IOB's
XC 2064	5.0 V	64	58
XC 2064 L	3.3 V	64	58
XC 2018	5.0 V	100	74
XC 2018 L	3.3 V	100	74

Tabla 5-2: Características principales de la serie 2000 de Xilinx

Figura 5-6: Estructura de un IOB de la serie 2000

Figura 5-7: Estructura de un CLB de la serie 2000

1.5.2 SERIE 3000

La familia o serie 3000 de Xilinx está compuesta por 6 tipos diferentes de circuitos. Se diferencian principalmente en el número de CLB's y de IOB's que contienen [Ukeiley 93] [Xilinx94]. En la tabla 5-3 podemos encontrar las características y propiedades principales de la Serie 3000.

Dispositivo	Nº CLB's	Nº IOB's	Coste Normalizado
XC 3020	64	64	1.00
XC 3030	100	80	1.36
XC 3042	144	96	1.84
XC 3064	224	110	3.15
XC 3090	320	144	4.83

Tabla 5-3: Características principales de la serie 3000 de Xilinx

Los CLB's de la serie 3000 están dispuestos, como en todos los dispositivos de Xilinx, en forma de Matriz $M \times N$, que va desde 8×8 para el 30010 hasta el de 16×20 que tiene el dispositivo con mayor número de bloques lógicos de la serie, el 30060. Cada uno de los bloques lógicos o CLB incluye los siguientes componentes:

- 9 Multiplexores
- Un generador de funciones lógicas
- 2 biestables
- Lógica combinatoria

El generador de funciones lógicas es el bloque principal del CLB y se puede configurar para que actúe o bien como un LUT de 32×1 , o bien como un LUT de 16×1 . Como ya hemos dicho, los LUT's (Look Up Tables) son bloques que pueden representar cualquier función Booleana del que dependan sus entradas. Debido a la rigidez que presenta este bloque en la Serie 3000 la utilización de la FPGA se limita mucho.

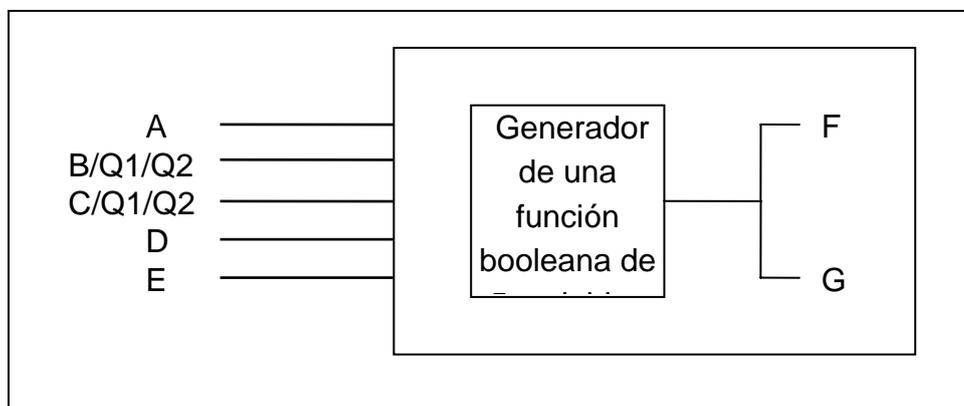


Figura 5-8 :Modo F de configuración del generador de funciones booleanas en la serie 3000 de Xilinx

La Serie 3000 permite configurar este generador de funciones de tres formas distintas: F, FG y FGM. En el modo F el generador de funciones actúa como un LUT de 32×1 que puede representar cualquier función que dependa de 5 variables. El segundo modo, el FG, funciona como un LUT de 16×1 y puede representar 2 funciones lógicas de 4 variables. El tercer tipo, FGM, es

una versión multiplexada del FG. Las salidas F y G del generador de funciones lógicas se pueden conectar posteriormente a los otros multiplexores del CLB, a las salidas del mismo, o incluso a cualquiera de los dos biestables D que contiene. En las figuras 5-8, 5-9 y 5-10 podemos ver los distintos tipos de conexión de los generadores de funciones (BFG). La figura 5-11 es un esquema general de un CLB de la serie 3000.

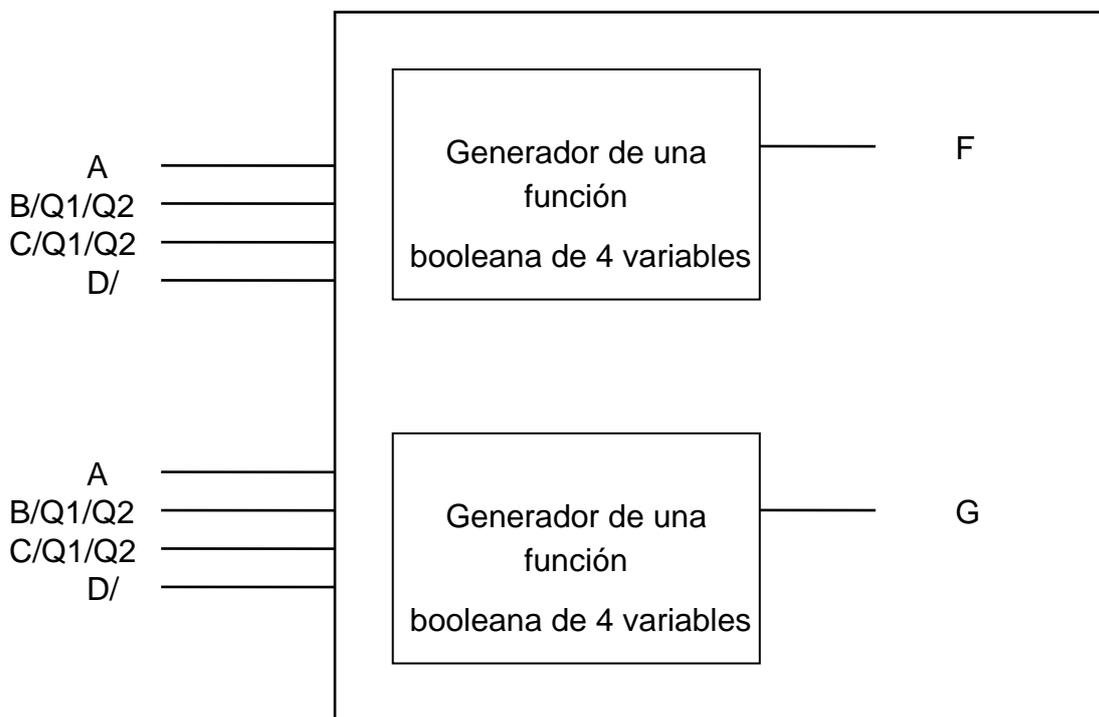


Figura 5-9: Modo FG de configuración del generador de funciones booleanas en la serie 3000 de Xilinx.

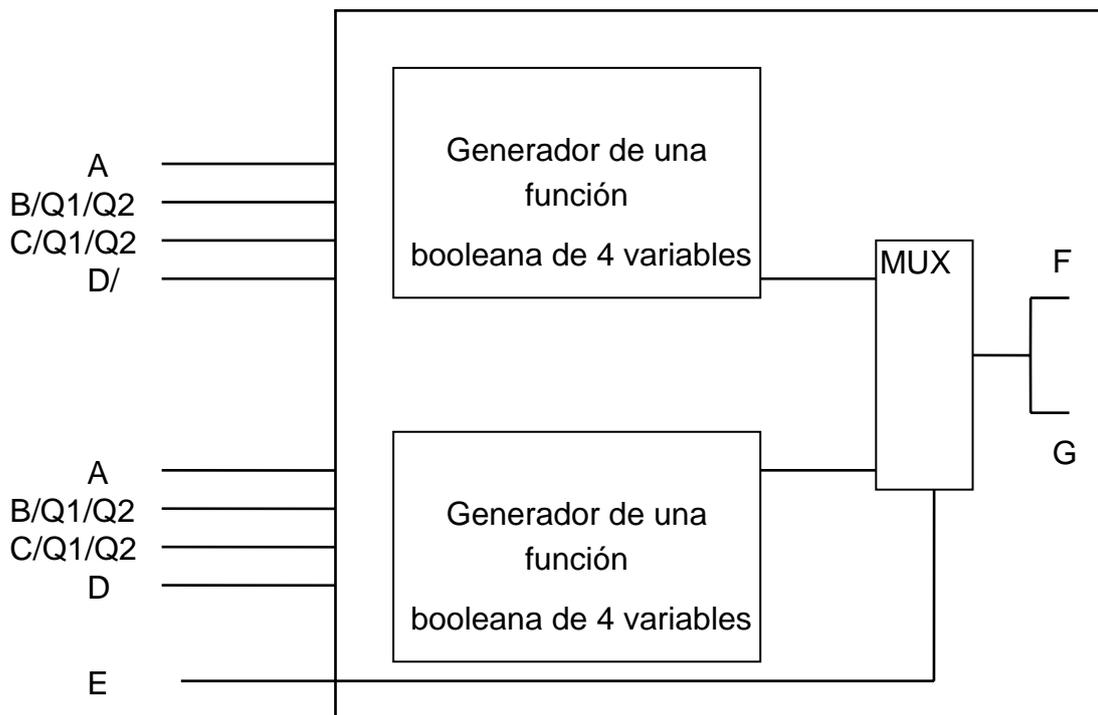


Figura 5-10: Modo FGM de configuración del generador de funciones booleanas en la serie 3000 de Xilinx

Figura 5-11: Estructura general del CLB de la serie 3000 de Xilinx

Por lo que se refiere a los bloques de entrada / salida, éstos están situados a lo largo del perímetro de la FPGA. La cantidad de IOB's que contienen los dispositivos de la serie 3000 va desde los 64 hasta los 144. Su misión es la de conectar los circuitos exteriores con la FPGA. Cada uno de ellos se puede programar para que actúe como bloque de entrada, de salida o bidireccional. En la figura 5-12 se puede ver un esquema del bloque de entrada salida de la serie 3000.

Figura 5-12: Esquema del bloque de entrada salida de la serie 3000.

1.5.3 SERIE 4000

La familia XC4000 de Xilinx proporciona varios de los beneficios de los circuitos custom CMOS de VLSI pero eliminando el coste inicial, los retrasos en el tiempo de diseño y el riesgo inherente al proceso de fabricación de las MPGA convencionales [Xilinx94]. Además esta familia cuenta entre sus miembros con un tipo de dispositivo, los XC4000H, que disponen de los mismos recursos de rutado que la XC4000 pero con el doble de recursos de entrada-salida. Otra característica importante es que se pueden programar tantas veces como se desee y de dos formas distintas:

- Mediante una PROM paralela
- Mediante una conexión periférica.

La serie 4000 presenta una serie de variaciones con respecto a la serie 3000 que citamos a continuación:

- Los CLB's disponen de 2 generadores de 4 entradas independientes que se pueden conectar con un tercer bloque para dar o utilizar una novena entrada.
- Los CLB's disponen de una capacidad de carry aritmético de alta velocidad.
- Los biestables contenidos en el interior del CLB disponen de set - reset asíncrono.
- El número de salidas de los CLB's es cuatro: 2 de los biestables y dos de los bloques combinacionales.
- Los bloques de entrada - salida son más versátiles, pues disponen de opciones de polaridad del reloj.
- Los tiempos de inicialización de los bloques de entrada - salida se pueden programar a la entrada. Se pueden configurar para que sean altos y evitar así problemas de potencial o para que sean cortos y mejorar de esta forma su comportamiento.
- Los bloques de entrada - salida se pueden conectar en parejas para aumentar el soporte de corriente.

- Las FPGA de la serie 4000 tienen 4 decodificadores anchos, es decir con un alto número de entradas.
- EL número de recursos de interconexión es considerablemente mayor que en la serie 3000.
- Todas las entradas y salidas de los CLB tienen acceso a la mayoría de las líneas de interconexión.
- Las matrices de interconexión se han simplificado para mejorar su velocidad. Sin embargo en cierta forma se limita el poder de conexión de los CLB's.
- A lo largo del dispositivo se encuentran ocho redes globales que se pueden utilizar para sincronizar o distribuir señales lógicas.
- El reset global se puede sincronizar con cualquier reloj de usuario

Todas estas mejoras en la arquitectura, así como la modernización de la tecnología hacen que la velocidad a la que puede trabajar un diseño implementado sobre una FPGA sea bastante elevada. El circuito por sí mismo es capaz de soportar frecuencias de hasta 50 MHz. Comparadas con las series más antiguas de Xilinx, las FPGA de la serie 4000 son más potentes ofreciendo una RAM incluida en el chip y los decodificadores anteriormente citados. Cabe destacar también que pueden contener hasta 20.000 puertas equivalentes y que hay 16 tipos diferentes de dispositivos.

Al igual que hicimos con la serie 3000, veremos a continuación una breve descripción de la estructura de sus CLB y de sus bloques de entrada / salida Comenzaremos con el CLB de cuya estructura tenemos un esquema en la figura 5-13:

Figura 5-13: Esquema general del CLB de la serie 4000 de Xilinx

Los elementos principales que contiene un CLB de la serie 4000 son dos biestables (FF) y dos generadores de funciones (GF). Estos generadores de funciones tienen la propiedad de ser independientes y proporcionan al CLB una gran flexibilidad pues la mayoría de las funciones lógicas combinacionales utilizan menos de 4 entradas y de esta forma se aprovechan mejor el conjunto de bloques lógicos. Este es uno de los problemas que se mejora con respecto a la serie 3000.

Los CLB's tienen 13 entradas y 4 salidas para acceder a los GF y a los FF. Hay 4 entradas independientes para cada uno de los GF, F1 a F4 para el primer GF y G1 a G4 para el segundo. Las salidas son F' y G' que están implementadas como LUT's. Hay un tercer GF que puede implementar cualquier función lógica que dependa de F', G', y una entrada adicional (H1). Por lo tanto en un CLB podremos implementar cualquiera de los siguientes tipos de funciones lógicas:

- 2 funciones de 4 variables
- 1 función de 5 variables
- 1 función de 4 variables y otra de 5
- 1 función de hasta 9 variables

Los elementos de memoria en el CLB son evidentemente los biestables tipo D que contiene. Estos FF tienen un reloj común (en la figura K) y una línea global SET/RESET que resetea cada registro en el momento de la inicialización o cuando el diseñador lo crea necesario. Estos FF se pueden activar por flanco de subida o de bajada y se pueden alimentar o bien desde los GF o desde el exterior.

Por su parte los bloques de entrada - salida (IOB's) de la serie 4000 tienen la estructura interna que se puede ver en la figura 5-14. Cada IOB controla un pin del encapsulado. Las señales de entradas pueden pasar directamente o almacenarse en los FF. Los IOB's disponen de un conjunto de resistencias de pull-up y pull-down programables que son muy útiles para unir pines sin utilizar Vcc o tierra y reducir así el consumo de potencia. En la tabla

5-4 podemos ver las características de la serie 4000 en cuanto a número de IOB y de CLB que contienen.

Dispositivo	Nº CLB's	Nº IOB's
XC4002	64	64
XC4003	100	80-160
XC4004	144	96
XC4005	196	112 - 192
XC4006	256	128
XC4008	324	144
XC4010	400	160
XC4013	576	192
XC4020	784	224
XC4025	1024	256

Tabla 5-4: Características de la serie 4000

Figura 5-14 : Estructura general de un IOB de la serie 4000

1.6 SISTEMAS MULTI-FPGA

En la actualidad hay un gran número de sistemas Multi-FPGA y para diferentes aplicaciones[Hauck 95]. Este tipo de sistemas no sólo incluyen los circuitos integrados, si no que además suelen incorporar memorias y circuitería de conexión o condensadores de desacoplo. La diferencias fundamentales que hay entre unos circuitos y otros son dos:

- La disposición de las FPGA
- Los recursos de rutado

Existe una amplia gama de sistemas Multi-FPGA que van desde los más simple que incluyen 2 FPGA's (Xilinx FPGA Demo Board) [Xilinx 95], hasta sistemas con módulos interconectables que se pueden ampliar tanto como sea necesario (Virtual Wires Emulation System) [Tessier 94]. Podemos encontrar también sistemas Multi-FPGA de propósito específico [Grahamp 95]. A continuación veremos una breve descripción de algunos de estos sistemas, incluyendo un esquema de la disposición de sus FPGA's.

1.6.1 XILINX FPGA DEMO BOARD

Es un ejemplo de sistema Multi-FPGA sencillo. La placa incluye una XC4003 y una XC3020, además de una serie de interruptores y un regulador de tensión. Para programar este sistema se utiliza un interface hardware de conexión mediante el cual se le transmiten los datos desde de la computadora donde se este ejecutando el software de programación y prueba (XACT).

Sus ventajas principales son su coste y portabilidad, pues puede funcionar sobre diferentes plataformas. El inconveniente principal es su poca capacidad por lo que el tipo de diseños que puede implementar está muy limitado. En la figura 5-15 podemos ver el aspecto exterior de la placa de demostración de Xilinx.

Figura 5-15: Placa de Demostración de Xilinx..

1.6.2 ESTRUCTURA DE MALLA

Es junto con la estructura de grafo bipartito la más común. Las FPGA están dispuestas en forma de matriz y están conectadas directamente con sus vecinas más próximas en las cuatro direcciones, salvo las de la periferia que estarán conectadas con los bloques de entrada - salida. Tiene dos ventajas fundamentales:

- Simplicidad
- Fácil de expandir

Como desventajas podemos citar la rigidez de sus recursos de rutado y el desperdicio que se hace en ocasiones de los mismos. En la figura 5-16 podemos ver una estructura de malla simple.

Figura 5-16 : Estructura o topología de malla para sistemas Multi-FPGA**1.6.3 ESTRUCTURA DE GRAFO BIPARTITO O CROSSBAR**

En este tipo de sistemas hay una diferenciación entre dos tipos de dispositivos, los que incluyen la lógica del sistema y los que soportan los recursos de rutado. Los circuitos de rutado están conectados únicamente con los circuitos de lógica y viceversa. Por lo tanto tenemos una separación clara entre dos zonas y de ahí su nombre.

La idea es que si necesitamos conectar dos circuitos solamente tendremos que pasar por un único circuito intermedio, mientras que en otro tipo de topología como la de malla tendríamos en algunos casos que atravesar varias FPGA para llegar a la que queremos conectar. Además al estar dispuestos los circuitos de una manera simétrica es indiferente el circuito que escogemos para realizar la conexión pues la distancia será la misma.

Sin embargo esta topología presenta dos inconvenientes. Al tener que estar interconectados todos los circuitos de lógica solamente con los de rutado, no es posible conectarlos con otros sistemas similares y por lo tanto no se puede expandir. Por otro lado es evidente que se produce un mal aprovechamiento de los recursos de la s FPGA al tener la mitad de nuestros circuitos destinados únicamente a la s conexiones. La figura 5-17 es un ejemplo de este tipo de estructuras.

Figura 5-17: Estructura o topología de grafo bipartito para sistemas Multi-FPGA

1.6.4 SISTEMA MP3 DE APTIX

Es un sistema que permite la inclusión de hasta 6 circuitos programables. Esta compuesto por dos tipos de dispositivos con una estructura de grafo bipartito y permite además la inclusión de otros tipos de circuitos integrados o dispositivos para completar la implementación [Aptix97]. Al igual que la tarjeta de Xilinx incluye un software de programación de los dispositivos y un analizador lógico que conecta ambos. Está especialmente diseñado para emulación de sistemas en tiempo real y permite el desarrollo conjunto de funciones hardware y software.

Como principales inconvenientes están primero su capacidad, pues realmente sólo disponemos de 3 circuitos programables para implementar la lógica de nuestro circuito. Segundo presenta todos los inconvenientes anteriormente citados para las topologías de grafo bipartito. En las figura 5-18 se muestra un esquema general del sistema MP3 y su forma de conexión.

Figura 5-18: Sistema MP3 de Aptix

1.6.5 SISTEMA SPRINGBOK

El sistema Springbok ha sido desarrollado en la Universidad de Washington. ES un sistema de prototipado rápido para diseños a nivel de placa. En lugar de hacer la correspondencia con la tecnología con la lógica, el sistema la realiza con los circuitos reales ya diseñados. Esta compuesto por una placa base en la que se conectan los circuitos integrados y los buses de conexión. La ventaja es que estos circuitos integrados además de FPGA's pueden ser memorias, procesadores, elementos de entrada - salida, interruptores o cualquier otro tipo de elemento que sea necesario para la implementación del circuito. Adicionalmente la placa base se puede conectar con otras placas iguales mediante unos pines de conexión laterales. A estas placas cuando se conectan varias, se les llaman placas hermanas [Hauck 94].

Es una estructura de tipo malla y por lo tanto presenta los inconvenientes de las mismas. Sin embargo es un sistema que posibilita una gran capacidad de comunicaciones entre los circuitos debido a su estructura de la placa base en la que las conexiones son directas. En la figura 5-19 tenemos una vista exterior de este sistema.

Figura 5-19: EL sistema Springbok. En los laterales se pueden apreciar los puntos de conexión con las placas hermanas

2. EL DESIGN FRAMEWORK II DE CADENCE

2.1 CARACTERÍSTICAS

El Design Framework II de CADENCE (DFWII) es un entorno de trabajo que facilita el acceso a herramientas de diseño automático de circuitos integrados. Estas herramientas pueden ser o no ser de CADENCE. Sus principales características son:

- **Portabilidad del software.** Puede correr sobre las estaciones de trabajo de Apollo, HP, NEC o SUN, entre otras.
- Posee un **interface de gráfico consistente** basado en el manager Motif de Windows. La manera de interactuar con el sistema es común a todas las aplicaciones, tanto si se trata de introducir diseños, como si se trata de ejecutar análisis.
- Posee una **base de datos unificada** lo que elimina las conversiones de datos entre las diferentes herramientas.
- Su **entorno es configurable, modular y versátil.** Gracias a él se pueden integrar nuevas herramientas con facilidad. Éstas pueden interactuar con el resto de las herramientas existentes.
- Utiliza el **lenguaje SKILL**, un lenguaje de programación de alto nivel muy poderoso .

2.2 NECESIDADES DE SOFTWARE Y HARDWARE

El DFVII es la capa más externa de un conjunto de capas de software que se apoyan en una plataforma de hardware. Las distintas capas de software son:

- El sistema operativo UNIX,
- El entorno X-Windows
- El administrador de ventanas Motif.

El sistema X-Windows capacita a la estación gráfica para mostrar una o varias ventanas al mismo tiempo. En cada ventana se puede ejecutar una aplicación independientemente del resto. Las ventanas que más interesan al usuario son:

- v **CIW** .- (Comand Interpreter Window) es la ventana principal que CADENCE mantiene abierta. Representa el entorno DFVII y proporciona un acceso a las herramientas a través de sus menús y de su línea de ordenes. En ella se pueden usar menús o comandos SKILL para crear, modificar e imprimir diseños. También se pueden crear entornos a medida o modificar la base de datos.

nota: En las prácticas esta ventana deberá permanecer abierta. En ella aparecen los mensajes de warning y de error, y en general , las respuestas del entorno a todas las acciones del diseñador, así como la descripción de las acciones que realiza el DFVII

- **Consola**.- es la primera ventana que se abre. No conviene trabajar en la consola. Se utiliza para introducir comandos en caso de que el sistema tenga problemas.

2.3 HERRAMIENTAS QUE CONTIENE EL DWFII

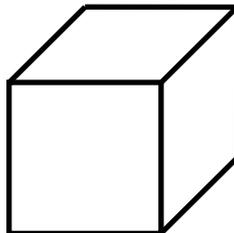
- Captura de esquemas
- Generación de símbolos
- Simulación eléctrica (HSPICE)
- Simulación lógica (VERILOG)
- Simulación de fallos (VERIFault)
- Análisis de tiempos críticos (VERITIME)
- Síntesis lógica (SINERGY)
- VHDL
- Editores de layout (VIRTUOSO)
- Verificadores interactivos (DIVA)
- Place & Route

3. COMO SE ORGANIZA UN DISEÑO

3.1 ESTRUCTURA DE UN DISEÑO

Un diseño se identifica mediante los siguientes elementos:

- una librería (library)
 - una celda (cell)
 - una vista (view)
 - un número de versión.
- v **Celda (cell).**- Es el objeto básico de diseño y representa una función electrónica. Esta función puede ser tan sencilla como un transistor o tan compleja como un microprocesador. Suelen utilizarse nombres que reflejan su función. Conviene recordar que una celda no se corresponde con un archivo físico de la base de datos sino que es un objeto lógico de diseño.



- v **Vista (View).**- Una celda puede tener diferentes representaciones. A cada una de ellas se la llama vista. Los tipos de vista más habituales son *schematic*, *symbolic*, o *layout*. Si consideramos el cubo de la figura como una celda inversor, cada uno de los lados de que consta se puede considerar una vista. Una vista se caracteriza por:

- Una propiedad (viewtype)
- Un nombre
- Atributos de lectura /escritura, unidades de grids y escala

- Propiedades adicionales como el color
- Ser un objeto lógico de diseño.

Existe una gran variedad de nombres de vistas. Estos nombres se hacen corresponder con tipos de vista en el archivo de tecnología. A continuación vemos una tabla que relaciona nombres de vista con sus tipos:

VIEW NAME	VIEW TYPE
abstract	maskLayout
autoLayout	maskLayout
behavioral	netlist.v
cmos.sch	schematic
flatten	maskLayout
Layout	MaskLayout
msymbol	schematicSymbol
routed	maskLayout
schematic	schematic
symbol	schematicSymbol

tabla1

- v **Cellview** .- Es un fichero de datos virtual que asocia una celda con una vista. Es un objeto lógico de diseño. El fichero asociado a una celda definida mediante una vista puede ser de tipo ASCII o del tipo binario CDBA (Cadence Database Acces).

En la Tabla 2 aparecen diferentes tipos de vista con el tipo de fichero que les corresponde y la aplicación de CADENCE sobre la que trabajan.

Viewtype	file type	aplicación
masklayout	CDBA	Graphics Editor
Layout	CDBA	Graphics Editor
Schematic	CDBA	Graphics Editor
Symbolic	CDBA	Graphics Editor
Behavior.vhdl	CDBA	VHDL

abel	CDBA	PLD
behavioral	CDBA	HDL
dataflow.vhdl	CDBA	VHDL(Architecture)
entity.vhdl	CDBA	VHDL(entity)
mixed.vhdl	CDBA	VHDL(architecture)
package.vhdl	CDBA	VHDL(package)
stimulus.vhdl	CDBA	VHDL(architecture for stimulus)
system	CDBA	HDL
verilog	CDBA	PLD

- v **Versión** .- Es el fichero de datos actual de un determinado cellview. Está restringido a una librería. Si no se le indica nada, CADENCE toma su valor por defecto
- v **Categoría de celdas**.- Es un grupo lógico de celdas. La categoría de las celdas no afecta a la localización física del diseño, pero se puede acceder a ellas como parte de una categoría. Una celda puede pertenecer a varias categorías. Ejemplo de categorías son la categoría CMOS que incluye puertas de tecnología CMOS o la categoría INVERSOR que incluye inversores en diferentes tecnologías. Darse cuenta que un inversor CMOS pertenece a ambas categorías.
- v **Jerarquía**.- Son los diferentes niveles de un diseño. Para crear una jerarquía hay que instanciar un diseño en el interior de otro. Físicamente un diseño no contiene un copia de otro, contiene una instancia y los datos permanecen en su localización original. De manera informal se puede definir una instancia como una especie de puntero a un determinado diseño .
- v **Librería**.- Una librería es un directorio que contiene una colección de celdas y cellviews. Cuando se crea una librería, el editor de esquemáticos crea un fichero que relaciona celdas lógicas, cellviews y nombres de versión con localizaciones físicas. Este fichero se llama *LibName*. La información que describe una librería se encuentra en el archivo de tecnología. Existen dos tipos de librerías:
 - **librerías de diseño**. Contiene una colección de celdas y views que describen un diseño. Se pueden leer y escribir.
 - **librerías de referencia**. Contienen celdas y vistas bien verificadas. Sólo se pueden leer.

A continuación aparece una tabla con diferentes librerías de referencia que vienen con CADENCE:

NOMBRE DE LIBRERÍA	DESCRIPCION DE LOS DISEÑOS
CMOS	De propósito específico
sample	Librería de referencia CADENCE
basic	Librería de referencia CADENCE

Antes de abrir un diseño, se debe colocar el camino de búsqueda de la librería en la que se encuentra para que el software de CADENCE pueda encontrarla. Si no se especifica el path de búsqueda CADENCE lo localiza por defecto en uno de los dos directorios siguientes:

- ~/.cdsinit
- el directorio desde el cual se ha arrancado el CADENCE

NOTA: Debido a que las librerías y diseños de CADENCE no son objetos físicos sino lógicos no se debe usar nunca comandos UNIX para manejar los archivos. Siempre se deben utilizar los comandos de SKILL, los menús del DFVII o el library browser. Los comandos UNIX no actualizan correctamente la base de datos.

3.2 COMO USAR EL LIBRARY BROWSER

El *Library Browser* permite al usuario moverse a través de las librerías y expandir versiones de cellviews que se encuentren dentro del camino de búsqueda de la librería. Vamos a describir a continuación los pasos que hay que seguir para trabajar con ella. (Nota: botón de la izquierda del ratón=I, botón de la derecha = D, botón del centro =C)

- 1) Para ver la library browser (LB) pulsar con I en BROWSE. Aparecen las librerías que se encuentran en el camino de búsqueda especificado. Debajo del título de menú del (LB) aparece un mensaje que indica el nivel de jerarquía en el que se encuentra. Si el mensaje que aparece es "LIB" esto indica que se está mostrando el nivel de las librerías.

Pulsando en la opción *COMANDO* del MENÚ se obtienen las siguientes alternativas:

- **show edited cellviews**- lista todas las cellviews de cualquier librería visible.
- **center**.- centra el listado de las librerías
- **unexpand**.- elimina los niveles de jerarquía inferiores
- **purge versions** .-. elimina versiones de librería
- **close browser**.- cierra y elimina una ventana de browser

- 2) *Para listar las celdas y las categorías de celdas de una librería se pulsa con I en el nombre de la librería. Las celdas y las categorías tienen diferentes colores para que puedan distinguirse. El nivel de jerarquía del library browser es LIB CELL.*

- 3) *Para ver los views de una librería hay que pulsar la opción EXPAND VIEWS del menú de librería. Este menú se obtiene pulsando C.*

También se pueden mostrar los views pulsando con I sobre el nombre de una celda.

- 4) *Para ver las celdas que aparecen en una categoría, o las categorías a las que pertenece una celda se pulsa con I en una categoría. Si se pulsa C aparece el menú correspondiente al nivel de celdas o a las categorías de celdas.*

El nivel de jerarquía del library browser es LIB CELL CELLVIEW y en el se muestran las librerías las celdas y las vistas.

- 5) *Para ver las versiones se pulsa I en una cellview. Pulsando C aparece el menú de cellviews. El nivel en el que nos encontramos es el LIB CELL CELLVIEW VERSION. Pulsando C aparece el menú del nivel de versiones.*

NOTA: Si se desea eliminar un diseño, no se debe hacer nunca desde UNIX. Para hacerlo utilizamos el library browser. Se selecciona con el puntero el diseño que se desea borrar. Pulsando con el botón central del ratón aparece un menú. Se selecciona la opción DELETE

4. EL ARCHIVO DE TECNOLOGÍA

El archivo de tecnología contiene información asociada con la librería. El sistema compila el archivo y lo asocia a cada librería que se crea. Este archivo está escrito en el lenguaje SKILL y se encuentra situado en el directorio :

`<cds_install_dir>/samples/techfile/default.tf.`

Alguno de los aspectos de la librería que se definen en el archivo de tecnología son:

- Nombre de layer.- Background, grid,axis.
- Propósito del layer.- Dibujo, nodo, frontera, warning.
- Número de layer.- 0..127 para el usuario ,128..255 reservados para CADENCE.
- Colores
- Estilo de líneas.
- Views y sus propiedades
- Reglas de diseño físico
- Recursos como transistores, pines y contactos

Los archivos de tecnología son de tipo binario. Para poder editarlos se realiza un “*dump*”, que consiste en convertirlos a código ASCII. Estos archivos se componen de una serie de comandos de SKILL. Se pueden ver algunos ejemplos de archivos de tecnología en el directorio *install_dir /samples/*.

Al generar una librería, se le asocia por defecto el archivo de tecnología *techfile.deff* situado en el directorio *insatall_dir/etc*. Este fichero de tecnología define

- el tipo de display que se tiene (estación de trabajo o terminal)
- los colores disponibles, las líneas de los patrones y los punteados de los patrones

- los layers usados por las aplicaciones, tales como el color por defecto que usan de los objetos seleccionados.

El fichero no define recursos, hilos o layers de diseño (tales como nwell layers).

Un layer es una representación visual (un color) de diferentes tipos de información. Los layer proporcionan una visión personalizada de la información. A esto se lo define como un par *layer/propósito*. Se pueden tener varios layer con el mismo nombre pero diferente propósito.

v **¿Como mostrar los layers en forma tabular?**

1. Seleccionar TECHNOLOGY FILE del menú de la ventana CIW
2. Selecciona LAYERS
3. Seleccionar LAYERS INFORMATION. Se puede seleccionar número, orden y prioridad

v **¿Como mostrar layers en formato gráfico?**

1. Seleccionar TECHNOLOGY FILE del menú de la ventana CIW
2. Seleccionar LAYERS
3. Seleccionar el comando OPEN LAYERS

Tras estas operaciones aparecen un conjunto de iconos cada uno de los cuales se refiere a un determinado layer. Cada icono se divide en tres partes:

- color,
- nombre
- propósito.

A continuación del icono aparecen dos casillas. La primera sirve para seleccionar el layer y la segunda, llamada visibilidad, para determinar si se puede ver.

5. BIBLIOTECAS DE CELDAS ESTANDARES

ES2 proporciona dos bibliotecas de celdas estándares cada una de ellas con una tecnología de fabricación diferente:

- ECPD10 de 1μ
- ECPD07 de 0.7μ

En las prácticas trabajaremos con la biblioteca ECPD07.

v CARACTERÍSTICAS GENERALES:

- Fabricante (*foundry*): ES2 (*European Silicon Structures*)
- Tecnología : $0.7\mu\text{m}$ CMOS
- Aplicaciones: diseño digital ASIC.

v DESCRIPCIÓN DE LA TECNOLOGÍA

- Dos layers de metal
- Un proceso de polisilicio sencillo.
- No existen layers disponibles para resistencias ni capacidades
- Anchura de canal $0.7\mu\text{m}$
- $V_t(\text{PMOS})=-1.06\text{V}$
- $V_t(\text{NMOS})=0.82\text{V}$

v BIBLIOTECA DE CELDAS ESTANDARES

Biblioteca digital

- 68 celdas estándares que incluyen puertas lógicas, biestables , mux...
- megaceldas compiladas de memorias RAM, DPRAM, ROM, multiplicadores.
- Retardo típico para una nand de dos entradas si capacidades de carga 0.18ns

- Tamaño de una puerta NAND de dos entradas $380\mu\text{m}^2$
- Rejilla de rutado $2.8\mu\text{m}$
- Alimentación a 5V

Librerías analógicas,

- Contiene muy pocas celdas. La mayoría destinadas a la generación de interfaces.

v SOPORTE CAD- PAQUETES DE DISEÑO

CADENCE frame-work

- Front-end (entrada de esquemáticos, y simulación VERILOG)
- Back-end (ensamblado de celdas y localización y rutado)
- Archivos de tecnología para diseño fullcustom
- Extracción y generación de netlist SPICE para diseño fullcustom.
- Reglas de diseño interactivas DIVA.

SOLO 1400

- diseño de celdas estándar desde esquemático a layout.

Synopsis

- Síntesis y optimización
- Transferencia de listas de nodos de CADENCE a mentor
- Trabaja con VHDL

Mentor v8.2

- Front-end (entrada de esquemático y simulación quicksim)
- Autologic
- Back-end (place and route)
- Reglas de chequeo, extracción generación de netlist de spice
- megaceldas(RAM ,ROM ,FIFO, DPR, multiplicadores)

6. COMO SE CREAN Y RECUPERAN DISEÑOS

6.1 COMO ABRIR UNA LIBRERÍA DESDE EL MENÚ OPEN

Marcar con I la opción OPEN del menú principal y seleccionar LIBRARY. Aparecerá una ventana de dialogo .

1. Escribir el nombre de la librería
2. Teclar el path de la librería. Se puede teclar el camino absoluto (/ dir1/dir2/dir3) o el camino relativo (../ dir3)
3. Abrir la librería en modo lectura.

NOTA:

El paso 3 lo realiza el sistema por defecto. Sólo se debe realizar en el caso de querer acceder a una librería diferente.

El paso 4 sólo cuando se quiera que la biblioteca no se pueda modificar.

NOTA: La librería sólo hace falta crearla la primera vez. Una vez creada este paso NO hace falta repetirlo al comienzo de cada sesión.

6.2 COMO ABRIR UN DISEÑO DESDE EL MENÚ OPEN

1. Pulsar en la opción de menú **OPEN**
2. Seleccionar **DESIGN**. Si no hay ninguna librería en el camino de búsqueda aparecerá un ventana de dialogo que lo indica
3. **Selección de la Biblioteca**. Usar el botón BROWSER de la ventana de dialogo para mostrar la *library browser*. Si se pulsa con I sobre una determinada librería, ésta se selecciona automáticamente.

También se puede hacer escribiendo el nombre directamente

4. **Selección de la celda**. Seleccionar en el directorio mostrado en la *library browser* la celda de diseño deseada con la tecla del ratón I.

También se puede escribir el nombre de la celda directamente.

5. **Selección de la vista (view)** . Seleccionar en el directorio mostrado en la *library browser* la la vista del diseño deseada con la tecla del ratón I.

También se puede escribir directamente

6. **Número de la versión**. Sólo se puede editar una versión del diseño al tiempo. Si se deja en blanco se selecciona automáticamente la versión más reciente.(OPTATIVO)
7. Seleccionar el **modo de acceso**. Los modos pueden ser edit, read y overwrite. Tener cuidado al utilizar la opción overwrite, porque se puede eliminar totalmente la versión.(OPTATIVO)
8. Indicar el **nivel de jerarquía** con el que se quiere trabajar. Por defecto se selecciona el nivel de jerarquía superior (OPTATIVO)
9. Seleccionar **OK**. Aparece **una ventana de diseño**, que varía según la aplicación que se esté utilizando. Para cada aplicación hay un conjunto diferente de comandos

6.3 OBTENCION DE INFORMACIÓN DE LA VENTANA DE DISEÑO

Desde la ventana de diseño se pulsa con I el MENÚ de CADENCE

1. Pulsar en OPCIONES
2. Pulsar SHOW ENVIRONMENT

NOTA: el slash (/) en el camino de jerarquía indica que nos encontramos en los alto una jerarquía.

7. CAPTURA DE ESQUEMÁTICOS

7.1 INSTANCIAS

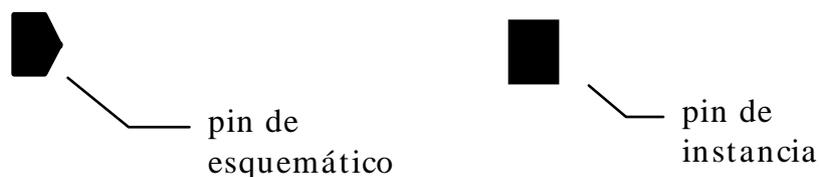
Un esquemático puede incluir los siguientes elementos:

- instancias
- pines
- hilos (nodos, buses, y bundles)
- etiquetas
- bloques
- puntos de soldadura
- notas de figuras

Una instancia es un símbolo localizado en una vista de tipo Schematic. Cada instancia tiene un nombre único, y se relaciona dinámicamente con el símbolo maestro. Por ejemplo, dentro de un esquemático que representa un biestable, se pueden instanciar inversores diseñados y simulados en un diseño aparte.

El símbolo maestro es una celda de tipo *symbol* que tiene asociada una celda de tipo esquemático. Esta última describe el comportamiento eléctrico del símbolo. Si se modifica el maestro se modifican todas sus instancias. CADENCE proporciona varias librerías de referencia de símbolos maestros. Cada símbolo representa un elemento electrónico.

Los pines de entrada y salida de la instancia no deben confundirse con los pines de entrada y salida de los esquemáticos. Los primeros proporcionan conectividad entre la instancia y el resto del esquemático.



A las instancias se le pueden modificar las propiedades. De esta manera se pueden tener dos instancias del mismo maestro con diferentes propiedades. Si se quiere que todas las instancias tenga las mismas propiedades, éstas se deben incluir en el símbolo maestro.

v **Iteración de símbolos.**- es una manera compacta de repetir instancias de un símbolo. Es particularmente útil en los tipos bus que se caracterizan por utilizar la misma estructura para cada bit del bus. Si se quiere que una instancia aparezca M veces, se le da el nombre *nom<M-1:0>*. Cuando se utiliza la iteración de símbolos se aplican las siguientes reglas:

1. Si un hilo de un bit conecta con un pin del símbolo iterado, la señal llega al pin de cada instancia. Internamente el bit se itera para emparejar con el número de instancias creadas.
2. Si un hilo multibit (bus, bundle) conecta con un pin del símbolo iterado, el número de hilos del multihilo debe emparejar con el número de instancias.

v **Arrays de símbolos.**- La misma instancia se repite un número determinado de veces en el mismo esquemático. En la ventana de dialogo que aparece al instanciar componentes existe la posibilidad de elegir el número de filas y columnas del array.

7.2 SÍMBOLOS

Un símbolo es la representación gráfica de un interface entre el nivel de jerarquía actual y el inferior. Se crean símbolos para instanciar un esquemático en otro diseño. De esta manera se asciende en la jerarquía. Cuando se instancia sus detalles internos permanecen ocultos.

Una de las ventajas de trabajar con símbolos es la consistencia. Si se necesita cambiar el símbolo, solo hay que modificar el símbolo maestro, y el editor de esquemáticos aplica los cambios a todas las instancias que se hayan realizado. Un símbolo consta de los siguientes elementos:

- **Pins.**- realiza la conexión eléctrica del símbolo con el resto del esquemático.
- **Etiqueta** .-(*Labels*) Identifica el símbolo, el pin y las propiedades eléctricas
- **Líneas** .- Conecta los pines con las figuras (*Shapes*)
- **Figuras.**- (*Shapes*) Representan partes abstractas del símbolo. Pueden ser círculos, cuadrados, rectángulos, líneas, polígonos, arcos y elipses.
- **Cajas de Instancia.**- Líneas exteriores que marcan el área seleccionable del símbolo.

v Generación automática de símbolos

Los símbolos se pueden crear manualmente o automáticamente. Para crearlo automáticamente desde la ventana del esquemático se seleccionan encadenadamente las siguientes opciones

DESIGN, CREATE CELVIEW.

En este momento aparece la ventana de diálogo *frame cellview* que pregunta por la *cellview* origen y la destino. La origen debe ser la de tipo esquemático y la de destino de tipo *symbol*.

v **Generación manual de símbolos**

Para generar el símbolo manualmente se abre un nuevo diseño con el mismo nombre de celda y el tipo de vista *symbol*. En este instante aparece una ventana de diseño que proporciona todas las facilidades para dibujar el símbolo.

NOTA: Es importante recordar que los pines deben tener los mismos nombres y sentidos de entrada o salida que los del esquemático.

El dibujo, una vez realizado, se chequea y se salva. Cada vez que se instancie un símbolo se tendrá como circuito eléctrico asociado el incluido en el esquemático de la celda.

NOTA :a la hora de diseñar un símbolo se deben tener en cuenta dos cosas. La primera comenzar todo símbolo con los pines. de esta manera se consigue que el símbolo guarde proporcionalidad con el resto de los esquemáticos. En segundo lugar utilizar siempre pines del tipo *square*.

7.3 HILOS, NODOS, BUSES Y “BUNDLES”

- v **Hilo** (wire) es una línea entre elementos del esquemático que se puede dibujar manual o automáticamente.
- v **Bus** es un hilo que lleva un conjunto de señales del mismo tipo. Para etiquetarlo se siguen las siguientes reglas:
 - A un bus de M líneas se le etiqueta como *bus<M-1:0>*. P.e un bus dato de 16 líneas se etiqueta *dato<15:0>*
 - Cada señal del bus se distingue con el nombre y el número de línea; por ejemplo *dato<6>*.
 - Para representar un subconjunto del bus se utiliza el nombre y ,entre ángulos, los números de los bits separados por comas. P.e. *dato<3,5,6>*.
- v **Bundle**.- Es un hilo que contiene un conjunto de señales de diferentes tipos. Para conseguirlo se listan los nombres separados por comas en la etiqueta . Por ejemplo, la etiqueta A,B,C , nombra un hilo de tres bits que lleva las señales A,B y C , en este orden.
- v **Conexiones al bus**.- De un bus o un bundle se pueden derivar subconjuntos de señales. Para realizar la conexión se hace lo siguiente:
 - 1.- conectar un hilo
 - 2.- etiquetarlo

Para hacer la conexión a un bus es suficiente especificar el número del bit en la etiqueta del hilo. Por ejemplo, del bus DATA<15:0> puede derivar la conexión DATA <3>. Para derivar una señal de un bundle se debe especificar el nombre de la señal en el nuevo hilo.
- v **Derivaciones en cascada**.- Consiste en derivar un subconjunto de bits del que a su vez se derivan subconjuntos. La forma de conseguirlo es nombrar las derivaciones sin utilizar el nombre base. Es suficiente con especificar qué subconjunto de bits es el que se deriva. Lo que sí es importante que el bus principal esté etiquetado con el nombre.
- v **Nodo (net)** .- Es un unión entre un pin y una instancia.
- v **Nodo global**.- Es un nodo que establece conectividad entre diferentes niveles de jerarquía. Para indicar que un nodo es global, añadir una

exclamación al nombre, por ejemplo vdd!. Se suelen definir como nodos globales:

- la tensión de carga del circuito,
 - la tierra del circuito
 - la señal de reloj que sincroniza el sistema.
- Para conectar un nodo con un pin sin utilizar hilos, se llama el nodo con el mismo nombre del pin.
 - Por defecto los nodos tienen el nombre del pin.
 - Si se quiere cambiar el nombre de un nodo unido a un pin, se debe etiquetar el nodo .
 - Para unir nodos diferentes, sin hilos basta con llamarlos por el mismo nombre.

v **Un patchcord** es un símbolo que se usa para conectar de manera rápida y provisional dos nodos. Se suele usar para corregir errores o realizar tareas adicionales. La conexión correcta se debe realizar con posterioridad. Una de las funciones es la de conectar nodos con diferentes nombres.

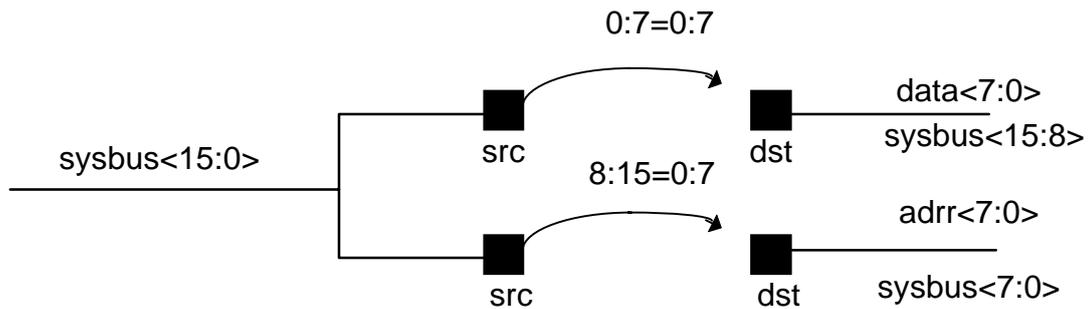
Este elemento se encuentra en la librería BASIC, en la categoría de celdas MISC y se llama **patch**. La forma que tienen es la siguiente:



La propiedad llamada **_schPatchExpr** muestra el modo en que los bits que están conectados al nodo fuente (src) se conectan a los bits conectados al nodo destino (dst) . El valor de esta propiedad tiene la siguiente forma:

$$\text{src_vector_expresion}=\text{dst_vector_expresion}$$

Donde **vector_expresion** tiene la misma sintaxis vista para los hilos con múltiples bits (pg 4.5). El número de bits en el vector src debe ser igual al número de bits de dst. El patch conecta el primer bit del nodo fuente con el primer bit del nodo destino.



La información que contiene la propiedad `schPatchExpr` **NO** es los bits del bus de sistema que se conectan con los buses de datos o dirección, lo que indica es el **ORDEN DE CONEXIÓN**. El orden del bit y el número de bit **NO** coinciden, por ejemplo en el `sysbus<15:0>` el bit 15 tiene el orden 0, el bit 14, el orden 1,..., el bit 0 el orden 15, por eso en la figura los valores de la expresión aparecen invertidos. En el caso de `data<7:0>` el bit 7 tiene orden 0 y el bit 0 tiene orden 7. Esto es muy importante para que funcione correctamente el patchcord.

NOTA: Los nombres de los hilos no deben coincidir con los nombres de los pines de las instancias maestras. Esto produce warnings y da problemas a la hora de realizar la netlist.

7.4 PINES Y TERMINALES

Un **pin** define las terminales de entrada y salida de un esquemático. Obligatoriamente se le debe dar un nombre cuando se crea.

Para crear un pin se acude al icono correspondiente del menú vertical de la ventana de esquemáticos. Se selecciona y aparece una ventana de dialogo en la que hay que introducir obligatoriamente el nombre del pin. También hay que seleccionar la dirección que indica si el terminal es de salida, de entrada o ambas cosas:

- input
- output
- input/output

Un pin se puede conectar a una señal sencilla o a una múltiple. La manera de nombrar los pines es similar a la utilizada para nombrar los hilos. Los buses e hilos que no se nombren toman el nombre y la anchura del pin al que se conecten.

Para aumentar la claridad del diseño, se puede usar más de un pin para representar el mismo terminal.

Pines multihoja (multisheet).- Cada hoja del diseño multihoja puede contener dos tipos de pines diferentes:

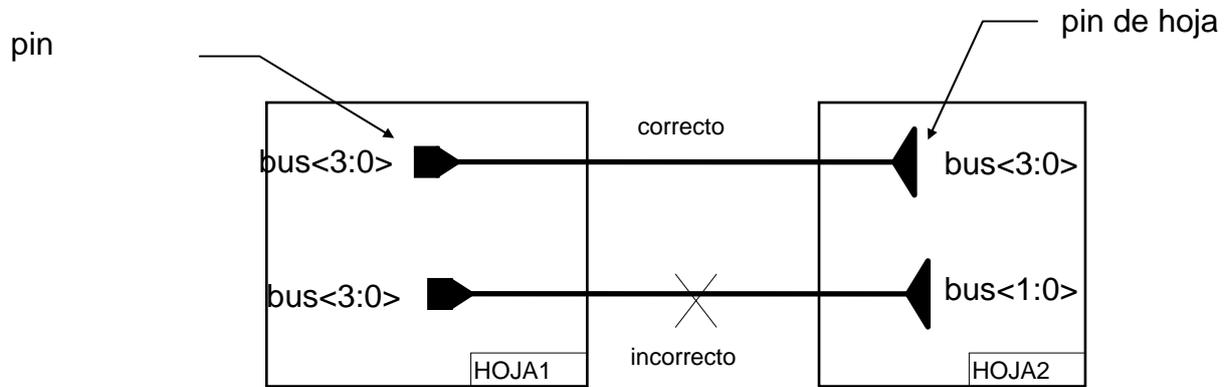
- *Pines jerárquicos.*- son los que aparecen en un símbolo del diseño. Ayudan a realizar las conexiones con los niveles superiores de jerarquía
- *Pines de hoja.*- Son los que conectan un hoja con la siguiente.

Los pines multihoja deben cumplir las siguientes reglas:

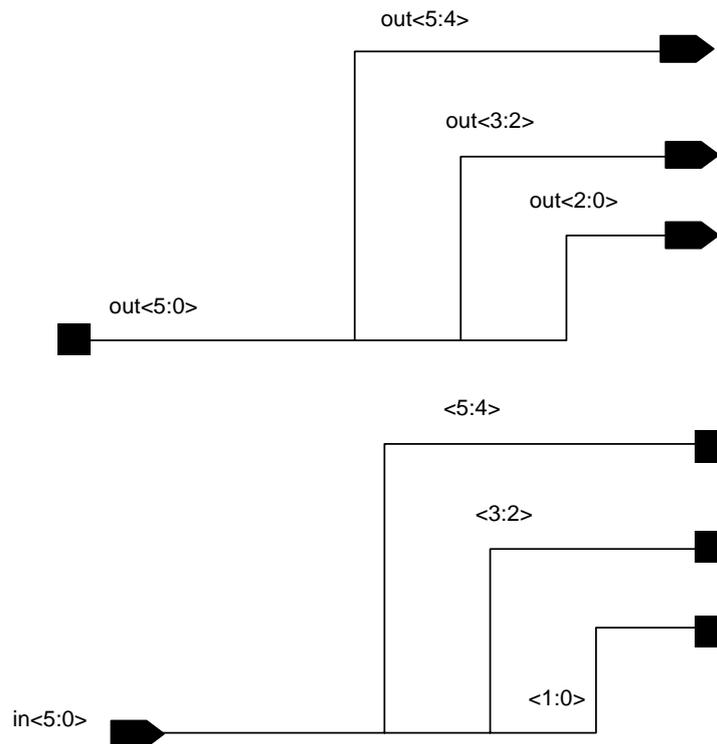
- los pines que conectan hojas deben tener el mismo nombre.
- el orden de los bits no es importante. Se puede conectar el pin BUS<3:0> al BUS<0:3>.
- un pin jerárquico puede aparecer únicamente en una hoja. Si se necesita que aparezca más de una vez,

lo que se hace es declarar uno de ellos como jerárquico y el resto como multihoja.

- si el pin de hoja se utiliza para que un bus conecte dos hojas, debe tener tantos bits como el pin jerárquico al que se conecta



v **Conexiones de pines a buses**



7.5 JERARQUÍA Y MULTIHOJAS

7.5.1 Jerarquía

Una forma de ir generando una estructura jerárquica ascendente es crear símbolos asociados a esquemáticos, e instanciarlos en otros esquemáticos.

El DFVII proporciona herramientas para moverse a lo largo de la jerarquía. La forma de acceder a ellas es la siguiente:

1. selecciona DESIGN del menú de la ventana de diseño
2. selecciona HIERARCHY. Dentro de esta opción aparece:
 - Descend edit
 - Descend read
 - Edit in place
 - Return

NOTA: Utilizando este menú se puede modificar y leer el master de cualquier instancia trabajando desde otro diseño de jerarquía superior.

NOTA: Se debe recordar que cualquier modificación en el master supone la modificación de todas sus instancias en cualquier diseño.

7.5.2 Multihojas

Cuando un diseño no entra en una sola hoja se puede utilizar la herramienta multisheet. Esta consiste en utilizar varias hojas para implementar el diseño. Cuando se generan multihojas, el editor de esquemáticos trata los esquemáticos multihoja como un conjunto de diseños individuales. No hay límite en el número de hojas que se pueden almacenar. Cada hoja se representa y describe en un índice como una instancia de hoja. El editor genera automáticamente los nombres de las instancias. La instancia de cada hoja contiene la siguiente información:

- Título
- Tamaño de la hoja
- Número de hoja
- Nivel de revisión

La utilización de multihojas es la otra forma de implementar jerarquías. El editor de esquemáticos implementa un esquemático multihoja utilizando dos niveles de jerarquía: el nivel superior y el inferior.

- el nivel superior contiene un índice de todas las hojas que forman la descripción multihoja.
- el nivel bajo contiene el esquemático contenido en cada hoja.

Generación de multihojas

En el menú horizontal de la ventana de esquemáticos debe aparecer una opción SHEET.

Seleccionar la opción create del menú sheet

aparece una ventana de dialogo con el mensaje:

Converting single sheet schematic to multisheet schematic

Puede ocurrir que no entre en una hojas todo el diseño y que haya que repetir el proceso previa selección de una determinada zona del diseño.

Pines multihoja.- Cuando se utilizan varias hojas para el diseño jerárquico de un subsistema, un esquemático contiene pines jerárquicos y de hojas. Los pines de hojas son los que se utilizan para conectar hojas entre sí. Los pines

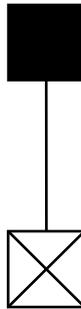
jerárquicos son los que se utilizan para localizar una instancia del multihoja en un nivel de jerarquía superior. Para poder utilizar un índice de esquemáticos en un diseño de mayor nivel hay que hacer lo siguiente:

1. añadir los pines jerárquicos que conectan la multihoja con el resto del diseño.
2. generar el símbolo.

7.6 CELDAS NOCONN Y LOG2

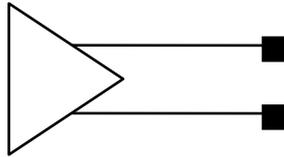
7.6.1 NoConn

En muchas ocasiones al hacer un diseño hay que dejar pines de un determinado símbolo sin conectar. Para evitar que al comprobar la conectividad mediante el comando *check and save* se produzcan warnings, se utiliza un elemento llamado noConn de la librería BASIC y la categoría de celdas misc, y cuyo símbolo es el siguiente:



7.6.2 LOG2

En ocasiones puede ser necesario introducir un valor lógico constante a un determinado esquemático, para ello se utiliza la celda estándar llamada log2 cuyo símbolo es:



Esta celdas no se debe utilizar NUNCA para realizar alimentaciones a las megaceldas.

8. SIMULATION AND TEST LENGUAJE (STL)

Se define como un lenguaje de alto nivel pensado para desarrollar estímulos de simulación y test. Sus principales características son:

- Es un lenguaje sencillo debido al pequeño número de comandos que lo componen y a la regularidad de sus sintaxis.
- Se basa en skill
- soporta la generación de :
 - * Estímulos de simulación
 - * Formateado de vectores de test
- Permite comparar los resultados de la simulación con los valores esperados.
- Proporciona interfaces para :
 - * Simuladores lógicos silos, hilo, cadart y verilog
 - * Simuladores eléctricos spice, hspice y sage
 - * Generación automática de test como sentry, genrad advantest y ando.

8.1 ESTRUCTURA

Los comandos se dividen en dos grupos,:

- **Comandos de definición** que definen los parámetros que necesita el programa para realizar la simulación
- **Comandos de ejecución** que determinan cuáles son los valores de las entradas que se quieren simular.

v Comandos de definición:

- Inicialización (stlint)
- Definición de nodos (defpin, defformat)
- Definición de tiempos (deftiming, defstrobe, defclock, deftimset , endtimset.)

v Comandos de ejecución

- Comienzo de ejecución (deftest)
- Vectores funcionales (xv,v,#, rptv)
- Fin de ejecución (endtest.)

8.2 COMANDOS

v **stlinit** .- Inicia la estructura de datos interna usada por el STL. Se debe utilizar siempre al principio del programa.

v **defpin** .-Define los nombres de las señales y sus características

defpin <nombre> [<dimension>]<tipodepin>[internal][<atributo>][<pinlist>]

argumentos:

- *nombre* es el nombre definido para el usuario para el grupo de pines. Debería coincidir con el nombre dado en el esquemático.
- *dimensión* (opcional) es el número de bits o el rango del bus. Tiene un máximo de 32 bits. El número por defecto es 1. El rango se especifica n1:n2 y se puede colocar entre ángulos.
- *tipodepin* describe la funcionalidad del pin: in, out, io, clk, pwr, gnd.
- *internal* (opcional) indica que el nodo es interno. Se usa sólo en la simulación.
- *atributo*.- (opcional) Especifica atributos utilizados en la simulación eléctrica y en la simulación lógica. Entre estos atributos destacan los tiempos de subida y de bajada para un nodo de entrada. Por ejemplo, suponiendo que un nodo llamado entrada tiene unos tiempos de subida de 1.5ns y de bajada de 0.9 ns esto se puede expresar:

defpin entrada in tr=1.5 ns tf=0.9 ns

- *pinlist*.- (opcional) lista ordenada de los números de pines externos.

v **deflevel**.- Comando que define el potencial máximo que representa un 0 lógico y el potencial mínimo que representa un 1 lógico.

deflevel [<atributo>] <lista de valores de voltaje>

argumentos:

- *atributo* (opcional). palabra clave que indica si es de tecnología *cmos*, *nmos*, *pmos* o *ttl*
- *lista de valores de voltaje* .- vil (voltage in low), vih (voltage in high) vol (voltage out low), voh (voltage out high)

deflevel vil=0.2 vih=4.8

- v **deformat**.- Define la plantilla que interpreta los datos funcionales en vectores posteriores. Sólo trabaja con los pines de tipo in, out o io.

deformat: <nombre>[:<radix>]...<nombre>[:<radix>]

argumentos :

- *nombre del grupo* de los pines definido en defpin.
- *radix* (opcional) indica como se interpretan los datos en vectores posteriores. Los valores permitidos son binario, octal, decimal, hexadecimal. El radix por defecto es el binario. **Su carácter es meramente informativo.(nota solo se utiliza esta información cuando los vectores de test son del tipo #)**

- v **deftiming**.- define las unidades básicas de tiempo, que se utilizan para llevar a cabo la simulación. Estas definiciones también se utilizan en otros comandos de definición de tiempo como *defstrobe* y *defclock*.

deftiming <resolucion> <unidades><periodo>

argumentos :

- *resolución*. Se define como el mínimo paso de tiempo con el que se puede trabajar. Es la base del resto de unidades que se utilizan.
- *unidades*.- Unidad de tiempo utilizada para definir strobes y relojes. Debe ser múltiplo de la unidad de resolución.
- *periodo*.- longitud del periodo de test. Debe ser un múltiplo de la unidad de tiempo. Tiempo durante el cual STL aplica datos a los recursos y muestrea los datos de salida.

- v **defstrobe**.- Define la ventana en la que se aplica la señal para cada periodo de test. Gracias a este comando se pueden estudiar diferentes formas de

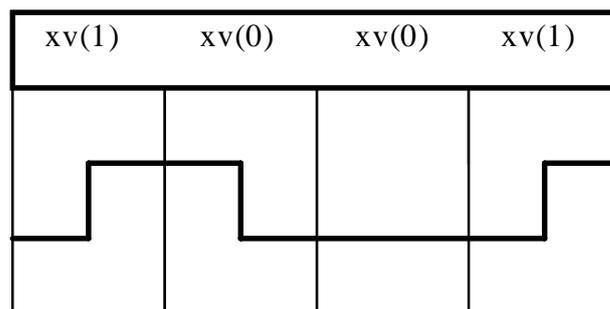
onda en un único periodo de test, lo que permite ahorrar tiempo de simulación.

defstrobe {in/out} [<tipo>]<character-waveform><lista de pines>

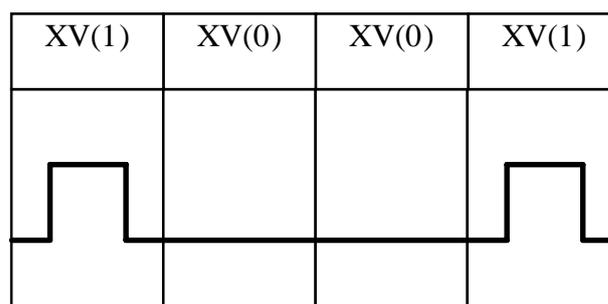
argumentos

- *in/out*.- indica si el nodo es de entrada o de salida
- *tipo*.- mediante este argumento se describe de manera cualitativa el tipo de ventana que se quiere generar. Existen cuatro tipos de ventana diferentes, *edge*, *window*, *rto* y *comp*.

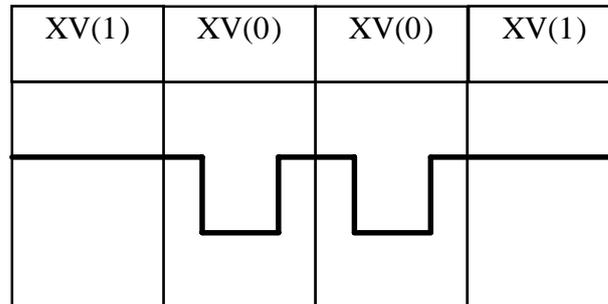
⇒ *edge*: también llamado NRZ (Non Return to Zero). Aplica el dato de entrada al comienzo de cada ventana. Se produce una transición simple en cada periodo. Suponiendo que se quiere simular una entrada definida por los vectores *xv(1)*, *xv(0)*, *xv(0)*, *xv(1)* la forma de la onda es la siguiente:



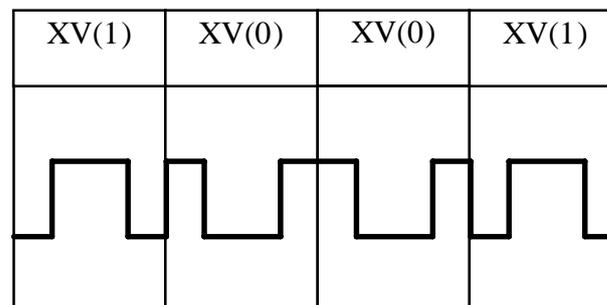
⇒ *window*.- también llamado RZ (Return to Zero). Este formato aplica la entrada durante toda la ventana. Este es el formato que proporciona STL por defecto para el *defclock*. Utilizando los mismos vectores de entrada del tipo anterior se obtienen la siguientes ondas .



⇒ *rto.*- Return to one. Es el complemento de window :



⇒ *comp.*- la señales se aplican en las ventana y sus complementos en el resto del periodo. Este formato emula las condiciones de timing utilizadas por los fabricantes para testar los circuitos. Sirve para comprobar los tiempos de setup y de hold.:



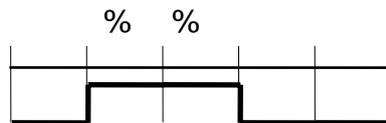
Estos cuatro tipos se puede aplicar a las entradas. A las salidas sólo se pueden aplicar el tipo edge y el window. El edge muestrea sólo al principio de la ventana y el window muestrea en toda la ventana.

- *character wave.*- define cuantitativamente el tamaño de la ventana. Un periodo se divide en tantas unidades como indica el deftiming. Cada una de estas unidades se

representa en este atributo mediante uno de los siguientes caracteres

1. (.) indica que la ventana no está activa
2. (%) indica que la ventana está activa.

Por ejemplo para un deftiming 1ns 10ns 50ns y un defstroke in window “.%%.” la forma de la onda para xv(1) es:



nota:

- cuando se hace un defstroke con buses no se deben poner los ángulos “<>”
- el tipo window se puede poner como de tipo edge
- Si en la definición defstroke no se pone a que señal se refiere, por defecto se aplica a todas.

v **defclock**.- sirve para definir la forma de los relojes del sistema. Por defecto se usa un tipo window.

defclock <character waveform> <nombre del pin>

argumento:

- character waveform. Igual en defstroke pero variando los caracteres que se utilizan:

(.) no activo

(1) activo.

v **defitest-enditest**.- define el comienzo y final de una descripción funcional. Dentro de la definición funcional se deben incluir los vectores de test o vectores funcionales. Se define un vector funcional como un conjunto completo de estímulos y valores de respuesta para *un periodo de test*. El dato funcional se especifica en términos de datos de entrada o respuestas

esperadas para cada grupo de pines del comando deformat. Puede ser, o un número, o uno de los siguientes símbolos.:

- X → don't care
- Z → alta impedancia
- ? → datos obtenidos de la simulación. sólo es válida para pines de salida.

v **xv.-** (execute vector) define un vector de test sencillo. Especifica datos funcionales para cada pin definido por el comando deformat. Para los grupos bidireccionales debería haber dos grupos de datos. Se debe usar 0b, 0, 0x para indicar que los datos son binarios, octales o hexadecimales. **Por defecto su valor es decimal.**

v **v.-** Define un vector de test sencillo en la definición de una secuencia.

v **#.-** Es similar al comando xv solo que no hace falta especificar el prefijo 0b, 0x. En su lugar se utiliza la definición de radix utilizada en deformat (Unico caso en que el radix deformat no es meramente informativo). El carácter # debe ser el primero de la línea.

v **rptv.-** Se utiliza para repetir la ejecución de un vector de test. Hay que especificar el número de repeticiones que se desean. Solo tiene aplicación cuando se trabaja con un testador ATE, en el resto de los casos actúa igual que xv

Cuando se utilizan los comandos, xv,v,#,rptv se debe especificar el dato funcional **para cada grupo** de pines definidos en el comando deformat, en **cada línea** de comando.

Se puede optimizar la utilización del comando xv de manera que sólo se tengan que modificar los valores de aquellos grupos de pines que interese. Para poder hacerlo en lugar de utilizar la correspondencia con el orden en deformat, se utiliza el nombre de la señal que se quiere variar de manera explícita.

Pines bidireccionales.- En ellos hay que especificar, en cada periodo de test, tanto el dato funcional como el sentido de entrada o salida. Para ello se usan dos columnas de datos. La primera para la entrada y la segunda para la salida,. Cuando el grupo de pines de tipo io se encuentra en un estado de entrada la primera columna debe contener los datos y la segunda el símbolo X. En un estado de salida la primera columna debería contener un Z y la segunda los datos de salida esperados.

EJEMPLO DE PROGRAMA STL

```
stlinit
defpin a in tr=1.5ns tf=0.9ns
defpin b in
defpin c out
defformat a b c
deftiming 1ns 10ns 50ns
defstrobe in window “..%%.” a
defstrobe in edge “..%%%” b
deftest
xv(0 0)
xv(0 1)
xv(1 0)
xv(1 1)
end test.
```

8.3 USO DE VARIABLES STL

En un programa STL se puede reemplazar un dato binario por una variable. Sea el programa en STL:

```
stlinit
defpin A in
defpin B in
defpin C in
defpin Y out
deftest
xv(0 0 0 0)
xv (0 0 1 0)
xv (0 1 0 0)
xv(0 1 1 1)
xv(1 0 0 0)
xv(1 0 1 0)
xv(1 1 0 0)
xv(1 1 1 1)
endtest
```

La ultima columna corresponde a los valores esperados. Si se supone que la variable a es el posible valor de A, la variable b el de B, y c el de C, y además suponemos que “y” es posible valor de Y se puede hacer:

$$y = (a\&b)|(a\&c)|(b\&c)$$

donde

- & es el símbolo de la función lógica AND
- “|” es el símbolo de la función lógica or.

Utilizando todo lo anterior se puede definir en una sola línea los 8 vectores XV que aparecen en el programa anterior. Para ello se debe tener en cuenta:

- se puede utilizar una variable “i” para indicar cada uno de los vectores que se desean probar.
- Puede nombrar a cada elemento del vector “i” metiendo su posición entre ángulos: el $i<2>$ se le haría corresponder al a, $i<1>$ se le haría corresponder al b, $i<0>$ se le haría corresponder al c.

El programa quedaría como sigue:

```
deftest
for(i 0 7 a=i<2> b=i<1> c=i<0> y=a&b|b&c|a&c xv(a b c y))
endtest
```

Esto es útil por que te permite generar de manera automática el conjunto de vectores de entrada, sin tener que especificar cada uno de los valores que toman. Por otro lado permite generar de manera lógica el valor que tomará la salida, para poder realizar la comparación.

8.4 USO DE PROCEDIMIENTOS

STL permite el uso de procedimientos, similares a los lenguajes de alto nivel, para generar los vectores de entrada y las señales de salida para comparar. Vamos ver un ejemplo:

```
procedure ( ejemplo(i)
a=i<2>
b=i<1>
c=i<0>
Y=a&b|a&c|b&c
xv(a b c y)
)
```

Donde:

- **procechure(--)** son las palabras claves que lo definen
- **ejemplo(i)** es el nombre del procedimiento que después se podrá instanciar
- **(i)** son los parámetros formales, que proporcionan las entradas al procedimiento
- las líneas que aparecen a continuación son el cuerpo del procedimiento.

La forma de utilizar el procedimiento es la siguiente:

```
deftest
for i 0 ejemplo(i)
endtest
```

8.5 NOTAS A STL

- Cuando se realiza la compilación de STL a VERILOG y existe un error, en ocasiones este se puede estudiar en el archivo *STLerrors.tmp* que genera el sistema
- Si en este archivo aparece un mensaje del tipo:

```
si:cannot map Cadence net name
```

puede ser que no se hayan unido los pads de entrada salida a un pin que le de conectividad.
- Si en la ventana de ondas, la salida de un bus es de tipo XXXXX, puede ser que la línea no tenga suficiente fuerza y haya que introducir una buffers para que funcione correctamente.

9. FLUJO DE DISEÑO SEMI-CUSTOM

9.1 COMO EMPEZAR

9.1.1 Entrada al kit de diseño de ES2

El kit de diseño de ES2 es una adaptación del DFVII de CADENCE al proceso de fabricación del *foundry European Silicon Structures* (ES2). En este caso el archivo de tecnología describe un proceso real de fabricación y proporciona descripciones de *layers* para llevar a cabo el *full-custom*. Además las descripciones de los componentes son reales. Los pasos para acceder a este kit de diseño son:

- Entrar en la cuenta del usuario
- Crear un directorio por debajo del directorio raíz p.e. `Mkdir curso(*)`. **A este directorio le llamaremos curso.**
- Cambiarse al nuevo directorio
- Ejecutar el comando **openwin**
- Ejecutar el comando **xhosts +**
- Ejecutar el comando **openDesignKit**
- Elegir la opción `ecpd07(*)`
- Elegir la opción `icfb(*)`

(*) indica que sólo se deben ejecutar la primera vez que se abra el kit de diseño

NOTA: Observar que en el comando `openDesignKit`, la **D y la **K** deben ir con mayúscula.**

Las opciones **ecpd10** y **ecpd07** representan dos procesos de fabricación diferentes. El primero corresponde a una anchura mínima de polisilicio de 1u y el segundo a una anchura mínima de polisilicio de 0.7u.

icfb es el comando del DFVII que utiliza todas las herramientas que proporciona CADENCE.

Cuando se arranca el kit de diseño de ES2, se crean las siguientes librerías de diseño:

- OscLib,
- PadLib,
- StdLib,
- AnaMac,
- ES2_74C,
- ES2_8255,
- JTAGLib,
- Syn_ind,
- Syn_mil,
- basic,
- sample,

Algunas de ellas las utilizaremos, como la librería

- StdLib que contiene las celdas estándar,
- PadLib que contiene los pads de alimentación, tierra y entrada /salida

Las otras, en cambio, no las utilizaremos. Este es el caso de la librería JTAGLib, que contiene pines y celdas específicos para realizar test cumpliendo unas normas específicas de IEEE, o la librería ES2_74C que contiene celdas independientes del proceso de fabricación.

NOTA: si sale el mensaje *error xtoolkit cant't open display* se ha olvidado el comando `xhost +`

9.1.2 Como crear una librería.

- Ir a la barra de menú de la ventana CIW
- seleccionar el comando **OPEN**,
- seleccionar **LIBRARY**.

En este punto aparece una ventana de dialogo (llamada forms) en la que se pregunta por :

- el nombre de la librería .
- por su path.

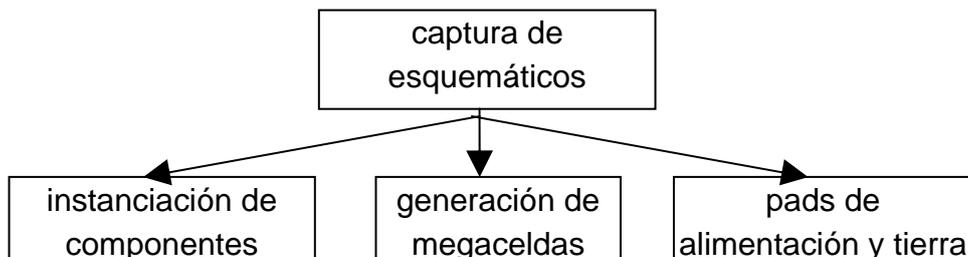
Como path introducir ~/ para indicarle que sitúe la librería en el directorio del usuario. El resto de valores los tomará por defecto.

NOTA: la librería sólo se debe generar la primera vez.

NOTA: Llamar a la librería “fundacion”

9.2 COMO GENERAR UN DISEÑO

9.2.1 Captura de esquemáticos



- Ir a la barra de menú de la ventana CIW
- Seleccionar el comando **OPEN**
- Seleccionar la opción **DESIGN**

En este punto aparece una ventana de dialogo que pregunta por

- La librería en la que queremos colocar el diseño.
- El nombre de la celda
- la *view* --> Se debe poner **Schematic**

El resto de los datos se pueden dejar por defecto. Una vez realizados todos estos pasos aparecerá la ventana de diseño Schematic.

Una vez en la ventana de diseño debemos instanciar los componentes que definen el esquemático. Existen dos casos diferentes, :

- El esquemático a instanciar ha sido diseñado por el usuario
- El esquemático es de los incluidos en la herramienta

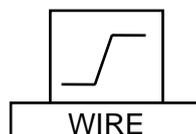
La forma de actuar en ambos casos es similar. Se selecciona el icono de la figura y aparece una ventana de dialogo en la que se preguntará el nombre del componente y la librería en que se encuentra. La forma más cómoda de trabajar es utilizar la opción LIBRARY BROWSER que aparece en la ventana de dialogo.

En el caso que el componente lo haya diseñado el usuario, debemos ir a la librería en la que se encuentre y seleccionar el diseño con la vista SYMBOL. De esta manera quedará automáticamente seleccionado. Aparecerá sobre la ventana de diseño de esquemáticos cuando se sitúe el cursor.

CADENCE proporciona librerías de referencia para instanciar objetos ya diseñados. Destacan las siguientes librerías:

- **SAMPLE.** En la clase TRANSISTOR Contiene transistores de diferentes tecnologías En nuestro caso utilizaremos las celdas NMOS.SYMBOL y PMOS.SYMBOL.
- **BASIC** contiene las alimentaciones y las tierras
- **StdLib** contiene las siguientes categorías de celdas:
 - **adders**- sumadores
 - **boolean**- diferentes puertas lógicas
 - **d_flipflps**- celdas de memoria de carga por flanco
 - **inv/buff**- inversores y buffers de diferente capacidad de carga
 - **latches**- celdas de memoria de carga por nivel
 - **mux**- multiplexores
 - **tri**- puertas triestate

Una vez instanciados los componentes se les une mediante hilos seleccionando el icono WIRE :



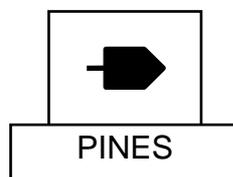
Los hilos se deben etiquetar para dar claridad al diseño. Para etiquetar un hilo se selecciona el icono WIRE LABEL, tras lo cual aparece una ventana de dialogo. En esta ventana se pueden seleccionar

- el nombre del hilo

- la opción *bus-expansion*.

Esta última opción se utiliza si el hilo que se etiqueta es un bus. Si no está activada el nombre aparece en forma vectorial. Por ejemplo, si se etiqueta un bus llamado *dato* de 4 bits se utiliza el nombre *dato<3:0>* y este es el nombre de etiqueta que aparece. En cambio si se selecciona la opción *bus-expansion* aparecen de manera sucesiva los nombres de cada uno de los bits del bus, es decir, *dato<3>*, *dato<2>*, *dato<1>*, *dato<0>*.

A continuación de los hilos se deben instanciar los pines del diseño para ello se debe seleccionar el icono PINES.



Igual que con los hilos aparece una ventana de dialogo que nos pregunta por el nombre del pin y que nos permite seleccionar algunas opciones. Si se quiere nombrar un bus de pines se hace *dato<3:0>*,

dato<3:0>

Si se selecciona la opción *bus expansión* aparecerán de manera sucesiva un pin para cada bit del bus.

dato<3>

dato<2>

dato<1>

dato<0>

Recordar que un hilo sin etiquetar toma el nombre del pin que se le asocia.

9.2.2 GENERACIÓN DE MEGACELDAS

Una de las facilidades que proporciona el kit de diseño de ES2 es la utilización de megaceldas. Una megacelda es una celda regular parametrizable, que se diseña sobre la marcha utilizando un compilador. El kit de diseño ES2 proporciona las siguientes megaceldas:

- Memorias RAM
- Multiplicadores
- Memorias ROM
- Dispositivos FIFO
- Memorias Ram de doble puerto

Para compilar una megacelda se deben realizar los siguientes pasos:

- Introducir en la línea de comandos de la ventana CIW el comando *ES2generate*
- Seleccionar en la ventana de dialogo que aparece el tipo de megacelda
- Especificar el nombre del bloque. Este debe empezar por uno de los siguientes prefijos: **ram**, **mul**, **rom**, **dpr**, **fifo**; y debe tener como máximo 10 caracteres.
- Como resultado de esta operaciones aparece una ventana x-term en la que se debe especificar la configuración de la megacelda. Esta ventana desaparece automáticamente al finalizar.

Nota.- Si la megacelda es una ROM, se debe especificar como se llena, en un fichero ./generate/<nombre_megacelda>.prg, usando las direcciones y datos en formato apropiados.

Una vez realizada la compilación se pueden ver las características de la megacelda en el archivo *./generate/<nombre_megacelda>.ds*

La generación de una megacelda conlleva la creación de las siguientes vistas:

- *Symbol* para la captura de esquemáticos
- Interface para el modelo de simulación
- *Abstract* para el *place&route*

la Instanciación de una megacelda se hace exactamente igual a la de cualquier esquemático. La librería en que se debe buscar las megaceldas se llama ***MegaCell***.

9.2.3 Instanciación de pads de i/o, pwr, gnd

La instanciación se realiza como en los casos anteriores. Las componentes se encuentran en la librería *PadLib*. Todos los pads se deben instanciar en el esquemático de más alto nivel de la jerarquía. Dados los siguientes tipos de pads:

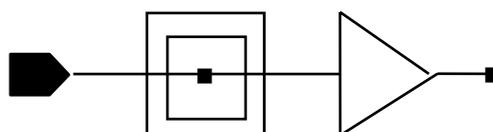
- * **LIBPWRPY** .- Vdd digital para la periferia
- * **LIBGNDPY** gnd digital para la periferia de CI
- * **LIBPWRCO** Vdd digital para el core del CI
- * **LIBGNDCO** gnd digital para el core de CI
- * **LIBCRNPWRPY** vdd digital para la periferia que se debe situar en un corner
- * **LIBCRNGNDPY** gnd digital para la periferia que se debe colocar en un corner.
- * **LIBTOPNETS** necesario para la comprobación layout versus esquemático (macro LVS)

Todo circuito integrado debe tener como poco una instanciación de los cuatro primeros. La separación de los pads en los que alimentan el core (el corazón) del CI y los que alimentan la periferia se realiza para conseguir aislar el core de los ruidos de conexión de los buffers de salida. Con esto se consigue que los rebotes de pwr y gnd debido a las conexiones y las inducciones del circuito tengan efectos mínimos sobre el core.

El core debe estar adecuadamente alimentado. Para circuitos pequeños es suficiente una pad de tierra y otro de alimentación. Para circuitos grandes se debe colocar un pad cada 40mm^2 o cada 64 celdas de i/o.

De todos los pads de entrada y salida que incluye la librería padlib nosotros vamos a seleccionar el LIBIPS8B como pad de entrada y el LIBOPS0U como pad de salida..

A la hora de realizar la simulación de los diseños que contiene pads de entrada y salida, se debe recordar colocarles a estos pines de entrada salida. Además se debe tener cuidado en no definir estos pads de tipo bus.



9.3 SIMULACIÓN FUNCIONAL DE UN DISEÑO

9.3.1 Utilización del calculador de retardos de CADENCE (CDC)

Una vez acabado el esquemático se selecciona la opción

check and save,

para comprobar que es correcto y salvarlo.

Las descripciones VERILOG de las celdas estándares incluyen los retardos asociados a las puertas lógicas pero no incluye los retardos debidos a los posibles fanouts y a los retardos RC de los hilos. Para que estos retardos se tengan en cuenta hay que utilizar el CDC que proporciona CADENCE. Este programa genera un archivo SDF(Standar Delay Format) que contiene estos retardos. Según el punto del flujo de diseño en que se realice la llamada al CDC, los archivos se calculan de una forma p de otra.

- Si se llama antes del floorplaning el retardo se basa en los fan-outs y en datos estadísticos
- Si se llama durante o después de la fase de floorplaning se basa en los fan-out, en las longitudes estimadas del hilo y en los lugares de ubicación de los bloques
- Si se llama después del place & route se basa en la ubicación física y en la longitud de rutado.

Para invocar al CDC se hace lo siguiente:

```
CIW> ES2generateSDF
```

aparece una ventana de dialogo en la que toda la información se puede dejar por defecto salvo la que nos pregunta por el tipo de retardo:

- Minimum
- Typical
- Maximum

La forma de actuar es la siguiente. Busca en el directorio de diseño un archivo con el siguiente nombre:

<nombre_celda>.spf

(spf Standar Parasitic File) que utiliza para crear archivos SDF según las condiciones seleccionadas y que pueden tener los siguientes nombre:

- <nombre_celda>_spf_min.sdf
- <nombre_celda>_spf_typ.sdf
- <nombre_celda>_spf_max.sdf

En el caso que el programa no encontrara archivos SPF en el directorio de diseño se generarían archivos con los siguientes nombres:

- <nombre_celda>_est_min.sdf
- <nombre_celda>_est_typ.sdf
- <nombre_celda>_est_max.sdf

NOTA: recordar que los archivos con extensión SDF aparecen en el directorio de diseño, no en el directorio de simulación.

9.3.2 Activación del entorno de simulación

Los pasos que hay que seguir para la simulación se muestran a continuación. De la barra de menú de la ventana de diseño se selecciona:

```
menu>tools->simulation->verilog-xl
```

En este momento aparece una ventana de dialogo preguntando por el nombre del directorio de simulación. Por defecto, el nombre de directorio que da la herramienta es el del diseño con la extensión **run1**. Por ejemplo si el diseño se llama *inversor* el nombre del directorio debe ser *inversor.run1*.

Además, el directorio de simulación debe colgar al mismo nivel que el diseño por lo tanto, en la casilla **run directory** se debe colocar el path de usuario. En el caso del ejemplo *,/home1/usuario/inversor.run1*. Esto se realiza de manera automática tras seleccionar la opción OK. En este instante aparece una nueva ventana, en este caso la ventana **VERILOG-XL**.

9.3.3 OPCIONES DE SIMULACIÓN

Desde el menú principal de la ventana *VERILOG* se puede llamar a la opción *setup* para seleccionar las opciones de simulación deseadas. La mayoría de ellas se pueden dejar por defecto pero hay alguna de ellas que se deben comentar.

v **Opción preserve bus**

La selección de la opción:

menú>setup->netlist

provoca la aparición de una ventana de dialogo en la que se puede modificar el valor *Preserve Buses*. Si el esquemático a simular tiene definidos pines del tipo buses la opción debe estar **on**. Se utiliza para evitar que una señal que se ha definido de tipo bus en STL se convierta en un conjunto de señales individuales al hacer la compilación a verilog.

v **Selección del tipo de retardo**

Tras seleccionar

menu>setup->simulación

surge una ventana de dialogo con un parámetro *delay* en el que se debe seleccionar el tipo de retardo:

- minimum
- typical
- maximum

La opción seleccionada debe coincidir con la seleccionada al invocar al CDC.

v **Conjunto de señales a simular**

Por defecto CADENCE solo guarda el trazado de simulación de los pines de entrada y de salida. Con esto suele ser suficiente para comprobar la funcionalidad de la celda que se está simulando.

Pero en ciertas ocasiones cuando la simulación no ha sido correcta conviene poder acceder a los nodos intermedios del esquemático para conocer sus valores. Para seleccionar qué señales se deben guardar en el trazado de la simulación hacer

menu>setup->simulation.

tras ello aparece una ventana de dialogo. Escoger para el parámetro save el valor *ON* y seleccionar alguna de la opciones

- top-level primary i/o
- all signals

Esta última opción produce unas necesidades de almacenamiento muy grandes. También lleva más tiempo. Se utiliza para visualizar las simulaciones de nodos intermedios que no sean pines de entradas y salidas.

NOTA: la selección de opciones se debe hacer antes de la netlist. En caso que por algún motivo haya que modificar alguna de las opciones habrá que repetir la netlist. La forma más sencilla de hacerlo es la siguiente

MENU> file> clean curren directory

Tener en cuenta en la utilización de esta opción que no elimina todo el directorio, solo un pequeño conjunto de archivos, entre los que se encuentra la netlist.

9.3.4 Generación del archivo de estímulos

A continuación se ven los pasos que hay que seguir para editar una archivo de simulación STL: Seleccionar del menú:

menu>stimulus->stl->edit stimulus

La primera vez que se lleva a cabo esta secuencia de pasos, aparece una ventana informando que no se ha hecho todavía la *netlist*, y preguntando si se desea realizar. La respuesta debe ser afirmativa. Una vez seleccionada esta opción aparecen en la ventana *CIW* unos mensajes indicando que se está llevando a cabo la *netlist*. El mensaje que aparece indicando que se ha realizado correctamente es:

Netlisting of cell-nombre view-nombre successful

siendo *cell-nombre* el nombre de la celda y *view* nombre el nombre de la vista. Simultáneamente aparece una ventana de diálogo llamada *edit stl stimulus file* que sirve para indicarle al programa si se desea llevar a cabo automáticamente la compilación de STL a VERILOG. Para seleccionar esta opción se debe seleccionar la opción *compile stl after editing*

De manera automática aparece una ventana VI (o con el editor UNIX que se haya seleccionado) con el programa en STL llamado *input.stl* que contiene una plantilla de un programa en lenguaje STL. Este archivo se debe editar para introducir los valores de simulación deseados.

NOTAS:

- v **comprobar si el primer argumento del deftiming es 100ps (0.1ns) que es el paso mínimo de simulación de ES2.**
- v **Recordar que por defecto el formato de los vectores es decimal.**
- v **Si se quiere modificar este formato no es suficiente modificar el radix del comando defformat Para modificar el formato habría que añadir delante de los valores de los vectores de simulación los siguientes prefijos:**
 - **0x** si el formato es hexadecimal,
 - **0b** si es binario
 - **0** si es octal

La ventana VI se cierra automáticamente al salir del editor. En CIW debe aparecer el mensaje

Num test vector generated for VERILOG simulation

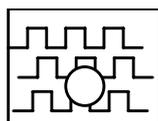
que indica que se ha llevado a cabo la compilación de STL a VERILOG (num es el número de vectores de datos que aparezcan en archivo STL). El **stl** compilado se vuelca automáticamente sobre un archivo llamado *testfixture.stl*.

v **Visualización de las ondas de entrada**

Antes de llevar a cabo la simulación se puede ver un cronograma de los estímulos de entrada al circuito. Estos datos se encuentran almacenados en el archivo *stl.shm.db* del directorio del usuario. Para ello se realizan los siguientes pasos.

menu>stimulus->stl->show stimulus waveform

De manera automática aparecerá la ventana *CWAVES* . Si no aparecen las ondas de entrada hay que seleccionarlas en la ventana de diseño de esquemáticos. Para ello nos trasladamos a la ventana de diseño, seleccionamos una de la entradas, volvemos a las ventana de ondas y seleccionamos la opción *añadir onda* del menú vertical cuyo icono es el siguiente:



Si las ondas de entrada no se corresponden con las que el usuario quería introducir se modifica el archivo STL y se repite el proceso. Si coinciden conviene cerrar la ventana *CWAVES*, para que no se produzcan errores más tarde.

9.3.5 Incorporación del Archivo .sdf al archivo de simulación VERILOG

Utilizando el rutina skill *ES2simTemplate* se toma el archivo *testfixture.stl* -que contiene las entradas de simulación en VERILOG - y se le modifica insertándole las directivas necesarias para poder utilizar el cálculo de retardos de ES2. Para utilizar esta rutina nos debemos asegurar que:

- hay un entorno *verilog* abierto, con un directorio de simulación especificado y la netlist realizada
- que se ha creado el conjunto de estímulos

Los pasos a seguir son:

- Escribir en la línea de comandos de CIW *ES2simTemplate*.
- seleccionar en la ventana de dialogo el modo retardo SDF
- Escribir en la ventana de dialogo el archivo que contiene los estímulos. En nuestro caso es *testfixture.stl*.

NOTA: si no se pone nombre de archivo la simulación se realiza correctamente pero todos los estímulos de entrada son cero.

- La rutina crea en el directorio de simulación un archivo de salida llamado *ES2testfixture.v*. Este nuevo archivo incluye
 - * las directivas de control para el cálculo de retardos de ES2.
 - * una llamada al archivo *testfixture.stl*
- Modificar el archivo *testfixture.stl*

Como tanto el archivo *testfixture.stl* como el archivo *ES2testfixture.v* son archivos verilog, ambos contienen información redundante que da lugar a que no se pueda realizar la simulación. Para prevenir esto hay que editar el archivo *testfixture* y eliminar o comentar todas las partes comunes, principalmente cabeceras y finales de módulo.

La mejor forma de hacerlo es abrir mediante un editor de textos ambos archivos para compararlos y eliminar del `testfixture.stl` la información redundante.

El último paso que se realiza es llamar desde la ventana CIW al procedimiento *ES2sdfTranslation*. Este procedimiento linka los archivos sdf que se han creado en el directorio de diseño.

NOTA:

- **es importante en las llamadas desde la CIW a procedimientos respetar las mayúsculas y las minúsculas.**
- **Debe quedar claro que la simulación funcional se puede realizar en dos fases del flujo diferentes. La simulación pre-layout, se lleva a cabo antes de la fase de place&route. La simulación post-layout se realiza después de haber realizase el place&route. En este caso se utiliza para calcular el retardo las capacidades extraídas del layout.**

9.3.6 Ejecución de la simulación funcional

Para ejecutar la simulación propiamente dicha, se selecciona el comando:

```
menu->simulation->start batch.
```

Para ver el estado en el que se encuentra la simulación se escoge la opción

```
menu>simulation->job monitor
```

y aparece una ventana llamada *analysis job monitor* que sirve para ver el estado de la simulación, para detenerla o para interrumpirla. En esta ventana se puede seleccionar la opción *show run log*.

Los resultados de la simulación se pueden ver en archivo *si.out*. Cuando la simulación acaba aparece una ventana de dialogo informando que ésta a acabado con éxito. Esta ventana es engañosa porque se refiere a la invocación del verilog y no tiene nada que ver con el éxito o el fracaso de la simulación en sí misma. Esto se ve en el archivo *simout.tmp* cuyas últimas líneas cuando la simulación está libre de errores deben ser similares a las siguientes:

```
DL>$check_design started
```

```
DL>$check_design completed
```

```
L26
```

```
"library9/users/peter/cdkv30/dec2to4/dec2to4cells.run1/Es2testfixture.v"
```

```
:$finish at simhulation time 50
```

```
1 warning
```

```
331 simulation events+1064 acelerated events+3952 timming check events
```

```
CPU time: 7.4 secs
```

```
end of verilog-xl
```

NOTA:

- si después de realizada la simulación hay que cambiar el archivo STL se deben repetir todos los pasos para que la simulación sea correcta y tenga en cuenta los cambios.
- Si tras la simulación se da cuenta que tiene que cambiar el esquemático recordar que el archivo input.stl sólo se genera automáticamente la primera vez que se llama y que el resto de las veces hay que generarlo automáticamente
- los manuales de CADENCE recomiendan que por defecto se utilice la simulación batch
- antes de arrancar la simulación percatarse de que la ventana de esquemáticos no tienen ningún nodo o pin seleccionado. Error GSM (modulo generador de señales)
- error: the netlister files needed by stl do no exist. Please set netlist option to use test fixture. El problema consiste en que alguno de los módulos que componen el esquemático han sido modificados y no se han salvado tras modificarse. En la ventana CIW aparece cual es la celda modificada. A continuación antes de repetir la simulación, desde la ventana VERILOG se selecciona la opción files clean current run para realizar de nuevo la netlist.
- error: can't determine what to tap for <15:0> indica que existen problemas en dicho bus. Causa es que el nodo intermedio está sin nombre y por lo tanto no se reconocen las derivaciones del bus. Solución poner nombre al bus.
- warning: signal name ual<0> collides with cell name ual solución modificar el nombre de la señal.

9.3.7 Visualización de las ondas

Acabada la simulación se selecciona el icono “ver ondas” y surge la pantalla de visión de ondas llamada **CWAVES**.

NOTA: Si esta pantalla no se cerró antes de la simulación, el sistema puede fallar. El principal problema que puede producir es la generación de archivos basura que luego no se pueden eliminar y que corrompen la base de datos de la simulación.

Para que funcione correctamente debe haber una ventana *schematic* abierta, el entorno de simulación *VERILOG* activo y un directorio de simulación definido. Para estudiar los resultados de la simulación hay que seleccionar los nodos que se desean ver. Los nodos se pueden seleccionar uno a uno, o varios a la vez:

- un nodo.- se selecciona en la ventana de diseño de esquemático. A continuación se va a la ventana *CWAVES* y se escoge el icono **añadir onda**.
- varios nodos .-el proceso es el mismo pero pulsando la tecla de **mayúsculas** al seleccionar los nodos.

NOTA:Para ver el periodo total de simulación seleccionar la opción FIT.

Las señales que se hayan definido como buses en el esquemático se pueden desdoblar y agrupar a voluntad, además de modificarles el radix. Para ello hay que seguir los siguientes pasos:

- se selecciona la señal
- edit->strips-
 - expand desdobla el bus
 - collapse agrupa las líneas del bus
 - radix

El archivo en el que se cargan los datos de visualización de las ondas se llama *shm.db* y cuelga del mismo nivel que el directorio de simulación. El archivo que contiene las ondas de entrada se llama *stl.shm.db*.

Si la visualización de las ondas no se produjera conviene asegurarse que el archivo que se carga es el correcto para ello en la ventana cWaves se selecciona :

file->load data

NOTA si se producen errores en la simulación pueden estar motivados por:

- se ha modificado el esquemático y no se ha comprobado, salvado ni generado la nueva netlist. Para generar la netlist sólo hay que hacer:

- *menu>file->clean current run*

antes de repetir los pasos de simulación.

- no se ha tenido en cuenta que el formato de XV en el archivo input.stl es decimal. Si se quiere modificar hay que añadirle 0x para formato hexadecimal, 0 para formato octal y 0b para formato binario.
- se modifico el nombre de un pin en el esquemático pero no se modifico en el archivo input.stl.
- no se han realizado todos los pasos de la simulación
- la opción *preserve bus* esta desconectada
- los resultados de los buses pueden aparecer en un radix diferente al seleccionado en el stl

10. PRACTICAS DE DISEÑO SEMI-CUSTOM

En las prácticas se va a desarrollar un computador simple completo para ver íntegro el proceso de diseño de un circuito integrado. Para ello se van a utilizar las bibliotecas de celdas estándar que proporciona ES2, en concreto se trabajara con la librería ECPD07 que es la de 0,7micras. Para este diseño se van a utilizar las siguientes herramientas de CADENCE:

- Captura de esquemáticos
- Generación de macroceldas
- VERILOG para la simulación funcional
- VERIFault para la simulación de fallos
- VERITIME para el análisis de caminos críticos
- Place&route
- Simulación postlayout
- Verificación

10.1 ESPECIFICACIONES DE ARQUITECTURA

10.1.1 Especificaciones generales

A continuación se describe la arquitectura y estructura del computador que se diseña en la asignatura.

- Es una arquitectura de tipo Von Neumann
- Tipos de datos enteros de 16 bits codificados en Complemento a dos
- Memoria RAM de 256 palabras de 16 bits cada una que puede almacenar tanto datos como instrucciones
- Bus de direcciones de 8 bits
- 2 buses de datos de 16 bits
- Secuenciamiento implícito
- Registro de instrucciones de 16 bits
- Contador de programa de 8 bits
- Banco de 8 registros de 6 bits cada uno
 - * (R0-R7)
 - * El R0 es un registro especial que contiene siempre el 0.
 - * Un puerto de entrada
 - * Un puerto de salida
- Modelo de ejecución registro - registro
- Dos indicadores de condición, cero y signo

10.1.2 Repertorio de instrucciones:

Tiene tres tipos de instrucciones diferentes:

- Aritmético lógicas
- Transferencia de datos
- Salto

Las instrucciones aritmético lógicas se dividen a su vez en dos grupos, las que utilizan un operando inmediato y las que no lo utilizan.

v **Operaciones aritméticas con dos operandos en registros :**

ADD Rf1, Rf2, Rd

$$Ra \leftarrow Rf1$$

$$Rd \leftarrow Ra + Rf2$$

SUB Rf1, Rf2, Rd

$$Ra \leftarrow Rf1$$

$$Rd \leftarrow Ra - Rf2$$

ASR Rf, Rd

$$Rd \leftarrow \text{deplaza}(Rf2)$$

AND Rf1, Rf2, Rd

$$Ra \leftarrow Rf1$$

$$Rd \leftarrow Ra \text{ and } Rf2$$

Formato de las instrucciones aritméticas con dos fuentes y un destino

CO	Rd	Rf1	Rf2	00	OP
----	----	-----	-----	----	----

Siendo:

- **CO** código de operación - identifica el tipo de instrucción. Utiliza dos bits

00 load

01 store

10 salto

11 aritmético - lógicas

- **Rd** Registro destino. 3 bits RI<13:11>
- **Rf1**. Registro fuente. 3 bits. RI<10:8>
- **Rf2**. Registro fuente. 3 bits. RI<7:5>
- **OP** identifica la operación aritmético - lógica concreta que se quiere realizar

100 ADD

101 SUB

110 ASR

111 AND

v Operaciones Aritméticas con operando inmediato

ADDI Rf, #n,Rd

$Ra \leftarrow Rf1$

$Rd \leftarrow Ra + \text{exte}(\#n)$

SUBI Rf,#n,Rd

$Ra \leftarrow Rf1$

$Rd \leftarrow Ra - \text{exte}(\#n)$

CO	Rd	Rf1	INMEDIATO	OP
----	----	-----	-----------	----

- **Rd** Registro destino. RI<13:11>. 3 bits
- **Rf1**. Registro fuente. RI<10:8>. Tres bits

- **INMEDIATO.** Numero inmediato con el que se opera. RI<7:3> 5 bits. Como se utiliza en una UAL de 16 bits habrá que hacer una extensión de signo con el.

- **OP**

000 suma

001 resta

v Instrucciones de acceso a memoria

LOAD A(Ri),Rd

$R@ \leftarrow \langle A+Ri \rangle$

$Rd \leftarrow m \langle R@ \rangle$

este dato llega al banco de registros a través de la UAL

STORE Rf, A(Ri)

$R@ \leftarrow \langle A+Ri \rangle$

$m \langle R@ \rangle \leftarrow Rf$

Siendo la dirección base un número natural incluido en la instrucción y el desplazamiento el contenido de un registro

CO	Rx	Ri	dirbase
----	----	----	---------

Rx Registro destino o fuente RI<13:11>

Ri Registro indice RI<10:8>

v Instrucciones de salto

Existen siete instrucciones de salto:

BL bifurcar si es menor	BG bifurcar si es mayor
BEQ bifurcar si es igual	BNE bifurcar si es distinto
BLE menor o igual	BGE mayor o igual
BR salto incondicional	

Los pasos de ejecución de todas ellas son idénticos:

si condición es cierta

$$R@ \leftarrow RI<7:0> + R0$$

$$PC \leftarrow R@ + 1$$

$$RI \leftarrow M[R@]$$

si no

$$PC \leftarrow PC + 1$$

CO	cond	000	dirección
----	------	-----	-----------

- **CO** = 10
- **COND** es la condición respecto a la que se hace el salto.
 $RI<13:11>$.
 - 000 incondicional
 - 001 igual
 - 010 menor
 - 011 menor o igual
 - 101 distinto
 - 110 si mayor o igual
 - 111 si mayor.
- **Dirección** es la dirección a la que se bifurca $RI<7:0>$

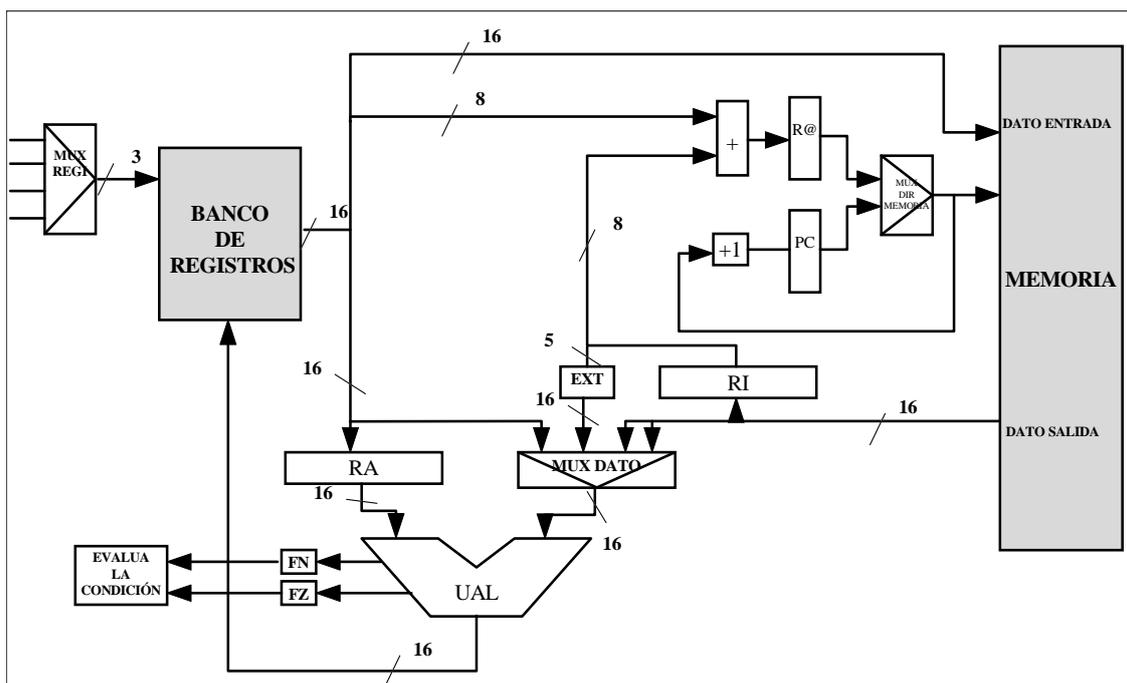
Es importante darse cuenta que todas las instrucciones están condicionadas por el hecho de que el conjunto de registros sólo tenga un puerto de salida, lo que impide que tanto las operaciones aritmético - lógicas como las de bifurcación y movimiento de datos se implementen en dos pasos.

10.2 CAMINO DE DATOS DEL PROCESADOR.

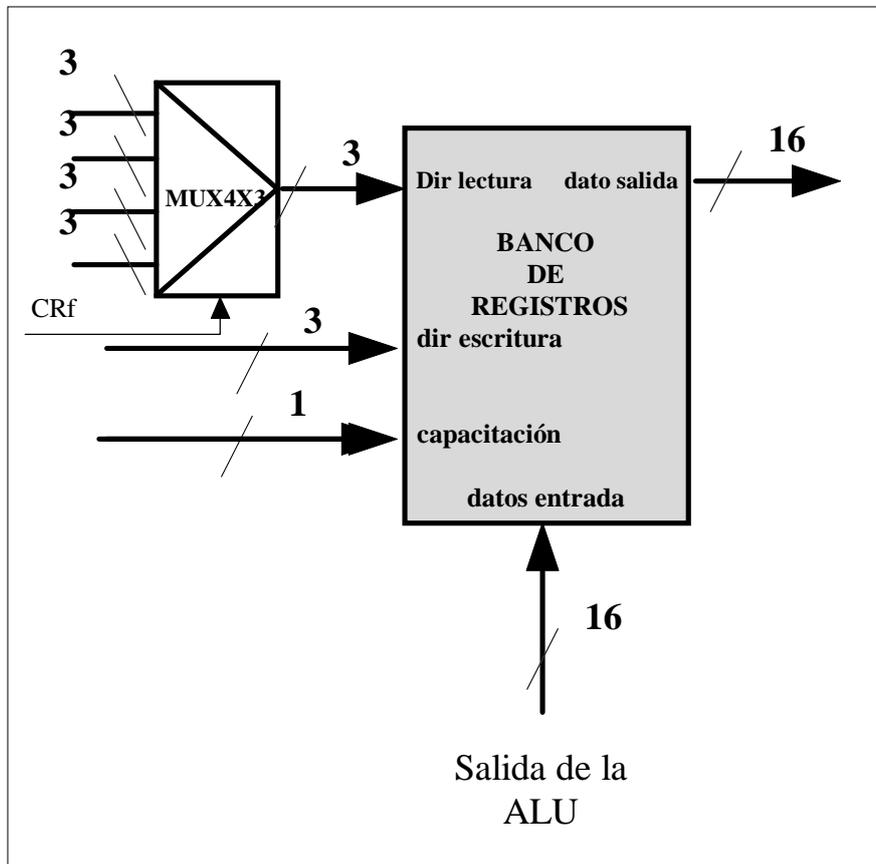
La figura que aparece a continuación contiene el camino de datos completo del computador. En el vamos a distinguir los siguientes bloques funcionales:

- El banco de registros
- La unidad secuenciadora
- La unidad aritmético lógica
- La memoria

en las siguientes secciones vamos a explicar en detalle cada uno de estos módulos para que el alumno los vaya implementando con la herramienta.



10.2.1 El módulo del Banco de registros

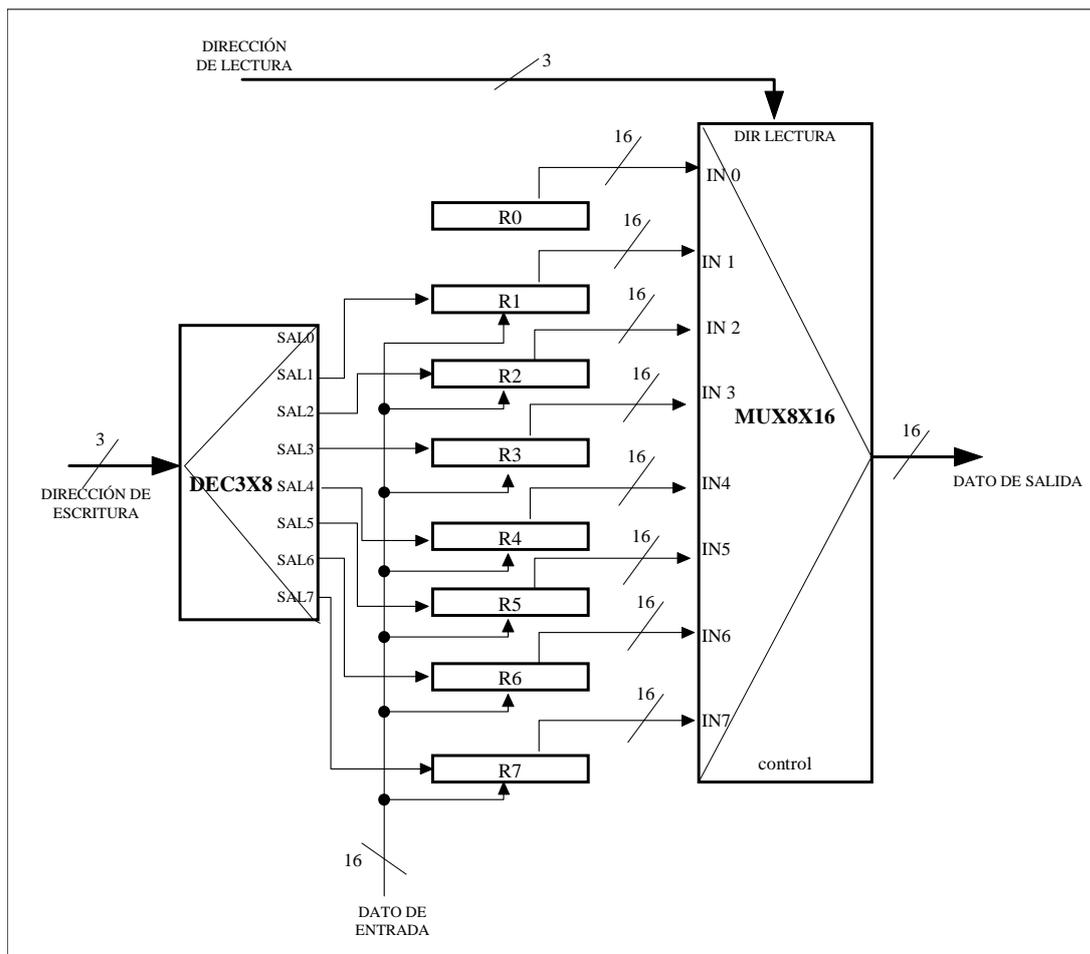


El banco de registros se descompone en un multiplexor MUX4x3 que selecciona la dirección de escritura y el banco de registro propiamente dicho.

Las señales del banco de registros son :

- Dirección de escritura
- Dirección de lectura
- Entrada de datos
- Salida de datos
- Señal de capacitación de escritura

10.2.1.1 El Banco de registros

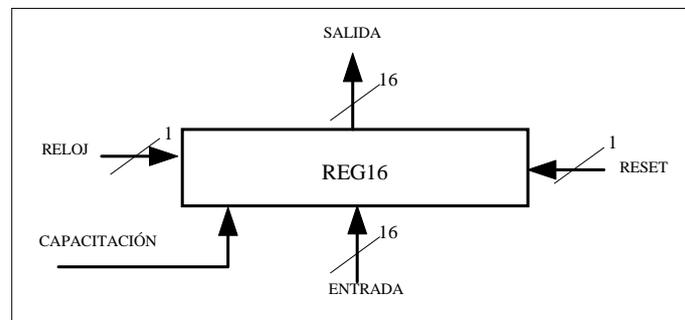


Como se puede ver en la figura el banco de registros se compone de los siguientes módulos:

- Registros de 16 bits
- Decodificador con una entrada de tres bits y 8 salidas de 1 bit
- Un multiplexor vectorial de 16 bit y tres señales de control (MUX8X16)

Nota: En la figura no aparece la señal de reloj de sistema que también se debe incluir. En las siguientes secciones se explica como se implementa.

10.2.1.1.1 Diseño del registro de 16 bits: REG16



Como se puede ver en la figura los registros del computador van a tener las siguientes señales de entrada y salida:

ENTRADA:

- Datos de entrada
- Señal de capacitación
- Señal de reset
- Señal de reloj.

SALIDA:

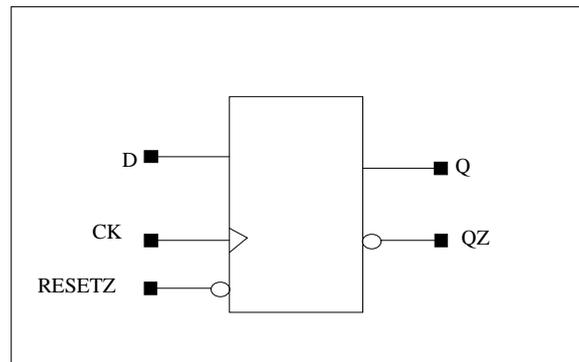
- Datos de salida

Si queremos cargar un dato en el registro se debe activar la señal de capacitación y posteriormente activar la señal de reloj. Existe la tentación habitual de implementar esto puertando la señal de reloj con la señal de capacitación. Esto no se debe hacer **NUNCA** porque se corre el peligro de introducir errores de sincronización en el computador. Ya veremos al final de la sección como se realiza esta capacitación del registro.

Para realizar el registro se utiliza como elemento básico de diseño la celda estándar DFFR que se encuentra en la librería de celdas StdLib. Este biestable tiene las siguientes señales de entrada y salida:

- Una señal reset asíncrono RESETZ
- Una señal de carga por flanco de reloj CK
- Una entrada de datos D
- Una salida de datos Q
- Una salida de datos complementada QZ

Su símbolo es el siguiente:



Y su tabla de verdad:

D	CK	RESETZ	Q	QZ
1	↑	1	1	0
0	↑	1	0	1
X	0	1	Q0	QZ0
X	1	1	Q0	QZ0
X	X	0	0	1

Donde ↑ indica el flanco de subida del reloj.

NOTA: Es importante darse cuenta que el reset funciona a 0.

Para que el biestable funcione correctamente se deben cumplir los tiempos de set-up y de hold de los datos:

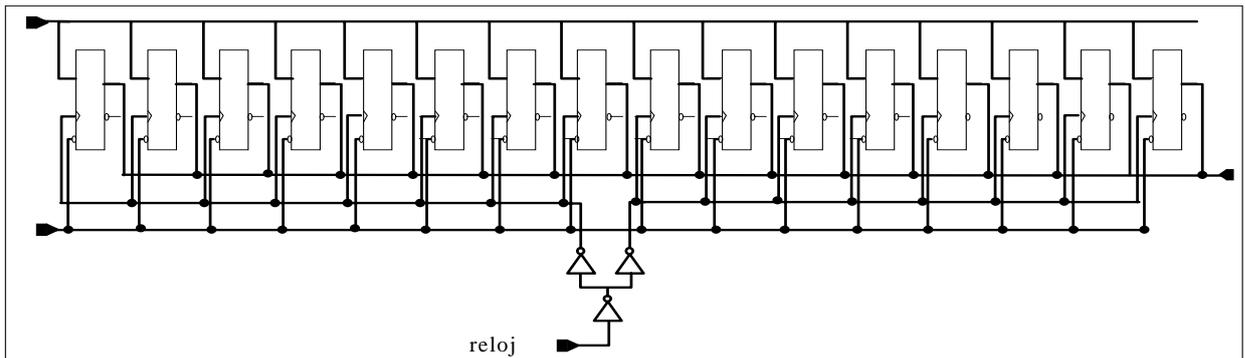
- **Tiempo de setup** del dato es el tiempo que debe estar estable el dato a la entrada del biestable antes de que llegue la señal de reloj.
- **Tiempo de hold** del dato es el tiempo que debe estar el dato estable a la entrada del biestable después de haber llegado la señal de reloj.

tiempos	NS
set-up	2
hold	0.75
anchura de 0 del ck	2

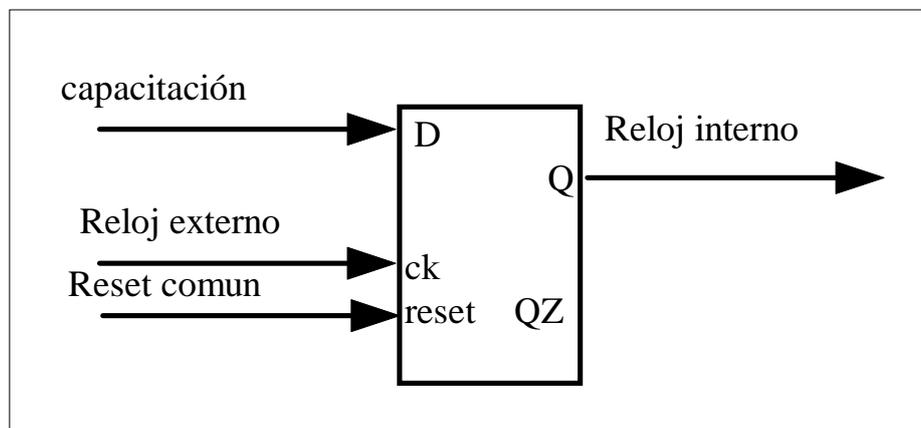
anchura de uno del ck	1.50
-----------------------	------

NOTA: Es importante que los archivos en STL que describen las entradas a los biestables cumplan los tiempos de setup y hold.

El diseño total sería como se ve en la siguiente figura:



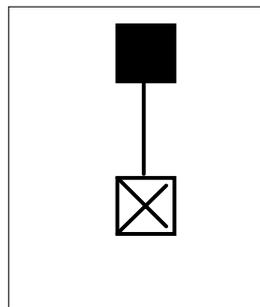
A este diseño le falta la señal de capacitación. En la siguiente figura se ve como se implementa



Notas para el diseño con CADENCE:

- Los buses se deben llamar de un modo especial:
Nombre<15:0>
- Cada línea que salga del bus principal debe llevar una etiqueta indicando el número de línea que es : <0>, <1>, ..., <15>

- Los pines deben estar definidos correctamente como de entrada (input) o como de salida (output)
- El motivo de diseñar la señal de reloj con ese árbol de inversores es disminuir el retardo y las desviaciones de reloj.
- El símbolo se puede hacer automáticamente o manualmente.
- Tener en cuenta los tiempos de set up y de hold en los archivos de simulación.
- No dejar nunca pines sueltos, puede provocar un exceso de **warnings** o de errores. Los pines que no se vayan a conectar a ninguna señal se deben conectar a la señal **noconn** que esta en la biblioteca **basic**, dentro de la clase de celdas **misc** y cuyo símbolo es el siguiente:



10.2.1.1.2 Diseño del multiplexor de 8 entradas de 1 bit (MUX8x1)

Vamos a utilizar la celda estándar MUX4x1 como módulo básico de diseño. Primero implementaremos un mux de 8 entradas de datos de un bit, para posteriormente utilizarlo en la realización de un mux vectorial de 16 bits.

El símbolo del MUX41 (que se encuentra en la librería StdLib) es el siguiente:

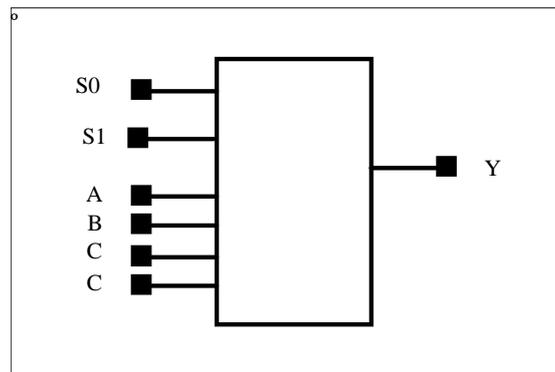


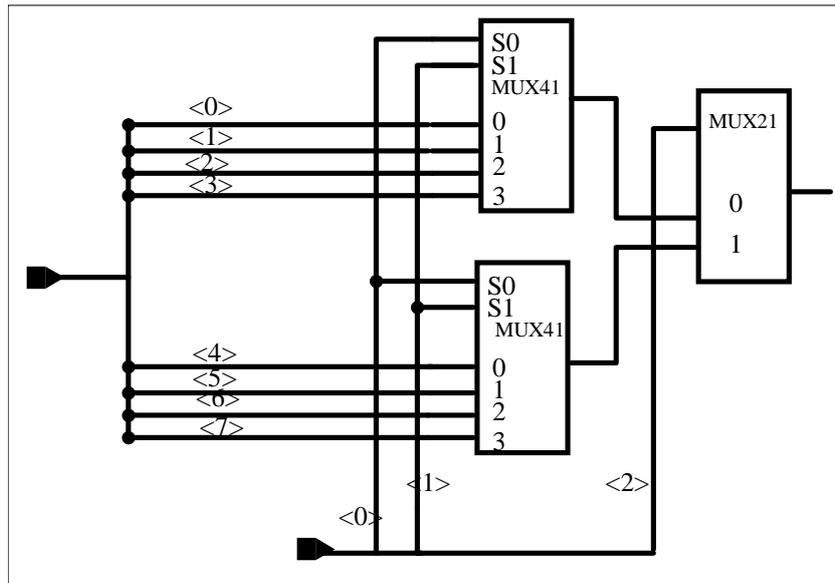
Tabla de verdad:

S1	S0	Y
0	0	A
0	1	B
1	0	C
1	1	D

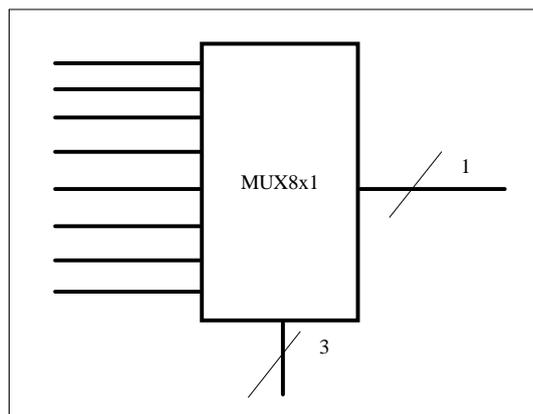
Siendo

- **<S1,S0>** las señales de control
- **Y** la salida
- **A,B,C,D** las entradas.

El esquemático del multiplexor de 8 entradas de control se puede ver a continuación:

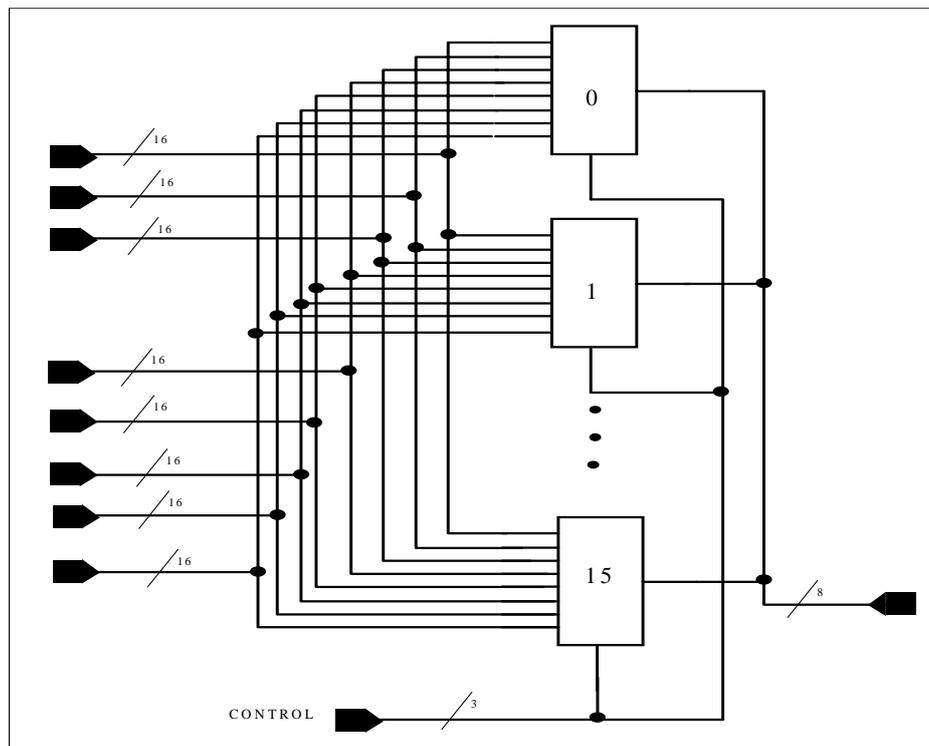


Y el símbolo podría ser es siguiente:

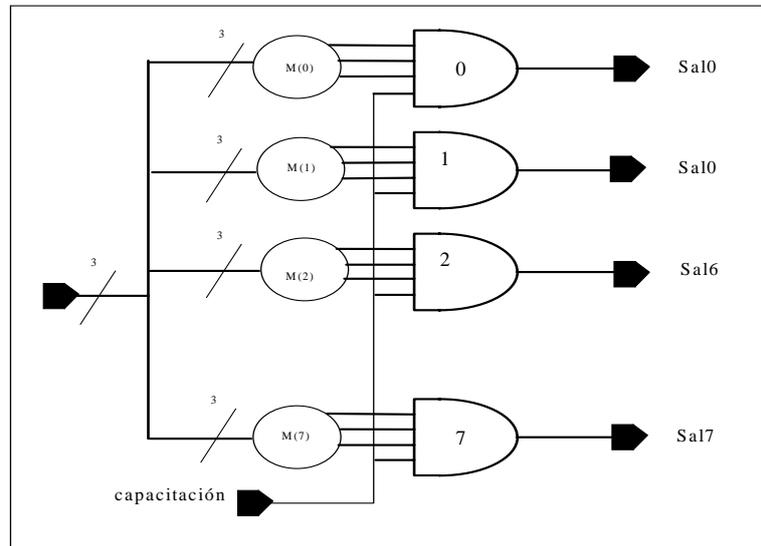


10.2.1.1.3 Diseño de un multiplexor de 8 líneas de entrada de 16 bits. MUX8x16

En el esquemático de multiplexor de 8 entradas de 16 bits cada una se utiliza el mux8x1 diseñado en la sección anterior. Su esquemático sería siguiente :



10.2.1.1.4 Descodificador de tres entradas DEC3X8



En esta figura los círculos representan sencillos circuitos combinacionales que implementan los minterminos de las tres señales de entrada.

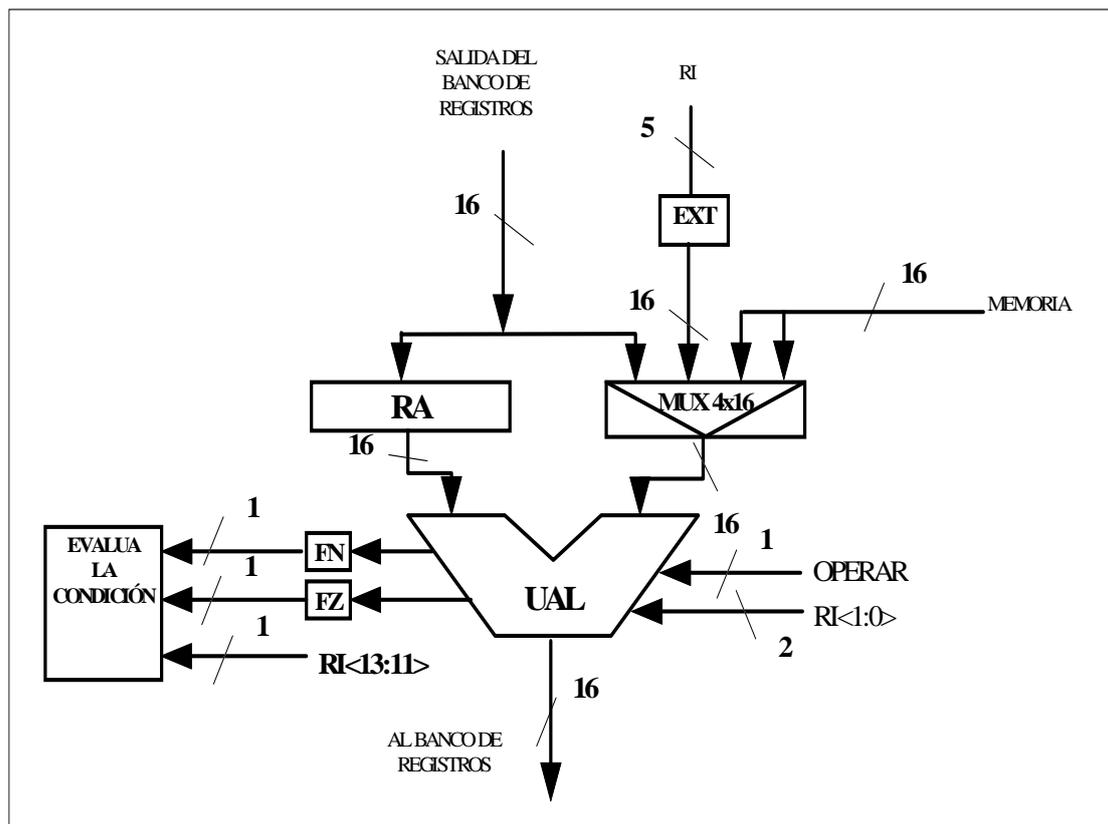
10.2.1.1.5 Multiplexor MUX4X3

Es el multiplexor que se utiliza para seleccionar la dirección del registro del banco de registros al que se desea acceder. Tiene cuatro entradas de datos de tres bits cada una de ellas. La entrada de control (CRf) tiene dos bits y proviene de la unidad de control.

No se entra en mayor detalle porque el método a seguir es similar al de los otros multiplexores

10.2.2 La Unidad aritmético lógica (ALU)

A continuación aparece el esquemático de la unidad aritmético lógica y de los registros y unidades funcionales que le rodean. Darse cuenta que de las tres señales que controlan la unidad solo la señal OPERAR proviene de la unidad de control y las otras dos ($RI<1:0>$) provienen del Registro de Instrucciones directamente.



De los módulos que se ven en la figura anterior voy a enumerar aquellos cuyo diseño o bien está ya estudiado, o bien es un diseño trivial:

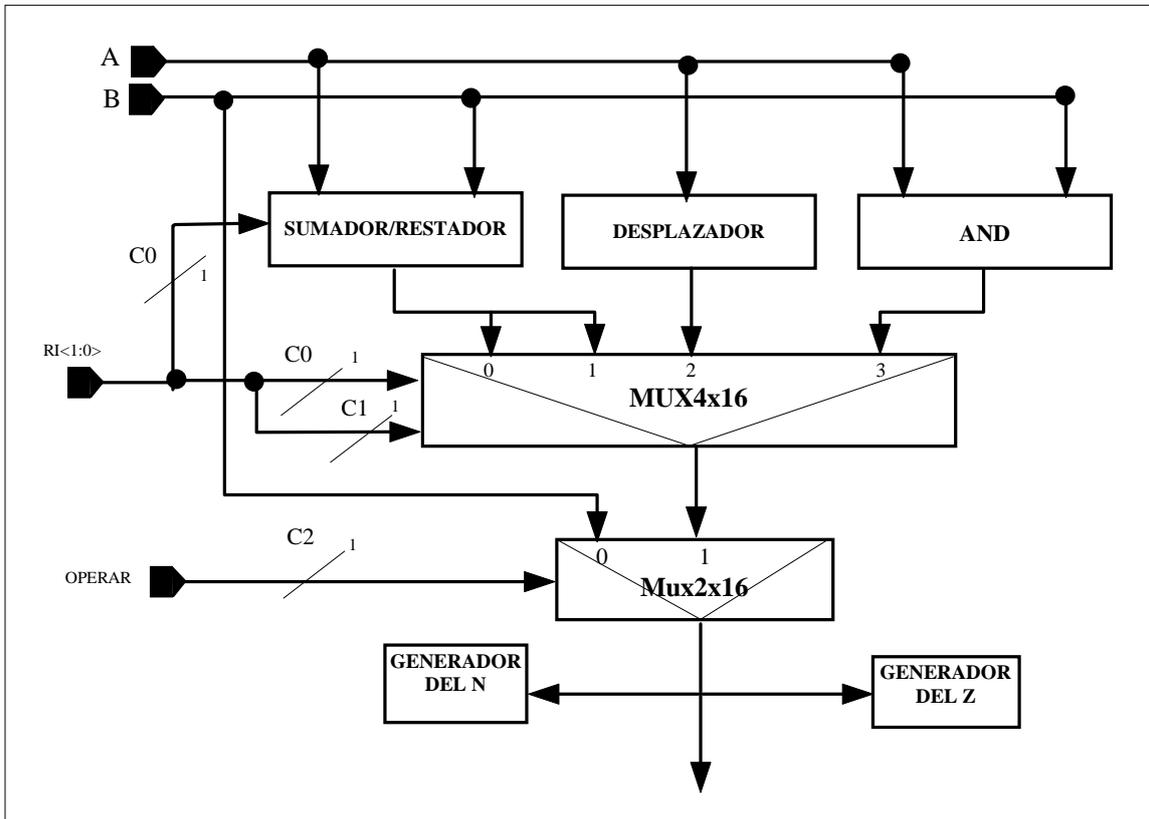
- el RA
- El MUX4x16
- FZ
- FN

Siendo Fz y Fn dos biestables a los que se les carga las salidas correspondientes de la ALU. Conviene que estos dos biestables, (como todos los registros y biestables del computador) estén conectados a una señal de reset que debe ser la misma para todo el computador. A continuación estudiamos cada uno de los siguientes submódulos:

- Unidad aritmético lógica

- Evalúa condición de salto
- Extensión

10.2.2.1 Unidad Aritmético Lógica



Las señales de control de la Unidad aritmético lógica son tres C0, C1 y C2:

C2	C1	C0	función
0	X	X	PASA B
1	0	0	A+B
1	0	1	A-B
1	1	0	DESP A
1	1	1	AND

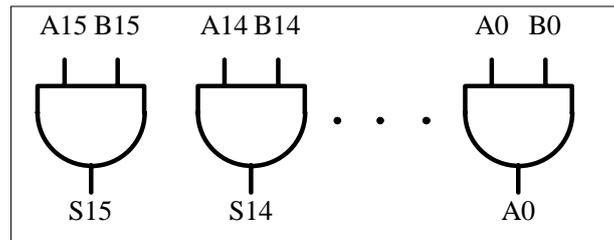
C1 y C0 vienen directamente del registro de instrucciones y C2 viene de la unidad de control. Esto simplifica el diseño del control del computador. La unidad aritmético lógica se descompone en los siguientes submódulos:

- Sumador/Restador en complemento a 2
- Desplazador
- AND
- Generador del N
- Generador del Z
- MUX4x16
- MUX2x16

Los multiplexores no se explican pues la dinámica de diseño es similar a la ya vista. El resto de los submódulos se ven en las secciones siguientes.

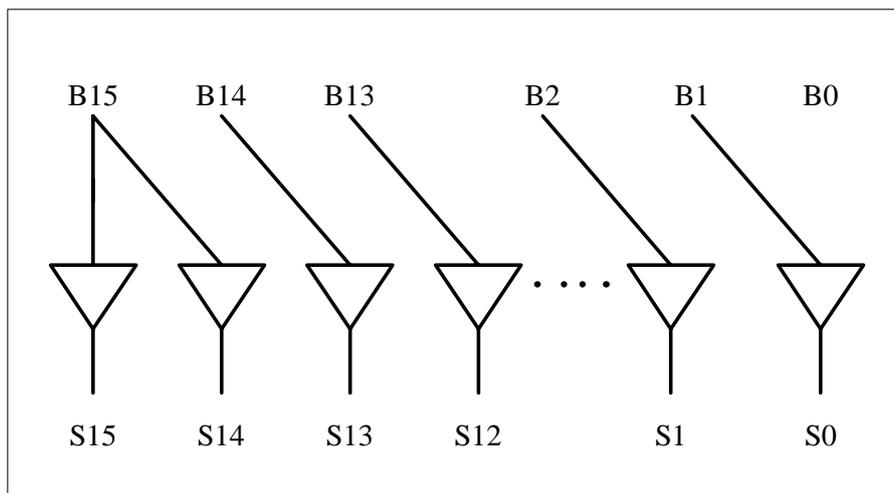
10.2.2.1.1 Módulo lógico AND.

Realiza una operación lógica AND bit a bit. Es muy sencillo de implementar. Únicamente se necesita la celda estándar AND2 de la biblioteca de celdas estándares StdLib. Un esquemático se puede ver a continuación.



10.2.2.1.2 Desplazador aritmético

Este módulo un realiza un desplazamiento aritmético a la derecha de un bit. El esquemático de este módulo aparece en la siguiente figura:

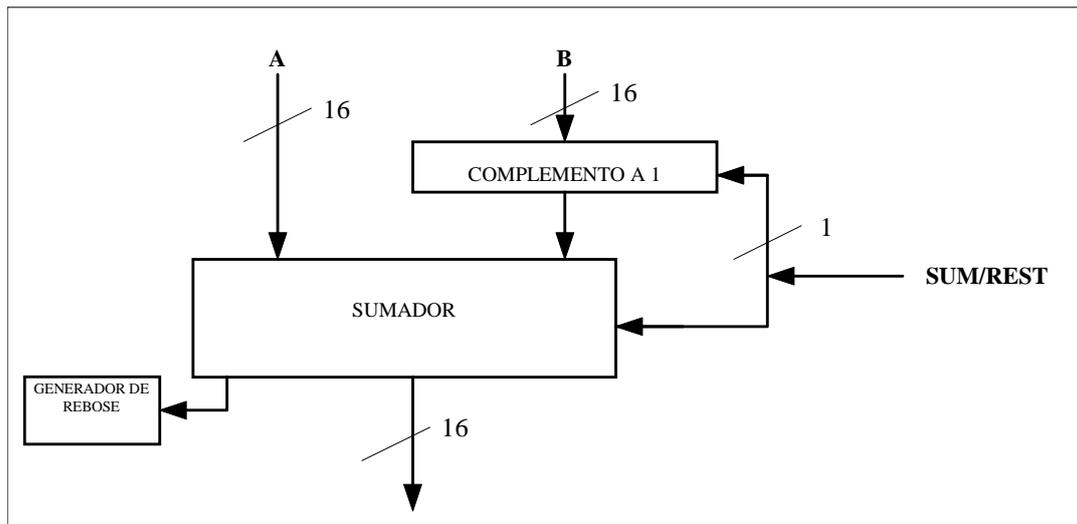


Para este módulo utilizamos la celda estándar de la librería Stdlib llamada BUF39

Darse cuenta que el bit de entrada B15 ataca a dos bits de salida el S15 y el S14. Esta es la forma de impedir que se pierda el signo en el desplazamiento aritmético. Además el bit B0 se pierde. Esto es equivalente a realizar una división por dos redondeada por defecto.

10.2.2.1.3 Sumador restador en complemento a dos

El esquemático del sumador restador es el siguiente:

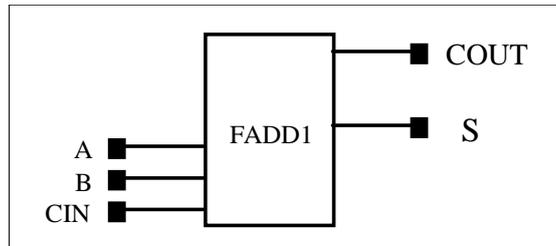


Los módulos del sumador /restador son los siguientes:

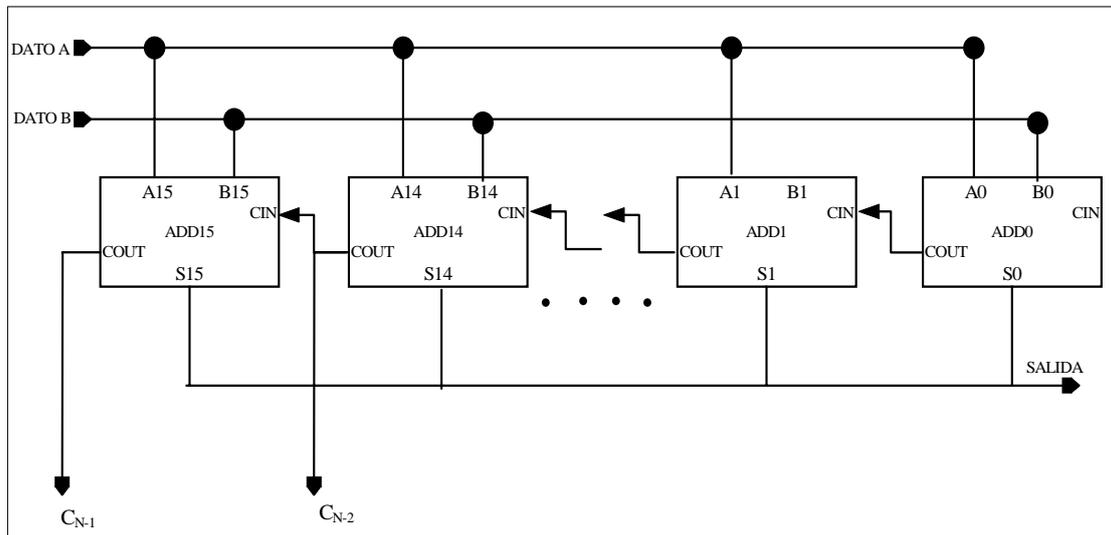
- El sumador binario
- Generador de complemento a 1
- Generador de rebose.
- Sum/resta es la señal de control que determina si el módulo suma o resta.

v **Sumador binario.**

Para implementar el sumador vamos a utilizar la celda estándar FADD1 de la librería StdLib cuyo símbolo es el siguiente:



El esquemático del sumado binario es el siguiente:



v Generador de complemento a uno

Recordemos que la función del generador del complemento a uno depende del valor de la señal de control.

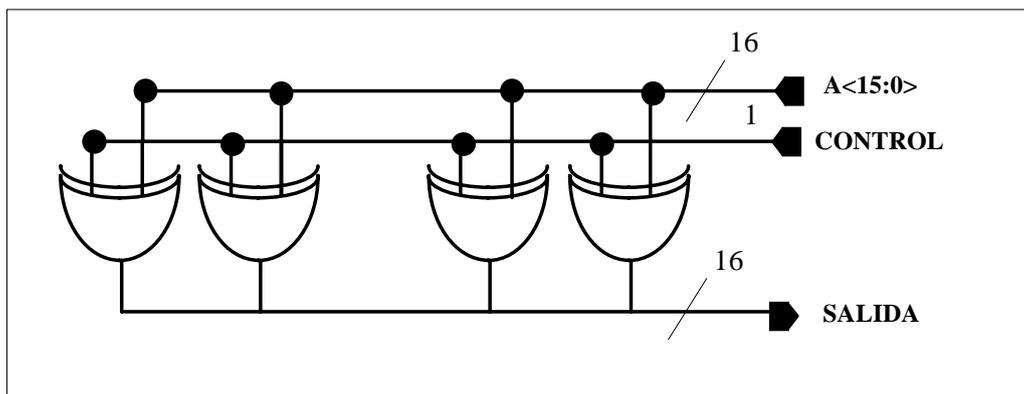
Si control = 0 salida = entrada

Si control = 1 salida = not(entrada)

Para implementarlo nada mejor que utilizar una puerta XOR cuya tabla de verdad es:

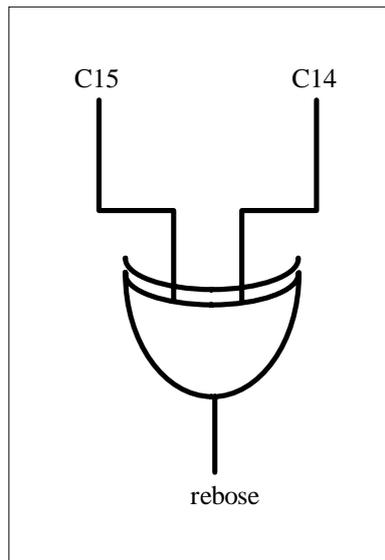
A<i>	control	salida
0	0	0
0	1	1
1	0	1
1	1	0

Para este submódulo utilizamos la celda estándar XOR de la librería StdLib. El esquemático es el que aparece a continuación:



v Generador de desbordamiento:

Aunque este módulo y la señal que genera no se utilizan posteriormente en el computador, se estudia para que el sumador sea lo más completo posible. Su esquemático aparece a continuación:



Siendo C15 y C14 los acarrees de los sumadores 15 y 14 del sumador restador.

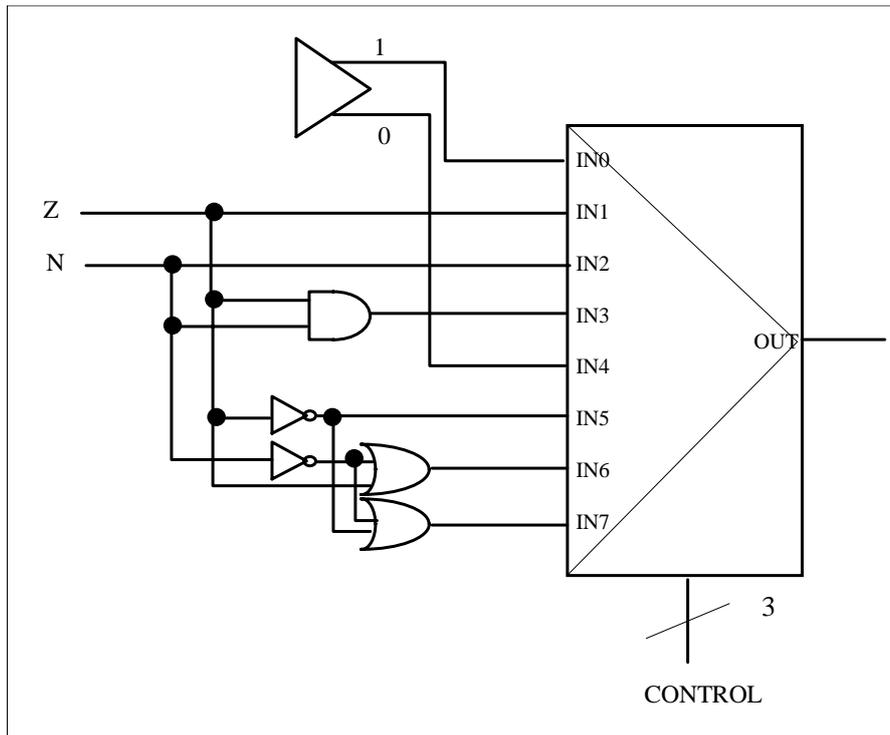
NOTA: Tener cuidado a la hora de realizar la simulación con el radix elegido en STL, y con el radix que se utiliza en la ventana de ondas.

10.2.2.3 Evaluador condición de salto

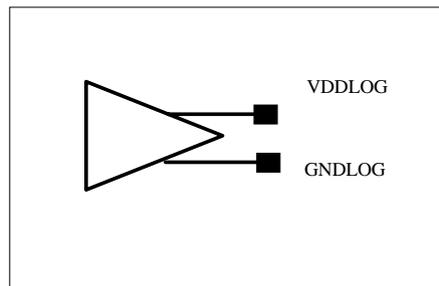
Es un módulo combinacional que recibe como entrada el contenido del flag Z, el contenido del flag N y los bits del registro de instrucciones que indican que tipo de salto queremos. La salida es de un solo bit e indica si se cumple la condición de salto o no se cumple. Esta salida es una de las entradas a la unidad de control. La razón de realizar esta evaluación fuera de la Unidad de control es que simplifica el diseño de la misma. a tabla de condiciones es la siguiente:

TIPO DE SALTO	CODIGO	CONDICION
BR salto incondicional	000	1
BEQ si igual	001	Z=1
BL si menor	010	N=1
BLE menor o igual	011	N=1 o z=1
BNE si distinto	101	Z=0
BGE si mayor o igual	110	N=0 o Z=1
BG si mayor	111	N=0 y Z=0

Esta tabla se puede implementar muy fácilmente mediante el multiplexor MUX8x1 que ya hemos estudiado. El esquemático resultante es el siguiente:



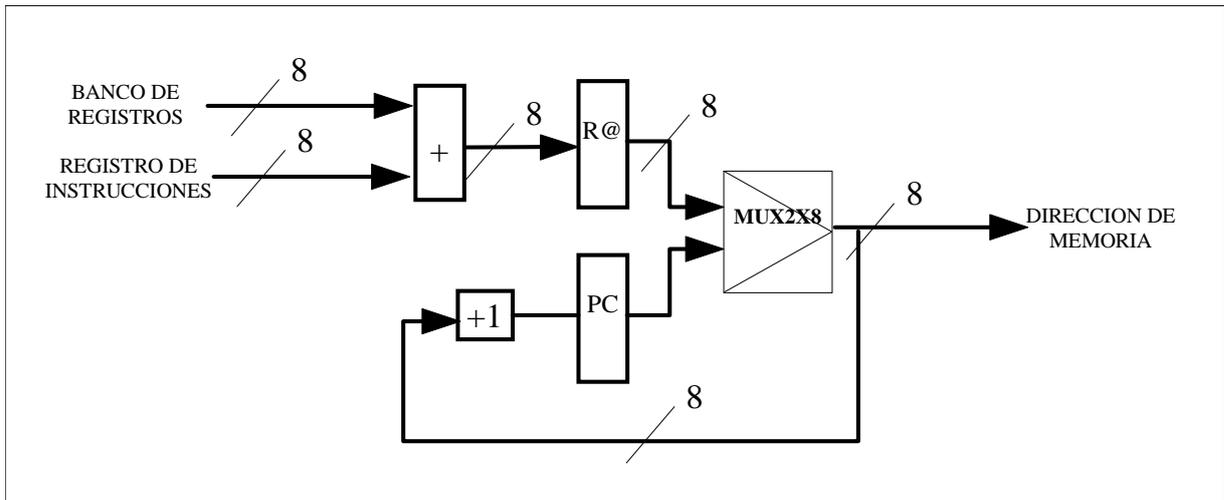
Donde la celda:



e la celda estándar LOG2 de la biblioteca y su única misión es proporcionar ceros y unos lógicos.

NOTA: este generador de valores lógicos no se debe utilizar nunca para alimentar megaceldas.

10.2.3 La unidad secuenciadora



Esta unidad se compone de los siguientes módulos:

- MUX2x8
- R@
- PC
- Incrementador
- Sumador binario de ocho bits

Todos estos módulos (o módulos parecidos) se han estudiado ya por eso no se entra en detalle en ninguno de ellos.

NOTA: recordar que el carry de entrada al incrementador debe ser 0

10.3 MEMORIA

La memoria del computador tiene las siguientes características:

- Memoria RAM
- Contiene datos e instrucciones
- 256 palabras de 16 bits cada una
- Dirección de 8 bits que permiten direccionar 256 palabras

Se va a utilizar las macroceldas de ES2. Se utilizará RAM cuya generación e instanciación se ha estudiado en capítulos anteriores. Recordar que el nombre de la celda debe comenzar por *ram*.

En la línea de comando CIW se escribe *ES2generate* y automáticamente surge una ventana de dialogo preguntando por el tipo de megacelda que se quiere implementar. En este caso se debe seleccionar RAM. A continuación se abre una ventana X en la que se deben ir contestando una serie de preguntas que determinan la configuración final de la memoria. Las preguntas son las siguientes:

BIST? yes, no

choose functional configuration

two unidirectional data buses (DI/DO)

with output latches on BUS d0(1)

with tristate outputs on bus d0(2)

number of words(4...1024)

bits per word: (1..128)

La elección de la opción BIST le indica al compilador que incluya en la memoria circuitos de autotest (Built in self test). A continuación se selecciona el tipo de buses que va a utilizar la memoria, en este caso un bus unidireccional para la entrada y otro para la salida.

La memoria puede conectar sus líneas de salida (DO) con el exterior o bien mediante latches o mediante puertas tristate. En el primer caso los datos permanecen estables en la salida hasta la siguiente operación de lectura. En el segundo caso, después que las capacidades se descargan el valor de salida es de alta impedancia.

Por último se debe seleccionar el número de palabras y el número de bits por palabra todo ello en los rangos especificados.

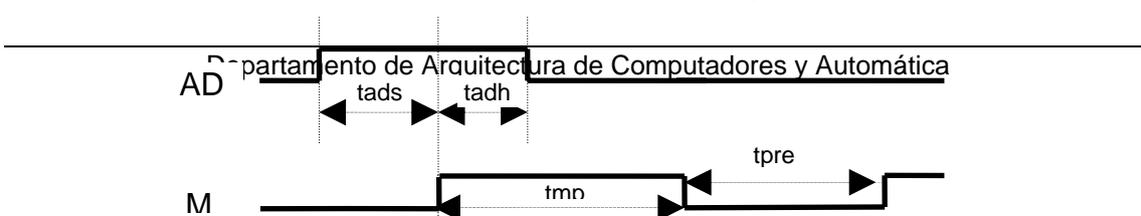
Con todos los datos anteriores, el compilador proporciona una tabla en la que aparecen diferentes configuraciones físicas de la memoria que puede seleccionar el diseñador.

Toda la información necesaria para la memoria RAM se almacena en el directorio físico de UNIX llamado *generate*. Dentro de este directorio los archivos con la extensión *.ds* (data sheet) contienen información sobre las megaceldas generadas.

Pines de la megacelda RAM:

- ME.- capacitación de memoria
- WE.- capacitación de la escritura, controla los modos de lectura escritura
- OE.- señal opcional de capacitación de las salidas de la memoria. En este caso se supone que en las salidas hay buffers triestate
- ADD.- dirección a la que se desea acceder.- Controla la selección de las celdas de memoria
- data i/o.- líneas de entrada y salida de datos pueden ser comunes, es decir bidireccionales o únicas(unidireccionales). La salida de datos se encuentran en modo de alta impedancia cuando la memoria esta descapacitada, cuando se está escribiendo en ella o mientras la señal OE esté descapacitada.
- Potencia (Vdd) y tierra (Gnd) . Están presentes para ofrecer al diseñador la posibilidad de de tener tierra y alimentación separadas para la megacelda. Esta capacidad es útil en grandes RAM. Si no se desean utilizar alimentaciones especiales para las megaceldas se deben seleccionar la tierra y la alimentación de la librería basic(por defecto los nombres *vdd!* y *gnd!* para indicar que las alimentaciones son las alimentaciones son las globales). Es importante que estos pines de alimentación de la megacelda no queden sueltos.

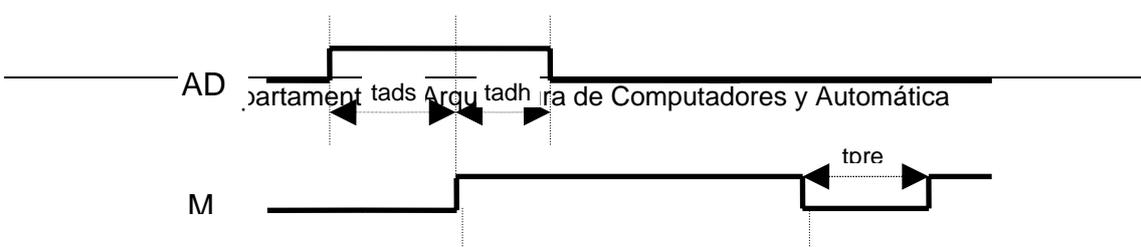
Las señales ME, WE y OE se pueden escoger activas a alta, o a baja. Es importante tener claro como deben ser las señales de control de la memoria en un ciclo de escritura y en un ciclo de lectura. A continuación aparece la forma de un ciclo de escritura suponiendo las señales ME y WE activas a alta.



Vamos a ver que significan cada uno de los tiempos que aparecen en la figura:

- tads.- tiempo de set up de la dirección respecto a la señal de capacitación de la memoria
- tadh.- tiempo de hold de la dirección respecto a la señal de capacitación de la memoria
- tmpw.- anchura mínima de la señal de capacitación de la memoria.
- twrs.- tiempo de setup de la señal de escritura respecto a la señal de capacitación de la memoria.
- twrh.- tiempo de hold de la señal de escritura respecto a la señal de capacitación de memoria.
- twds.- tiempo de setup del dato de entrada respecto a la señal de escritura
- twdh.- tiempo de set up del dato de entrada respecto a la señal de escritura

Tener en cuenta que el twrs debe ser ≥ 0 , es decir la señal de ME y WE pueden coincidir siempre que se cumplan las condiciones de tmpw y twrh. Los valores de estos tiempos dependen del tamaño de memoria y de la configuración elegida. Los tiempos de subida y de bajada para las señales de entrada rondan los tres nanosegundos y los tiempos de subida y de bajada de las salidas rondan los seis nanosegundos. A continuación se ve el ciclo de lectura



Los tiempos del ciclo de lectura son las siguientes:

- tads.- tiempo de setup de la dirección respecto a la señal de capacitación de la memoria
- tadh tiempo de hold de la dirección respecto a la señal de capacitación de la memoria
- tpre.- tiempo de precarga de la memoria
- trds.- tiempo de setup de la señal de lectura respecto a la señal de capacitación de la memoria
- trdh tiempo de hold de la señal de lectura respecto a la caída de la señal de capacitación de la memoria.
- tacc.- tiempo de acceso desde la señal de capacitación de la memoria
- tmed.- tiempo que tarda en hacerse la precarga de las líneas de salida desde la bajada de la señal de capacitación de la memoria.

Conviene recordar que la señal de capacitación de la memoria es equivalente a la señal de reloj que sincroniza la memoria con el resto del sistema. El trds y trdh pueden ser 0.

Para su instanciación se selecciona de la librería Megacell.

NOTA:

En la memoria RAM la sincronización se realiza mediante la señal de capacitación de memoria ME, por lo tanto los tiempos de setup y hold del resto de señales se toman respecto a esta señal.

Si $ME=0$ fase de precarga

si $ME=1$ fase de Lectura/escritura

la escritura se realiza siempre que $ME=0$ y $WE=0$, es decir no hace falta precarga

En el caso que la escritura se acabe porque $WE=1$ se entra en una fase de lectura con todos sus retardos y ligaduras

La señal con la que se debe tener más cuidado es la de capacitación de la escritura WE.

Par realizar lectura recordar que se debe realizar precarga poniendo $ME = 0$

Al simular la RAM no hace falta poner la opción Preserve Bus ON

Para implementar las señales de control de la memoria RAM pueden empezar la DIRECCIÓN la WE y el DATO DE ENTRADA al mismo tiempo y generar la ME un poco mas tarde.

10.4 LA UNIDAD DE CONTROL

La unidad de control es un sistema secuencial cuya misión es determinar el orden en que los distintos elementos de la Unidad de proceso deben operar para que se realice la ejecución de una instrucción. La forma de determinar este orden es activando las siguientes señales de control

- **LDRA** capacitación del registro acumulador
- **LDRI** capacitación del Registro de Instrucciones
- **LDPC** capacitación del contador de programa
- **LDR@** capacitación del registro auxiliar de direcciones
- **LDFZ** capacitación del flag cero
- **LDFN** capacitación del flag de signo
- **ERD** capacitación de escritura en el banco de registros
- **L/E** señal de escritura lectura de la memoria
- **MUXPC** selecciona la dirección de memoria
- **MUXRG<1:0>** selecciona la dirección del banco de registros
- **OPERAR** selecciona el tipo de operación de la ALU

La unidad de control tiene como entradas el código de operación y el resultado de evaluar la condición de salto. La unidad de control actúa sobre los elementos de la unidad de procesos:

- Permitiendo la carga de registros
- Indicando si la unidad aritmético lógica debe operar
- Indicando a la memoria el tipo de operación que debe realizar.

De manera general la ejecución de una instrucción se divide en varias fases:

- Fase de búsqueda de la Instrucción
- Fase de decodificación
- Búsqueda de operando y evaluación de condiciones de salto
- Fases de ejecución de la instrucción

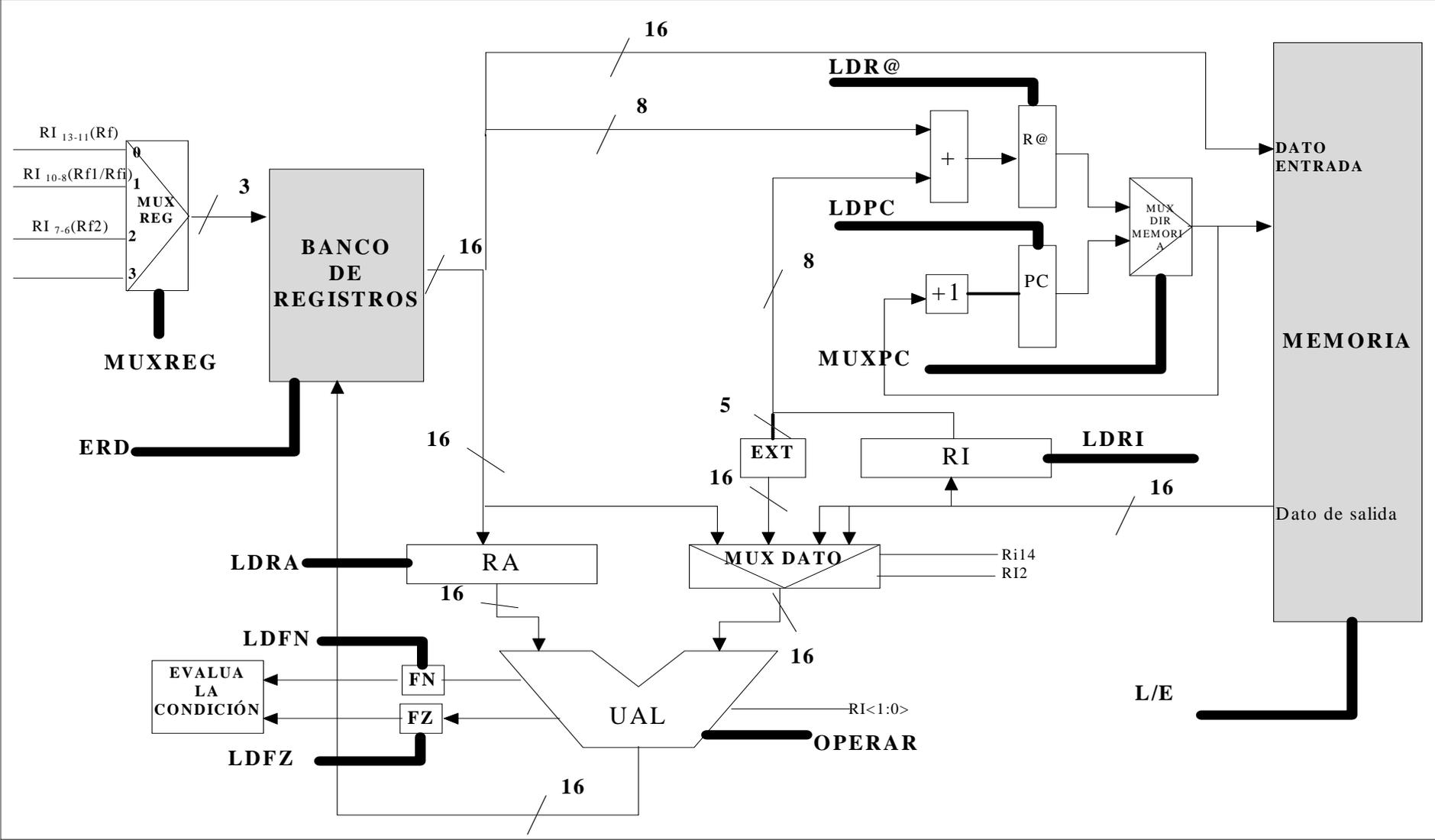
Cada fase se divide en un conjunto de operaciones y para llevar a cabo una operación se deben activar alguna de las señales de control que se han

visto. En función del camino de datos implementado habrá operaciones que se puedan realizar simultáneamente, cuando no utilizan los mismos buses ni las mismas unidades funcionales, y habrá operaciones que se deben realizar en serie, cuando utilizan los mismos buses o las mismas unidades funcionales.

El número de conjunto de operaciones que se deben realizar en serie para implementar una fase determina el número de ciclos de reloj necesarios para ejecutar la fase.

El número de ciclos de una instrucción es la suma de todos los ciclos necesarios para ejecutar las cuatro fases ya nombradas.

En las siguientes secciones se estudian las operaciones, el número de ciclos, el grafo de estados y las señales de control que tiene que generar la unidad de control para implementar cada instrucción del computador



10.4.1 Fases comunes

Para este computador existen dos fases comunes a todas las instrucciones,

- La fase de búsqueda (fetch)
- La fase de decodificación

10.4.1.1 Fase de búsqueda

En esta fase se realizan las siguiente operaciones

$RI \leftarrow m[PC]$

$PC \leftarrow PC+1$

En el camino de datos se puede ver con facilidad que estas dos operaciones no utilizan los mismos módulos en ningún caso y por lo tanto se pueden llevar a cabo en el mismo ciclo de reloj.

Nota: Conviene recordar que los registros se actualizan en los flancos de subida de la señal de reloj. Las señales que llegan desde la unidad de control a los registros son señales de capacitación.

10.4.1.2 Fase de decodificación

Durante la fase de decodificación la lógica de la unidad de control evalúa el código de operación de la instrucción con el objeto de decidir cuales son las siguientes acciones a realizar. El código que corresponde a cada clase de operación es el siguiente:

00 instrucciones LOAD

01 instrucciones STORE

10 instrucciones de SALTO

11 instrucciones aritmético - lógicas

Durante esta fase la unidad de proceso no debe realizar ningún tipo de operación. El contenido de todos los registros de la unidad de proceso se preserva durante esta fase poniendo a cero sus señales de carga.

10.4.2 Instrucciones aritmético lógicas

Por simplicidad el procesador del computador ha sido diseñado de modo que todas las instrucciones aritmético lógicas se ejecuten siguiendo las mismas secuencias de operaciones, tanto si se opera con dos operandos almacenados en registros, con un operando en registro y otro inmediato o con un sólo operando. Las dos operaciones en que se dividen las instrucciones aritmético lógicas son las siguientes:

1. Búsqueda del primer operando que se encuentra en el banco de registros
2. Búsqueda del segundo operando, ejecución de la instrucción y almacenamiento en los registros del resultado.

10.4.2.1 Búsqueda del primer operando

La operación a ejecutar es la siguiente:

RA←RF1

Es decir se carga en RA el contenido del registro RF1 del banco de registros cuya dirección proviene del registro de instrucciones, para ello hay que activar la señal de capacitación del registro RA.

En el caso de la instrucción ASR (desplazamiento aritmético a la derecha) este primer paso se realiza para homogeneizar el tratamiento de todas las instrucciones aritmético lógicas. La información que se carga en el registro RA en este caso no tiene importancia.

10.4.2.2 Búsqueda del segundo operando y ejecución

Según que tipo de instrucciones aritmético lógicas quieras ejecutar así serán las operaciones a realizar.

- v **Para las instrucciones con dos operandos en el banco de registros las operaciones son:**

$RD \leftarrow RA \text{ OP } RF2$

FZ

FN

- v **Para las instrucciones con un operando inmediato las operaciones son:**

$RD \leftarrow RA \text{ OP } \text{EXT}(RI_{\langle 7:3 \rangle})$

FZ

FN

Siendo EXT una operación auxiliar de extensión de signo necesaria para transformar un número en complemento a dos codificado con 5 bits a un número en Ca2 codificado a 16 bits.

- v **Para las instrucciones de desplazamiento:**

$RD \leftarrow Rf2 \gg 1$

FN

FZ

El tipo de operación que se quiere ejecutar no viene explicitado en el campo código (bits 15 y 14) sino en el campo operación (bits 1 y 0). La señal que proporciona la unidad de control en estos casos es **OPERAR** que debe estar activada a uno para que la operación aritmético lógica se realice.

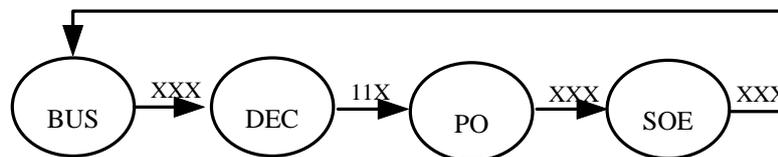
10.4.2.3 Diagrama de estados de la operaciones aritmético - lógicas

Para interpretar los grafos de estado debemos saber que:

- los círculos representan los estados
- las flechas indican la relaciones entre estados
- los códigos sobre las flechas indican la información de entrada a la unidad de control es decir **RI<15:14> y condición.**

Los cuatro estados en los que se divide la ejecución de una instrucción aritmético lógica son:

1. Búsqueda de la instrucción (**BUS**)
2. Descodificación (**DEC**)
3. Búsqueda del primero operando (**PO**)
4. Búsqueda del segundo operando y ejecución (**SOE**)



Conforme a este diagrama de estado y al camino de datos las señales de control que deberían activarse son las siguientes:

señales de control	BUS	DEC	PO	SOE
LDRA	0	0	1	0
LDRI	1	0	0	0
LDPC	1	0	0	0
LDR@	0	0	0	0
LDFZ	0	0	0	1
LDFN	0	0	0	1
ERD	0	0	0	1
L/E	0	0	0	0
MUXPC	0	X	X	X
MUXREG<1:0>	X	X	1	2
OPERAR	X	X	X	1

10.4.3 Instrucciones de acceso a memoria

Existen dos instrucciones de acceso a memoria diferentes:

- **LOAD** lectura de memoria
- **STORE** escritura en memoria

en ambos casos se utiliza un registro del banco de registros mas un campo de la propia instrucción para calcular la dirección de memoria a la que se accede. Estas dos instrucciones tiene realizan dos operaciones cada una:

1. cálculo de la dirección de memoria y posterior almacenamiento en el registro auxiliar de direcciones R@
2. acceso a memoria

v **Cálculo de la dirección de memoria:**

$$R@ \leftarrow R_i \langle 7:0 \rangle + RI \langle 7:0 \rangle$$

donde

R_i es el registro índice especificado por el RI

RI<7:0> son los siete bits menos significativos del registro de Instrucciones.

v **Acceso a memoria**

Depende si la operación es load o store:

- **STORE** (Escritura en memoria)
- **LOAD** (Leer la memoria y cargar el contenido en el banco de registros)

$$M[R@] \leftarrow RF$$

$$RD \leftarrow M[R@]$$

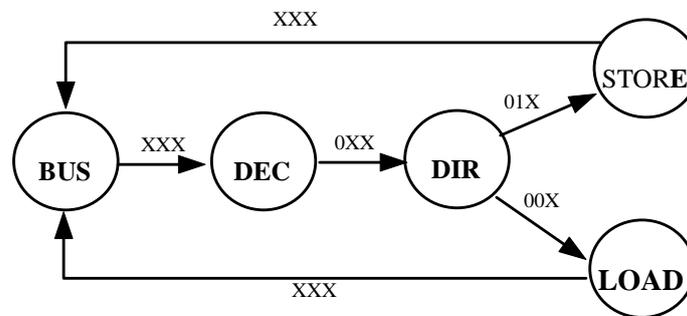
FN

FZ

Conviene recordar que el camino de datos de lectura de memoria pasa por la entrada B de la unidad aritmético lógica.

10.4.3.1 Diagrama de estados

En definitiva las operaciones de load y store tiene el siguiente diagrama de estados



Donde

- BUS y DEC son los estados comunes a todas las instrucciones,
- DIR es el estado en el que se calcula dirección de memoria a la que se quiere acceder y
- LOAD y STORE son los dos estados de acceso.

Fijándose en la figura del camino de datos se puede ver que la tabla de señales de control que se deben activar es la siguiente:

señales de control	dir	store	load
LDRA	0	0	0
LDRI	0	0	0
LDPC	0	0	0
LDR@	1	0	0
LDFZ	0	0	1
LDFN	0	0	1
ERD	0	0	1
L/E	0	1	0
MUXPC	X	1	1
MUXRG<1:0>	1	0	X
OPERAR	X	X	0

10.4.4 Instrucciones de salto

Todas las instrucciones de salto se ejecutan de forma similar. Una vez descodificada, la instrucción de salto tiene dos fases bien diferenciadas

- Evaluación de la condición de salto
- En caso de realizar el salto cálculo de la nueva dirección
en caso contrario salta a la fase común de fetch.

La evaluación de la condición de salto se realiza mediante un módulo combinacional implementado en el camino de datos. Este módulo genera un único bit que indica exclusivamente si realiza el salto o no.

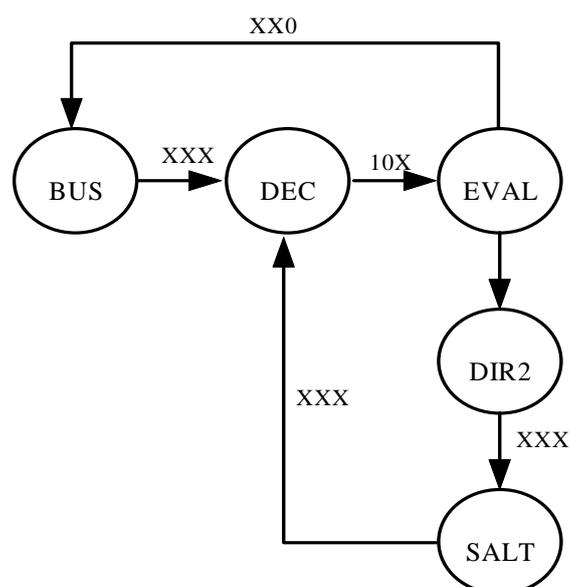
La fase de calculo de la nueva dirección sólo se realiza en caso que si haya salto y consiste en calcular la dirección y ejecutar el salto propiamente dicho.

primera fase $R@ \leftarrow RI < 7:0 >$

segunda fase $PC \leftarrow R@ + 1$

$RI \leftarrow M[R@]$

10.4.4.1 Diagrama de estados



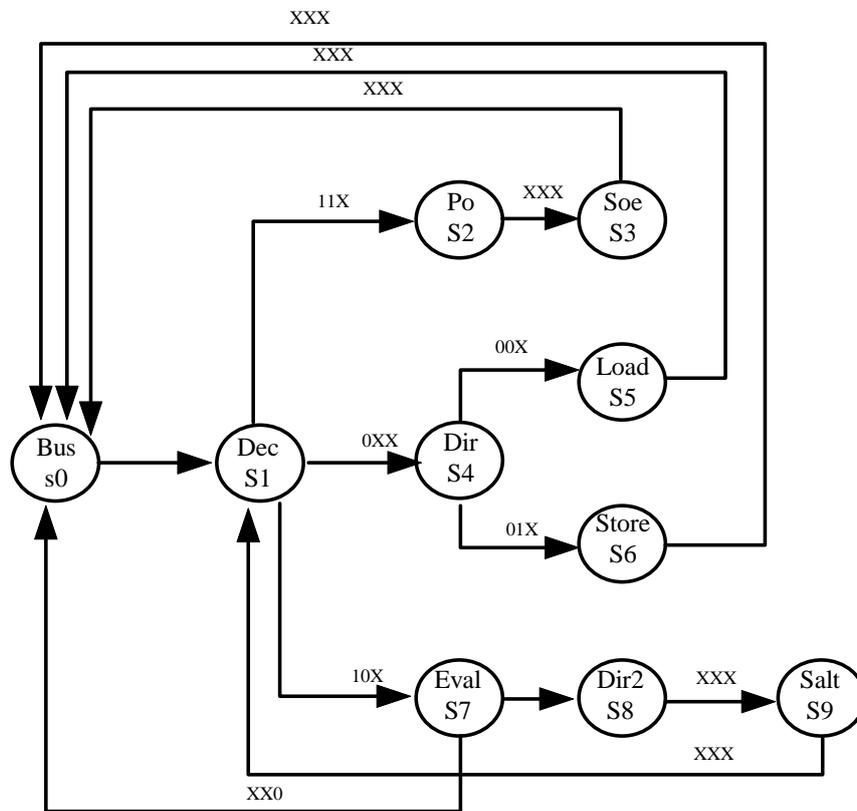
Donde :

- EVAL** es el estado en el que se evalúa la condición de salto,
- DIR2** es el estado en el que se calcula la nueva dirección y
- SALT** es de hecho el estado BUS de la siguiente instrucción.

Fijándose en el camino de datos de la figura se puede ver que las señales de control que tiene que activar la unidad de control para cada uno de los estados es la siguiente:

señales de control	eval	dir2	salt
LDRA	0	0	0
LDRI	0	0	1
LDPC	0	0	1
LDR@	0	1	0
LDFZ	0	0	0
LDFN	0	0	0
ERD	0	0	0
L/E	0	0	0
MUXPC	X	X	1
MUXRG<1:0>	0	1	X
OPERAR	X	X	x

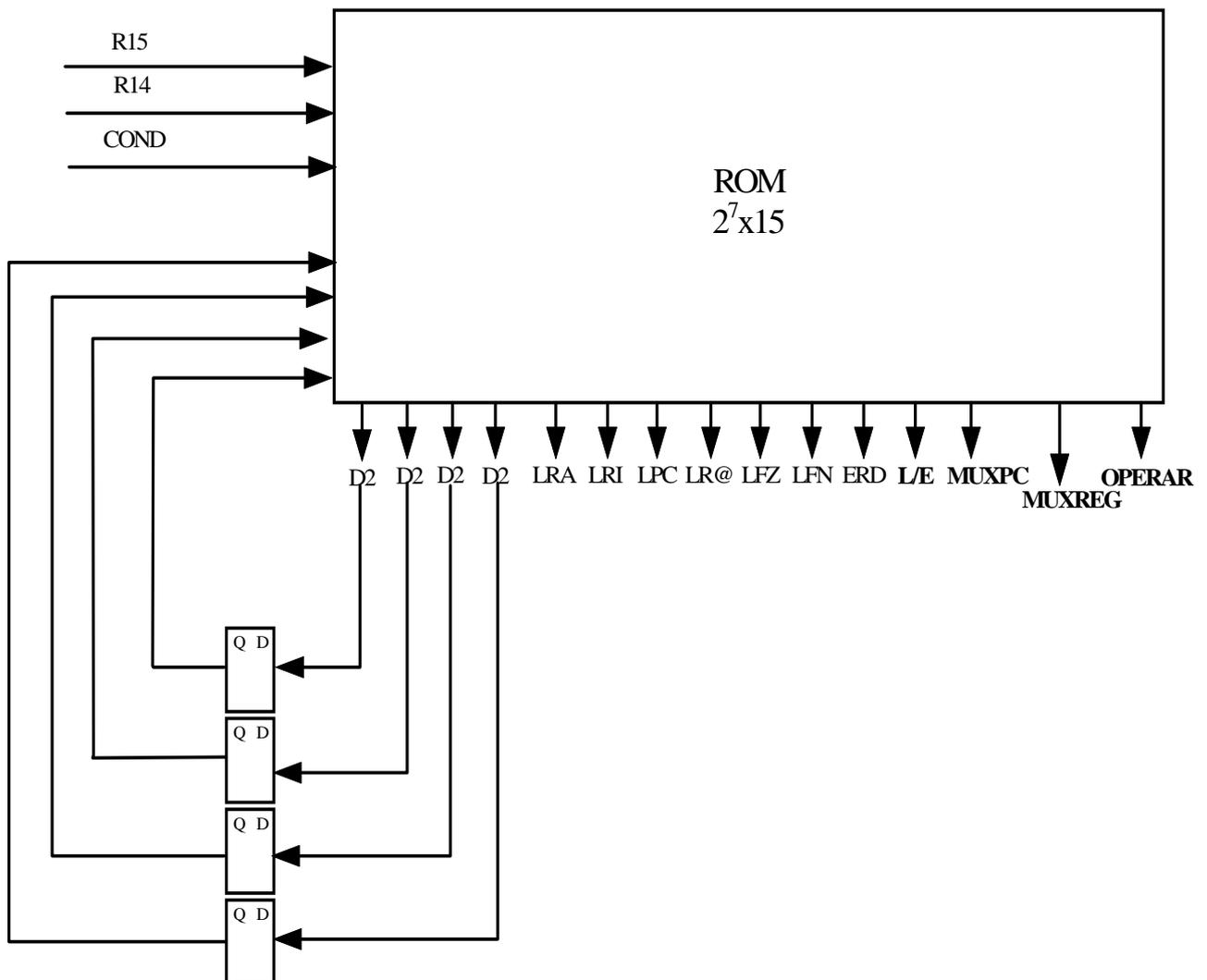
10.4.5 Diagrama de estados y tabla de salida de la unidad de control



A este diagrama de estado le corresponde la siguiente tabla de salidas de la unidad de control:

salidas	bus	dec	dir	load	store	po	soe	eval	dir2	salt
LDRA	0	0	0	0	0	1	0	0	0	0
LDRI	1	0	0	0	0	0	0	0	0	1
LDPC	1	0	0	0	0	0	0	0	0	1
LDR@	0	0	1	0	0	0	0	0	1	0
LDFZ	0	0	0	1	0	0	1	0	0	0
LDFN	0	0	0	1	0	0	1	0	0	0
ERD	0	0	0	1	0	0	1	0	0	0
L/E	0	0	0	0	1	0	0	0	0	0
MUXPC	0	X	X	1	1	X	X	X	x	1
MUXRG<1:0>	X	X	1	X	0	1	2	X	1	X
OPERAR	X	X	X	0	X	X	1	X	x	X

10.4.6 Realización de la unidad de control con una ROM



Para implementar la unidad de control necesitamos una memoria ROM y un conjunto de biestables. En los biestables se guarda el estado actual de la máquina de estados y en la memoria ROM se guarda el siguiente estado y las señales de control correspondientes al estado actual. Por lo tanto si tenemos diez estados necesitamos cuatro biestables para codificarlos.

La dirección a la que se accede viene dada por el contenido de los biestables y por las entradas, por lo tanto en nuestro caso se necesita un bus de direcciones de 7 bits lo que indica que la memoria tendrá 128 posiciones de las cuales no todas se utilizan. En cuanto a los bits por palabra tendrá cuatro para representar el estado y doce señales de control propiamente dichas.

A continuación vamos a estudiar como habría que rellenar la memoria. Las direcciones vienen dadas por los siguientes bits

Q3 Q2 Q1 Q0 RI₁₅ RI₁₄ COND

es decir Q3 es el bit de dirección de mayor peso y COND es el bit de dirección de menor peso:

<i>Si</i>	Q3	Q2	Q1	Q0	RI15	RI14	COND	D3	D2	D1	D0	LRA	LRI	LPC	LR@	LFZ	LFN	ERD	LE	MPC	MREG	OPER
<i>BUS</i>	0	0	0	0	X	X	X	0	0	0	1	0	1	1	0	0	0	0	0	0	x	x
<i>DEC</i>	0	0	0	1	1	1	X	0	0	1	0	0	0	0	0	0	0	0	0	x	x	x
<i>DEC</i>	0	0	0	1	0	X	X	0	1	0	0	0	0	0	0	0	0	0	0	x	x	x
<i>DEC</i>	0	0	0	1	1	0	X	0	1	1	1	0	0	0	0	0	0	0	0	x	x	x
<i>PO</i>	0	0	1	0	X	X	X	0	0	1	1	1	0	0	0	0	0	0	0	x	1	x
<i>SOE</i>	0	0	1	1	X	X	X	0	0	0	0	0	0	0	0	1	1	1	0	x	2	1
<i>DIR</i>	0	1	0	0	0	0	X	0	1	0	1	0	0	0	1	0	0	0	0	x	1	x
<i>DIR</i>	0	1	0	0	0	1	X	0	1	1	0	0	0	0	1	0	0	0	0	x	1	x
<i>LOAD</i>	0	1	0	1	X	X	X	0	0	0	0	0	0	0	0	1	1	1	0	1	x	0
<i>STORE</i>	0	1	1	0	X	X	X	0	0	0	0	0	0	0	0	0	0	0	1	1	0	x
<i>EVAL</i>	0	1	1	1	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x
<i>EVAL</i>	0	1	1	1	X	X	1	1	0	0	0	0	0	0	0	0	0	0	0	x	x	x
<i>DIR2</i>	1	0	0	0	X	X	X	1	0	0	1	0	0	0	1	0	0	0	0	x	1	x
<i>SALT</i>	1	0	0	1	X	X	X	0	0	0	1	0	1	1	0	0	0	0	0	1	x	x

En la tabla anterior aparece la información que se debe cargar en la memoria ROM. Vamos a ver un ejemplo de como se llenaría la memoria. En la tabla se ve que la información relativa al estado BUS se debe cargar en las posiciones de memoria determinadas por los valores 0000xxx, para todas las posibles combinaciones de xxx, es decir desde 0000000 hasta 00001111,

10.4.7 La memoria ROM de la unidad de control

Esta tabla de salidas es la que se debe cargar después en la memoria ROM de la unidad de control. Esta memoria se debe crear a partir de una megacelda de tipo ROM.

La tabla no refleja las señales de control de la memoria RAM. Su conversión a la tabla real es muy sencilla. Solo hay que añadir las señales que aparezcan en la megacelda RAM utilizada para implementar la memoria RAM de instrucciones.

A la hora de realizar la implementación de la unidad de control, se debe tener en cuenta que las señales de control de la memoria deben guardar unos tiempos de set up y de hold. Se puede aplicar dos posibles soluciones a este problema:

- utilizar una misma señal de control para activar las señales de control de la memoria, WE y ME, utilizando buffers de retardo para generar los tiempos de set-up y de hold
- utilizar varios pasos de control para implementar correctamente las señales de escritura y lectura.

Vamos a ver con un poco de más detalle la manera de trabajar con esta memoria ROM. Se genera utilizando las megaceldas de las librerías de ES2. La forma de hacerlo es la siguiente:

Hay que especificar el dato que se almacena en cada dirección de memoria. Esto se hace creando en el directorio *generate* un archivo *nombre.prg*, siendo nombre el nombre elegido para la megacelda, que en este caso debe empezar por *rom*.

Este archivo consta de dos columnas. La primera indica la dirección de la palabra de memoria, y la segunda indica el dato que almacena dicha dirección. Estas dos columnas tienen tantas filas como palabras tiene la memoria. Las direcciones se pueden especificar en

- binario,
- decimal
- hexadecimal.

Los datos se pueden expresar

- binario
- hexadecimal

Una vez creado el archivo *nombre.prg*, activamos el compilador mediante el comando *ES2generate* que pregunta por la megacelda que se quiere generar (ROM, RAM, PLA..) y por el nombre que se le quiere dar.

A continuación surge una ventana X que va preguntando por una serie de parámetros parecidos a los que se preguntaban en el caso de la RAM. Además hay que responder a las siguientes preguntas:

Expect pattern file is: nombre.prg

specify the address format

binary

decimal

hexadecimal

specify the data format

binary

hexadecimal.

Un ejemplo de un archivo *nombre.prg* podría ser el siguiente:

0	3f
1	1
2	14
3	0
4	a
5	3a

en el que las direcciones y los datos los damos en hexadecimal. Las señales de control de la ROM son las siguientes:

ME.- capacitación de la memoria

ADD.- dirección de la la palabra de memoria a la que se quiere acceder

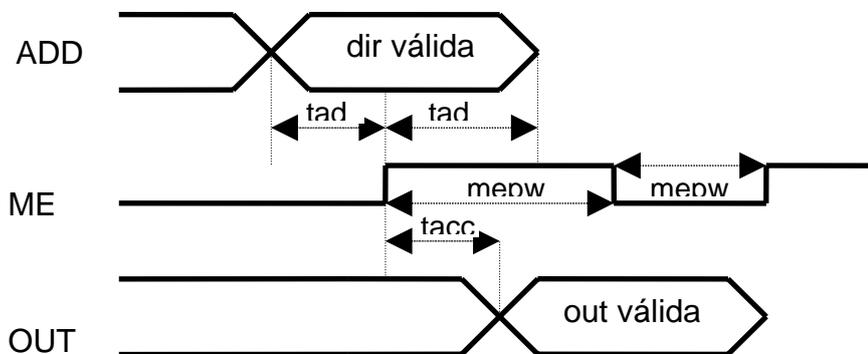
DO.- La salida de datos

OE .- señal de capacitación de la salida triestate cuando es esta opción la que se selecciona.

VDD, GND.- Cuyo uso principal es indicar si se quiere que lleguen a la megacelda líneas independientes de vdd y gnd.

Una vez que se ha generado la memoria ROM se puede ver en el directorio *generate* en un archivo nombre.ds (data sheet) todos los datos de tiempos, retardos, consumos y capacidades de dicha megacelda.

A continuación se estudia el ciclo de lectura de la memoria. Conviene saber que una vez que se ha leído el dato, la ROM ,de manera automática, realiza la precarga de las líneas de datos, es decir los datos no permanecen constantes en las líneas de salida. De esta forma con una fase de reloj sencilla que ataque a la señal ME se puede leer un dato.



Donde:

- **tads**.- tiempo de setup de la dirección respecto a la señal de de capacitación de la memoria.
- **tadh**.- tiempo de hold de la dirección respecto de la señal de capacitación de la memoria
- **tacc**.- tiempo de acceso de la memoria
- **mepw1**.- anchura en alza de la capacitación de la meoria
- **mepw0**.- anchura del pulso en baja

11. PLACE & ROUTE

11.1 OPERACIÓN DE FLATTENING DE LA NETLIST.

Antes de comenzar la operación es importante comprobar que se han instalado los pads de alimentación y de tierra del esquemático de más alto nivel. También se debe comprobar que se ha colocado la celda LIBTOPNETS , en el caso que se desee realizar la verificación layout versus esquemático(LVS).

A continuación desde la ventana del esquemático del más alto nivel de la jerarquía se activan los siguientes comandos:

Menu>tools→floorplan/schematics

Menu>floorplan→hierarchy browser

surge una ventana de dialogo preguntando por el nombre de la librería el nombre de la celda y el tipo de vista, tipo que habrá que seleccionar como schematic.

Al pulsar la opción OK aparece una nueva ventana con una representación del diseño. A esta ventana se le llama *Hierarchy Library Browser*.

a continuación se debe pulsar en el interior del diseño aparecido que cambiará su color a rojo

Menu >hierarchy→properties→set master

aparece una ventana de dialogo en la que se dejan todos los valores por defecto.

Menu>hierarchy→generate physical hierarchy

este comando genera la vista autolayout . Este no es un proceso interactivo hay que esperar hasta que se acaba. El mensaje de finalización correcta aparece en la ventana CIW. Una vez concluido correctamente el proceso se puede cerrar la ventana *Hierarchy Browser*

11.2 ACTIVACIÓN DE PLACE & ROUTE

Para activar las opciones de place and route hay que abrir la celda de tipo autolayout que se ha generado en el paso anterior. para ello:

menu>open→design

y aparece una ventana de dialogo preguntando por:

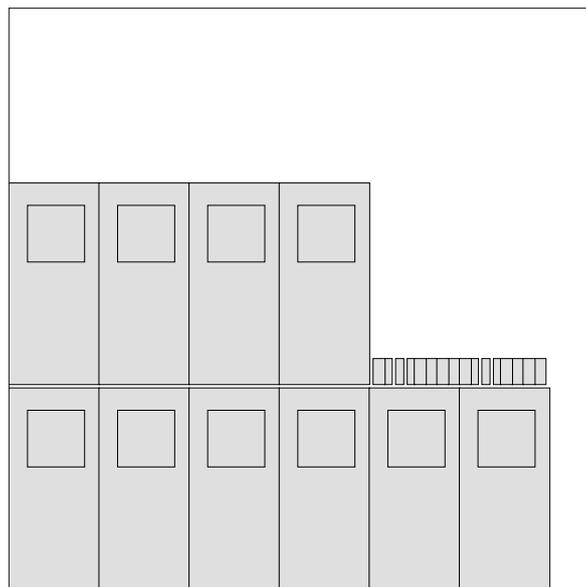
el nombre de la librería

el nombre de la celda

el tipo de vista que habrá que seleccionar como autolayout

surge una ventana en la que aparecen todos los elementos físicos del diseño en el interior de un cuadrado. Estos elementos son:

- Pads de entrada/salida
- Pads de alimentación
- Celdas estándares
- Macroceldas:



La opción de *place&route* se activa desde la ventana de autolayout de la siguiente manera:

menu>tools→floorplan/P&R→cell ensemble

En este momento las opciones de P&R aparecen en la parte superior de la ventana.

Aparte de la ventana de autolayout aparece otra ventana a la izquierda llamada Object Selection Window (OSW)

11.2.1 ASIGNACIÓN DE PRIORIDADES DE LOS NODOS

Este paso sirve para cargar el archivo de prioridades netData

```
menu>route→modify Net→net properties file
```

```
form>net properties file [read]
```

```
form>nets [selected]
```

para el nombre de fichero de propiedades utiliza el defecto:

```
form> properties file name <install-dir>/<rocess>/utl/startup/netData
```

!!!! ATENCION ESSTE ARCHIVO HAY QUE LOCALIZARLO Y COPIARLO EN EL DIRECTORIO DEL DISEÑO !!!

11.2.2 INICIALIZACIÓN DEL FLOORPLANING

menu>floorplan→reinitialise

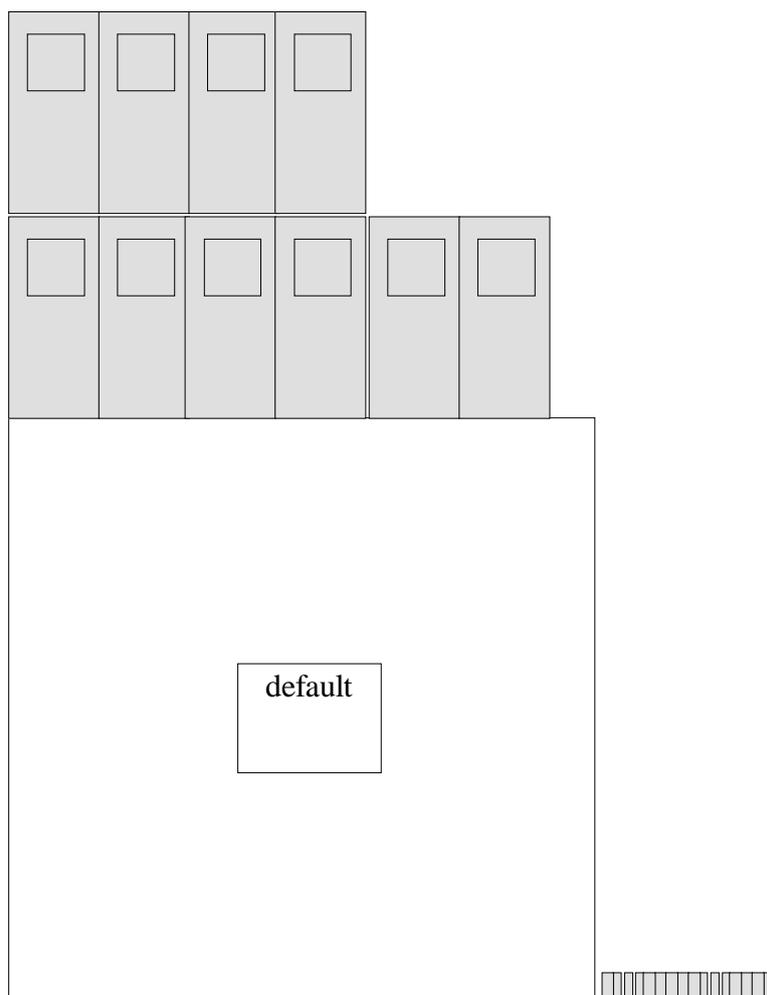
se dejan todas las opciones de la ventana de dialogo por defecto.

El proceso no es interactivo y por lo tanto hay que esperar hasta que sea completa. Esto genera una región a los pads en el exterior y una región interna en la que se colocarán las celdas estándar y las megaceldas.

Una vez acabada la inicialización se puede ver toda la ventana siguiendo los siguientes pasos:

menu>window→fit

el resultado es el siguiente:



A continuación habrá que colocar los pads de entrada salida en el floorplan para ello hacer:

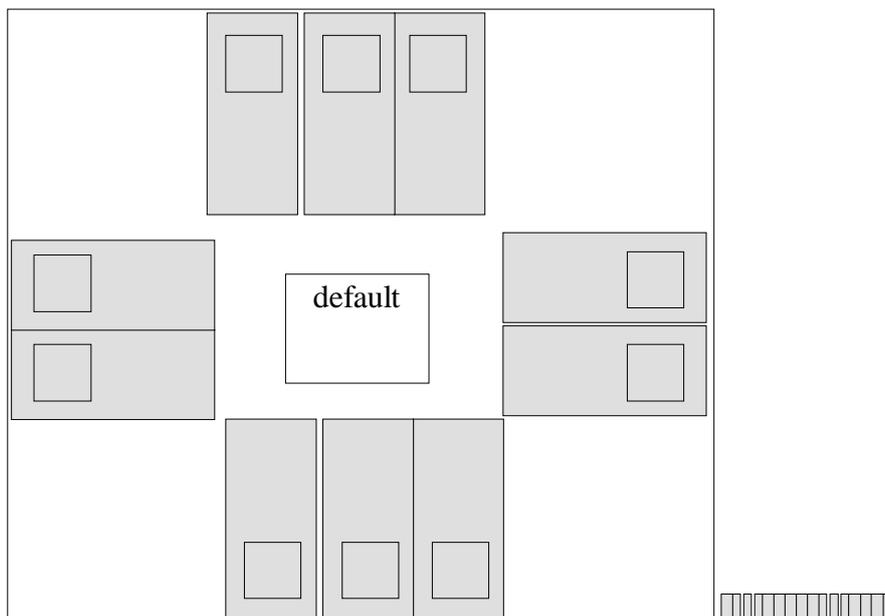
menu>Floorplan→i/o place

surge una ventana de dialogo en la que si el diseño es lo suficientemente complejo se pueden dejar todas las opciones por defecto.

En caso que el diseño sea excesivamente simple, las opciones por defecto provocan un descolocamiento de los pads de I/O cuando se hace la compactación final del layout, quedando el diseño con una periferia irregular. Para estos casos hay que modificar la siguiente opción:

min IO to core distance 100.00.

El resultado de la colocación de los pads de entrada salida es el siguiente:



11.2.3 UBICACIÓN DE MEGACELDAS:

Para ubicar las megaceldas se seleccionan una a una y se colocan en la posición deseada con la orientación deseada

Cuando las megaceldas estén ya ubicadas cambiar su estado a ubicadas:

menu>design→propiedades

ventana>status [placed]

11.2.4 CREAR LA REGION DE LAS CELDAS ESTÁNDARES

- v **Borrar la región de celdas estándares que existe :**

Menu>edit→delete

- v **Crear una nueva región (o regiones) de la siguiente manera:**

Menu>create→region

Form>number of rows

El resto de las opciones dejarlas por defecto. En particular si la opción

Fit instance after creation

se deja por defecto las celdas estándares se ubican en la región dando de manera inmediata una idea de si esta es válida o no.

- v **Para comprobar que la región es adecuada:**

Menu>floorplan→check→region

Con las opciones por defecto. Cualquier región errónea queda inmediatamente marcada.

- v **Para eliminar estas marcas:**

Menu>floorplan→check→clear errors

- v En el caso que sea necesario ajustar el tamaño de una región

Menu>region→stretch

11.2.5 UBICACIÓN DE LAS CELDAS ESTÁNDARES

menu>place→automatic

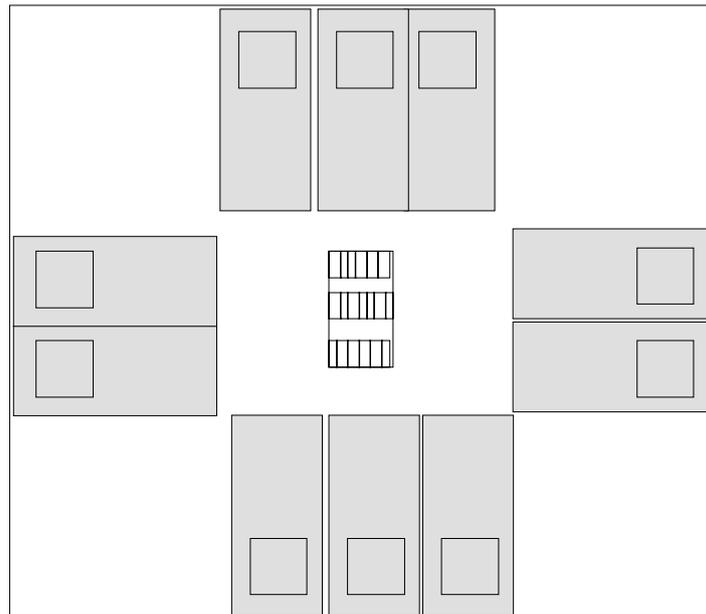
form>

method	[both]
timing driven placement	[off]
insert feedthru	[on]
feedthru library name	[stdlib]
feedthru master name	[LIBFEED]
feedthru master view	[abstarct]
placement snap grid	0.1
mirror cells	[on]

En la ventana CIW se da un mensaje completo de la operación de ubicación.

v **Notas:**

- En este punto conviene guardar el diseño con el tipo de vista corePlaced.
- De las dos opciones posibles de ubicación automática la primera vez se debe seleccionar siempre initial y las demás veces improve.
- Cuando al realizar la ubicación aparecen las megaceldas solapadas con las celdas estándares o desplazadas sobre los pads de entrada/salida la única manera de arreglarlo es moverlas manualmente realizar de nuevo la operación de ubicación.



11.2.6 AÑADIR LAS CELDAS DE ESQUINA

Si el diseño no incluye pads de alimentación de esquina se deben añadir cuatro celdas de esquina de la siguiente manera:

menu>place→io comands→add corners

modificar las opciones por defecto de la ventana:

form>

glue cell library name

glue cell master name

<botton_left_corner_cell>

<botton_right_corner_cell>

<top_right_corner_cell>

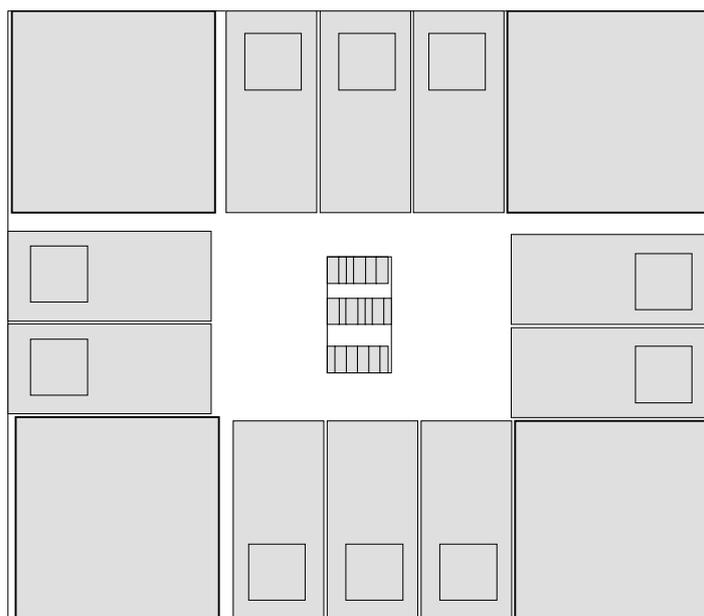
<top_left_corner_cell>

glue cell master view [abstact]

net association [matchTermName]

donde el nombre de la librería es PadLib y la celda es LIBCORNER.

Al finalizar este paso se debe poner a todas las celdas como ubicadas. Esto se realiza mediante el comando EDIT propiedades



11.2.7 JUSTIFICACION DE CELDAS DE ENTRADA SALIDA

menu> place→i/o comandas→justify

form> IO align style	[free]
align feature	supplypins
IO to core spacing	100 (defecto)
IO to IO spacing	1 para evitar que las celdas de IO solapen cuando los chips están limitados por los pads de entrada/salida
shift IO fame to origin	[on]
Placement snap grid	0.125 para ecpd10 y 0.1 para ecpd07

Este diseño conviene salvarlo como IOJust

11.2.8 REMATES A LAS FILAS DE CELDAS ESTÁNDARES

(Este paso se realiza solo si no se ha utilizado Power Grid Routing)

Se deben colocar celdas tapón a cada fila de celdas estándares para rematarlas.

- v Seleccionar la región de la celdas estándar

menu>place→power cell→add auto

las opciones de la ventana se pueden dejar por defecto salvo:

form>max bar separation 1

form>cut search range 1

Es decir no deben tener valores cero.

- v seleccionar la opción define power cells para que surja la ventana Power cell definition

form> library name StdLib

left cap cell name LIBLCAP

right cap cell name LIBRCAP

dejando el resto de las opciones por defecto

- v Si hay que borrar alguna de estas celdas los pasos a seguir son:

seleccionar la región de celdas estándares

menu>place→power cell →delete

- v en el caso que la adición de celdas tapón no funcione correctamente realizar una de las tres siguientes acciones:

1. comprobar que todas las celdas ubicadas tiene el estatus placed. Seleccionando la celda y haciendo:

menu>design→propieties

Form> Satus placed

2. Realizar de nuevo el paso de justificación de celdas de entrada/salida con un mayor espacio entre celdas

3. hacer mayores las regiones de ubicación de las celdas estándares.
(alargar la distancia IO to IO y funciona

Atención si aparece el error :

“1 es de tipo string” es porque hay que seleccionar la región de celdas estándar.

11.2.9 AJUSTE DE LAS CELDAS A LA REJILLA

Al final de las operaciones de ubicación es necesario asegurarse que todas las celdas están ubicadas sobre el grid.

Seleccionar todas las instancias

menu>Place→Snap to grid

modificar las opciones por defecto:

selected cells [on]

Placement Snap Grid 0.1 ecpd07 y 0.125 par ecpd10

11.2.10 COMPROBAR Y SALVAR LOS RESULTADOS

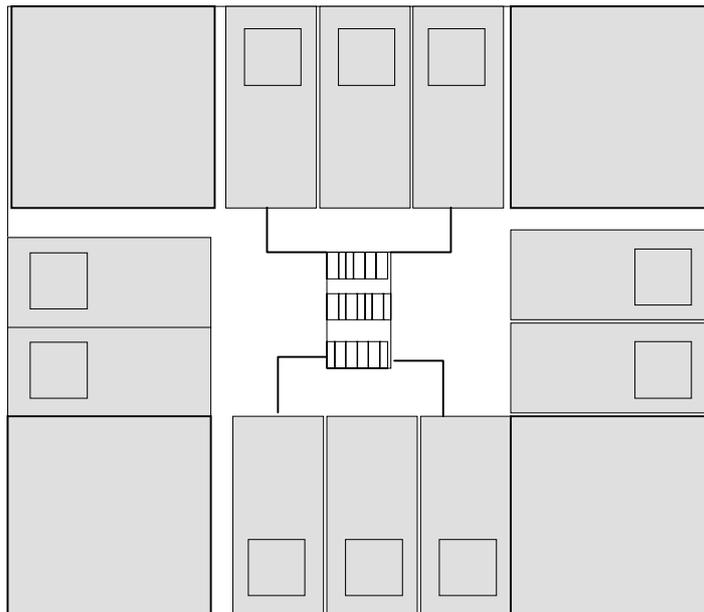
menu→place→chek

salvar con una vista PLACED.

11.3 RUTADO

11.3.1 CREAR CANALES DE RUTADO

- v Abrir la ventana PLACED
 - v Activar las herramientas P&R
 - menu>tools→floorplan/P&R→cell ensemble
 - v borrar las regiones de diseño:
 - seleccionar la región de diseño
 - menu> region→delete
 - menu>route→channels→create
 - v Salvar como vista channels
- si se produce algún error se pueden borrar los canales.



11.3.2 RUTADO GLOBAL

menu>route→global routeautomatic

form> Feedthru	Library Name	Stdlib
Feedthru	Master Name	LIBFEED
Feedthru	view Master	Abstarct

salvar como Routed

11.3.3 RUTADO DETALLADO

Menu>Route→Detail Routed →Automatic

Form>	Compaction Mode	[Automatic]
	Contact Style	[Offcentred]
	Routing Layer	2
	Options	Compact

se expande la ventana

outing snap grid 0.1 para ecpd07 y 0.125 para ecpd10

el resto de las opciones se pueden dejar por defecto.

12. FLUJO DE DISEÑO FULL-CUSTOM

12.1 INTRODUCCIÓN

En este apartado explicamos los pasos que hay que seguir para desarrollar diseños full-custom con el Kit de diseño de ES2. La tecnología con la que se trabaja es la de 1 micra del foundry ES2. Además se incluyen las reglas de diseño básicas (DRC) para poder desarrollarlo.

CADENCE permite dos formas diferentes de generación de full custom que pueden funcionar independientemente o complementariamente. Por un lado se encuentra la captura directa de los layouts, que se desarrolla mediante el dibujo de formas geométricas de diferentes layers utilizando para ello las herramientas gráficas del programa denominado VIRTUOSO.

Por otro lado, se puede realizar un diseño mediante la captura de esquemáticos a partir de los cuales se realiza la síntesis del layout.

12.2 ENTRADA AL SISTEMA

Los pasos a ejecutar son lo siguientes

1. Introducir el login y el password
2. Openwin
3. Opendesignkit->surge una ventana x interactiva
4. Seleccionar la opción ecpd10-> selecciona la tecnología de 1micra
5. Seleccionar la opción icfb-> activa todos los recursos de CADENCE
6. Abrir una librería->
7. Abrir un diseño-> para ello selecciona una vista **layout**
8. Aparecen dos ventanas:
 - una ventana de layout
 - una ventana LSW (layers and selection window)

En la ventana LSW aparecen los layers con los que se hacen las figuras geométricas del layout. En esta ventana se pueden encontrar cuatro opciones de trabajo con los layers:

- AV- hace visibles todos los layers
- NV hace invisibles todos los layers
- AS permite seleccionar layers
- NS impide la selección de layers

Existen dos opciones referidas a las celdas de instancias en el interior del layout

- inst.- hace las instancias seleccionables
- pin hace los pines seleccionables.

Vamos a ver que significan cada uno de los layers:

- **CNWI**.- Implante del pozo N
- **CPOL** (rojo)- polisilicio

- **CTOX** (verde).- thinox o zona activa.- indica donde va a ir un transistor
- **CNPI** (rosa punteado).- implante tipo N. si se hace sobre el sustrato P rodeando un cruce de polisilicio y thinox indica una zona en la que se fabrica un transistor N. Si se hace en el pozo N indica contacto ohmica de polarización de pozo.
- **CPPI** (amarillo punteado).- implante tipo p. en un pozo N rodeando un cruce de un polisilicio y un thinox indica un transistor tipo P. En el sustrato un contacto ohmico de polarización
- **CME1** (azul).- metal 1
- **CME2** (azul).- metal 2
- **(CCON)**.- contacto

12.3 ENTRADA DE DISEÑOS MEDIANTE LAYOUT

12.3.1 ENTRADA DE FORMAS GEOMÉTRICAS

La forma habitual de desarrollar un layout es la siguiente:

- Seleccionar en la ventana LWS el layer de la mascara que se desea dibujar
- Seleccionar en la barra vertical la forma geométrica que se desea dibujar
- Dibujar la forma siguiendo la reglas DRC

Igual que ocurría en el diseño semi-custom desde una ventana se pueden instanciar componentes desarrollados en otros diseños. La forma de realizarlo es la siguiente

- Seleccionar la opción instancia de la barra de iconos vertical de la ventana layout.
- Aparece una form llamada **create instance**.
- Pulsar browser y surge el library browser.

Desde el library browser se puede seleccionar cualquier diseño. ES2 proporciona un conjunto de layouts que facilitan la tarea del diseñador. Estos diseños contienen las mascararas de elementos básicos del diseño full-custom como son transistores y contactos. Para acceder a ellos desde el library browser se debe hacer lo siguiente:

seleccionar de la librería creada la categoría de celdas SymDevices

Esta categoría contiene un conjunto de celdas de tipo symbolic. El tipo symbolic es un layout. Este conjunto de celdas son diseños de dispositivos básicos que cumplen las reglas de diseño y que se van a utilizar como base al desarrollo de toda diseño full-custom. Las celdas mas importantes que contiene son las siguientes:

- **NTR.**- transistor de canal N. Se compone de una zona de thinox (zona activa) cruzada con polisilicio y rodeada de una layer de implante N
- **PTR.**- transistor de tipo P. Thinox y polisilicio cruzados, rodeados de un layer de implante P

- **M1_N**.- contacto entre el metal 1 y una difusión de tipo N. Se compone de metal1, thinox, implante N y contacto .
- **M1_p**.- contacto del metal 1 con difusión de tipo P. Se compone de implante P, metal 1, thinox, contacto.
- **M1_poli** Contacto de metal 1 con polisilicio. se compone de polisilicio, metal1 y contacto.

Recordar que en la tecnología CMOS un layout debe contener:

- Transistores NMOS (NTR)
- Transistores PMOS (PTR)
- Los transistores PMOS deben estar en el interior de un pozo N (CNWI)
- Contactos n (M1_N) en el pozo N son contactos ohmicos
- Contactos p (m1_P) en el sustrato p son contactos ohmicos
- Contactos n (m1_N) en sustrato p son las zonas de disfusion (fuente y drenador)
- Contactos p (M1_p) en pozo N son las zonas de difusión (fuente y drenador)
- El cruce de un polisilicio (rojo) y una zona activa (verde) en el interior de una zona activa P (amarillo punteado) es un transistor P .
- El cruce de un polisilicio (rojo) y una zona activa (verde) en el interior de una zona activa n (rosa punteado) es un transistor N

12.3.2 LOS PINES DEL LAYOUT

Igual que ocurría en el diseño semi-custom, un pin indica como y donde una celda layout se conecta a otra celda layout en una jerarquía de diseño. En definitiva muestra donde una instancia de una celda puede conectar a un routing a otras instancias

Para dibujar un pin manualmente se debe hacer lo siguiente:

1. Seleccionar la opción PIN del menu create
2. Seleccionar el layer del pin
3. Aparece una form automaticamente. Escribir el nombre del terminal
4. Seleccionar la forma del pin
5. Dibujar el pin en el layout

12.3.3 REGLAS DRC.

Para pasar las reglas DRC primero hay que seleccionar el archivo de tecnología adecuado. Este paso solo se realiza una vez. Para ello hacer lo siguiente

- Ir al la ventana ICFB principal
- Menu->tools->layout
- Technology file->compile technology
- En la form introducie el nombre diva.dr

Cada vez que se quieran pasar las reglas DRC :

1. Ir a la barra de menú principal y seleccionar verify
2. Selecciona la opción drc
3. Aparece una form.-
4. En la opción switch names la opción slow
5. Ok

En la ventana icfb aparece los mensajes indicando las reglas que se están comprobando. Si existen reglas que se incumplen aparecen marcadores parpadeantes que indican los lugares de la violación.

De los marcadores se puede obtener información de la siguiente forma:

- Selecciona la opción verify
- Seleccionar la opción markers
- Aparecen las opciones explain, find, delete, delete all.

v **mensajes de error:**

- floating CPOL #ERC.- Existe algún polisilicio sin contacto(pe M1_pol)
- bulk tap with.- no se ha colocado el pozo N. los dispositivos p se fabrican sobre sustrato P, es decir mediante contactos ohmicos.
- Ngate with cppi.- no se ha colocado el pozo N

Conviene saber que los metales si pueden aparecer sin contactos en el layout.

12.3.4 EXTRACCIÓN

Mediante la extracción se obtiene la descripción electrónica del layout incluyendo los elementos parásitos como las capacidades que se utilizarán en la simulación post layout.

Al hacer la extracción aparece una nueva vista llamada *extracted* en la que aparecen sobre el layout los símbolos de los transistores creados.

La extracción se lleva a cabo desde la opción verify del menú principal.

12.3.5 SIMULACIÓN FUNCIONAL HSPICE

El diseño debe estar extractado y salvado. El diseño extractado contiene información sobre la conectividad. El generador de netlist utiliza esta información junto con las primitivas de las componentes para producir una lista de nodos como entrada de la simulación.

Inicialmente el menú de simulación no aparece en la barra de menu horizontal. Para mostrar el menú de simulación:

tools-> simulate-Other.

en este momento cambia de barra de menú apareciendo la opción simulation

Para inicializar el entorno:

- tools->simulation->initialize
- aparece una form preguntando por el directorio, que por defecto puedo llamar hspice.run1
- ok
- aparece una nueva form
- seleccionar other->hspice
- ok

La entrada de estímulos se puede hacer o bien directamente a partir de archivos hspice o mediante archivos STL. Si se hace la entrada en hspice directamente habrá que modificar el archivo llamado control. Mediante archivos STL se hace de la siguiente manera:

- simulation->stimulus-> edit STL
- Surge una form preguntando si se desea compilar. La compilación sirve para comprobar si el archivo input.stl tiene errores. Si no se selecciona se compila directamente en la simulación.
- aparece un archivo input.stl configurado para es2 0.7mm hspice

Para simular se selecciona la opción netlist/simulation

Para ver los resultados se hace simulation->show waveforms.

Por defecto el nombre de archivo de simulación es waves.

12.4 ENTRADA DE DISEÑOS MEDIANTE ESQUEMÁTICOS

Es en todo idéntico a la captura de esquemáticos para semi-custom. El diseño debe generarse en una vista de tipo schematic. Los transistores se deben seleccionar de la librería

sample->trans → de tipo symbol.

Nota: Para que el esquemático se pueda simular con spice los transistores utilizados deben ser de tipo nfet y pfet,.

A estos transistores se les deben añadir los parametros de longitud y anchura. Para ello se utiliza la opción *property* de la barra de iconos horizontal.

form->user property->add.

v **para añadir la longitud del canal:**

nombre l (debe ser minúscula)

type string

value valor deseado seguido de la unidad "u"

v **para la anchura lo mismo pero con w**

La tierra (gnd) y la alimentación (vdd) se cogen de la librería basic.

NOTA: Cuidado al elegir las tierras no es lo mismo elegir las en mayúscula que en minúscula. Recomiendo se seleccionen en minúscula para que concuerden con las plantillas que genera el STL

Para que se generen los valores del transitorio que necesita el hspice de manera automática se debe hacer lo siguiente:

ir a la librería sample->spice y seleccionar la celda tran cuyos valores se deben modificar con los valores de simulación que se desee. mediante el comando edit propertie→ add

name Tran

Type NLPEXPR

value 1n 100n

12.5 SINTESIS DE LAYOUT A PARTIR DE ESQUEMATICOS

Partiendo del esquemático de transistores conseguido en el apartado anterior se hace lo siguiente:

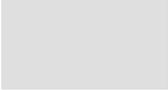
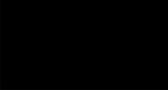
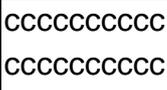
tools→design synthesis→layout synthesis

aparece en la barra de herramientas la opción LAS (Layouty Syntehsis)

hacer la siguiente selección:

LAS→generate

12.6 REGLAS DE DISEÑO DE ES2 PARA LA TECNOLOGÍA DE 0.7 MICRAS ECPD7

	zona activa (75%)		polisilicio (50%)
	pozo N (12.5%)		implante P
	metal1 (40%)		implante N
	via		contacto

