TENDENCIAS EN SÍNTESIS DE SISTEMAS ELECTRÓNICOS

Diego Gómez Vela
Pedro Galindo Riaño
Carmen García López

SERVICIO DE PUBLICACIONES • UNIVERSIDAD DE CÁDIZ

Tendencias en síntesis de sistemas electrónicos

Diego Gómez Vela
Pedro Galindo Riaño
Carmen García López

Grupo de Diseño Microelectrónico
Escuela Superior de Ingeniería
Universidad de Cádiz

UNIVERSIDAD DE CÁDIZ
SERVICIO DE PUBLICACIONES
1997

Diego Gómez Vela
Pedro Galindo Riaño
Carmen García López

Edita: SERVICIO DE PUBLICACIONES DE LA UNIVERSIDAD DE CÁDIZ
ISBN: 84-7786-428-4
Imprime: *Servicio de Autoedición e Impresión*
Universidad de Cádiz

A nuestras hijas: Lucía, Irene y Eva

y..... Laurita

PRÓLOGO

En un campo de investigación como el diseño de circuitos microelectrónicos, una década de actividad es suficiente margen de tiempo para comprobar que la velocidad de obsolescencia de los sistemas electrónicos y sus propios métodos de desarrollo, no son sólo un tópico.

La experiencia propia y la observación de lo acontecido en otros campos afines, pone en evidencia la necesidad de entrar en una dinámica, que se caracteriza fundamentalmente por: un aumento de complejidad en los sistemas electrónicos, la tendencia a razonar los desarrollos desde un grado de abstracción mayor, la necesidad de integrar un esfuerzo multidisciplinar para resolver los problemas y la necesaria experimentación que en la práctica validan las ideas.

El cuadro descrito, aunque no exclusivo del diseño microelectrónico, nos ha llevado recientemente a plantearnos la asimilación de las técnicas de síntesis de sistemas electrónicos, bajo una concepción multidisciplinar, donde el eje del codiseño de los módulos de circuitos y sistemas electrónicos y los programas que gobiernan a muchos de sus elementos, se alinean en una metodología, que une a los ingenieros de diseño microelectrónico con los ingenieros de computación para recorrer la misma ruta compartiendo asiento en un vehículo común.

En el sentido expresado previamente, el Grupo de Microelectrónica, cuyos componentes proceden de diferentes departamentos de la Universidad de Cádiz, (Ingeniería de Sistemas y Automática, Tecnología Electrónica y Electrónica - Ingeniería Eléctrica - Lenguajes y Sistemas Informáticos) ha realizado un esfuerzo de integración desde diferentes áreas de conocimiento logrando la cohesión de un grupo humano para abordar esta actividad investigadora en técnicas de codiseño y síntesis de alto nivel sobre sistemas electrónicos.

Esta monografía, es una introducción teórica al tema de la síntesis de sistemas electrónicos de naturaleza digital acompañada de una bibliografía seleccionada y actualizada. Probablemente, sea esta la primera contribución en castellano sobre esta temática, y es el fruto de una labor de estudio y experimentación, llevada a cabo en un seminario interno realizado durante el curso 95/96, que culminó con una primera presentación en la 47ª edición de los Cursos de Verano de la Universidad de Cádiz. Entre sus propósitos incluye motivar el estudio e investigación en esta línea de trabajo y, como miembros de la comunidad universitaria, deseamos contribuir a la difusión de estos nuevos conocimientos.

El temario desarrollado recorre la idea del codiseño y plantea la captura de especificaciones con herramientas informáticas adaptadas a ese fin. Aborda el concepto de síntesis de sistemas, lo ilustra con ejemplos recogidos de la literatura especializada, para en último término arribar a los conceptos de síntesis de alto nivel, que son los más próximos a las capacidades de las actuales herramientas de diseño asistido por ordenador.

Cádiz, abril de 1.997.
Los autores.

AGRADECIMIENTOS

Esta obra ha sido posible gracias a la colaboración y apoyo de varias personas e instituciones que nos complace indicar.

En primer lugar nuestros compañeros de equipo, los profesores Ángel Quirós Olozábal, Juan Manuel Barrientos Villar, Arturo Morgado Estévez y Francisco José Lúcas Fernandez.

Igualmente reconocemos las aportaciones, especialmente en la confección de algunas gráficas realizadas por nuestros alumnos Musti Cepero Boullra, Juan Fráncisco Jiménez Pascual y Juan J. Pérez Alfonso.

De otra parte queremos expresar nuestro agradecimiento a los departamentos siguientes de la Universidad de Cádiz :

- Ingeniería de Sistemas y Automática, Tecnología Electrónica y Electrónica.
- Lenguajes y Sistemas Informáticos.
- Ingeniería Eléctrica.

así como a la Escuela Superior de Ingeniería, y al Servicio de Publicaciones de la Universidad de Cádiz, cuya inestimable ayuda ha permitido que el presente libro vea la luz.

Cádiz, abril de 1997.
Los autores.

ÍNDICE DE CONTENIDOS

CAPÍTULO I

EL CODISEÑO Y LA CAPTURA DE ESPECIFICACIONES

1. INTRODUCCIÓN AL CODISEÑO

1.1. Sistemas empotrados

Un sistema empotrado es un sistema electrónico, normalmente muy específico, compuesto de un conjunto de componentes *software* que se ejecutan en elementos *hardware* (normalmente microcontroladores y/o microprocesadores) y que permiten controlar una aplicación, de la cual forman parte [34]. Los sistemas empotrados se clasifican con frecuencia dentro de los sistemas reactivos, es decir, reaccionan al entorno ejecutando funciones en respuesta a estímulos de entrada específicos, utilizando para ello sensores y actuadores. Sistemas tan simples como un horno microondas, un contestador automático, una impresora láser o un teléfono móvil o tan

complejos como un piloto automático, el controlador de un robot, los sistemas de frenos ABS o los sistemas de inyección de combustible se realizan mediante sistemas empotrados. Este emparejamiento entre *software* y *hardware*, implica que una vez definidas las especificaciones del sistema a diseñar, debemos encomendar cierta funcionalidad al *software* y otra al *hardware*. Este reparto no se debe hacer de forma arbitraria, sino atendiendo a factores tales como la sencillez del diseño, rapidez de la fase de desarrollo, rendimiento final del sistema, restricciones de espacio físico, etc. También pueden existir otros criterios de decisión, que en la mayoría de los casos son evaluados por el propio diseñador de acuerdo a su experiencia o a factores restrictivos, bien sean de tipo técnico, económico, o incluso, a decisiones de tipo comercial.

El objetivo del *codiseño* es, por tanto, el desarrollo de *sistemas empotrados*, que no sólo cumplan con las especificaciones iniciales de funcionamiento, sino que además, el diseño tanto del *software* como del *hardware*, como la decisión de qué se hace en *software* o en *hardware* se tome utilizando unos criterios objetivos de optimización.

El principal factor a considerar en el diseño de todo producto suele ser su coste final. Sin embargo, en muchas ocasiones, es igualmente importante el coste de diseño, entendiendo por tal, no sólo coste en pesetas, sino también en tiempo de diseño, factor clave en los tiempos que corren. Puede ser más importante tener un producto en el mercado antes que la competencia a un precio algo superior, que prolongar la fase de desarrollo para optimizar costes. Por último, se deben considerar factores de tipo técnico, tales como la fiabilidad del sistema obtenido en términos de control de la calidad, así como la eficiencia obtenida para el objetivo para el que está diseñado.

El hecho de utilizar una metodología de diseño que no optimice los factores mencionados, nos puede llevar al desarrollo de sistemas con un excesivo coste de diseño y/o implementación y, consecuentemente, no competitivos en la sociedad de libre mercado.

1.2. Metodologías de diseño

El proceso de diseño de un sistema electrónico tal como se muestra en la figura I.1, comienza con una idea que suele partir del departamento de marketing, bien por iniciativa propia o bien como resultado de un estudio de mercado. Una vez que se tiene una idea de lo que se quiere diseñar, se formalizan las especificaciones, y se elabora un informe con los requerimientos y las especificaciones formalizadas que debe cumplir el sistema. Dicho informe se confecciona y se entrega al siguiente grupo de trabajo, formado por el/los Ingeniero/s Jefe, los técnicos y los expertos en CAD (*Computer Aided Design*) y CASE (*Computer Aided Software Engineering*). Este segundo grupo ya posee conocimientos técnicos suficientes para abordar el problema desde el punto de vista del sistema, es decir, realizan un estudio de las posibles configuraciones, y de las formas de abordar el problema propuesto. El tercer grupo implicado está compuesto por el equipo de diseño *hardware* y por el equipo de diseño *software*. El trabajo de éstos es más concreto, y su función consiste en implementar los diferentes bloques *hardware* y *software* que componen nuestro sistema.

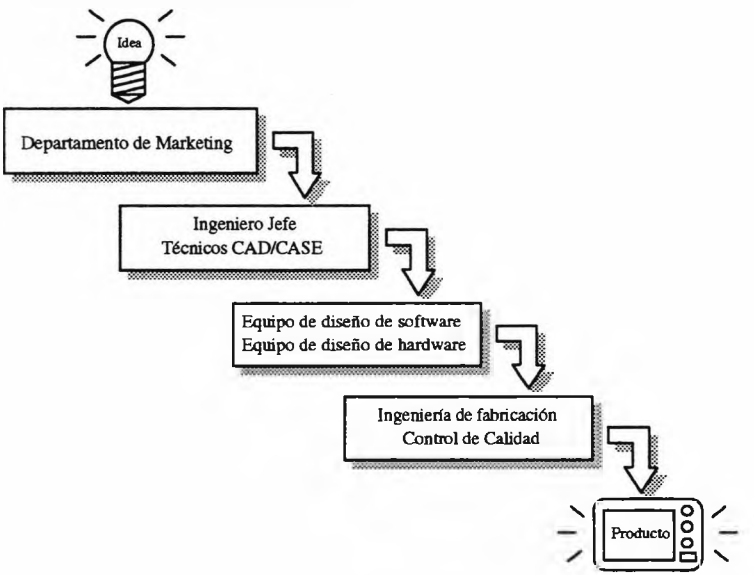


Figura I.1. Proceso típico de diseño de un sistema electrónico.

Por último, una vez el proceso de diseño ha culminado, se debe realizar la implementación física. En el caso de tratarse de fabricación en grandes series, como suele ser lo normal, nos encontraremos con un cuarto y último grupo formado por los ingenieros de fabricación y de calidad, cuya principal meta es que el sistema final se ajuste a las especificaciones de diseño.

En este ciclo de diseño, no solo intervienen personas de diferentes departamentos, sino que a medida que el proceso avanza, hay más personas ocupándose del desarrollo del producto. Esto puede desembocar en la aparición de conflictos de intereses, en la existencia de puntos de vista contrapuestos entre los distintos grupos de trabajo, en dificultades de comunicación entre los distintos equipos, así como otros problemas relacionados. Es vital para la buena marcha del proyecto que estos enfrentamientos de ideas sean evitados o, al menos, minimizados. Para ello, es imprescindible la utilización de herramientas de formalización, así como el seguimiento de una metodología preestablecida.

Dicha metodología de diseño de sistemas se debe desarrollar [34] de forma que permita cumplir al menos los tres objetivos siguientes:

- Predecir costes de implementación, en términos de presupuesto y de tiempo.
- Refinar un diseño de forma incremental, en sus distintos niveles de abstracción, introduciendo mejoras progresivas en el producto. Esto significa que podamos crear algo que nos dé la posibilidad de mejorarlo en un futuro.
- Crear una implementación que funcione, y que lo haga bajo las condiciones que hemos impuesto al diseño.

Normalmente estos tres objetivos no se cumplen para una metodología dada, por lo que suele ser necesario un compromiso entre los diferentes criterios a seguir para obtener un resultado satisfactorio.

1.3. Síntesis de Sistemas. Codiseño

Desde aquellos años en los que la escasa complejidad del sistema permitía el *diseño a mano* hasta nuestros días, en los que se hace totalmente indispensable la

ayuda de herramientas de diseño y simulación por ordenador, se ha producido una evolución que ha seguido las tres tendencias siguientes :

- Capturar y Simular
- Describir y Sintetizar
- Especificar y Explorar

Cada uno de estos métodos de diseño se corresponde con un época determinada, aunque éstas no están bien definidas debido, sobre todo, a que normalmente han coexistido (y coexisten en la actualidad) las tres. La captura y la simulación se emplea cuando la sencillez del sistema electrónico a diseñar lo permite. En cuanto al diseño a mano, ha quedado prácticamente relegado al olvido, pues son escasos los sistemas que pueden realizarse de esta forma en la actualidad.

Si echamos una mirada al pasado, vemos que se ha producido un gran cambio en la forma en la que se diseñan los sistemas electrónicos. En la década de los 80 se capturaba el esquemático con un programa de CAD y se realizaba la simulación posteriormente. Actualmente y cada vez más, se emplea en la mayoría de los diseños un lenguaje de alto nivel para describir el comportamiento del sistema, con lo que se ha entrado de lleno en una nueva fase. Pero ya está apareciendo una tendencia, aún no consolidada, de captura de las especificaciones del comportamiento del sistema de forma abstracta, es decir, independiente de su implementación final. El proceso típico [16] consiste en convertir nuestra especificación en un diagrama de bloques. Dicho diagrama se traduce automáticamente a una descripción, por ejemplo en VHDL. A continuación, el sistema se subdivide en una serie de módulos, tales como ASIC's, FPGA's, memorias, procesadores, etc., de tal forma que la funcionalidad del sistema se implementa, bien mediante componentes físicos (*hardware*), bien mediante programas (*software*) que se ejecutan en un procesador. En función de todas las posibles alternativas, y siguiendo una serie de criterios objetivos, se sintetizan diferentes soluciones. Con la ayuda de herramientas *software* adecuadas, se observa y simula el resultado obtenido con cada una de ellas. Por último, se selecciona ya sea de forma manual, automática, o semiautomática la que mejor se adapta a nuestras necesidades.

El objetivo actual del codiseño es describir un sistema sin tener en cuenta su implementación final, esto es, el diseñador debe suministrar las especificaciones, y no

tiene por qué saber, en principio, cómo se implementará finalmente. Esta metodología de diseño lleva asociadas varias ventajas [34] :

- Se pueden crear juegos de prueba y con ellos detectar errores funcionales desde las primeras fases del diseño. En el enfoque clásico, dichos errores se detectan en una fase más tardía y por tanto, el coste de su corrección es mucho mayor.
- Se pueden aplicar herramientas de simulación y síntesis en las diferentes etapas del diseño para evaluar diferentes implementaciones. De esta forma podemos escoger de entre todas las posibles, la que mejor se adapte a nuestros criterios de optimización.
- Las tareas de rediseño se realizan con menor dificultad, con el consiguiente ahorro de coste y tiempo.

1.4. Fases en el codiseño de sistemas

Encontramos varias fases o pasos a seguir [14] durante el codiseño de sistemas:

1. **Captura de especificaciones:** consiste en plasmar formalmente mediante un lenguaje de captura de especificaciones la visión del sistema que se quiere diseñar. Así se consigue una estructuración de las ideas para obtener una especificación formal de la funcionalidad del sistema, que no es más que una descripción del comportamiento que se desea obtener del mismo. Para especificar la funcionalidad deseada de un cierto sistema, se descompone en módulos, creando un modelo conceptual del sistema. A continuación se genera una descripción de este modelo en un cierto lenguaje formal. Dicha descripción se comprueba que es correcta mediante técnicas de validación y verificación, obteniéndose la especificación funcional resultante de esta primera fase.
2. **Exploración:** en esta fase se exploran numerosas alternativas de diseño hasta encontrar la que mejor satisface nuestras necesidades. Para ello, se transforma la especificación inicial en una más adecuada para su implementación. Se seleccionan una serie de componentes del sistema, y se especifican sus

restricciones físicas y de funcionamiento. Se subdivide la especificación entre los diferentes componentes, estimando la calidad de cada una de las posibles alternativas de diseño. Como vemos, las fases que podemos distinguir son transformación, reserva, particionado y estimación. Posteriormente estudiaremos en detalle cada una de estas fases.

3. **Refinado de las especificaciones:** como resultado del análisis previo, podemos refinar la especificación funcional, obteniendo otra especificación mejorada que refleje las decisiones tomadas en la exploración, y que será depurada de nuevo hasta obtener una descripción del sistema satisfactoria. Con esto buscamos la alternativa óptima, la opción que satisfaga todos los requisitos de la forma más eficiente. Como resultado de este refinamiento, se obtiene una descripción a nivel de sistema que posee algunos detalles acerca de la implementación final, aunque sigue siendo eminentemente funcional.
4. **Diseño del Sw y Hw:** es entonces cuando comenzamos el diseño propiamente dicho, tanto para el *hardware* como para el *software*. Realizaremos varios procesos de síntesis, síntesis de *software* [14] y síntesis de *hardware*: síntesis de alto nivel [13] y síntesis lógica. El resultado de la síntesis de alto nivel es una descripción a nivel de registro (RTL - *Register Transfer Level*), que contiene código fuente para el *software* y *netlists* de este nivel RT para los componentes *hardware*. La síntesis lógica es el paso previo al diseño físico.
5. **Diseño Físico:** finalmente se obtienen los datos de fabricación para cada componente. Para el *software*, esta fase se reduce al proceso de compilación y enlazado, y para el *hardware*, consiste en los datos de la descripción geométrica (*layout)* para los circuítos integrados, ficheros de configuración para los dispositivos lógicos programables, ya sean FPGA's (*Field Programmable Gate Array*) o CPLD's (*Complex Programmable Logic Device*), y el diseño de las placas de circuíto impreso - PCB's (*Printed Circuit Board*) -.

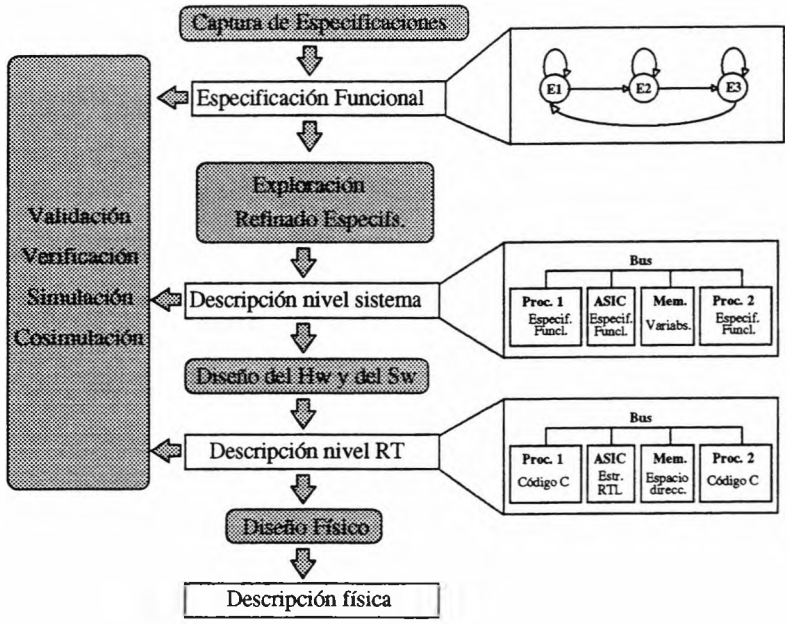


Figura I.2. Fases en el codiseño de sistemas (adaptado de [14]).

Todo el proceso sufre, al final de cada fase, un conjunto de verificaciones, pruebas y simulaciones para comprobar que todo progresa según lo deseado. En caso contrario, se deberá dar marcha atrás en el proceso y corregir los errores cometidos. En la figura I.2 queda reflejada la secuencia de pasos, de forma gráfica.

2. CAPTURA DE ESPECIFICACIONES

En este apartado comentaremos la primera fase en el codiseño de sistemas. Analizaremos el concepto de modelización y estudiaremos los modelos más empleados en la captura de especificaciones para describir el comportamiento de sistemas empotrados, así como las características de los lenguajes empleados para realizar dicha tarea y las subfases de que consta la captura de especificaciones.

Podemos decir que *la captura de especificaciones* es el proceso por el que definimos formalmente el comportamiento de un sistema, obteniendo de esta forma un modelo del mismo. Esto, a veces, no es tan sencillo como pueda pensarse en un principio, ya que no existe un modelo universal que se adecúe a todos los sistemas. Normalmente, el modelo que se adapta a un problema, no es útil para resolver otro problema diferente. Además, es difícil encontrar un modelo que responda al 100% a nuestras necesidades. Por ello, surgen modelos mixtos que nos permiten aproximarnos al objetivo ideal.

2.1. Modelización

El uso de modelos en la síntesis de sistemas permite especificar sin ambigüedades el comportamiento de los sistemas. Para la creación del modelo asociado al sistema debemos seleccionar un lenguaje para la captura de especificaciones. Esto es necesario para poder formalizar dicha captura. Podría pensarse en el lenguaje humano como un lenguaje de especificación, pero enseguida se pone de manifiesto un gran inconveniente: es ambiguo y falto de detalle. Este es uno de los problemas que han de salvar los lenguajes de especificación. Un lenguaje de especificación debe definir claramente y sin ambigüedad alguna el modo de actuación del sistema a diseñar. Si esto no es así, la especificación funcional no será correcta, y este fallo se hará patente a lo largo de todas las fases del diseño, con los consecuentes costes que se derivan de ello.

Debemos decir que un modelo, para ser útil, debe cumplir al menos las características que a continuación detallamos:

- Debe ser **FORMAL**, lo que permite evitar la ambigüedad de los lenguajes no formales.
- Debe ser **COMPLETO**. Es decir, el modelo tiene que ser capaz de describir al sistema en su totalidad, para todas las situaciones que puedan presentarse y en cualquier instante.

- Debe ser **COMPRENSIBLE**. Se trata de que sea fácil de entender por los usuarios (diseñadores). Si esto no es así, la dificultad de comprensión echaría por tierra el resto de las ventajas de nuestra metodología.
- Debe ser **FÁCIL DE MODIFICAR**. Con ello nos ahorraremos importantes esfuerzos cuando se desee realizar modificaciones al modelo de partida.
- Debe **AYUDAR A COMPRENDER EL SISTEMA**. Esta característica es muy importante ya que, en muchas ocasiones, el proceso de modelización facilita la comprensión de un sistema físico determinado.

2.2. Tipos de modelos

Podemos diferenciar [15] entre cuatro tipos diferentes de modelos de acuerdo a la forma en la que están definidos:

- *Modelos orientados a los estados*
- *Modelos orientados a la actividad*
- *Modelos orientados a la estructura*
- *Modelos heterogéneos*

A continuación daremos una breve descripción de cada uno de ellos, así como de los diferentes métodos de especificación que se acogen a ellos y sus características más importantes.

2.2.1. Modelos orientados a los estados

Este tipo de modelo, para describir un sistema, hace uso de las técnicas de representación de estados y transiciones entre ellos. Este tipo de descripción es muy familiar a los diseñadores de sistemas electrónicos pues, como a continuación se verá, se emplean los diagramas de estados que se han venido usando para el diseño de sistemas secuenciales comunes.

Podemos destacar tres tipos de modelos pertenecientes a esta familia: máquinas de estados finitos, máquinas de estados finitos con caminos de datos y máquinas de estados finitos concurrentes jerárquicos.

- **Máquinas de Estados Finitos**

Se trata de representar cada uno de los estados que puede tomar el sistema. Éstos se representan por circunferencias, dentro de las cuales aparece el nombre del estado. Las transiciones entre estados se representan por arcos terminados en punta de flecha que indican el sentido de la transición cuando se da una condición determinada. Esta condición, que viene dada por un cambio en las variables de entrada del sistema, se representa en alguna zona cerca de la línea de transición correspondiente. La figura I.3 muestra el caso de un sistema que controla el funcionamiento del mando a distancia de un televisor con 3 canales. Se ha hecho corresponder un estado por cada uno de los canales en que puede permanecer la televisión. Las flechas indican el canal al que debe cambiar la TV en el caso de que se dé la condición indicada, ya sea mediante la pulsación de un número de canal, o de las teclas P+ o P- correspondientes a "subir canal" o "bajar canal". Cuando un cambio en la variable de entrada no supone un cambio en el estado del sistema, se representa mediante una flecha que sale y termina en el mismo estado.

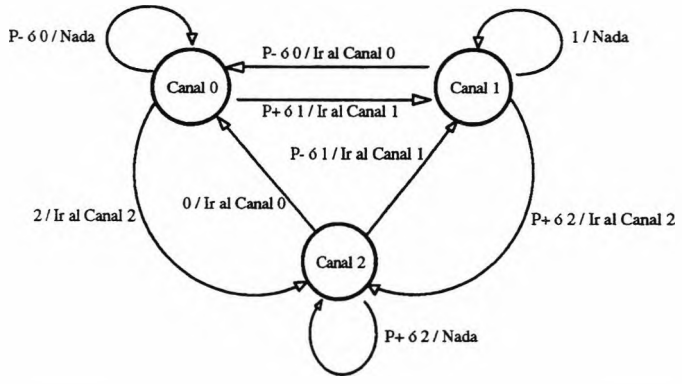


Figura I.3. Máquina de estados finitos del funcionamiento simplificado del mando a distancia de un televisor.

- **Máquinas de Estados Finitos con Caminos de Datos**

Este método presenta una ventaja que es evidente al comparar las figuras I.3 y I.4. Como puede verse, se ha reducido de forma considerable el número de estados que presenta el sistema. Esto tiene sus lógicas ventajas. Sin embargo, se complica la forma de presentación de las condiciones de transición. Antes estas condiciones estaban dadas por el cambio en una de las variables de entrada del sistema, mientras que ahora la condición a verificar es algo más compleja.

Si el canal que solicita el usuario es diferente del canal actual, entonces el TV cambiará de canal. Si el canal solicitado resulta ser el mismo, entonces no se producirá cambio alguno (salida nula) y el sistema permanecerá en el mismo estado en el que estaba.

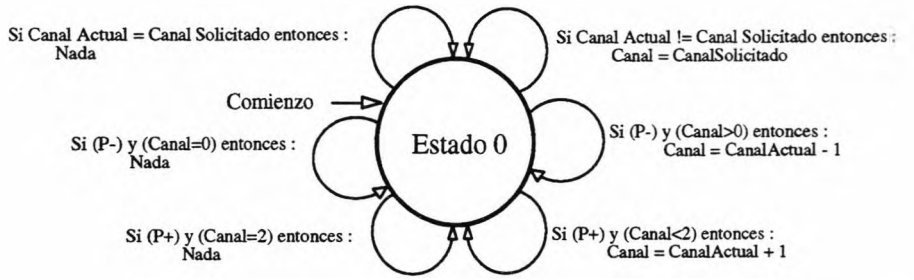


Figura I.4. Máquina de estados finitos con camino de datos del funcionamiento simplificado del mando a distancia de un televisor.

- **Máquinas de Estados Finitos Concurrentes Jerárquicos**

Este método de descripción es el más complejo de los modelos orientados a los estados. Nos ofrece dos potentes herramientas que son la *concurrencia y la jerarquía.* La concurrencia permite definir dos o más procesos que son necesarios para realizar una función determinada del sistema, pero que se pueden separar uno del otro, de forma que cada uno de estos procesos se ejecuta de forma autónoma, y cuando el último de ellos termina, los resultados de todos se emplean para obtener la función deseada en el sistema. Esto permite obtener mejoras en la velocidad.

En cuanto a la jerarquía, se trata de definir los diferentes procesos en niveles, de forma que un proceso determinado pueda ser considerado como un subproceso para

otro. En la figura I.5 se muestra un ejemplo, donde 'A' es el proceso padre, el cual consta de los subprocesos 'B' y 'C', que se ejecutan de forma concurrente; cuando éstos acaban finaliza el proceso padre 'A', que hace uso de los resultados generados por 'B' y 'C'. En los subprocesos 'B' y 'C', se indica el comienzo con un pequeño círculo relleno, y el final con un cuadrado.

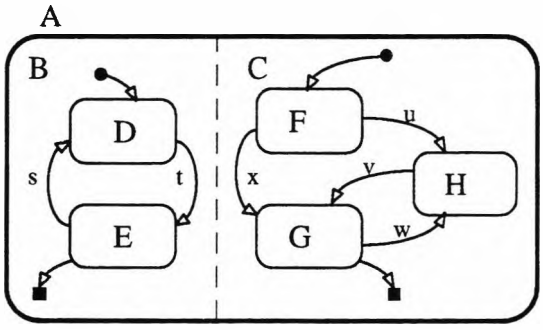


Figura I.5. Ejemplo de máquina de estados finitos concurrentes jerárquicos.

2.2.2. Modelos orientados a la actividad

Vemos a continuación dos tipos de modelos orientados a la actividad: grafos de flujo de datos y grafos de flujo de control.

- **Grafos de Flujo de Datos**

Este modelo descompone la funcionalidad en actividades que transforman datos, y el flujo de datos de estas actividades. Normalmente se representa por las estructuras de tipo árbol. En los nodos del árbol se representan por un lado los datos, y por otro las operaciones a realizar con éstos. El siguiente árbol es una representación de la función: j=j+1

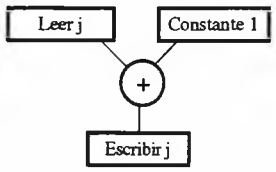


Figura I.6. Ejemplo de grafo de flujo de datos para la operación j=j+1.

- **Grafos de Flujo de Control**

Este tipo de modelo se representa mediante los diagramas de flujo u ordinogramas. Se trata de indicar el flujo del proceso mediante caminos que llevan a determinadas acciones, dependiendo de los sucesos que vayan ocurriendo en cada instante. En la figura I.7 se muestra un ejemplo de un grafo de flujo de control para un sistema simple de detección del valor máximo de un vector de datos.

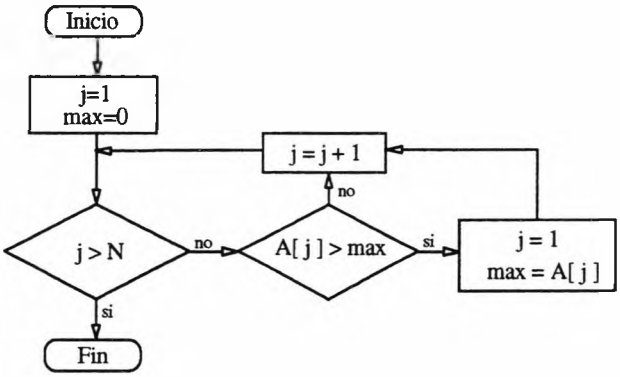


Figura I.7. Grafo de flujo de control para la detección del máximo de un vector de datos (adaptado de [15]).

Las acciones o tareas básicas a realizar por el sistema se representan por rectángulos, dentro de los cuales se indica la acción a realizar. La toma de decisión booleana (si/no) se representa mediante un rombo, de forma que se tomará un camino u otro dependiendo de si se cumple la condición indicada o no.

2.2.3. Modelos orientados a la estructura

Los modelos orientados a la estructura se acercan un poco más a la constitución física del sistema, puesto que dan información relativa a los bloques funcionales que compondrán el sistema, así como la forma en la que están relacionados/conectados.

- **Diagramas Conectividad - Componente**

La figura I.8 representa un diagrama conectividad-componente, en el que se pueden observar los bloques principales de un sistema (Procesador, memoria, ...) así como la interconexión entre éstos. Como puede verse, el grado de detalle es escaso, pero se tiene una idea global de cómo funciona un sistema de forma general.

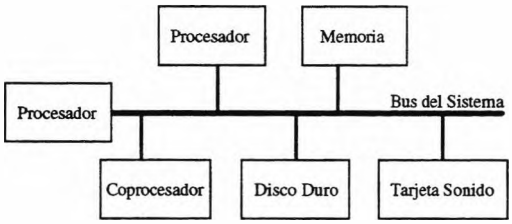


Figura I.8. Ejemplo de diagrama conectividad-componente.

2.2.4. Modelos heterogéneos

Para finalizar, los modelos heterogéneos, que no son más que híbridos de los anteriores, aprovechan las características que interesan de cada uno de los anteriormente expuestos. Veremos aquí cuatro tipos diferentes de modelos heterogéneos: grafos de flujo control/datos, lenguajes de programación, procesos secuenciales comunicados y máquinas de programas-estados.

- **Grafos de Flujo Control/Datos**

Como puede verse en el ejemplo y, como el propio nombre indica, se trata de emplear simultáneamente los grafos de flujo de control y los de flujo de datos. Lo que se hace es describir el modelo asociado al sistema mediante el método de grafo de flujo de control y, posteriormente, se *refinan* cada una de las acciones que forma el flujo de control, haciendo uso de grafos de flujo de datos. La figura I.9 ilustra la forma de empleo de este método.

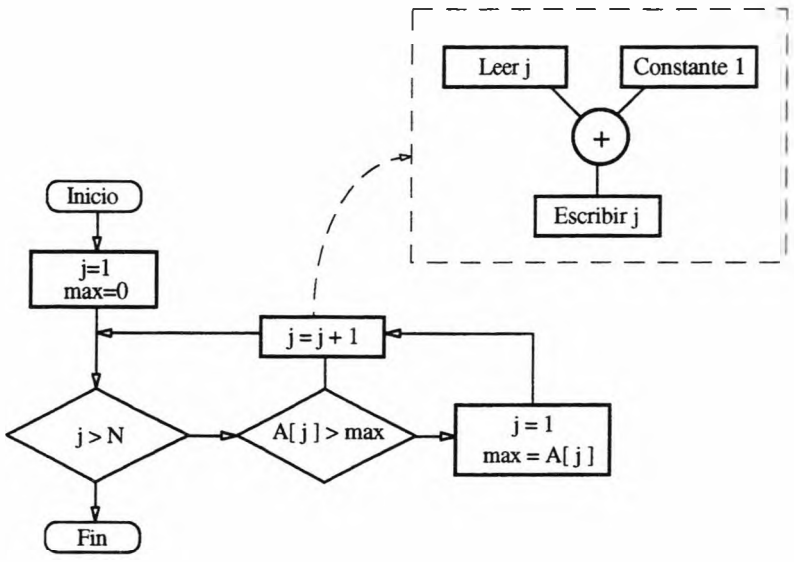


Figura I.9. Grafo de flujo de control/datos para la detección del máximo de un vector de datos (adaptado de [15]).

Tenemos aquí la ventaja de que podemos especificar a un nivel más bajo la forma en que se llevan a cabo las acciones del grafo de flujo de control. Por tanto, el nivel de detalle de la descripción funcional es mayor.

- **Lenguajes de Programación**

Los lenguajes de programación se estructuran en base a diferentes paradigmas, como Programación Estructurada, Funcional, Lógica, Orientada a Objetos, Visual, etc.

No todos se adecúan a la resolución de un problema. De hecho, problemas que pueden resolverse en unas pocas líneas mediante uno de ellos, pueden suponer cientos de ellas en otro. Por tanto, es de vital importancia el conocimiento de todos ellos para realizar una selección adecuada en un caso concreto.

El siguiente es un ejemplo de un modelo escrito en un lenguaje estructurado que realiza el mismo proceso mencionado anteriormente para la localización del máximo de un vector de datos.

```
funcion maximo ( entrada A : Vector ) : entero;
    variable i, max : entero ;
    max = 0
    desde i = 1 hasta 20
        si A[ i ] > max entonces
            max = A[ i ]
        fin si
    fin desde
    devolver max
fin funcion
```

Figura I.10. Ejemplo de lenguaje de programación para la detección del máximo de un vector de datos.

- **Procesos Secuenciales Comunicados**

Este modelo descompone el sistema en una serie de procesos concurrentes, cada uno de los cuales consiste en una secuencia de instrucciones de programa correspondientes a algún lenguaje de programación. La forma en que se comunican los diferentes procesos da lugar a variantes basadas en subprogramas, en corrutinas y/o concurrencia pura (*threads*). La estructura básica de cada una de estas posibilidades se muestra en la figura I.11.

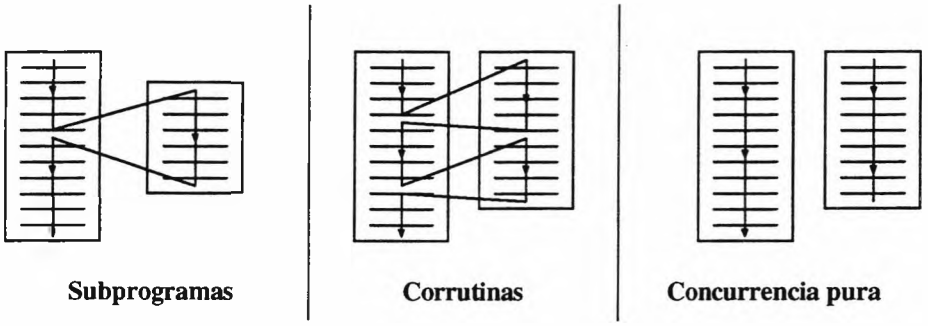


Figura I.11. Alternativas de comunicación en procesos secuenciales comunicados.

La primera técnica consiste en la interrupción de la ejecución normal de la secuencia de instrucciones, para ejecutar al completo un conjunto de instrucciones, denominadas subprogramas, que, cuando finalizan, devuelven el control al programa llamante. En el caso de las corrutinas, si bien sigue existiendo un único ciclo de ejecución, no existe ningún programa que lleve el control sobre el otro, sino que cualquier corrutina puede interrumpir su flujo de ejecución, y continuar en el punto donde lo dejó cuando la corrutina es invocada de nuevo. La concurrencia pura permite que diferentes procesos se ejecuten de forma absolutamente independiente.

- **Máquinas de Programas-Estados**

Las máquinas de Programas-Estados hacen uso simultáneo de los modelos orientados a estados y de los procesos secuenciales. En el ejemplo mostrado en la figura I.12 se observa un diagrama de estados jerárquico-concurrente, en el cual se ha usado la descripción en lenguaje de programación para la descripción funcional del proceso 'C'.

A modo de resumen, se muestra en la tabla I.1 los diferentes tipos de modelos, con sus correspondientes subdivisiones. Esta tabla no pretende en modo alguno presentar una tipología exhaustiva de la modelización de sistemas, sino aquellos que mejor se adecúan a su representación, de cara a la captura de las especificaciones de sistemas que vayan a ser codiseñados mediante herramientas automáticas.

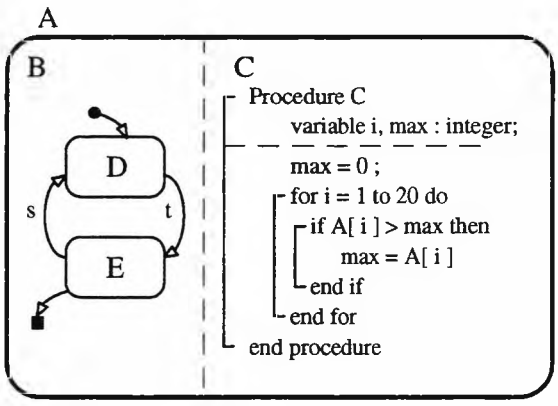


Figura I.12. Máquina de programas-estados para la detección del máximo de un vector.

Como conclusión, baste decir que ningún modelo se adecúa a todas las clases de sistemas. El mejor modelo es aquel que más se acerca a las características del sistema que modeliza.

TIPOLOGÍA DE MODELIZACIÓN	
Modelos orientados a los estados	– Máquinas de estados finitos – Máquinas de estados finitos con caminos de datos – Máquinas de estados finitos concurrentes jerárquicos
Modelos orientados a la actividad	– Grafos de flujo de datos – Grafos de flujo de control
Modelos orientados a la estructura	– Diagramas conectividad-componente
Modelos heterogéneos	– Grafos de flujo de control/datos – Lenguajes de programación – Procesos secuenciales comunicados – Máquinas de programas-estados

Tabla I.1. Tipos de modelos más usuales utilizados en las herramientas de captura de especificaciones.

2.3. Lenguajes de especificación

He aquí una serie de características [15] que han de cumplir los lenguajes de especificación para describir sistemas de una forma lo más parecida posible a su comportamiento real:

- **Concurrencia:** define si es posible la ejecución de diferentes procesos al mismo tiempo. Todos los lenguajes de especificación de *hardware* permiten la concurrencia, ya que es una característica fundamental incluso en el desarrollo de sistemas sencillos.
- **Transiciones entre estados:** define la forma en que el sistema puede cambiar de estado, si se puede detectar dicho cambio, y finalmente, si es posible ejecutar alguna acción como resultado de dicho cambio. De esta forma, los sistemas se encuentran en cada momento en un cierto estado, y un cambio en la situación del sistema se traduce en un cambio de estado del mismo.
- **Jerarquía:** indica la posibilidad de definir unos módulos dentro de otros, subordinados a bloques cada vez mayores. Existen dos tipos de jerarquías: jerarquía de estados, establecida a partir de una descomposición estructural, y jerarquía de subprogramas dentro del conjunto, como consecuencia de una descomposición de comportamiento.
- **Construcciones de Programación:** existen una serie de construcciones sintácticas (sentencias) que podemos emplear para implementar instrucciones. Así, encontramos diferentes tipos de sentencias: de asignación, condicionales, iterativas o bucles.
- **Capacidad de indicar la finalización de un proceso:** algunos lenguajes permiten avisar a un módulo del estado de otro módulo, ya sea en activo o para advertirle que dicho módulo terminó su tarea. Se indicarán como estados finales en máquinas de estados finitos, como puntos de fin en el caso de las máquinas de estados-programa.
- **Comunicación entre procesos y Sincronización:** define la forma en que se van a comunicar los diferentes procesos entre sí, así como la forma en que se van a

establecer puntos de sincronismo, de ruptura y/o de interrupción parcial de la ejecución. Normalmente la comunicación atiende a dos modelos básicos, mediante memorias compartidas y por flujo de mensajes.

- **Manejo de Excepciones:** las excepciones permiten interrumpir un proceso de forma inmediata, independientemente del estado en que se encuentre el sistema, según ocurra un cierto suceso (lo que en programación se conoce como interrupciones).

Los lenguajes de captura de especificaciones de uso más extendido en la actualidad son VHDL, Verilog, HardwareC, SDL, Silage, StateCharts, Esterel, Argos, SpecCharts, etc.

Hoy en día uno de los más importantes y conocidos es VHDL [4], es una norma del IEEE, sirviendo de plataforma o transición entre otros dos lenguajes cualesquiera cuando deseamos realizar una 'traducción'. La tabla I.2 puede ayudar a comparar estos lenguajes de especificación, atendiendo a las características más comunes que poseen. De esta forma podemos seleccionar el lenguaje que más se adapte a nuestras necesidades. Sin embargo, dada la disponibilidad y el coste de las herramientas más complejas, esto no siempre es posible. Por ello, puede ser necesario simular algunas características que el lenguaje seleccionado no incorpore de forma explícita, a costa de aumentar la complejidad de la descripción.

Como puede verse, SpecCharts [32] cumple prácticamente todos los requisitos que mencionábamos anteriormente para los lenguajes de especificación. Se puede considerar, por tanto, que SpecCharts es la mejor opción siempre y cuando el diseño a realizar justifique la inversión.

CARACTERÍSTICAS DE LOS LENGUAJES						
Característica / Lenguaje	Transiciones	Jerarquización	Concurrencia	Programación	Excepciones	Finalización
VHDL	NO	PARCIAL	SI	SI	NO	SI
Verilog	NO	SI	SI	SI	SI	SI
HardwareC	NO	PARCIAL	SI	SI	NO	SI
StateCharts	SI	SI	SI	NO	SI	NO
Esterel	NO	SI	SI	SI	SI	SI
SpecCharts	SI	SI	SI	SI	SI	SI

Tabla I.2. Características de lenguajes de descripción de *hardware* más usuales (adaptada de [14]).

2.4. Fases en la captura de especificaciones

El proceso de captura de especificaciones consta de varias fases, que son:

1. **Modelización:** realizamos una descripción del sistema mediante un modelo conceptual, que será el correspondiente al lenguaje de captura de especificaciones empleado. Este es el proceso más importante en la captura de especificaciones, ya que es donde el diseñador toma parte activa, y es el encargado de definir la funcionalidad del sistema. El proceso se realiza al completo de forma manual. Si bien, es posible la creación de bibliotecas de módulos que permitan su utilización en diseños posteriores que incluyan funcionalidades similares. Es de vital importancia, como hemos visto, la selección de un modelo que se adecúe al máximo a las características del sistema a diseñar.

Asimismo, la experiencia del diseñador permitirá reducir al mínimo el tiempo de desarrollo.

2. **Descripción:** se crea la descripción de este modelo usando un lenguaje de captura de especificaciones. Para que esta fase sea realizada fácilmente es imprescindible que el modelo seleccionado en la fase anterior no posea características impropias del lenguaje. Por ejemplo, si el modelo seleccionado incluye excepciones, es importante que el lenguaje seleccionado las contemple, pues en otro caso, puede ser muy difícil describir el sistema. De hecho, puede ser necesario, incluso, rediseñar el sistema en su totalidad.
3. **Simulación**: se valida la descripción formal mediante técnicas de simulación y/o verificación. Normalmente, estos procesos se realizan mediante baterías de datos de test que nos permiten confirmar la corrección del diseño. Si se descubre algún error, se corrige, y se vuelve a la fase inicial. Esta fase, junto con las anteriores forman un proceso cíclico. Es decir, las fases de modelización, descripción y simulación se realizan de forma consecutiva hasta que el diseño describe perfectamente nuestras especificaciones.
4. **Generación de la Especificación final:** se obtiene la especificación funcional sin ningún detalle de implementación. Esta fase suele ser automática. A partir de la descripción libre de errores procedente de las fases anteriores, el sistema genera la especificación funcional, que como ya vimos, puede contener tanto instrucciones directamente escritas por el diseñador como generadas de forma automática por el lenguaje de descripción seleccionado. Normalmente se utiliza VHDL como lenguaje de salida standard.

Como resumen de esta primera parte, hacemos hincapié en los puntos importantes. Hemos de seleccionar un lenguaje para la captura de especificaciones. No sirven los lenguajes naturales ya que, como se ha dicho, tienen el inconveniente de la ambigüedad. Para estudiar el sistema nos ayudamos del modelo asociado, el cual formaliza y simplifica el estudio del mismo. Hemos de establecer una correspondencia entre el modelo conceptual del sistema y un lenguaje con el que hacer realidad dicho modelo. Sin esta correspondencia no se puede llevar a la práctica el modelo.

3. EJEMPLO PRÁCTICO

Veremos a continuación la captura de especificaciones en el diseño de un sistema empotrado como puede ser un teléfono móvil. A continuación veremos la descripción funcional de este sistema, tanto en lenguaje natural como en un lenguaje formal de especificación, el SpecCharts. Veremos la gran diferencia que existe entre una especificación de tipo formal y una no formal como el lenguaje humano.

Figura I.13. Ejemplo de captura de especificaciones : un teléfono móvil.

Descripción en Lenguaje Natural

- Cuando el interruptor está apagado, la máquina ignora la línea de teléfono y el resto de los botones. El estado es inactivo.
- Cuando el interruptor se pasa a la posición encendido, el sistema pasa al estado activo en espera.

- El estado activo en espera supone que el teléfono está en espera, y por tanto permite recibir llamadas procedentes de la línea telefónica.
- Cuando se pulsa la tecla descolgar, el teléfono pasa al estado descolgado. Da comunicando si recibe alguna llamada, y responde a la pulsación de las teclas.
- Si estando en el estado descolgado se pulsa la tecla colgar, el teléfono pasa al estado activo en espera
- Cuando se pulsa memoria, y un dígito, el teléfono lee de su memoria el número de teléfono almacenado correspondiente al dígito en cuestión, y lo marca de forma automática.
- etc., etc., etc.

Como podemos observar, quedan muchos interrogantes acerca del funcionamiento del sistema que no se responden con la especificación en lenguaje natural. Es imposible sacar conclusiones correctas con sólo estos datos.

Especificación en SpecCharts

Tenemos un módulo, el controlador, que vamos a analizar. Por supuesto, el sistema consta de otros muchos módulos, pero son demasiados como para analizarlos todos (y además no es ese el objetivo de este ejemplo). Debemos hacer una puntualización con respecto a los símbolos empleados en los diagramas. El comienzo del proceso se representa por el símbolo ▼, mientras que el final del mismo se indica con un símbolo cuadrado ■. Las flechas indican acciones y cambios de estado.

Vemos en la figura I.14 los dos estados en los que puede estar el controlador, *apagado* o *encendido*. La transición de uno a otro estado se realiza según los cambios en la variable asociada al botón de encendido.

A continuación se van refinando cada uno de estos estados, de forma que lleguemos al final a una descripción detallada del funcionamiento de nuestro sistema. Vemos que el estado *apagado* no necesita de refinamiento, puesto que en dicha situación el controlador no debe realizar ninguna operación. Sin embargo, el estado *encendido* debe ser especificado con mayor detalle, puesto que en esta situación el controlador del teléfono debe realizar una serie de funciones de acuerdo a ciertas

variables de sistema. Vemos que este estado comienza con una inicialización general del sistema y, a continuación, se pasa al estado activo en espera, que deberá ser refinado posteriormente.

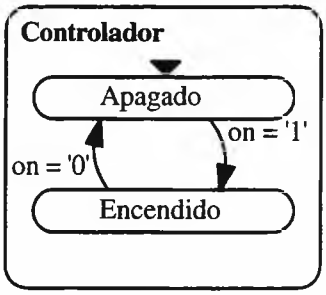


Figura I.14. Diagrama parcial en SpecCharts de la especificación funcional del proceso Controlador.

Cuando el botón ON esté pulsado ('1') se encenderá el contestador. Cuando ON no esté pulsado ('0') o vuelva al estado '0' estará apagado.

Comenzamos ahora una descomposición jerárquica: pasamos a analizar el bloque 'Encendido' (figura I.15) Cuando encendemos el móvil, se inicializa el sistema y se responde a la línea. Cuando se pulse un botón, el sistema responderá a dicha pulsación de botón.

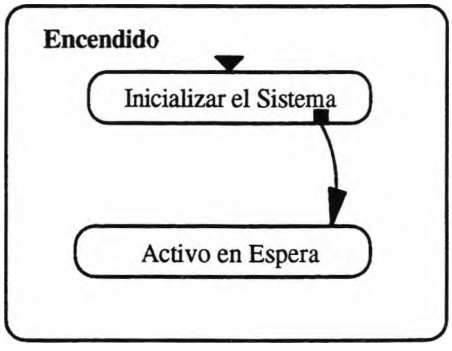


Figura I.15. Diagrama parcial en SpecCharts de la especificación funcional del proceso Encendido.

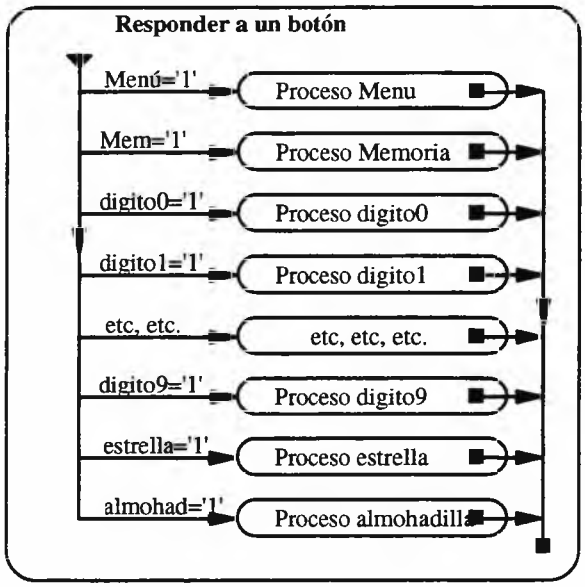


Figura I.17. Diagrama parcial en SpecCharts de la especificación funcional del proceso Responder a un botón.

Ante una pulsación de botón, la variable asociada a dicho botón tomará valor '1', y esto provocará que se ejecute un cierto proceso (se repite la llamada para el botón de la estrella, se inserta una pausa automática en la marcación para el botón de almohadilla, etc.).

Por ejemplo (figura I.18), para el botón de menú, que nos permitirá acceder a otras opciones del teléfono, hemos de mostrar un menú en el *display* del teléfono móvil, y a continuación leer la opción seleccionada por el usuario.

El proceso de Encendido inicializará el sistema, y pasará al estado Activo en Espera, de tal forma que pueda recibir una llamada, o responder a un botón pulsado por el usuario.

En el estado 'Esperando Llamada' (figura I.16), el sistema estará pendiente de si llega alguna llamada. Si ésta se produce, el sistema deberá activar la alarma del móvil, y esperar a que el usuario descuelgue el mismo. Si estando en el estado Esperando Llamada se detecta la pulsación de un botón por parte del usuario, pasaremos a otro estado en el cual se atiende al proceso asociado a dicho botón y, una vez acabado éste, volvemos al estado anterior.

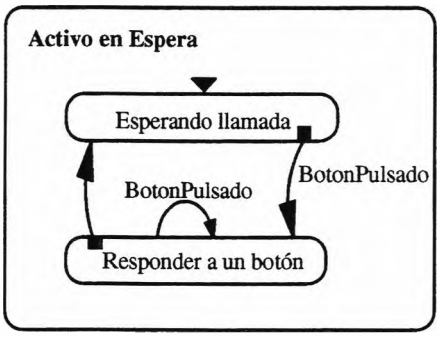


Figura I.16. Diagrama parcial en SpecCharts de la especificación funcional del proceso Activo en Espera.

Pero, ¿de qué manera responderá el sistema, dependiendo del botón que pulsemos? ¿qué debe hacer el sistema si pulso el botón memoria, o menú, o cualquiera de los posibles dígitos que existen en el receptor, o el botón para reproducir la última llamada?

La especificación refinada del estado Responder a un botón se muestra en la figura I.17.

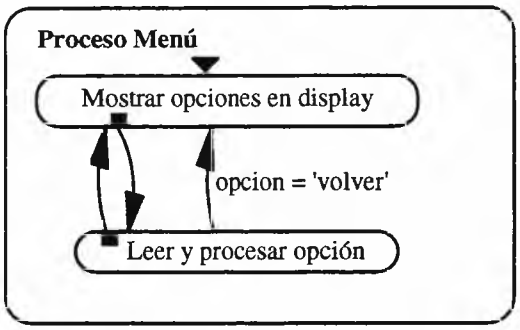


Figura I.18. Diagrama parcial en SpecCharts de la especificación funcional del proceso Menú.

Para el botón de memoria (marcar un número de teléfono almacenado en la memoria), hemos de leer el número de memoria que el usuario desea, y a continuación realizar la marcación automática (figura I.19).

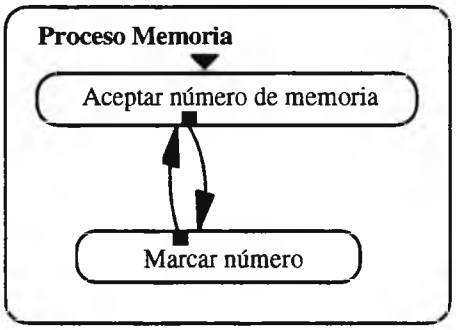


Figura I.19. Diagrama parcial en SpecCharts de la especificación funcional del proceso Memoria.

Siguiendo con el proceso de refinamiento, obtenemos la descripción del resto de procesos y subprocesos que configuran el sistema. Aquí podemos ver también que hacemos uso de un lenguaje de programación para definir el estado MarcarNumero. A continuación se muestra una simplificación del código correspondiente al proceso MarcarNumero escrito en SpecCharts :

```
behavior MarcarNumero(numero integer range 0 to 9)type code is
   begin
         Longitud = MemoriaLong[numero];
         for i in 1 to longitud loop
               digito = Memoria[numero];
               marcar(digito);
         end loop;
   end;
```

Si nos fijamos, hemos ido profundizando en la estructura de forma jerárquica, adentrándonos en los diferentes niveles de abstracción, aumentando el nivel de detalle, hasta llegar a un punto en que debemos completar el modelo con instrucciones en el lenguaje de especificación que vayamos a emplear: hemos profundizado tanto que basta con describir el comportamiento deseado con unas pocas sentencias. De hecho, todos y cada uno de los módulos que se definan deben ser subdivididos a su vez, hasta que al final, todos los submódulos del nivel inferior han de ser codificados en un lenguaje formal. Así, la herramienta que estemos utilizando genera instrucciones correspondientes a los estados, transiciones, condiciones de fin, etc. y el usuario introduce el código correspondiente a todos y cada uno de los submódulos de que consta el diseño. Por último, todas las instrucciones son ensambladas de forma automática, generándose el código fuente correspondiente al diseño completo en el lenguaje de especificación de *hardware* utilizado por la herramienta seleccionada, normalmente VHDL. Dichas instrucciones constituyen la entrada para los módulos sucesivos en el proceso de diseño.

Para finalizar, la figura I.20 muestra una parte de la especificación funcional de nuestro teléfono móvil, donde el único módulo en que se ha llegado a descomponer completamente es MarcarNumero, que como se ve está escrito en código fuente.

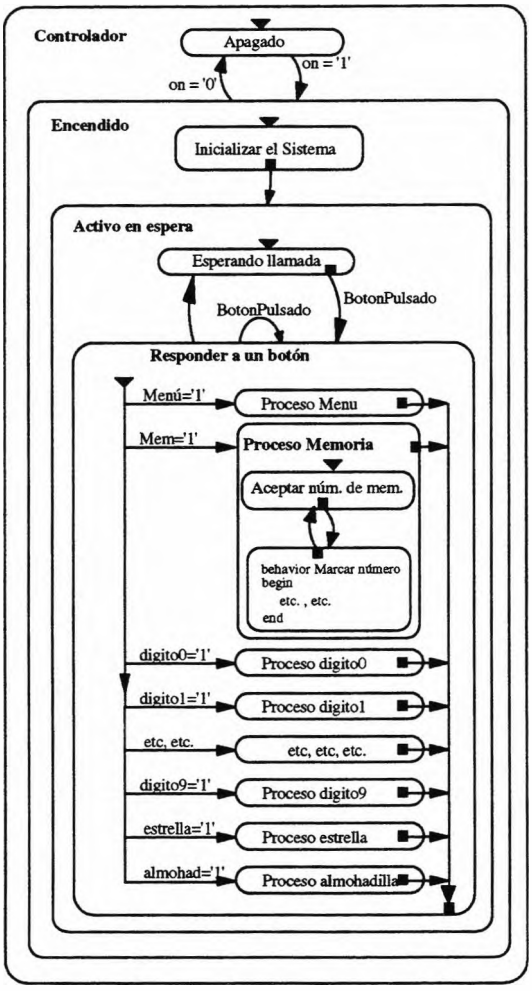


Figura I.20. Diagrama parcial en SpecCharts de la especificación funcional de un teléfono móvil.

4. CONCLUSIONES

Como hemos visto, la captura de especificaciones es el proceso por el que definimos formalmente el comportamiento de un sistema. Este proceso se realiza mediante la descripción en un lenguaje de especificación de la funcionalidad del

sistema. Existen múltiples alternativas para la selección del lenguaje de especificación. El criterio más importante para su selección debe ser la adecuación entre el modelo implícito del lenguaje y el sistema que queremos diseñar. Una vez realizada la descripción de la funcionalidad del sistema, se procede a realizar una validación del diseño mediante múltiples simulaciones realizadas de forma sistemática que confirme la corrección de la descripción realizada. Por último se genera de forma automática la descripción mediante un lenguaje formal de especificación, que normalmente es VHDL.

La fase de la captura de especificaciones es el primer paso en el codiseño de sistemas empotrados. Posiblemente es la fase crítica, ya que es donde el factor humano sigue siendo clave para la consecución de un buen diseño. Las fases posteriores pueden ser automatizadas en mayor o menor medida, pero la captura de especificaciones precisa de personal altamente especializado y con gran experiencia si se quiere llegar a la obtención de un producto óptimo en coste, tiempo y calidad.

CAPÍTULO II

SÍNTESIS DE SISTEMAS

1. INTRODUCCIÓN

El primer paso en el codiseño de sistemas *hardware/software* ha sido la captura de especificaciones para describir funcionalmente el sistema.

Una vez descrito el comportamiento del sistema deseado, se obtiene una especificación funcional. La síntesis de sistemas consiste en implementar esta especificación mediante un conjunto de componentes interconectados, tales como procesadores, circuitos integrados de aplicación específica (ASIC), dispositivos lógicos programables (FPGA), memorias y buses; es decir, buscar una arquitectura o conjunto de componentes interconectados, y asignar cada una de las funciones de la especificación a estos elementos, cumpliendo las restricciones que se impongan al sistema. Son muchas las posibles arquitecturas que pueden obtenerse de un modelo. Es tarea del diseñador evaluar las restricciones impuestas al sistema para cada una de las posibles arquitecturas y seleccionar aquella que mejor se adapte a sus necesidades.

La entrada a la síntesis de sistemas es la especificación funcional. Este proceso produce como salida una posible arquitectura. El esquema general del codiseño se

muestra en la figura II.1, donde se sitúa la síntesis de sistemas dentro del marco general del codiseño.

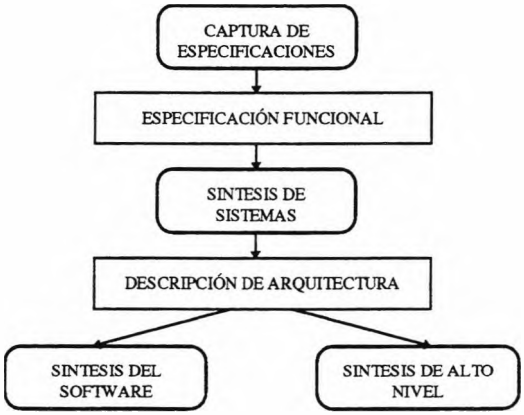


Figura II.1. **Esquema general del codiseño.**

Como ilustración de la síntesis, se presenta un sistema procesador de TV interactivo multimedia (ITVP) descrito en [14]. El sistema almacena imágenes de vídeo que pueden luego mostrarse acompañadas de texto y audio. El usuario interactúa seleccionando elementos de un menú en un teclado (figura II.2).

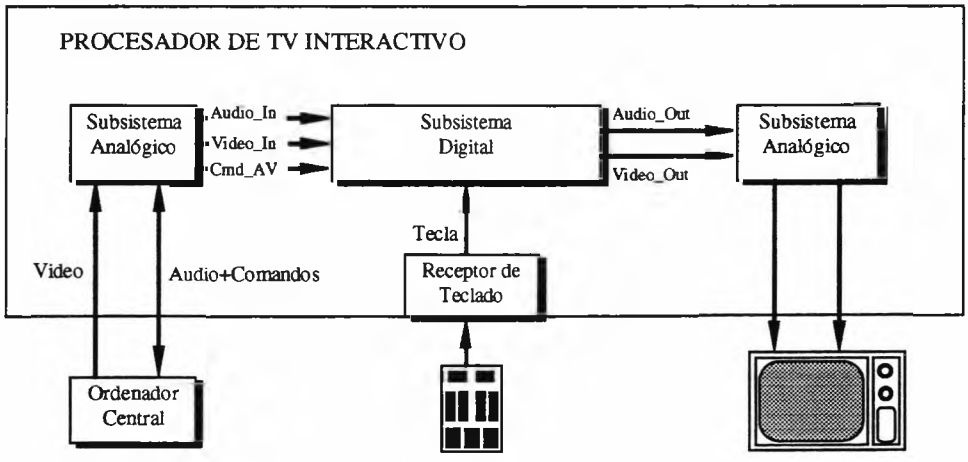


Figura II.2. Sistema de TV interactivo multimedia.

El diseño de este sistema implica describir la especificación funcional (figura II.3) y posteriormente implementarlo en una determinada arquitectura (figura II.4).

El sistema está compuesto por seis componentes, tres memorias, dos ASIC's y un procesador. La memoria 1 almacena información de audio; la memoria 2, almacena información de vídeo; la memoria 3 almacena fuentes y caracteres que van a ser mostrados en la pantalla. El circuito ASIC 1 implementa la función que almacenará las señales de audio entrantes y generará las señales de salida. El circuito ASIC 2 implementa la función de almacenamiento y generación de imágenes de vídeo y además almacena los comandos especiales codificados en la entrada de vídeo-audio. Finalmente el procesador implementa las funciones que procesan los comandos especiales de vídeo-audio, los principales comandos del sistema y los comandos del usuario.

```
entity itve is port (
  audio_in: in bit_vector (7 downto 0);
  audio_out: out bit_vector(7 downto 0);
  .......);
end itve;

architecture itva of itve is
begin
behavior itv type concurrent subbehaviors is
  type audiomemtype is array.....
  signal audio1, audio2: audiomemtype;
  .....

begin
  behavior AlmacenaGeneraVideo.....
  ...
  behavior AlmacenaAVCmd...
  ....
  behabior SobreimpresionCaracteres
  ...
  behavior AlmacenaAudio
  ......
  behavior GeneraAudio type code is
  begin
    wait until gen_audio;
    for i in 1 to num_audio loop
      audio_out <= audio1(i);
     ....

  behavior ControlPrincipal type sequential
subbehavior is
  begin
    Iniciar: (TOC, true, ModoTV);
    ModoTV:(TOC, true, ModoITV);
    ModoITV: (TOC, true, ModoTV);
     ..........
  end itva;
```

Figura II.3. Especificación funcional del sistema ITVP.

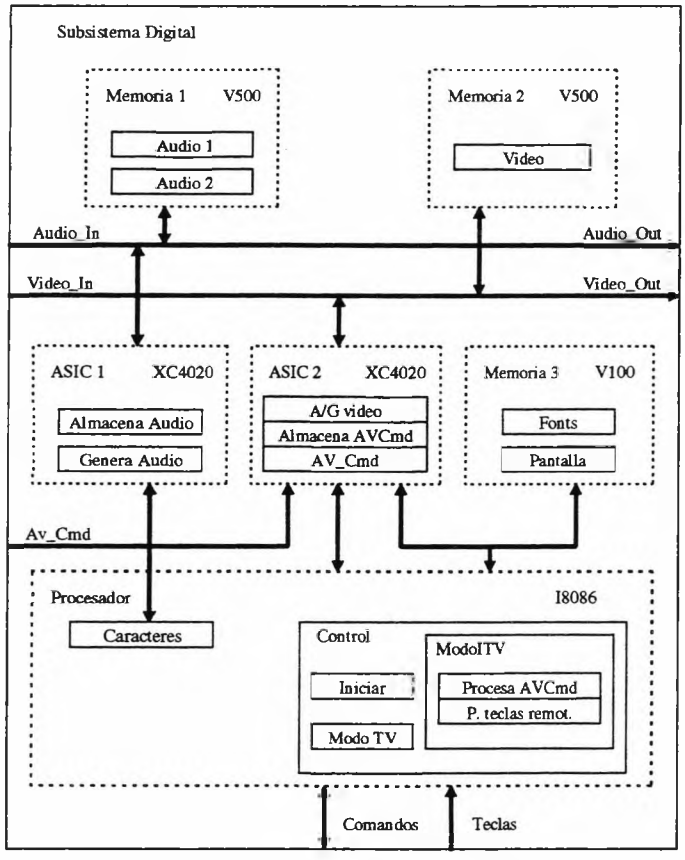


Figura II.4. Arquitectura para el sistema ITVP.

No existe ninguna metodología o herramienta que se utilice habitualmente para ayudar al diseñador a crear la especificación funcional y mapearla en una determinada arquitectura de forma automática.

La mayoría de los diseñadores trabajan de forma expresa para cada situación que se le plantea, y la experiencia anterior juega un papel importante. Una metodología jerárquica puede facilitar el desarrollo de estos sistemas. En estas metodologías, primero se especifica el sistema funcionalmente, a continuación se exploran distintas implementaciones con ayuda de herramientas y se genera una descripción más refinada. Mediante la evaluación de las métricas impuestas al sistema, se encuentra la arquitectura que mejor se adapta a las necesidades del diseño.

2. SÍNTESIS DE SISTEMAS

En el codiseño de sistemas constituidos por elementos *hardware* y elementos *software*, se comienza por una descripción del comportamiento deseado para el sistema, descrito mediante una especificación funcional y el resultado final es una implementación del sistema. Para conseguir esta implementación se siguen una serie de etapas bien definidas: captura de especificaciones, síntesis de sistemas y síntesis de los componentes *hardware* y *software*.

En la figura II.1 se muestran las distintas etapas en el desarrollo de sistemas *hardware/software*. La primera etapa consiste en la captura de especificaciones vista en el capítulo I, que produce como resultado una especificación funcional del sistema. Ésta se utiliza como entrada a la segunda etapa. La Síntesis de Sistemas consiste en implementar la especificación funcional en una determinada arquitectura a nivel de componentes (*hardware* y/o *software*). Estos componentes se desarrollan posteriormente bien con la Síntesis de alto nivel para los componentes *hardware* (capítulo III), o bien con la Síntesis del *Software* para los componentes *software*.

Una vez definido el comportamiento del sistema, en la síntesis se exploran distintas alternativas de diseño hasta encontrar una que satisfaga nuestras necesidades. Para hacer esto, se transforma la descripción inicial en una implementación más adecuada. Se realiza una reserva de componentes y se especifican sus características. A continuación se realiza una partición de la especificación funcional entre distintos componentes reservados. Se estudian distintas alternativas y se realiza una estimación de la calidad de cada uno de estos diseños. La especificación funcional inicial se refina para tener una nueva descripción que refleje las decisiones anteriores, y se genera una descripción del sistema detallando los procesadores, memorias y buses del sistema. Se utilizan técnicas de simulación para verificar que la descripción refinada es equivalente a la descripción inicial. El resultado de este refinamiento es una descripción a nivel de sistema que contiene algunos detalles acerca de la implementación de la arquitectura del sistema que se ha desarrollado, pero todavía bastante funcional.

A continuación se crea una implementación para cada componente usando técnicas de diseño *hardware* y *software*. Para un procesador se requiere una síntesis

software, es decir, crear una serie de instrucciones que realicen la tarea deseada. Para un ASIC se realiza una Síntesis de alto nivel.

3. FASES EN LA SÍNTESIS DE SISTEMAS

Dada una especificación funcional de un sistema, el diseñador debe crear un diseño mediante componentes interconectados. Cada componente implementa una porción de esta especificación. La aceptación de un diseño depende de que satisfaga las métricas deseadas en el mismo, como el tamaño, potencia, coste. La tarea de evaluar cada diseño es larga, así que el diseñador normalmente sólo examina unos determinados diseños, frecuentemente los que pueden evaluarse más rapidamente. Usando una especificación formal se puede explorar automáticamente un gran número de diseños.

La exploración implica cuatro subproblemas interdependientes: reserva, particionado, transformación y estimación. No es necesario resolver estos problemas en un determinado orden y normalmente son necesarias varias iteraciones antes de encontrar el diseño más adecuado.

3.1. Reserva

La reserva es el problema de encontrar un conjunto de componentes para implementar las funciones del sistema. La figura II.4, muestra un ejemplo de reserva para el sistema ITVP. La reserva proporciona dos memorias V500 y una V100, dos ASIC's de Xilinx XC4020, un procesador Intel 8086 y tres buses.

El diseñador normalmente dispone de cientos de componentes donde elegir. En un extremo están los componentes *hardware* específicos (*custom*). Son muy rápidos, pero caros. En el otro extremo están los microprocesadores programables de propósito general que son baratos, pero lentos. Entre los dos extremos, hay innumerables componentes que varían en precio, potencia, tamaño, fiabilidad y esfuerzo de diseño.

Hay una gran variedad de microprocesadores, microcontroladores, FPGA's, procesadores paralelos y los ASIP's (*Application-Specific Instruction-set Processors*). Además hay cientos de componentes prediseñados que implementan una función particular, como memorias, árbitros, controladores DMA, multiplicadores en coma flotante, etc.

Los componentes se caracterizan por el conjunto de instrucciones, descripción parametrizada o número de objetos *hardware*. Los procesadores de propósito general, se caracterizan por la secuencia de instrucciones que representa la función implementada en el mismo. Los componentes de propósito especial, como multiplicadores en coma flotante, o controladores DMA se caracterizan por una función parametrizada. Estos componentes ejecutan el mismo programa con ligeras variaciones definidas por los parámetros. Cualquier parte de la especificación implementada en estos componentes debe ser transformada en una descripción parametrizada que concuerde exactamente con la función. Finalmente los ASIC's, FPGA's y *arrays* de puertas, se caracterizan por el número de objetos *hardware* tales como transistores, bloques combinacionales o puertas.

El trabajo del diseñador es elegir el conjunto apropiado de componentes entre una gran número de distintas posibilidades.

3.2. Particionado

Dada una especificación funcional y una reserva de componentes, hay que repartir o particionar la especificación y asignar cada parte a cada uno de los componentes reservados. Se pueden distinguir tres tipos de objetos en la especificación que pueden ser particionados por separado:

- Variables: Almacenan valores de datos. Se asignan a componentes memoria.
- Procesos: Transforman los valores de los datos. Un proceso consisten en un conjunto de sentencias en la especificación funcional. Se asignan a procesadores específicos o estándar.
- Canal: Transfiere datos de un proceso a otro. Se asignan a buses.

El particionado de la especificación debe satisfacer una serie de ligaduras o restricciones. Estas restricciones pueden ser el número de *Bytes* del microprocesador o microcontrolador, el número de puertas o *pines* de un ASIC, el número de palabras de memoria, el tiempo de ejecución de una función o la velocidad de un puerto E/S.

Hay dos tipos de particionado, estructural y funcional. En el particionado estructural el sistema se diseña con objetos estructurales (de grano fino), tales como puertas. Estas puertas luego se particionan entre distintos componentes específicos. Aunque es fácil de automatizar, no considera la implementación del *software*. En el particionado funcional se dividen las distintas funciones del sistema en grupos y cada grupo se asigna a un componente. Cada grupo se implementa mediante *software* o *hardware*. Hay que considerar varias cuestiones a la hora de realizar un particionado funcional.

- Definir la granularidad de los objetos, la cual establece el objeto funcional más pequeño a dividir. Una alta granularidad significa pocos objetos, pero complejos, que permiten una fácil interacción, tiempo de ejecución rápido para los algoritmos de particionado y rápida estimación, pero pocas posibles particiones.
- Seleccionar las métricas de diseño que se van a utilizar para definir una buena partición: coste, realización, velocidad de comunicación, consumo, área, tamaño, testabilidad, fiabilidad, tamaño de programas, tamaño de datos, tamaño de memoria.
- Adoptar un modelo para el cual se va a estimar los valores de la métrica. La estimación es necesaria debido a que pueden ser necesarias varias horas o días para construir un diseño para cada posible partición.
- Al estimar la métrica debe hacerse una optimización con una función objetivo o función de coste que se emplea para definir la calidad de la partición. Así podemos comparar dos particiones y seleccionar la que mejor cumple las restricciones.
- Son necesarios algoritmos eficientes para el particionado. Algunos como el algoritmo de agrupamiento (*clustering*), son rápidos, otros son más lentos, como los algoritmos genéticos, pero encuentran mejores soluciones.

Hay muchas técnicas que permiten ayudar al diseñador a realizar el particionado. Las tres principales categorías son: particionado *hardware*, particionado *hardware/software* y entorno de particionado interactivo.

El particionado *hardware* es una técnica donde con ayuda de herramientas se realiza un particionado funcional entre módulos *hardware*, como ASIC's o bloques dentro de un ASIC.

Las técnicas de particionado *hardware/software*, realizan el particionado funcional entre componentes *hardware* y *software*.

Los entornos interactivos realizan el particionado de los tres tipos de objetos en la especificación (variables, procesos y canales) entre distintos componentes (procesadores, ASIC's, memorias y buses). Estos entornos pueden utilizarse tanto para el particionado *hardware* como para el particionado *hardware/software*.

3.3. Transformación

Se ha supuesto que una especificación consiste en funciones que pueden implementarse una a una en los componentes del sistema. Sin embargo, algunas funciones en la especificación pueden hacerla más legible, pero si se implementan directamente es probable que no se obtenga el mejor diseño. Por ejemplo, un procedimiento que se implemente en un módulo *hardware*, puede producir un cuello de botella. Es preferible implementar el procedimiento como parte del proceso que lo llama. Como segundo ejemplo, considérese dos procesos concurrentes en la especificación, si se implementan con dos controladores por separado puede resultar muy caro, en cambio, se pueden mezclar los dos procesos para que se ejecuten secuencialmente en un sólo controlador, simulando concurrencia.

Estos son ejemplos de transformación en la especificación. Una transformación reorganiza la especificación para mejorar la implementación.

3.4. Estimación

Sería deseable poder evaluar métricas para un gran número de diseños con el fin de encontrar el que mejor satisfaga las restricciones impuestas al sistema. Podemos obtener estos valores derivados de la implementación del mismo, pero esto requiere mucho tiempo si se examinan muchos diseños. En cambio, se pueden estimar valores de la métrica creando una implementación *hardware* y *software* menos detallada, pero rápida de simular para cada componente.

Exactitud y velocidad son factores que compiten en el desarrollo de una estimación. Si se hace una implementación más completa, se obtiene mayor exactitud, pero menor velocidad.

Por ejemplo, se puede estimar rápidamente el tamaño *hardware* reservando y contando unidades funcionales y hacer una rápida estimación estadística del número de registros multiplexores y puertas controladoras. El tiempo ahorrado sobre una implementación más detallada tiene el precio de la menor exactitud.

En general, sólo es necesaria una estimación grosera durante el diseño del sistema. Por ejemplo, para saber si un conjunto de funciones se puede implementar mediante un *array* de puertas de 10^5 puertas, sólo es necesario saber si la función requiere más o menos de 10^5 puertas, no el número exacto de ellas.

Los parámetros para estimar las métricas más comunes son, tamaño *hardware*, tamaño *software* y realización. No son del todo exactas porque la implementación de la descripción de comportamiento en componentes no es una a una.

Para estimar el tamaño *hardware*, se sintetiza una determinada función a nivel RT y se determina el número de objetos requerido (registros, unidades funcionales, multiplexores, buses, registros de estado). Los algoritmos que realizan esta función son computacionalmente caros. Los estimadores sólo generan un subconjunto de objetos. Una vez determinados el número de objetos requeridos se determina el tamaño en varias tecnologías. Para implementación en FPGA de Xilinx, se suman el número de CLB's. Para *array* de puertas, se suman el número de puertas, y para circuitos integrados dedicados el número de transistores.

Para determinar el tamaño del *software* de una determinada función, se compila la función utilizando el juego de instrucciones del procesador. Si el compilador no está disponible, se utiliza un juego de instrucciones genérico. Una vez traducida la función, se obtiene el tamaño del *software* sumando el número de instrucciones.

Normalmente en la métrica adoptada, interesa estimar dos tipos de factores de la realización, el tiempo de ejecución y la velocidad de comunicación del bus. Para cada una de estas métricas, es interesante conocer los valores extremos, máximo y mínimo, y su valor medio. Estas métricas se pueden estimar con varios niveles de exactitud.

4. SIMULACIÓN Y COSIMULACIÓN

De alguna manera es necesario comprobar que la especificación inicial es completa y correcta. Una especificación es completa si tiene en cuenta todas las posibles secuencias de entrada que el entorno va a proporcionar al sistema. Una especificación es correcta si genera la salida esperada para cada secuencia de entrada. Para validar si una especificación es correcta o completa se pueden aplicar técnicas de verificación formal o técnicas de simulación.

Las técnicas de verificación formal implican realizar una afirmación sobre la especificación y entonces, comprobar que la afirmación se mantiene. Por ejemplo, se puede afirmar que todos los estados de una máquina de estados finitos son accesibles, y utilizar un teorema de prueba para comprobar tal afirmación. La simulación implica ejecutar la especificación y comparar la secuencia de salida generada con la secuencia de valores esperados. Hoy en día la simulación es la técnica de verificación más utilizada. Ni la verificación formal ni la simulación pueden validar si una especificación es completa debido a la gran cantidad de posibles secuencias de entrada que existen en cualquier sistema, aunque sea de un tamaño moderado.

La simulación es útil no sólo para verificar la especificación funcional inicial, sino también para verificar las descripciones del diseño más detalladas generadas en el proceso de diseño. En particular, debe asegurar que la funcionalidad de un diseño es consistente con su especificación inicial, debe detectar posibles cuellos de botella en el

camino recorrido desde la implementación de la especificación abstracta hasta llegar a una implementación con componentes reales con sus limitados recursos y asegurar que el diseño satisface las limitaciones de tiempo para la comunicación y sincronización.

Una característica de los sistemas es la jerarquización en niveles, por ello la simulación de los detalles de la descripción deben tener lugar en varios niveles de abstracción. El proceso de diseño que se presentó en la figura II.1, incluye cuatro diferentes modelos para el sistema,

- La especificación funcional, que describe sólo la funcionalidad del sistema sin ningún aspecto de implementación.
- La descripción a nivel de sistema, que incluye el modelo de buses y componentes.
- La descripción a nivel RT.
- La descripción a nivel lógico.
- La descripción física del sistema.

Para conocer todos los aspectos durante el proceso de diseño, se modela el sistema a diferentes niveles de abstracción. Este modelado jerárquico requiere diferentes simuladores. Integrar la simulación de varios modelos se denomina cosimulación. Un ejemplo común de cosimulación es la simulación de componentes *hardware* junto con instrucciones ejecutándose en un procesador (*software*). Es la cosimulación *hardware/software*. Esta cosimulación persigue dos objetivos contrapuestos: velocidad y corrección. La velocidad está relacionada con el tiempo requerido para la simulación. Debido a que las simulaciones son frecuentemente de varios órdenes de magnitud más lentas que una implementación real, la velocidad es crucial si se quiere simular un número razonable de secuencias de entrada. La corrección se refiere a la generación de los valores de salida apropiados para la simulación. Pueden aparecer valores incorrectos cuando se simulan diferentes partes del sistema por separado. Y produce salidas falsas cuando las distintas partes acceden a datos compartidos.

Como tercer objetivo, que compite con la velocidad, puede considerarse el poder disponer de una depuración interactiva, para poder avanzar paso a paso en la ejecución del sistema, examinando valores intermedios.

5. HERRAMIENTAS PARA SÍNTESIS

Son muchas la herramientas que se han desarrollado para el diseño de sistemas *hardware* o *software*, sin embargo, las herramientas para el codiseño aún están comenzando a desarrollarse.

5.1. Herramientas para el desarrollo de sistemas *hardware*

La mayoría de los trabajos en diseño asistido por ordenador (CAD) de sistemas *hardware*, se han enfocado al desarrollo de circuitos integrados (CI), así los ASIC's pueden desarrollarse a partir de descripciones estructurales, a nivel de transferencia de registros (RTL) o a partir de una descripción de comportamiento, utilizando entornos CAD integrados tales como CATHEDRAL III y IV, DESIGN FRAMEWORK II, System Architect´ Workbench, Olympus, etc. Sin embargo, el diseño de tarjetas está todavía desarrollado exclusivamente a nivel de estructura (esquemático de CI's) y de capas físicas. Describir una tarjeta compleja mediante un conjunto de sentencias que especifican los elementos de una tarjeta, sus interconexiones y jerarquía (*netlist* jerárquica) es complicado, equivalente a diseñar un CI a partir de puertas lógicas. En el diseño de CI's, la solución ha sido incrementar el nivel de abstracción y considerarlo compuesto por módulos de alto nivel disponibles en una librería parametrizable y reutilizable (sumadores, multiplicadores, RAM, etc.). En el diseño de tarjetas aún no se ha llegado a un nivel de abstracción alto. Los requerimientos básicos son: disponer de una librería amplia de módulos, de herramientas de síntesis y de una amplia variedad de *layout*, y todo ello integrado en un entorno de trabajo.

Uno de los trabajos desarrollados en esta línea es una herramienta CAD denominada Micon de CMU, descrita en [5]. Esta herramienta utiliza un sistema experto que diseña un ordenador en una sola tarjeta. El sistema permite que la tarjeta disponga de microprocesador, memoria ROM, memorias RAM estática y dinámica, memoria caché, puertos de E/S serie y paralelo, bus interfase estándar y circuitos que soportan funciones como decodificador de direcciones, generador de reloj, etc. El

usuario especifica el tipo de microprocesador, cantidad y tipo de memoria, y número y tipo de dispositivos de E/S.

5.2. Herramientas para el desarrollo de sistemas *software*

En el lado del *software* existen herramientas denominadas *CASE (Computer Aided Software Engineering)*, que permiten al diseñador desarrollar un sistema *software* complejo desde su análisis hasta su implementación. En los grandes proyectos, el uso de una potente herramienta facilita el diseño y desarrollo del mismo, ya que son muchas las personas que trabajan en paralelo y es necesario utilizar métodos y técnicas que permitan la coordinación del trabajo. En las herramientas CASE la integración es una de las cualidades más importantes para el usuario del entorno. Estas herramientas permiten a los diseñadores documentar un sistema desde la fase inicial de requerimientos hasta el diseño e implementación, permitiendo aplicar test que comprueben si el diseño es consistente y completo.

Hoy se tiende a herramientas integradas que además permiten intercambio de información entre sistemas abiertos. Estos entornos están basados en marcos de trabajo flexibles y herramientas portables, que facilitan el intercambio y desarrollo de la información. Ejemplo de este tipo de herramientas son StateMate de i-Logix, Matrixx/SystemBuild de Integrated System Inc., SES/workbench de Scientific & Engineering Software Inc., y ADARTS de Software Productivity Consortium.

5.3. Herramientas para el desarrollo de sistemas *hardware* y *software*

Aunque algunas herramientas CASE tales como StateMate y SES/workbench proporcionan un ligero acoplamiento entre el desarrollo de componentes *hardware* y *software* de un sistema, no hay demasiados trabajos que traten el problema conjunto del diseño de sistemas *software/hardware*. El desarrollo de herramientas de Codiseño *Hardware/Software* para poder diseñar concurrentemente sistemas *hardware* y

software de aplicación específica, es un área que se empieza a desarrollar en la actualidad.

Ptolemy de Berkeley es un entorno de simulación que principalmente está orientado a la cosimulación *hardware/software* de sistemas procesadores digitales de señales (DSP), y a la generación de código de sistemas de flujo de datos síncronos [20].

Chiodo et al. [9] utilizan un modelo de especificación formal para codiseño denominado red de máquinas complejas de estados finitos (CFSM) para describir sistemas de control caracterizados por una baja complejidad algorítmica, y describe técnicas para implementar un CFSM como una máquina de estados finita *hardware* o *software*.

Se van a describir a continuación las herramientas SIERA [29, 30] y CHOP [19]. La herramienta SIERA ilustra los trabajos que se hacen actualmente en el campo del desarrollo de sistemas *hardware/software* y que se ha empleado en el diseño de sistemas reales. Esta herramienta contempla desde la descripción del comportamiento del sistema hasta la descripción física de las tarjetas, CI's y *software*. El aspecto más importante introducido por SIERA es la utilización de una plantilla de arquitectura en la que se distribuyen los distintos módulos tanto *hardware* como *software*. La herramienta CHOP aunque se emplea sólo en el desarrollo de sistemas *hardware*, ilustra algunas de las técnicas de estimación de parámetros para implementación de módulos *hardware*.

6. HERRAMIENTA SIERA

6.1. Organización de SIERA

SIERA presenta el problema del diseño de sistemas proporcionando una metodología de diseño que soporta todos los estados implicados en el desarrollo de sistemas de aplicación específica, desde la descripción de alto nivel a la generación del

software o de las tarjetas. Como se muestra en la figura II.5 el diseño de un sistema en SIERA se realiza en dos fases: generación de arquitectura y generación de módulos.

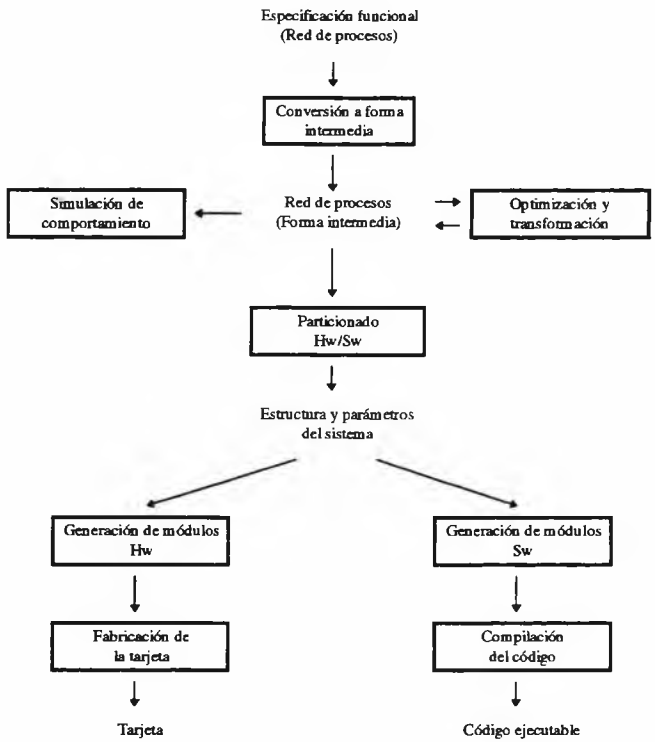


Figura II.5. Esquema general de diseño en la Herramienta SIERA.

La *Generación de Arquitectura* se refiere al proceso de generación de la arquitectura apropiada para el sistema comenzando desde la descripción de alto nivel. En SIERA, el sistema se especifica como una red parametrizada de procesos concurrentes que se comunican utilizando un canal FIFO, con un sólo elemento que escribe y un sólo elemento que lee, para interconectar los puertos de entrada y salida de los procesos. La especificación de la red de procesos se simula para comprobar que está de acuerdo con el modelo de computación abstracto. La red de procesos, después de las apropiadas transformaciones para optimizarla, se particiona en la plantilla de arquitectura. La plantilla define el sistema como una jerarquía multicapa de buses

implementada en varias tarjetas de aplicación específica, en las que se organizan de forma escalable y parametrizada los módulos *hardware* dedicados, los módulos *software* que se ejecutan en procesadores programables y la sincronización entre ellos.

La *Generación de Módulos*, se refiere a la implementación física del sistema dada su arquitectura como una composición de módulos *hardware* y *software*.

Cada proceso en la especificación es mapeado bien como un proceso *hardware* en un módulo *hardware* dedicado o como un proceso *software* secuencial ejecutándose en un módulo procesador programable que puede ser compartido por otros procesos *software*. Así el sistema se particiona de acuerdo con la granularidad en la descripción del sistema.

Hay dos aspectos importantes en el problema de generar la arquitectura. De un lado el modelo predeterminado sobre el cual se organizan los módulos *hardware* y *software*, y de otro se ha de crear una particularización de este modelo para implementar una determinada especificación funcional.

6.2. Plantilla de arquitectura

Basándose en el análisis de sistemas en tiempo real existentes, se elige una plantilla de arquitectura parametrizada con una capa, y organización jerárquica de buses. La plantilla define una familia de arquitecturas que permite una interconexión sistemática de un número arbitrario de nodos de procesos heterogéneos que pueden estar basados bien en un *hardware* dedicado o en un procesador programable por *software*.

El factor más importante al seleccionar la plantilla es la elección de un esquema de interconexión entre varios módulos. Los requisitos para elegir una determinada plantilla son,

- Bajo coste de implementación.
- Conectividad sistemática que simplifique la comunicación y sincronización.
- Permitir diseñar sistemas de distintos tamaños.

La interconexión entre distintos módulos se realiza mediante buses ya que es uno de los esquemas más apropiados en una tarjeta para comunicar sus componentes.

Una vez determinado el sistema de interconexión mediante buses, hay que precisar,

- Cómo organizar la interconexión entre varios buses si el ancho de banda no es suficiente.
- Si el bus va a ser controlado por un maestro o por varios.

Un bus con un maestro, donde sólo un módulo puede iniciar una transacción es fácil de implementar porque no se necesitan mecanismos de arbitrio, pero tiene como inconveniente que toda la comunicación debe hacerse a través del módulo maestro y esto requiere mecanismos de interrupción para requerir el servicio del maestro.

Un bus con varios maestros necesita caros mecanismos de arbitrio para gestionar el acceso al bus.

La elección del esquema de interconexión de buses en SIERA se ha basado en la observación de sistemas DSP y de control, que tienden a seguir un esquema jerárquico con un ancho de banda de comunicación alto entre tareas del mismo nivel, relativamente bajo entre niveles y decreciendo conforme nos movemos de niveles bajos a los altos, debido a que en los niveles más bajos del sistema se trata directamente con dispositivos E/S y se hace un tratamiento de la información para reducir la frecuencia de transición de datos visto desde niveles más altos, en los cuales se tiende al control del proceso de datos de los niveles inferiores.

El control jerárquico sugiere un sistema organizado en capas donde las más bajas tienen un mayor ancho de banda que las superiores y donde el *hardware* dedicado es dominante en las capas bajas de alta velocidad mientras el *software* es dominante en las capas altas de baja velocidad.

Se selecciona un modelo de arquitectura que puede ser visto como una interconexión híbrida consistente en varios buses con un maestro organizados en forma de árbol. El procesador maestro de cada bus es a su vez esclavo en el bus de nivel superior en la jerarquía, actuando como puente entre dos capas.

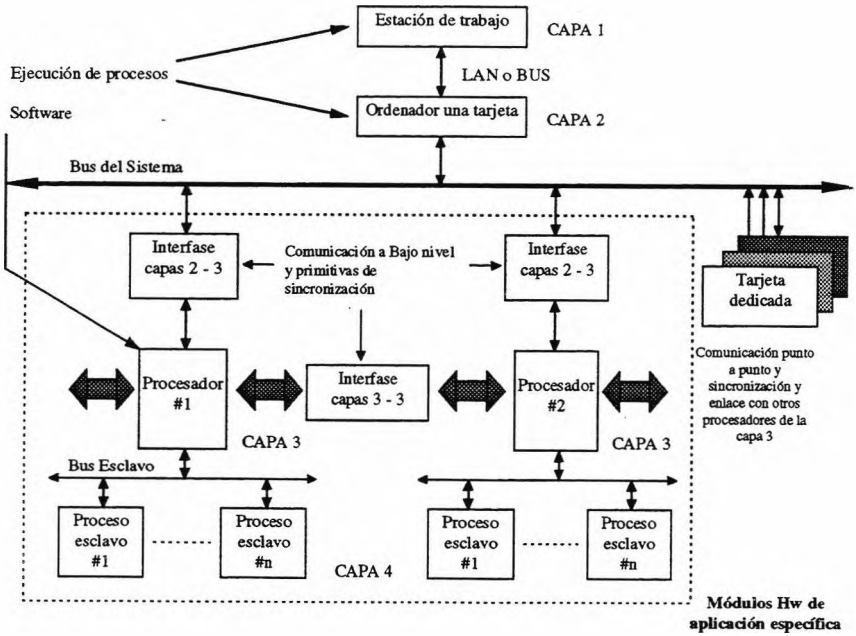


Figura II.6. Esquema general de la plantilla de arquitectura para un sistema SIERA.

SIERA utiliza una plantilla de cuatro capas que lleva el sistema de interconexión mencionado (figura II.6). Las dos capas de abajo en la jerarquía están ocupadas por tarjetas dedicadas. Cada tarjeta tiene uno o más procesadores programables mediante *software*. Cada uno de estos módulos procesadores ejecuta el núcleo de un sistema operativo (*kernel*) multitarea en tiempo real. Cada uno de estos procesadores coordina un determinado número de módulos esclavos de aplicación específica que pueden ser *software* o *hardware*. Estos módulos esclavos en la capa más baja de la jerarquía son el único lugar donde aparecen módulos *hardware* dedicados.

Los procesadores de la capa tres de la tarjeta dedicada interaccionan con el procesador maestro de la capa dos a través de un bus de interfase que requiere un conjunto de primitivas de sincronización y comunicación *hardware* y *software*.

La capa dos, es a su vez esclava de una estación de trabajo convencional que constituye la primera capa en la jerarquía. La comunicación entre la capa una y dos, utiliza un protocolo TCP/IP y es normalmente una red de área local como Ethernet.

6.3. Mecanismos de comunicación y sincronización en la plantilla de arquitectura

En la plantilla de arquitectura sólo se especifica la organización de los módulos en los que se ha mapeado la descripción del sistema mediante la red de procesos. Un aspecto importante es la comunicación entre varios módulos *hardware* y/o *software*. La comunicación entre procesos *software* que son implementados en el mismo procesador, puede realizarse completamente mediante *software* utilizando la memoria del procesador. El mismo *kernel* que se ejecuta en el procesador se encarga de realizar la comunicación entre distintos procesos. Para la comunicación entre distintos módulos, es necesario una interconexión física. La forma de realizar esta comunicación en el sistema descrito por la metodología SIERA consiste en utilizar memorias compartidas donde se implementan mecanismos de arbitrio, tales como semáforos para evitar colisiones entre distintos procesos que accedan simultáneamente a la misma dirección de memoria. Es importante tener en cuanta a la hora de realizar el particionado, considerar los procesos que están directamente conectados. Éstos deben implementarse en módulos contiguos, ya que de otra forma la comunicación debería hacerse a través de otro procesador intermedio, consumiendo recursos innecesarios.

6.4. Implementación del sistema

Una vez definido el modelo general o plantilla, la síntesis de cualquier sistema implica realizar una particularización de la arquitectura, que implemente correctamente la especificación funcional realizada para el sistema. El particionado se realiza a mano, con ayuda de herramientas que permiten simular el comportamiento de los distintos módulos, y la comunicación entre ellos. Una vez decidido cuáles de los módulos serán implementados mediante *hardware* y cuáles mediante *software*, la implementación de cada uno de ellos se realiza también de forma manual para el *software*, o mediante técnicas de síntesis de alto nivel para los módulos *hardware*.

Los trabajos actuales tienden a integrar las distintas tareas en el diseño de sistemas en un único entorno, pues un módulo *hardware* puede ser requerido por varios módulos de *software*, por ello no se pueden disociar totalmente la generación del módulo *hardware* y el *software*. SIERA evita soluciones *ad hoc* de bajo nivel estimulando la reusabilidad, uniformidad, modularidad y grado de abstracción de los módulos, y librerías *hardware/software*, fomentado que sean optimizadas y parametrizadas para ganar en reusabilidad.

Un ejemplo que puede presentarse en la implementación del núcleo de un procesador, a incluir en el sistema, sería diseñado con parámetros como tamaño de memoria, interfases de entrada/salida y otros atributos variables, a los que se les pasa el valor apropiado de estos parámetros al llegar a la implementación.

El diseño de un ASIC se contempla como un elemento más de los sistemas, y la base de datos de SIERA permite el acceso a las herramientas CAD específicas de la tarea de implementación de un ASIC.

El entorno SIERA es abierto y puede incorporar más módulos de librerías y herramientas, haciendo crecer el entorno integrado. Las librerías son una parte importante del entorno. Las hay de tres tipos, encapsuladas, primitivas de componentes y librería de subsistemas. Todas son parametrizables y modulares, la más importante es la librería de subsistemas, que están compuesta de otros módulos y/o primitivas especificada por una jerarquía de ficheros, que describen las interfases estructurales, información de ubicación, módulos apropiados de simulación, protocolos de entrada/salida y otra documentación.

6.5. Aplicación de SIERA al diseño de un sistema de control de un robot multisensorial

La herramienta SIERA se ha empleado en el diseño de varios sistemas multitarjetas y en sistemas dedicados. Se muestra a continuación un ejemplo de un sistema de control de un robot multisensorial (figura II.7). Las especificaciones del robot son el control en tiempo real de las posiciones del brazo del robot Panasonic con

seis grados de libertad y de la agarradera, con sólo un grado de libertad, además el control de la fuerza ejercida por la agarradera. Las entradas de datos provienen de la posición de las articulaciones, sensor de fuerza, sensor de proximidad y un subsistema de visión en dos dimensiones.

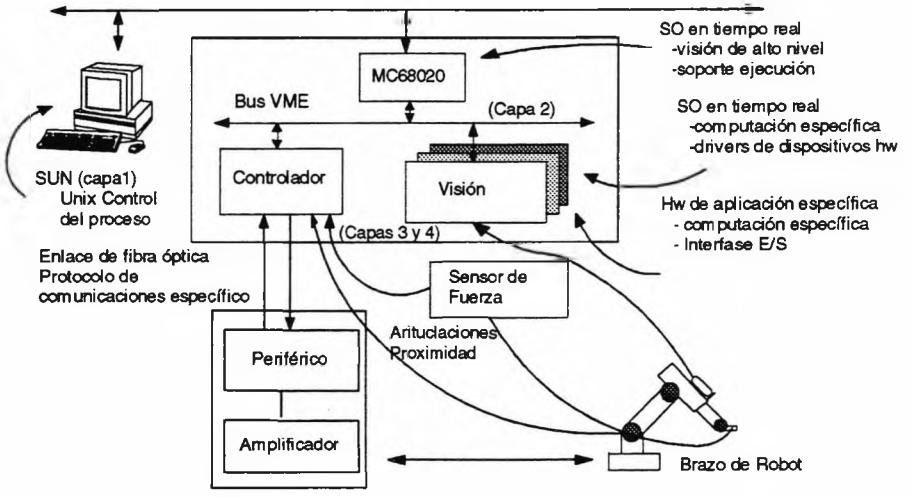


Figura II.7. Sistema de Robot.

El sistema de control requerido se muestra en la figura II.8. Aparecen como módulos independientes el control de posición y el control de fuerza. Algunos de los cálculos deben hacerse a la velocidad de muestreo, ya que se requiere un control en tiempo real.

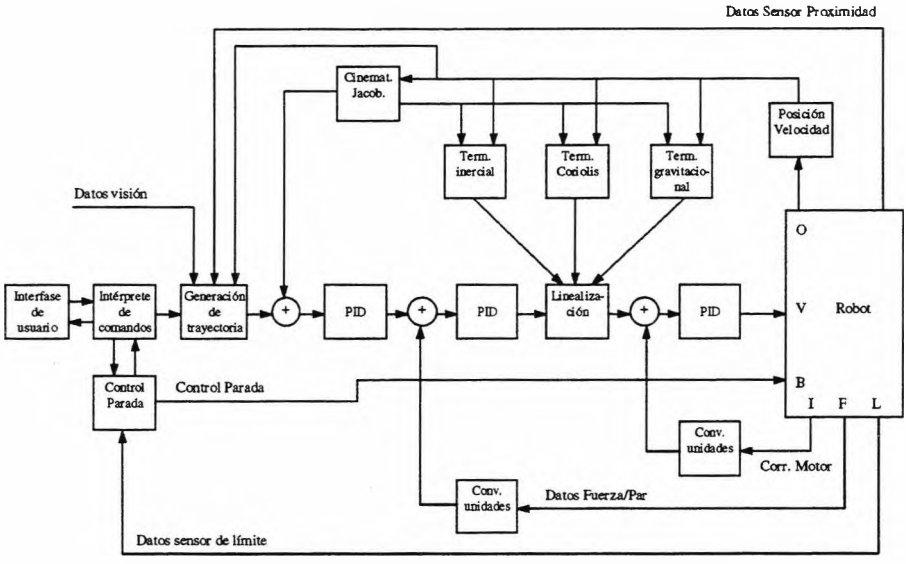


Figura II.8. Sistema de Control del Robot Panasonic.

La figura II.9 muestra la arquitectura del sistema y de control del robot siguiendo la plantilla de arquitectura descrita en los apartados anteriores. El sistema de control de fuerza y posición se encuentran localizados en una sola tarjeta y se corresponden con los niveles cuatro y tres de la plantilla. El sistema de visión se encuentra localizado en otra tarjeta también en los niveles cuatro y tres.

Estas tarjetas se unen mediante un bus VME a un procesador en la capa dos, que realiza un tratamiento de alto nivel de las imágenes y se utiliza como soporte para la ejecución de los módulos de las capas tres y cuatro.

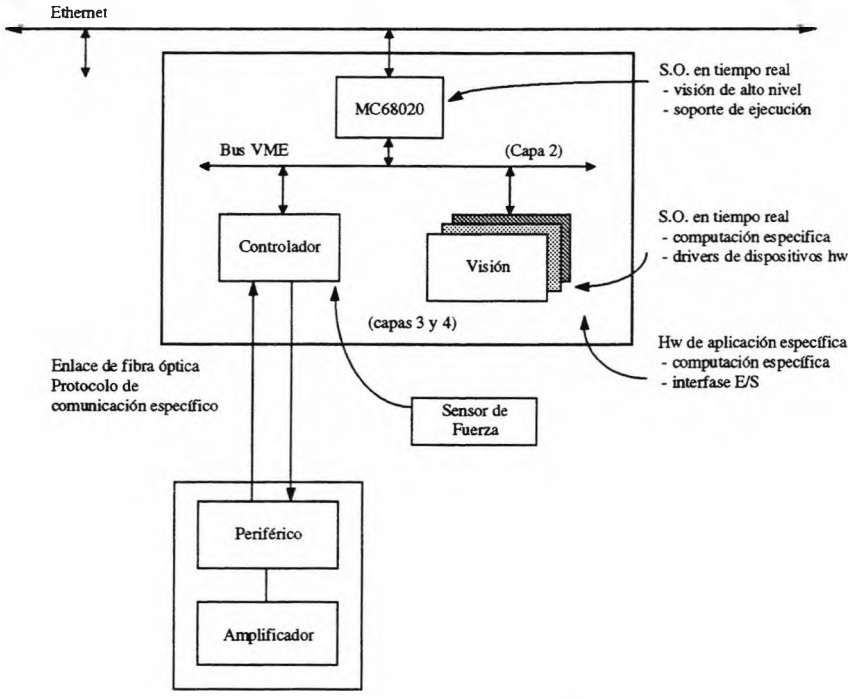


Figura II.9. Arquitectura del sistema.

En el nivel uno, se encuentra una estación de trabajo SPARC con el software que sirve de interfase para el usuario y es el que centraliza todo el sistema. La estación de trabajo se comunica con el procesador de la capa dos mediante una red Ethernet.

La figura II.10 muestra la organización de los distintos módulos de la tarjeta del controlador distribuidas en las capas tres y cuatro de la plantilla de arquitectura. En el nivel tres se encuentran dos procesadores TMS320C30 que se comunican entre sí y con el procesador de la capa dos mediante una memoria RAM con semáforos *hardware*. El procesador 1 tiene tres esclavos, un procesador DSP32C dedicado al proceso de cálculo de trayectoria; un ASIC para comunicación con otra tarjeta periférica; el tercer módulo es el sensor de posición y velocidad que está basado en dos ASIC. Este módulo está compartido con el procesador 2, el cual a su vez tiene dos esclavos más, un procesador DSP32C dedicado al cálculo de términos cinemáticos y jacobianos y un módulo de sensor de fuerza.

7. HERRAMIENTA CHOP

CHOP es una herramienta de diseño que proporciona mecanismos de evaluación para determinar la calidad y viabilidad de un particionado de forma rápida. CHOP ayuda al diseñador o a otras herramientas a particionar una especificación funcional en múltiples CI's cumpliendo unas determinadas restricciones físicas.

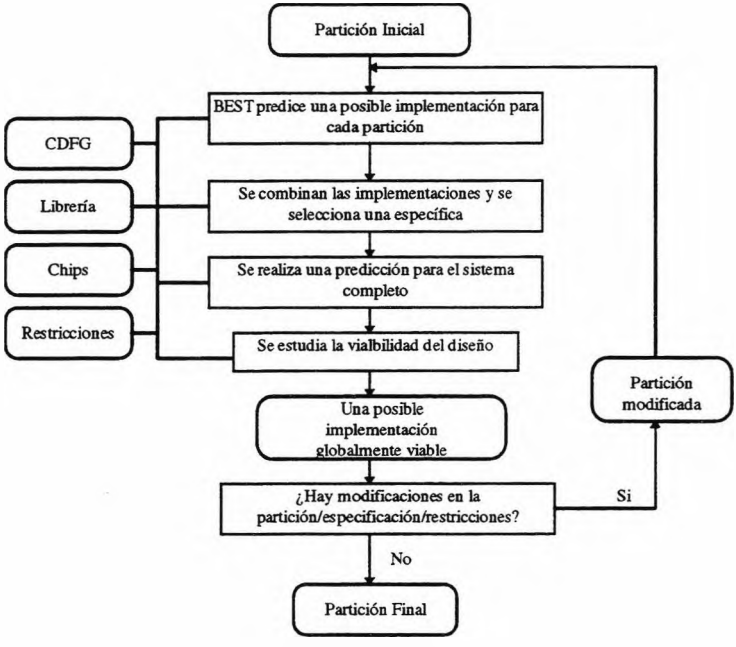


Figura II.11. Esquema de operación de CHOP

El modelo de particionado de CHOP permite implementaciones en uno o varios circuitos integrados con una o varias particiones en cada integrado, así que se pueden explorar diferentes configuraciones de particionado y producir diseños eficientes.

Los datos de entrada que requiere la herramienta son la especificación de comportamiento, una librería de componentes, el conjunto de integrados en los cuales el sistema va a ser particionado, los módulos de memoria usados, la asignación de los

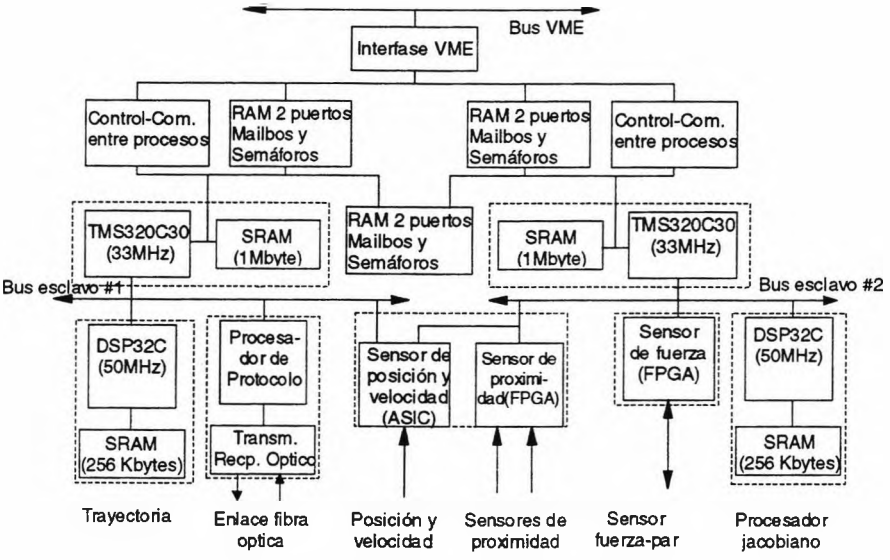


Figura II.10 Organización de módulos.

La metodología de SIERA permite desarrollar un sistema desde la descripción de alto nivel, a la generación del *software* y las PCB. El tiempo requerido para el diseño de esta tarjeta fue de aproximadamente dos meses. El uso de una plantilla de arquitectura predefinida simplifica el mapeado de las distintas funciones del sistema sobre una arquitectura real. Sin embargo, no es una buena estrategia cuando la métrica es la calidad del diseño y no el tiempo requerido para su desarrollo.

El desarrollo de las PCB se facilita con la combinación de una librería de módulos de *hardware* parametrizados, con herramientas de síntesis para generar la *netlist* de la PCB. Los módulos de librería son subsistemas completos como memorias, procesadores empotrados, módulos de interfase, adquisición de datos, etc., que hacen posible alcanzar un elevado grado de reutilización en distintos proyectos.

módulos de memoria a los distintos integrados, las particiones, la asignación de las particiones a los integrados, el estilo de arquitectura y los criterios de viabilidad.

El esquema de operación de la herramienta CHOP se muestra en la figura II.11. El diseñador realiza una partición inicial y con ayuda de la herramienta BEST se estudian posibles implementaciones para cada partición. Se combinan estas implementaciones y se elige una posible implementación global del sistema. A continuación se realiza un análisis del sistema completo para comprobar su viabilidad. En caso de que el diseño sea factible, el trabajo ha terminado, pero en caso contrario, se buscan otras posibles implementaciones.

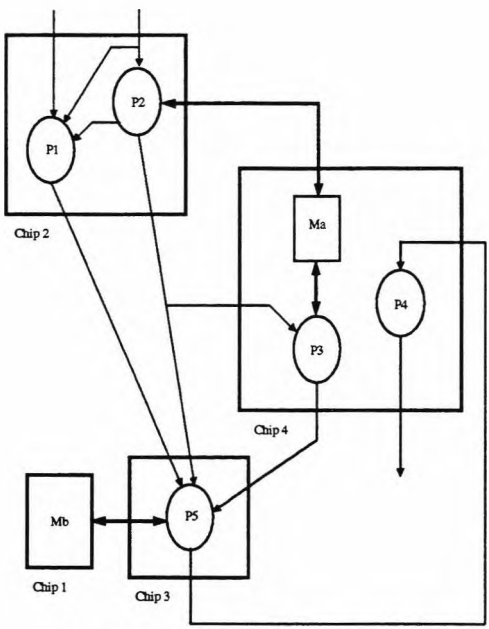


Figura II.12. Ejemplo de particionado.

La metodología de partcionado de CHOP puede verse más claramente con un ejemplo. Supongamos que se tienen cinco particiones (P1-P5) y dos unidades de memoria (M_A y M_B) y se realiza un particionado inicial tal como se muestra en la figura II.12. Los bloques de memoria pueden estar como un circuito integrado independiente (M_b en la figura II.12) o también aparecer junto a alguna partición en un integrado (M_a

en la misma figura). El primer paso es la creación de un grafo de tareas en el que los nodos representan las particiones y las flechas representan el flujo de datos. La creación del grafo de tareas implica determinar la cantidad de datos transferido y reservar suficientes *pines* para cada señal.

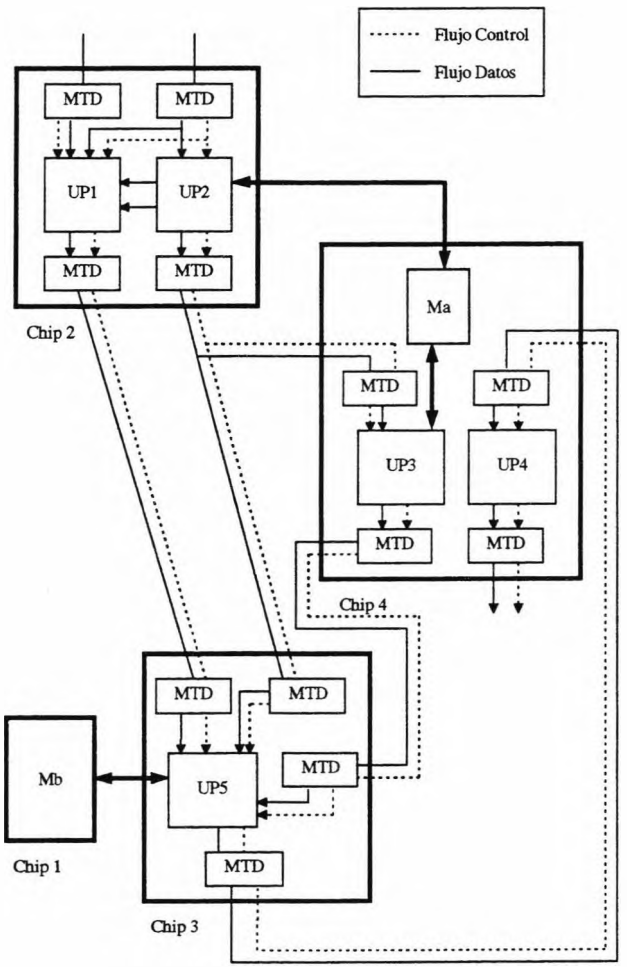


Figura II.13. Implementación del sistema.

La implementación final del ejemplo se muestra en la figura II.13, donde se disponen de unidades de proceso (UP) y de módulos de transferencia de datos (MTD). El siguiente paso consiste en estimar para cada partición posibles implementaciones y estudiar sus características de área y retardo (figura II.14)

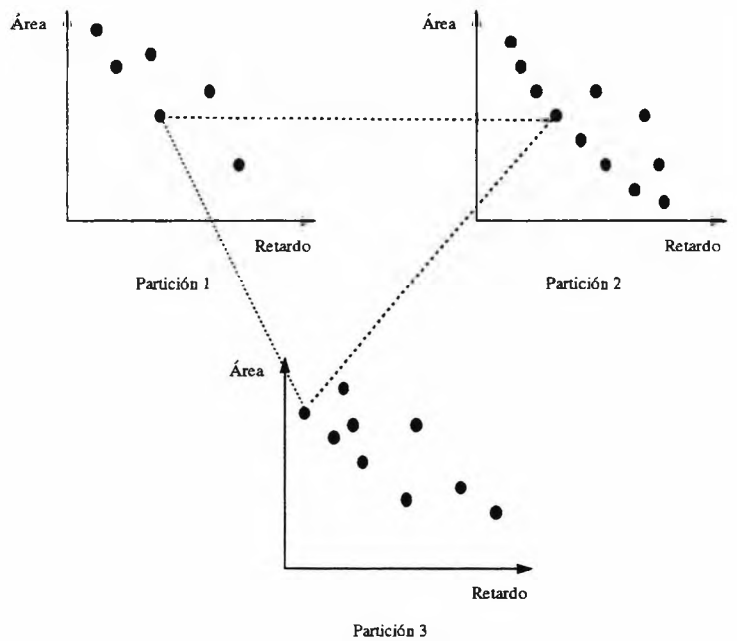


Figura II.14. Características de área y retardo.

La implementación global se obtiene como combinación de diferentes implementaciones individuales de cada partición. A la hora de hacer el estudio global no se examinan todas las posibles combinaciones sino que se aplican los siguientes teoremas con el fin de eliminar combinaciones que no serán viables.

Teorema I: *Para la partición i, el área A_k de la implementación k debe cumplir*

$$A_k \leq \text{Restriccion de Area del Chip} - \sum_{k \neq i} A_{i,min}$$

donde $A_{i,min}$ es el área mínima de la implementación de la partición i.

Teorema II: *Sea $D_{i,k}$ el retardo de la k-esima implementación de la partición i. Y $D_{i,min}$ el mínimo retardo de la implementación de la partición i, dado por*

$$D_{i,min} = \min_{k}(D_{i,k})$$

Sea $CP_{i,k}$ el retardo global del diseño usando $D_{i,min}$ para todas las particiones excepto para la partición i, y $D_{i,k}$ para la partición i. Si

$$CP_{i,k} \geq \text{Restriccion global de Retardo}$$

entonces la k-esima implementación de la partición i no se puede usar en el diseño global

El teorema I tiene como fin eliminar aquellas implementaciones cuya área no va a cumplir las restricciones de área globales para el sistema, mientas que el teorema II se aplica para eliminar las implementaciones que no cumplirán las restricciones de retardo.

Una vez eliminadas las implementaciones no válidas, se analizan las demás para encontrar aquella con la que se obtenga un diseño óptimo.

CAPÍTULO III

SÍNTESIS DE ALTO NIVEL

1. INTRODUCCIÓN

En el desarrollo de sistemas, con un planteamiento de codiseño, ahora volcamos la atención sobre el terreno de la circuitería física, el "*Hardware*". En síntesis de alto nivel, la entrada al proceso a seguir es una descripción del comportamiento del sistema en un lenguaje específico para la descripción del *hardware*, VHDL por ejemplo, y la salida a generar es una micro-arquitectura o representación estructural de los principales bloques funcionales del sistema.

En la actualidad se hace prácticamente impensable el diseño de un sistema electrónico sin la ayuda de herramientas CAD. De hecho, muchas de las vías de implementación disponibles para el diseñador a tal fin, como son los circuitos integrados programables, no serían abordables ni rentables si no llevaran asociadas herramientas CAD especializadas. La introducción de la síntesis de alto nivel en los diversos métodos de diseño ha tenido como consecuencia inmediata la reducción del tiempo empleado en realizar el diseño aunque ha seguido aumentando la complejidad de los sistemas.

Con frecuencia la síntesis de alto nivel se denomina síntesis arquitectural. El objetivo final de la síntesis es obtener una descripción detallada del circuito que pueda

ser apta para su fabricación. La síntesis de alto nivel es una primera fase de paso desde un nivel de abstracción alto próximo al de la concepción del sistema por el diseñador hasta desembocar en una descripción mediante una arquitectura de bloques algo más detallada, que posibilita una primera contrastación con las especificaciones pero que no es más rica en detalle.

La descripción de alto nivel proviene de las herramientas de captura de especificaciones [32] o como primer ejercicio en la concreción de una idea de diseño, donde ya el diseñador ha realizado un particionado del sistema y un modelado del comportamiento de los subsistemas, susceptibles de ser sometido a un proceso automatizado, en buena medida, de síntesis y optimización. Este último procedimiento suele llevarse a cabo gracias a un proceso iterativo, hasta dar con la topología adecuada y está encaminado a mejorar la calidad del diseño evaluando la métrica o conjunto de indicadores que son cifras de mérito del diseño obtenido.

El diseño de un sistema electrónico no es más que un proceso de refinado sucesivo de especificaciones iniciales, para alcanzar descripciones cada vez más detalladas y complejas que conducen a un circuito físicamente realizable. En este camino una etapa posterior sería la síntesis lógica que transcurre desde la descripción del modelo lógico de los bloques, mediante máquinas de estados finitos por ejemplo, a la descripción estructural más detallada de las primitivas o puertas lógicas.

Conceptualmente se denomina metodología descendente (*top-down*), como se expresa en la figura III.1, el camino que conduce desde las especificaciones al diseño listo para fabricar.

Tras la síntesis lógica, donde hemos obtenido una descripción con puertas genéricas abstractas, se presentan muchas alternativas de realización del producto, según las diversas tecnologías disponibles. Así, por ejemplo, podría realizarse con un conjunto de celdas de una librería de celdas específicas o bien con dispositivos lógicos programables del tipo FPGA (*field programmable gates array*) [6]. Este paso, denominado mapeado o proyección tecnológica, podría realizarse para ciertos bloques después de la síntesis de la arquitectura en el nivel denominado RTL (nivel de transferencia entre registros). En cualquier caso, cada mapeado que planteemos origina prestaciones distintas en área, consumo, tiempo etc., por lo que cada vez es mayor la

necesidad de disponer de herramientas (CAD) de ayuda al diseño, que permitan la traslación tecnológica, para que al pasar un diseño de una tecnología a otra nos dieran parámetros de evaluación de sus prestaciones.

Por último, en la figura III.1 se hace referencia a la fase final de implementación, la síntesis física en la cual según la tecnología adoptada, tendríamos las mascaras de fabricación de circuitos integrados, o los ficheros de configuración de la programación de los dispositivos FPGA.

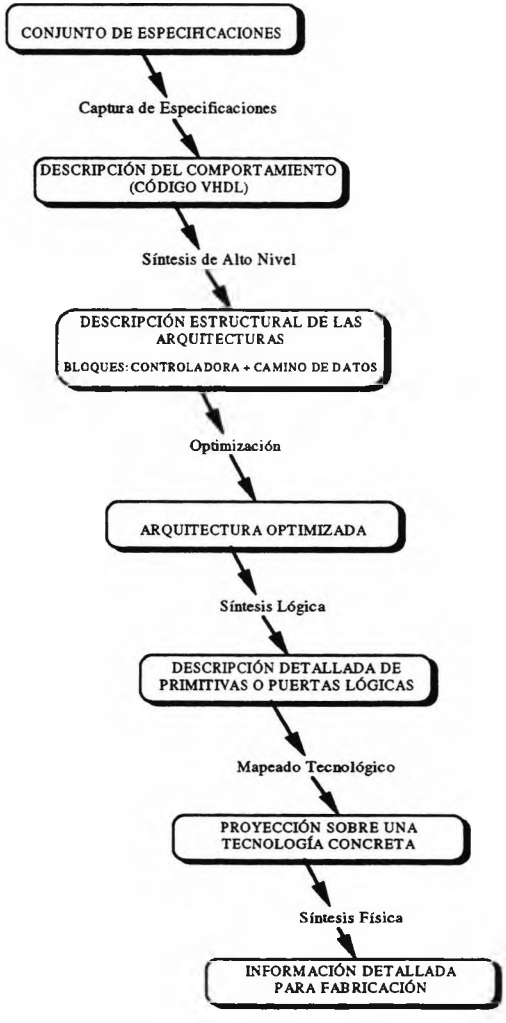


Figura III.1. Posibles fases de elaboración de un diseño siguiendo la metodología descendente (*top-down*).

Evidentemente la realización de un diseño no es una secuencia lineal, de arriba hacia abajo y debe estar abierta a múltiples lazos de retorno entre las etapas de la figura III.1 para depurar el diseño. En este flujo de trabajo y desde una etapa cada vez más temprana, se incorporan recursos de test interno o externo al circuito, porque la cuestión del test, ha pasado de ser deseable a ser imprescindible, sobre todo, con el vertiginoso aumento de la complejidad de los sistemas incorporados en los actuales circuitos integrados.

Tanto en tareas de síntesis como de optimización, la experiencia del diseñador es clave para interactuar con las herramientas. La ventaja es que la evaluación de alternativas se puede hacer desde una fase muy temprana del proceso de diseño, y que los tiempos de desarrollo de un producto se ven fuertemente reducidos merced a estas técnicas. Así si el comienzo de la década de los ochenta se caracterizó por el avance que supuso la disponibilidad de herramienta que ayudan a describir la geometría detallada (*Layout*) de las distintas capas que definen un proceso de fabricación de un circuito integrado, el final de década ha estado marcado por la consolidación de las técnicas de síntesis lógica. Tras esa etapa y desde el arranque de la década de los noventa, se ha observado cómo las técnicas de síntesis de alto nivel son las que han llegado al mercado mostrando una introducción acelerada.

Para algunos autores [24] el esfuerzo que significaba realizar un diseño de una cierta complejidad (veinte mil puertas) a principios de los ochenta, se ha reducido a una quinta parte con la síntesis lógica y hasta un orden de magnitud por debajo con la introducción de la síntesis de alto nivel. Esto se muestra en la figura III.2 medido en meses de trabajo de un diseñador que hubiese realizado el mismo diseño en distintas épocas. En definitiva, vemos cómo se ha pasado de una metodología en la que se empleaba mucho tiempo tanto para el diseño del sistema y su diseño lógico, como para el diseño físico (*layout*), a otras metodología que en la actualidad hacen que el tiempo de diseño del sistema se reduzca al máximo, y que las fases de diseño lógico y diseño físico estén prácticamente automatizadas al 100%. Recientemente [12] se ha presentado una metodología de trabajo que fomenta la descripción del sistema desde un nivel de abstracción alto y utiliza un ejemplo como prueba para justificar que se espera llegar a ciclos de diseño de 100 horas.

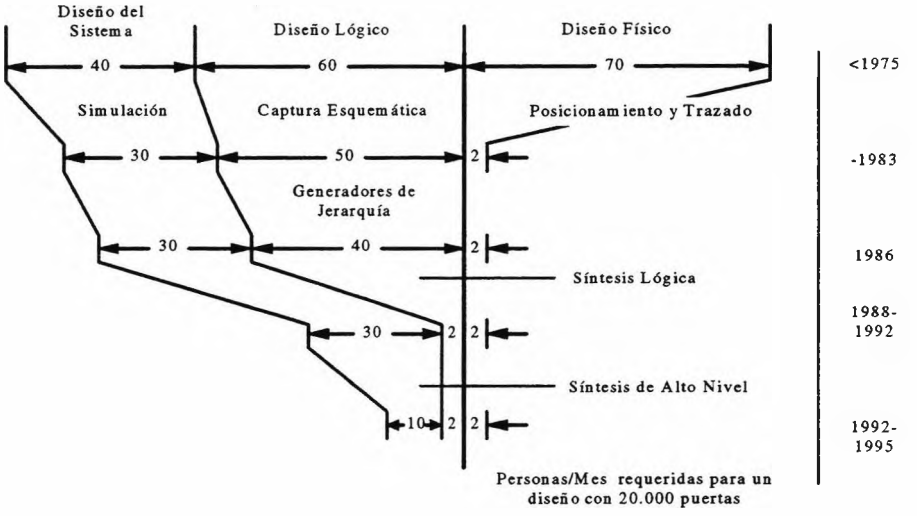


Figura III.2. Reducción relativa del esfuerzo preciso para realizar un diseño al introducir ayudas que automatizan tareas parciales, adaptado de [24].

El impacto de estas técnicas se centra en la reducción del ciclo de diseño, junto con la posibilidad de evaluar varias alternativas para un diseño concreto, ponderando entre ellas objetivos de costo como el tiempo, área ocupada o la facilidad que presenta el diseño por el que se ha optado para ser sometido a pruebas de verificación, todo ello en un marco de trabajo de herramientas integradas que muestran una sola interfaz con el usuario.

Desde un punto de vista metodológico, a la síntesis de alto nivel se ha llegado después de una época donde las tareas básicas eran la de captura esquemática y simulación, figura III.3, avanzando en sentido ascendente con ayuda de herramientas de CAD (*Computed Aided Design*) Electrónico, partiendo de los detalles circuitales más simples hacia estructuras jerárquicamente superiores. Sin embargo, esta metodología termina haciéndose inviable para circuitos complejos. En la síntesis de alto nivel se plantea una metodología descendente donde se describe el sistema desde una descripción abstracta, en un lenguaje de alto nivel para la descripción del *hardware* y se sintetiza automáticamente. Ésta es la situación de trasfondo en las denominadas herramientas EDA (*Electronic Design Automation*). Se trata en definitiva de realizar una descripción más abstracta con menos grado de detalle, pero capaz de afrontar

sistemas más complejos. El paso siguiente en la evolución está siendo presentando por nuevas herramientas de origen Universitario, aunque algunas ya en fase comercial, donde se parte de una captura de especificaciones y se explora el espacio de diseño, lo cual se está dando en llamar síntesis de sistemas ESDA (*Electronic Systems Design Automation*), aquí la captura de especificaciones tiende a ser la fase inicial de mayor nivel de abstracción desde donde se inicien las tareas de automatización del desarrollo de una aplicación.

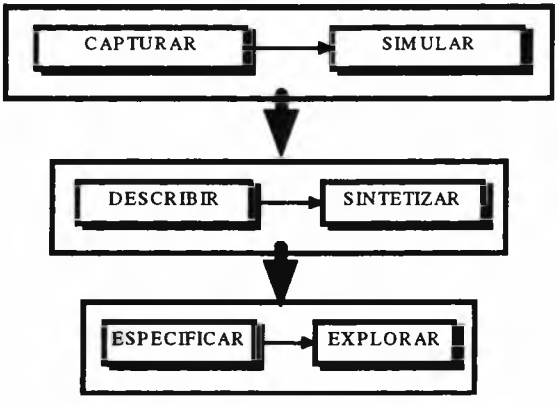


Figura III.3. Evolución histórica hacia la síntesis de sistemas.

2. LA SÍNTESIS DE ALTO NIVEL EN EL ESPACIO DE DISEÑO

Dependiendo de la tecnología adoptada hemos de recorrer un número distinto de fases para el desarrollo completo de un diseño electrónico. En la situación de diseño microelectrónico más clásica, se pueden diferenciar cuatro fases, mientras que se reduce a dos en la tecnología de dispositivos lógicos programables.

Para el primer caso y en primer lugar, se precisa pasar de la idea a unas especificaciones, para obtener un modelo apropiado a ese nivel de representación. Para ello se acomete las síntesis del sistema y/o una síntesis de alto nivel según la metodología seguida. Tras esto, las simulaciones funcionales y las pruebas de cobertura de fallos con un conjunto de vectores de test, sirven para validar la descripción lograda.

La tercera fase se dedica a definir la información detallada de las máscaras del proceso de fabricación de las obleas, y en la última fase indicada en la figura III.4-a, se obtienen los dados o *chips* sin encapsular sobre los que se pasan el mismo conjunto de vectores de test que fueron definidos y empleados en análisis de cobertura de fallos durante la segunda fase del diseño, procediendose en última instancia a la obtención de los *chips* ya encapsulados.

Para la tecnología de los dispositivos lógicos programables del tipo CPLD o FPGA [28], de la primera fase pasamos a una última, figura III.4-b, donde se adopta una configuración para los elementos programables sobre los que descansará físicamente el diseño, mediante la operación denominada descarga (*download*) del diseño sobre el dispositivo, el cual queda ya disponible para su empleo, evitando las tres últimas fases del procedimiento convencional.

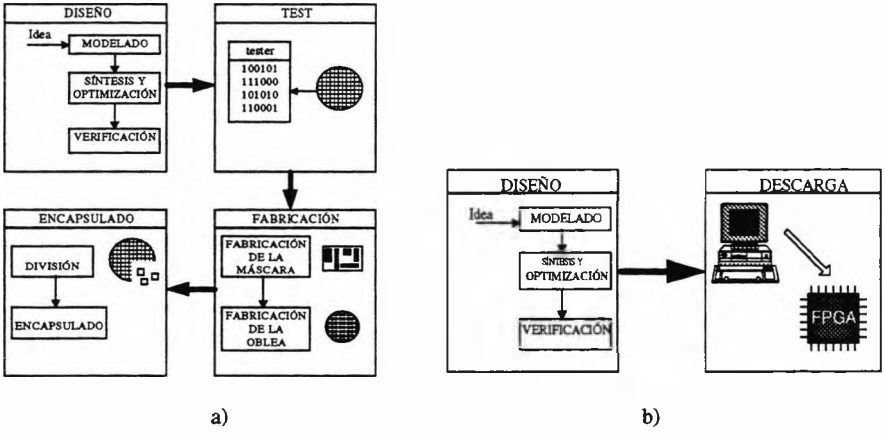


Figura III.4. Fases en el desarrollo de un diseño microelectrónico. a) Tecnologías clásicas. b) Lógica programable.

Cada elemento de un diseño electrónico se puede describir en tres dominios distintos, que se denominan Comportamiento, Estructural y Físico/Geométrico. Los cuales son como la visión de un mismo objeto desde distinta perspectiva.

En el dominio del comportamiento, la descripción es el comportamiento ideal esperado. "Interesa lo que hace y no cómo lo hace (o cómo está construido)", y esto

lleva asociado cierto grado de abstracción. En el dominio estructural se trata con una jerarquía de elementos funcionales e interconexiones. El sistema se describe con un conjunto de módulos, cada uno con una función propia y otra función global realizada gracias a las interconexiones y temporización a definir. Nos hallamos ante una aplicación de una representación (la del comportamiento) frente a varias representaciones estructurales posibles, siendo el diseñador el que establecerá con cuál de ellas se quedará, de acuerdo a las restricciones impuestas al sistema. Es en el dominio físico cuando se produce la traslación de la estructura hacia su representación física en el espacio o en el silicio, sin referencia en lo posible a su funcionalidad.

Gajski y Kuhn en 1983 proponen la idea del diagrama en "Y", para el espacio de diseño que recoge los tres dominios, y le confiere una estructura organizativa a dicho espacio de diseño, donde los dominios podemos decir que son idiomas para expresar el diseño [11].

Un círculo en este diagrama representa toda la información conocida sobre el diseño en un instante concreto de su desarrollo, mientras que el grado de abstracción adoptado en la descripción crece al alejarnos del centro. En este diagrama se pueden definir diversas acciones o tareas de diseño que se muestran mediante arcos orientados que son transiciones entre puntos del diagrama, véase la figura III.5. El proceso de diseño consiste en recorrer una cierta trayectoria en espiral desde el exterior (lo más abstracto) hacia el interior (lo más detallado) finalizando en el centro de ese espacio de diseño.

Según la formación del diseñador, sus hábitos de trabajo, experiencia o medios disponibles, se pueden describir distintas trayectorias tanto para un diseño digital, como para su extensión al caso analógico [17]. Por esto, el diagrama en Y juega un papel de ayuda o esquema mental para reconocer alternativas al proceso de diseño. Incluso podríamos recorrer el diagrama en sentido creciente, o sea, alejándonos del centro en cada fase del diseño. Este es el caso seguido por la Ingeniería inversa, que utiliza el método de "Extracción de parámetros" para averiguar la funcionalidad (dominio del comportamiento) de un sistema a partir del circuito físico (dado de silicio).

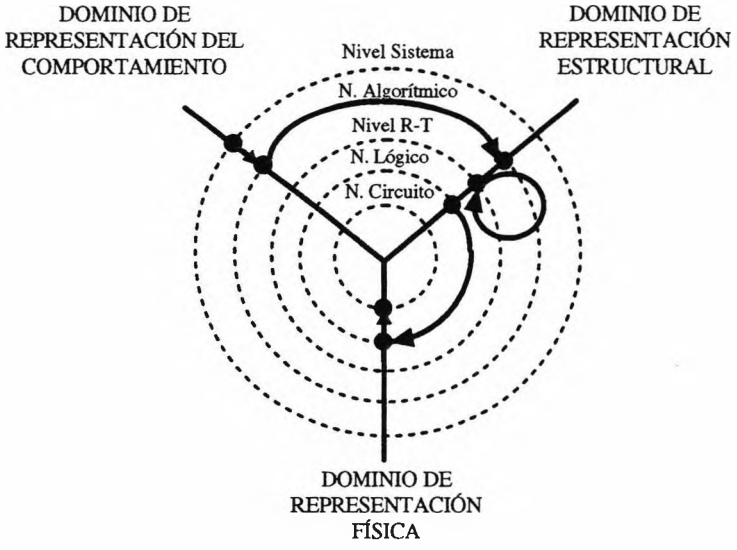


Figura III.5. Representación de un proceso de diseño mediante una trayectoria en el espacio de diseño según D. Gajski.

Sobre el diagrama de la figura III.6 podemos acometer la definición de algunos procesos básicos [24] con los que nos hemos de encontrar. Así denominamos: síntesis a un proceso de traslación desde la descripción de comportamiento a la descripción estructural. El análisis será el proceso de traslación desde la descripción estructural a una de comportamiento a menudo con propósito de verificación. También podemos redefinir ahora el concepto de optimización, que será el proceso de transformación en un mismo dominio y nivel con el propósito de mejorar la calidad del diseño, según una función objetivo específica. De otro lado denominamos refinamiento al proceso de traslación dentro de un dominio que añade información más detallada al diseño. Y Generación al proceso de traslación desde una descripción estructural a otra física, que se puede realizar sobre una pastilla de silicio.

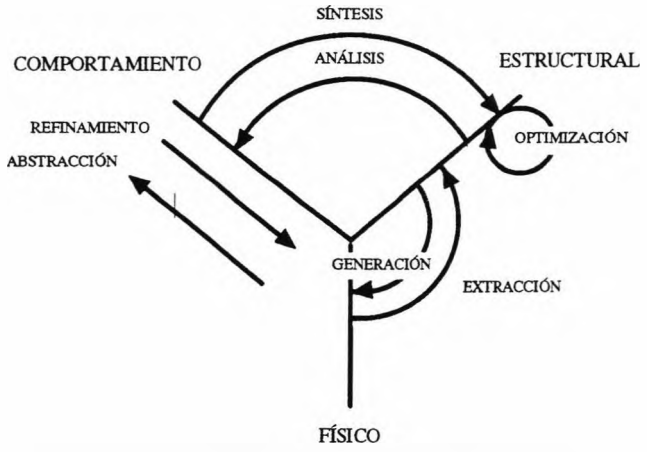


Figura III.6. Definición de algunos procesos básicos, según [24].

Con el diagrama en Y a la vista podemos situar tres niveles de síntesis de interés [26], figura III.7, la síntesis arquitectural o de alto nivel donde se genera una representación estructural de un modelo arquitectural. En segundo lugar, en un grado más detallado de información, la síntesis lógica nos genera una representación estructural mediante puertas lógicas de un modelo lógico. Y la última clase de síntesis, que denominaríamos síntesis geométrica (generación), donde se logra una representación física de la distribución de componentes sobre la superficie del CI en cada una de sus capas y sus posiciones concretas.

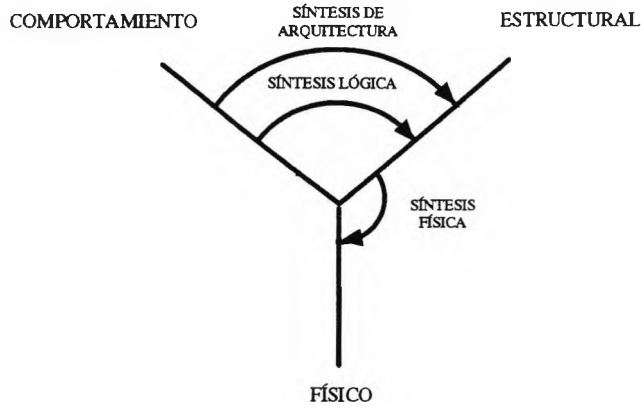


Figura III.7. Tipos básicos de síntesis, según [26].

3. OPTIMIZACIÓN

Es una acción que acompaña a la de síntesis, no solo por el objetivo de obtener la mejor calidad del diseño, sino porque la síntesis sin optimización puede llevarnos a circuitos no competitivos.

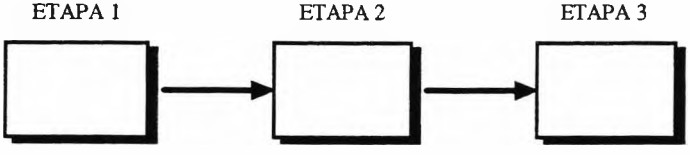
Para evaluar la calidad se acude a estimar el área y las prestaciones, aunque la testabilidad del circuito es otro factor que puede ser decisivo para lograr un diseño de calidad. La optimización busca hacer máxima la calidad del diseño a través de indicadores tales como el área y las prestaciones. El área en particular es un factor extensivo, suma de dos términos, el área de los componentes, y la ocupada por las interconexiones. La primera se calcula conociendo el área de cada elemento en la jerarquía de elementos que conforman el diseño, y la segunda se puede valorar estadísticamente cuando ya se tiene una representación estructural del diseño.

Las prestaciones del circuito son magnitudes intensivas no aditivas, y hay que matizar su significado según el tipo de circuito digital que nos ocupe. Para conocer las prestaciones de un circuito nos ayudamos de ciertos parámetros medibles que permiten comparar diferentes circuitos según una determinada escala de valores (la métrica). Su evaluación exige del conocimiento estructural y del comportamiento de los mismos.

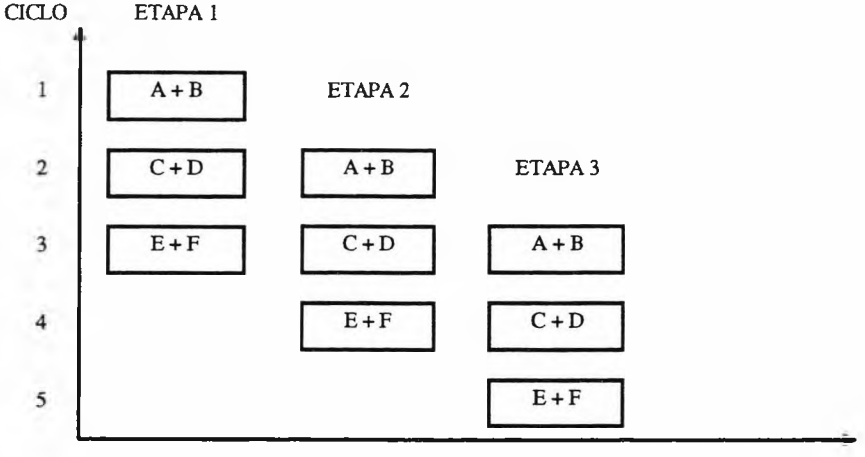
En circuitos combinacionales las prestaciones se evalúan midiendo el tiempo de propagación y los caminos críticos, mientras que en circuitos secuenciales se evalúa el período del reloj más rápido que admitirá el circuito siendo el retraso de la parte combinacional la cota inferior. En circuitos secuenciales ejecutando un conjunto de operaciones las prestaciones se miden por la latencia, o tiempo empleado en ejecutar las operaciones. A menudo esta latencia se expresa a través del número de ciclos de reloj que consume. En circuitos con segmentación, donde se realizan concurrentemente operaciones sobre distintos conjuntos de datos, el rendimiento o tasa entre datos producidos y consumidos es el parámetro a valorar.

El concepto de rendimiento queda mejor descrito si planteamos la siguiente situación comparativa entre dos casos, uno con varias etapas en serie, caso a) y otro con un esquema de segmentación, caso b). Para el caso a) supongamos que se realizan tres operaciones como parte de una tarea y que cada una necesita 5 ns para realizarse.

En el caso b) tenemos también que realizar las tres operaciones con duración de 5 ns cada una y se precisan de registros intermedios para la transferencia entre etapas a cada ciclo de reloj, de forma que van entrando sucesivamente en operación conjuntos de datos distintos (a+b), (c+d) y (e+f) respectivamente, transfiriéndose hacia la siguiente etapa según el esquema mostrado en la figura III.8, de forma que cada 6 ns se comienza una nueva operación si estimamos en 1 ns el tiempo de retraso introducido por los registros interetapas. Así podemos deducir que mientras el esquema directo tipo serie realiza una operación en 15 ns, el circuito segmentado en ese tiempo realiza 15/6=2.5 operaciones, lo cual es indicativo de su aumento de rendimiento y por ende de las prestaciones del circuito que es lo que siempre debemos buscar.



a)



b)

Figura III.8. El rendimiento mejora al pasar de una concepción serie, caso a), a una técnica de segmentación o *pipe-line*, caso b).

Podemos ahora plantear que optimizar el diseño es buscar la combinación de menor área (área < cota superior del área) y mejores prestaciones (prestaciones > cota inferior de las prestaciones). La evaluación del diseño es un problema multidimensional en un espacio definido por los objetivos impuestos, con un conjunto discreto de puntos de optimización producidos por la funciones de evaluación de área y prestaciones, que se interpolan para obtener una curva o superficie de compromiso. Los puntos de ese espacio representan las distintas realizaciones estructurales posibles para el diseño.

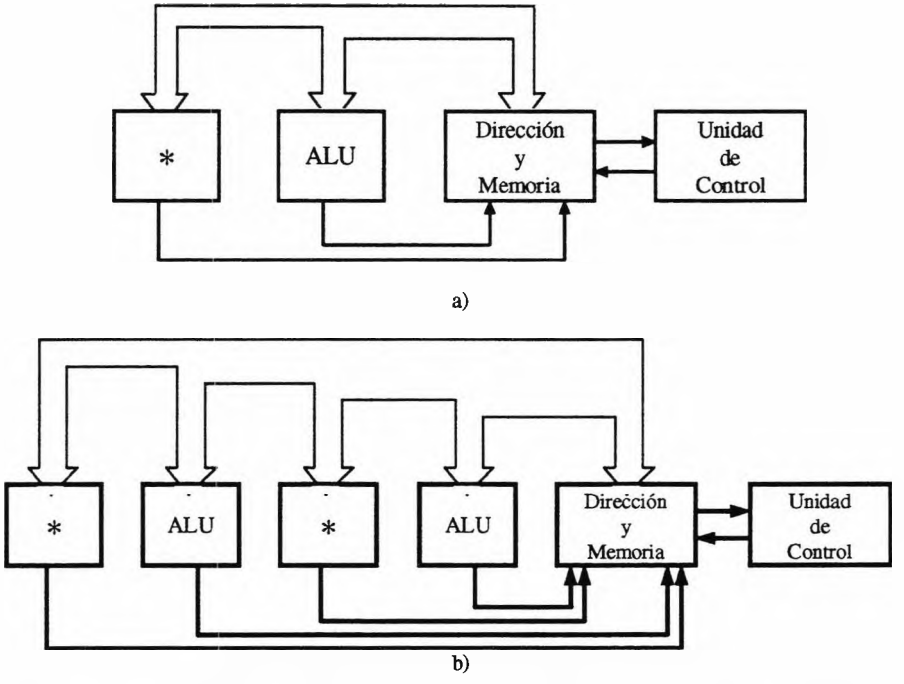


Figura III.9. La disyuntiva ante dos soluciones para resolver un mismo problema implica en el caso a), un número reducido de recursos de circuitería (unidades funcionales), y en el caso b) un número mayor de recursos, penalizando el área y/o el consumo, pero mejorando la velocidad de procesado, adaptado de [26].

Así por ejemplo si pretendemos resolver la ecuación $y'' + 2y' + 5xy = 0$, esto se podría realizar en base a una configuración formada por una ALU y un multiplicador más un bloque de control y otro de almacenamiento en memoria con su lógica de interconexión. Pero también podría ser planteada otra solución que implicase dos unidades del tipo ALU y dos multiplicadores con sus bloques de control, memoria y

lógica de interconexión, según se expresa en la figura III.9. Si consideramos que el área ocupada por la ALU son cinco unidades de área, que el multiplicador ocupa una y las unidades restantes también una unidad de área y si no tenemos en cuenta la unidad de control, la primera solución arroja un computo de área A=7 y latencia L=7. La segunda solución nos da A=13 y L=4. Estos valores junto con otras alternativas se representan gráficamente en la figura III.10 que es la representación del espacio de diseño en dos dimensiones, siendo cada uno de los puntos una solución óptima de las funciones de evaluación. La optimización del circuito es buscar el mejor diseño, es decir, aquella configuración que optimiza todos los objetivos.

La figura III.10 es el espacio de evaluación del diseño de nuestro circuito. En ella se pueden ver cuatro puntos correspondientes a cuatro posibles configuraciones cada una con ciertas características de área y latencia. Vemos que podemos reducir la pareja de puntos (1, 2) y (1, 1) a un solo punto que sería el (1, 1); esto tiene una explicación muy sencilla. Ambos puntos representan a dos configuraciones, las cuales tienen la misma latencia, pero la (1, 2) representa una implementación con mayor área, luego podemos descartarla y quedarnos con la (1, 1). Este tipo de simplificaciones la podremos hacer siempre que tengamos más de un punto sobre una misma vertical o una misma horizontal.

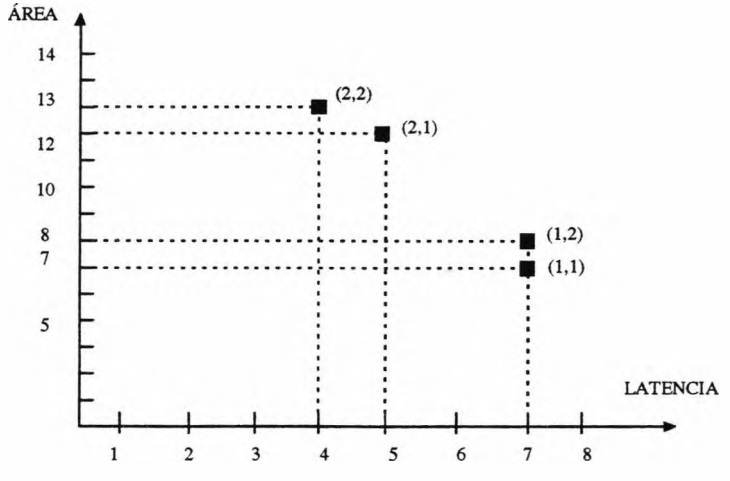


Figura III.10. Evaluación de prestaciones, en este caso área frente a latencia, según [26].

Llamamos puntos "pareto" a aquellos del espacio de evaluación del diseño que son puntos de optimización cuyas condiciones no dependen de los otros, en un espacio monodimensional serían puntos óptimos globales. En nuestro ejemplo, los puntos "pareto" son los siguientes: (1, 1) (2, 1) (2, 2).

Para distintos tipos de circuitos podemos presentar ejemplos ilustrativos de su espacio de diseño. Así con circuitos combinacionales de un tipo fijado, por ejemplo sumadores de n bits, tendríamos una representación del diseño en la figura III.11, en base al área y al retraso como parámetro evaluador de prestaciones. El extremo superior izquierdo representaría la solución mediante sumadores con acarreo adelantado y el extremo inferior derecho la solución en base a sumadores con propagación de acarreo.

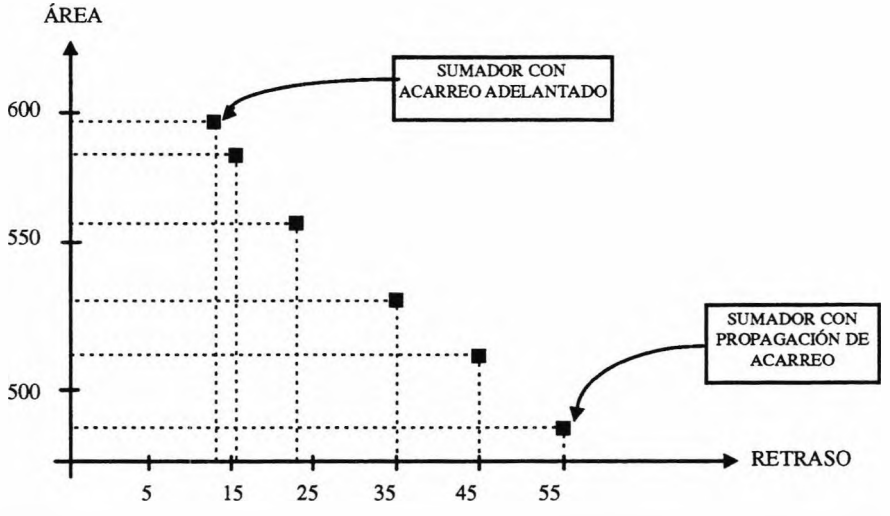


Figura III.11. Evaluación de prestaciones en un tipo prefijado de circuitos combinacionales, realizados con distintas arquitecturas internas, según [26].

En circuitos secuenciales empleando modelos a nivel lógico los objetivos serian área y ciclo (período del reloj más rápido aplicable al circuito), resultando unas curvas de evaluación en el espacio de diseño similares en su aspecto al caso combinacional. Para modelos arquitecturales sin segmentación los objetivos serían área, latencia y

ciclo, mientras que en arquitecturas segmentadas se hace necesario considerar en la evaluación del espacio de diseño el cuarteto área, latencia, ciclo y rendimiento.

El problema de la optimización es difícil de resolver por la naturaleza discreta de las funciones objetivos y del propio espacio de diseño, al estar constituido por la implementaciones realizables del circuito. Por ello sólo se tienen soluciones aproximadas a los puntos de ese espacio. Por ejemplo con modelos a nivel de arquitecturas para circuitos síncronos, las soluciones para distintos valores del ciclo se corresponden con obtener, o bien el mínimo de área ante restricciones de latencia, o bien la mínima latencia ante restricciones de área.

4. FASES EN LA SÍNTESIS DE ALTO NIVEL

El resultado de la síntesis de alto nivel en términos generales es un sistema formado por un camino de datos (*data path*) y una controladora (*controller*) [13], figura III.12. Para alcanzar ese objetivo se parte de una descripción algorítmica, que establece el comportamiento de los módulos, es decir nos indica lo que hacen estos módulos, pero no cómo lo hacen.

Para no dar lugar a confusiones, podemos aclarar ahora que cuando se hace referencia a una *descripción algorítmica* nos referimos a un procedimiento de resolución computacional (totalmente formal y sin ambigüedades). Por otro lado, entendemos por *comportamiento* al conjunto de operaciones y su secuencia de computación. Conviene en este punto matizar que vamos a obtener una representación estructural a partir de un modelo de la arquitectura del sistema, y para ello se ha de concretar la asignación de las funciones a realizar por operadores concretos, que serán elementos de la circuitería genéricamente denominados "recursos". A esta fase se le denomina reserva de recursos (*resource allocation*), para seguidamente establecer la secuencia temporal de operaciones en la fase denominada planificación (*Scheduling*) y por último establecer el enlace con los recursos y las interconexiones, fase denominada enlace (*binding*) [8].

Con estas fases se trata de asignar las variables a registros o señales, figura III.13, los operadores a unidades funcionales y las estructuras de control a los controladores, recorriéndose dos etapas fundamentalmente. En la primera se sitúan las operaciones en el tiempo (planificación) y espacio (enlace), y en la segunda se detallan las interconexiones del camino de datos, especificando la unidad de control a nivel lógico.

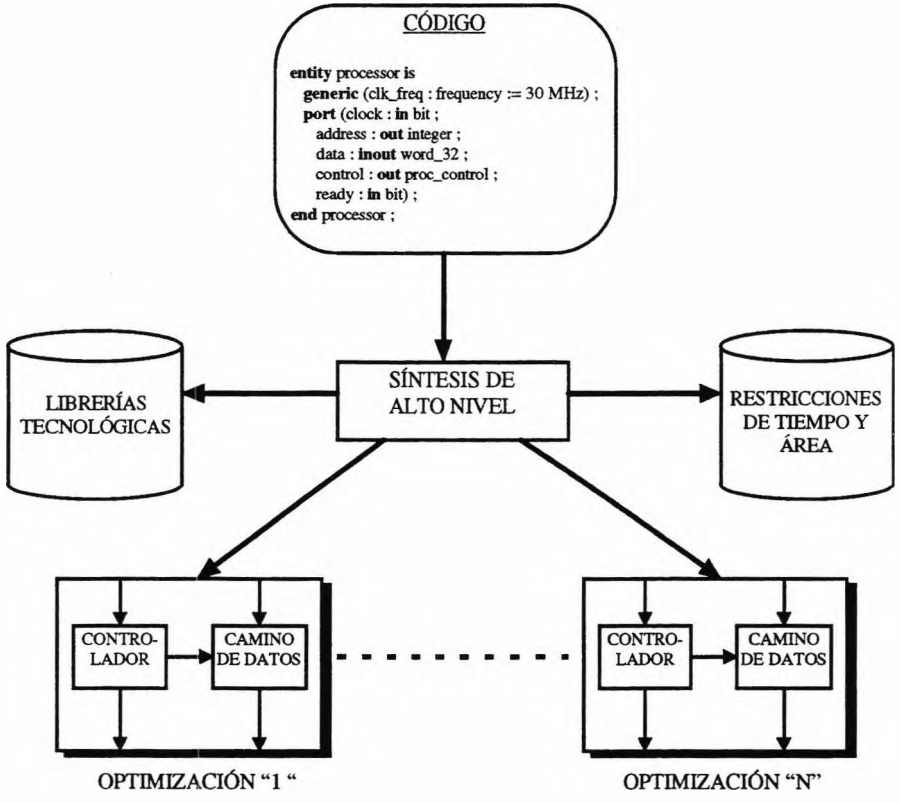


Figura III.12. La entrada en el proceso de síntesis de alto nivel es un código en un lenguaje específico de descripción de los circuitos, y la salida generalmente se compone de una descripción basada en un controlador y un camino de datos.

4.1. Descripción de la entrada

La forma de realizar la descripción de la entrada es la algorítmica del tipo comportamiento. Con este enfoque no se aporta información estructural que nos

indique qué tipo de componentes e interconexiones se van a presentar, qué clase o cuántas secciones de segmentación se incluirán, de cuántas fases es el reloj, etc. Pero la ventaja es que se reduce el tiempo preciso para hacer la descripción con un lenguaje normalizado de descripción de los circuitos (*hardware*) como puede ser VHDL [4,27].

El lenguaje VHDL soporta varios niveles de abstracción y admite asignaciones secuenciales y concurrentes, así como la mayoría de las construcciones típicas de los lenguajes de alto nivel, pero no soporta de forma directa la descripción de secciones de segmentación, el tratamiento de las interrupciones o el comportamiento jerárquico [15], además es un lenguaje pensado de forma nativa para simulación más que para síntesis de alto nivel, por lo cual ésta se hace más difícil si no se adoptan reglas para modelar [7, 35], y se restringe el conjunto de construcciones. Las técnicas de verificación formal son necesarias en este contexto para determinar la validez de descripciones semánticamente equivalentes, pero que pueden dar diseños de distinto grado de calidad según la sintaxis empleada (tipo y orden de las construcciones empleadas). Esto puede ocasionar que diseñadores distintos que describan un mismo sistema, pueden obtener resultados diferentes en cuanto a la estructura (aunque no cabe esperar diferencias en la funcionalidad).

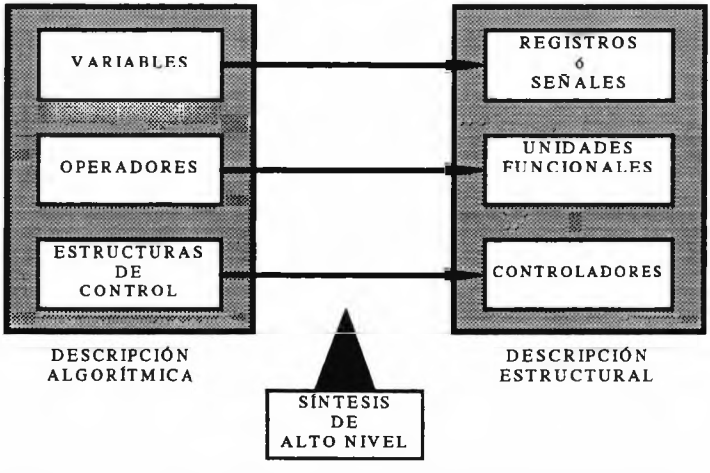


Figura III.13. Correspondencia espacio-temporal entre la descripción algorítmica y estructural en síntesis de alto nivel.

4.2. El modelo de la máquina de estados finitos y datos (FMSD)

En las técnicas de síntesis de alto nivel el modelo subyacente que se adapta a muchas situaciones es el modelo de la máquina de estados finitos y datos (FMSD) introducido en el primer capítulo, el cual es una descripción universal a nivel de transferencias entre registros, válida para diseños digitales dominados por la sección de control. Es el caso de un convertidor serie a paralelo el cual implica una notable sección de control y un sencillo registro de desplazamiento para el camino de datos. También es útil para aquellos diseños dominados por la ruta o camino de datos, como sería el caso de un filtro digital de respuesta finita al impulso (FIR), el cual tiene una mínima unidad de control y un extenso camino para la ruta de datos.

El modelo FMSD, que se muestra en forma esquemática en la figura III.14, incluye en la unidad de control registros de estado, circuitos combinacionales para calcular el próximo estado a través de una función "f" y otra sección combinacional para determinar una función "h", que establece las señales de salida para el control. Tanto las entradas como las salidas de la unidad de control son bits [13].

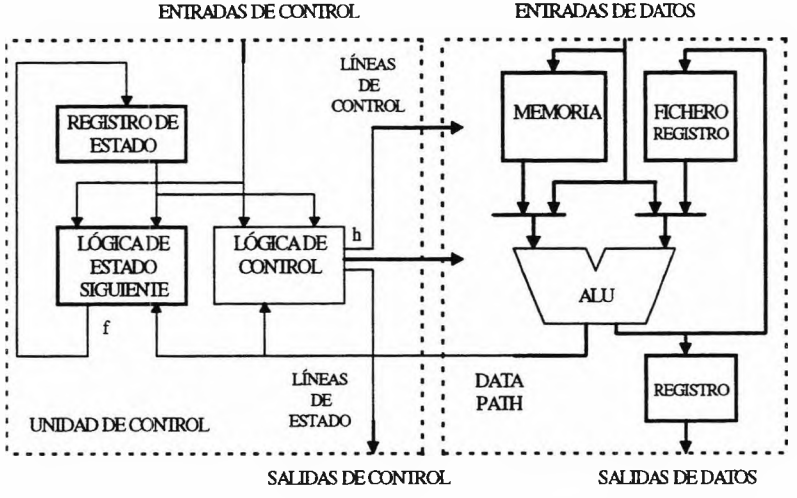


Figura III.14. Modelo de la máquina de estados finitos y datos (FMSD), compuesta de una unidad de control y un camino de datos (*data-path*), según [13].

La unidad que implementa la ruta o camino de datos, incluye unidades de almacenamiento de los tipos registros, contadores, ficheros de registros o memorias, unidades funcionales como ALU´s, desplazadores, Multiplexadores y comparadores y otras unidades de interconexión de los tipos hilos, Buses o Muxes. En este caso tanto las entradas como las salidas del camino de datos son palabras digitales.

Las prestaciones de la FMSD se pueden mejorar con distintos tipos de técnicas de segmentación, que afectan tanto a componentes del tipo de las unidades funcionales, como a la unidad de control o al camino de datos.

4.3. La fase de reserva

Para cada estado del modelo se determina el cómputo a realizar y se establece el número y tipo de unidades (recursos) funcionales en el camino de datos, el esquema de reloj a adoptar, la jerarquización de la memoria y el estilo de segmentación a introducir. Entonces, se hace necesario realizar exploraciones comparativas de optimización (*trade-off*) del costo frente a prestaciones. De esta forma se podría detectar, por ejemplo, si existe un grado de paralelismo inherente a la descripción de alto nivel del comportamiento del sistema que va a implicar mayor número de elementos de circuitería, con el consiguiente aumento del área y en definitiva del costo, pero también con un aumento en la calidad del circuito si valoramos su velocidad. De otra forma, reducir el paralelismo y aumentar la secuencialidad de ciertas operaciones va a redundar en una disminución de los recursos de circuitería, con la correspondiente reducción del área y costo, pero a base de una reducción de la calidad del circuito si miramos su velocidad de funcionamiento.

La métrica para evaluar se aplica a área y prestaciones, basándose, por ejemplo, en contar número de unidades funcionales y pasos de control, o bien basado en considerar modelos de librerías descritos al nivel de detalle de transferencias entre registros (nivel RTL). Así la realización del grafo de flujo de datos (DFG) de la figura III.15, con las cinco elecciones distintas de elementos de esa librería RTL recogidas en

la tabla III.1, hacen físicamente viable el circuito y nos llevaría a la representación de prestaciones frente a área de la figura III.16.

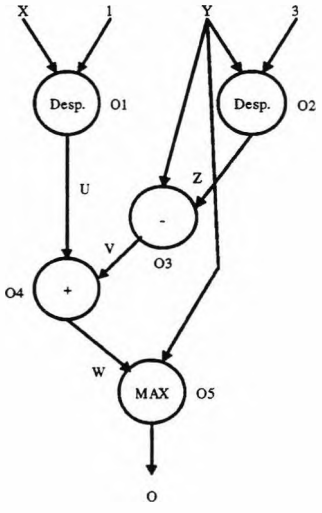


Figura III.15. Grafo de flujo de datos, donde intervienen cuatro operaciones y se establece un orden de precedencia entre ellas.

POSIBLES RESERVAS DE COMPONENTES				
Reserva	**N° de ALU's rápidas**	**N° de ALU's lentas**	**N° de MAX**	**Área**
A	0	1	1	1200
B	1	0	1	1400
C	0	2	1	1600
D	1	1	1	1800
E	2	0	1	2000

Tabla III.1. Evaluación de prestaciones para cinco posibles reservas de recursos que realizan el DFG anterior, a partir de la información disponible para cada unidad funcional.

La elección de la solución a sintetizar depende de la aplicación y se exploran alternativas a partir de la gráfica de la figura III.16 y otras curvas que evalúen prestaciones frente a número de unidades de almacenamiento, número de puertos en las unidades de almacenamiento o número de interconexiones. Al seleccionar el punto

apropiado en esas curvas tendremos concretada la reserva de unidades de cada tipo a realizar para implementar el diseño.

Las herramientas actuales más que una exploración exhaustiva del espacio de diseño, seleccionan tipos de recursos y sus combinaciones, y hacen métricas para determinar el área, predecir el deslizamiento de las señales del reloj, el uso de recursos etcétera, como guía al diseñador para que pueda elegir mejor. Pero el grado de madurez de las herramientas todavía no contempla el hacer consideraciones sobre el dominio de aplicación del sistema en desarrollo, el estilo de segmentación o la topología de interconexión.

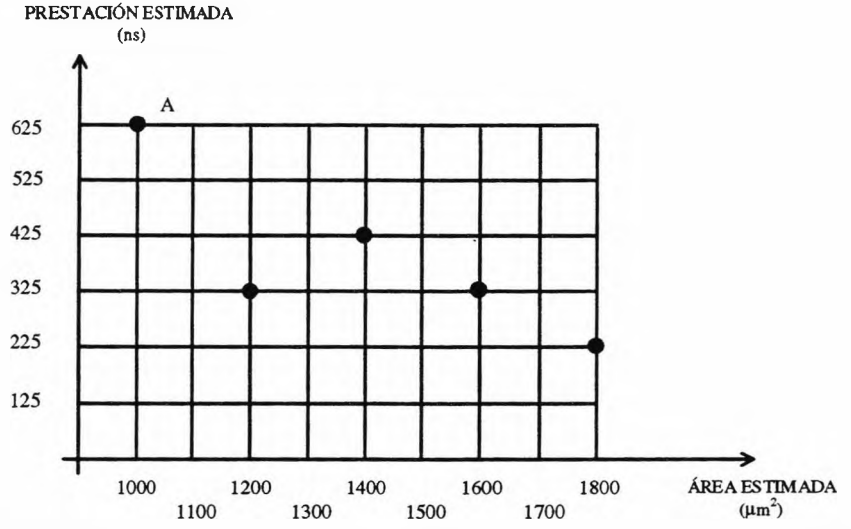


Figura III.16. Prestaciones frente al área para las cinco reservas indicadas en la tabla III.1.

4.4. Algoritmos de planificación

Planificar lo que se está haciendo consiste en particionar la descripción de comportamiento en intervalos de tiempo, estados o pasos de control, organizando en ciclos de reloj las operaciones y accesos a memoria. Pero además se ha de preveer qué operaciones son realizables en paralelo considerando funciones de costo, el tiempo de ejecución permisible y los recursos que se han reservado.

Durante la fase de planificación se convirte una descripción en VHDL, por ejemplo, en un grafo del tipo CDFG. En esta acción se tienen en cuenta criterios como la precedencia de ciertas operaciones o la dependencia entre ciertos datos. Por ejemplo planificar la operación de obtener G=AB+CD+EF, implica disponer previamente de A y B y realizar antes la operación AB y similares antes de obtener G. Es en esta fase donde se hace posible el enlace posterior de cada operación O_i con la unidad funcional k donde pueda ejecutarse la operación.

La planificación y la reserva de recursos están íntimamente ligadas y son interdependientes. No obstante podemos definir distintos tipos de planificación:

- PSR, planificación sin restricciones.
- PRR, planificación restringida por recursos, figura III.17-a donde se aprecia que no hay más de dos unidades funcionales idénticas en cada ciclo de reloj porque existe una restricción de circuitería disponible.
- PRT, planificación restringida por tiempo figura III.17-b donde se aprecia que la planificación alcanzada no dura más de un número determinado de ciclos (4 en este caso), pero pueden aparecer el número de recursos que sean precisos en cada paso de reloj.
- PRRT, la planificación restringida por recursos y tiempos [33].

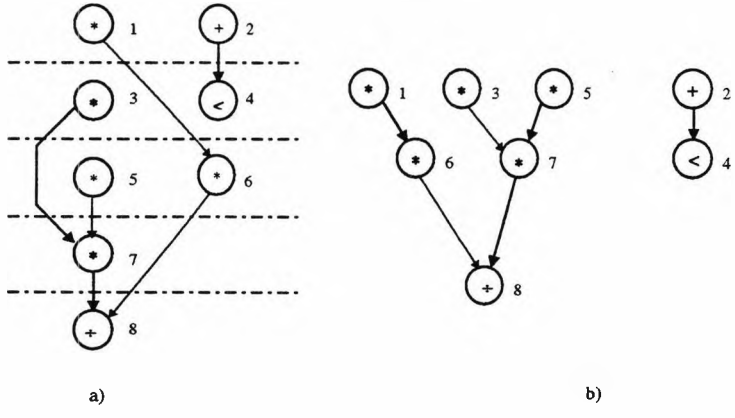


Figura III.17-a y III.17-b. Dos situaciones extremas en la planificación de los recursos: caso a) planificación sin restricciones de tiempo, b) planificación sin restricción en el número de recursos disponibles.

Los principales algoritmos de planificación son:

- ASAP/ALAP.
- Lista planificada.
- Planificación de fuerza directa.
- Programación lineal entera.

Los tres primeros tratan cada operación a planificar por separado de una en una, de forma iterativa, llevando cada operación a su ciclo, de forma que dan una primera evaluación y no abordan el problema de forma global.

ASAP/ALAP es un algoritmo rápido de implementar y ejecutar que resuelve bién el caso de planificación sin restricciones, donde se planifica una operación cada vez, lo antes posible en el algoritmo ASAP (*as soon as possible*), o lo más tarde posible en ALAP (*as late as possible*). La latencia, es el número de ciclos precisos para ejecutar toda la planificación, y no varía al aplicar el algoritmo de planificación ALAP. La descripción en forma de pseudo-código para el caso ASAP es la que se muestra a continuación:

```
FOR cada Oi
      IF Oi no tiene predecesor inmediato
            cstep (Oi)=1
      ELSE
            cstep (Oi)= máximo cstep de las operaciones
            inmediatas predecesoras + 1
```

El resultado de los algoritmos ASAP y ALAP lo podemos apreciar en la figura III.18-a y 18-b.

La escritura del pseudo-código para el caso de la planificación siguiendo el algorítmo ALAP es semejante.

El tercer algoritmo a considerar sería la lista planificada. Aquí se procesa cada paso de control secuencialmente (en vez de cada operación) y se elige la operación más apropiada según las ligaduras establecidas. Entonces se ordena la lista de operaciones disponibles en cada paso de control con un criterio, que tiene en cuenta la probabilidad de aumento de la longitud total de la planificación al introducir una u otra de las operaciones a planificar. Para ello un criterio es por ejemplo, la movilidad de las operaciones, de forma que a menor movilidad mayor prioridad. La lista se puede

actualizar cada vez que se elige una operación y se pueden mantener listas separadas para cada unidad funcional.

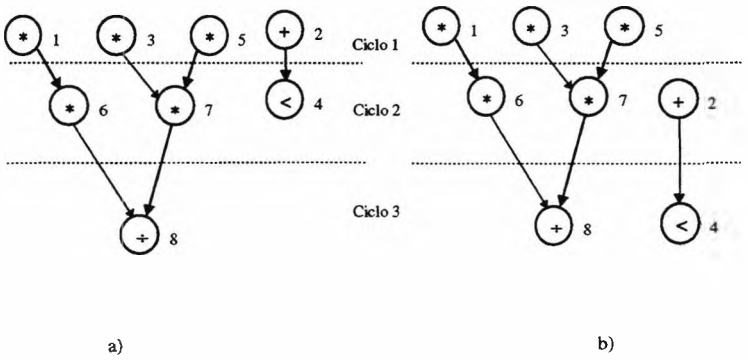


Figura III.18. Resultado de una planificación de operaciones: a) Según algoritmo ASAP. b) Según algoritmo ALAP.

Este algoritmo se adapta bién al caso de planificación restringida por recursos y selecciona globalmente la siguiente operación a planificar de entre el conjunto de las pendientes de planificar. Es un algoritmo computacionalmente más eficiente que ASAP/ALAP. Queda descrito en pseudo-código en la siguiente forma:

```
cstep actual=0
WHILE hay operaciones sin planificar
      cstep actual = cstep actual + 1
            colocar operaciones disponibles en la lista
            evaluar la prioridad de cada operación
            ordenar la lista por prioridad
      WHILE hay operaciones en la lista que verifiquen
            ligaduras
            elegir la operación Oi de mayor prioridad
            cstep (Oi)=cstep actual
```

El algoritmo de fuerza directa, da buenos resultados en el caso de planificación restringida por tiempo, hace mínimo el número de unidades funcionales y las distribuye lo más uniformemente posible. La selección de la operación que ha de ocupar un paso

concreto de la planificación, se realiza utilizando el concepto de fuerza para evaluar una función de coste.

El algoritmo se encarga de calcular el aumento del coste, al asignar la unidad funcional que se trate a cada uno de los pasos de su intervalo de ubicación, eligiendo el de menor coste. La función de coste es proporcional a la suma de un conjunto de fuerzas directas e indirectas, que por analogía mecánica informan de la atracción o repulsión producida al ubicar una operación en un paso concreto de la planificación, así como de su incidencia en el resto de las operaciones pendientes de planificar

La ventaja del algoritmo de planificación lineal entera (ILP), es la calidad de la solución alcanzada en los casos de PRT, PRR y PRRT, y que se planifican todas las operaciones de forma global, pero el dilema es el tiempo de computación para lograr esa calidad.

Con algoritmos diseñados cuidadosamente y técnicas de acotación se pueden abordar circuitos pequeños y medianos en tiempos razonables, además hay paquetes de *software* generales para resolver la planificación con estos algoritmos ILP, siendo factible añadirles restricciones y extensiones (por ejemplo, segmentación).

En un problema de aplicación de ILP, se trata de hacer mínima o máxima una función objetivo de varias variables, sujeta a un conjunto de restricciones mediante ecuaciones e inecuaciones lineales, con condiciones de integrabilidad en todas las variables. Para el caso PRT, por ejemplo, se establece la formulación del algoritmo buscando el mínimo de una función que permite hallar el número de unidades funcionales de cada tipo.

Existen otros muchos algoritmos como por ejemplo, la compartición de recursos, éste es un algoritmo avanzado que minimiza los pasos de control y coste de la circuitería, siendo especialmente útil en grafos con ramificaciones condicionales anidadas. Se fomenta compartir recursos y es aplicable al caso PRR. Se inicia en los bucles internos y progresa hacia los externos. Obtiene un grafo equivalente sin ramificaciones y entonces, lo planifica. A partir de aquí, deduce una planificación para el grafo inicial planificado.

Los recursos se pueden compartir de forma incondicional, lo que significa tener operaciones planificadas en pasos diferentes compartiendo los mismos recursos. Pero

también de forma condicional, lo que significa tener operaciones planificadas en ramas condicionales mutuamente excluyente que comparten los mismos recursos.

4.5. Fase de enlace

El enlace se plantea al implementar físicamente el camino de datos. En este proceso para cada paso de reloj se enlazan las operaciones y los accesos a memoria con los recursos *hardware* disponibles. Se establecen además, en cada estado, las siguientes correspondencias: se asignan las variables a las unidades funcionales, las transferencias de entrada y salida de datos a unidades genéricas, y las transferencias entre estas unidades a unidades de interconexión.

Existen tres tipos de enlaces posibles: enlaces de unidades de almacenamiento, enlaces de unidades funcionales y enlaces de interconexión. En cualquier caso un recurso puede ser compartido por diferentes operaciones y accesos o transferencias de datos, a condición de ser asignados a distintos pasos de control.

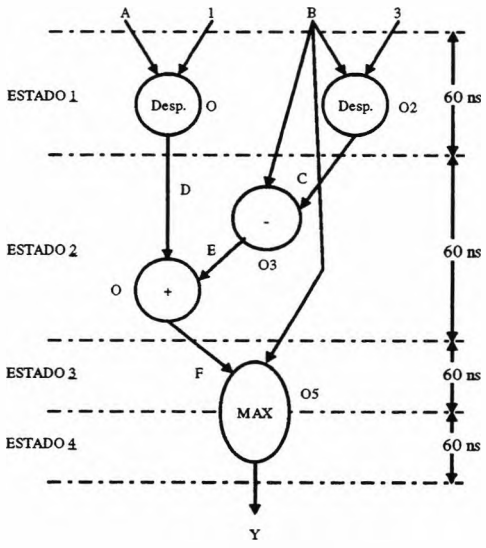


Figura III.19. Planificación para el DFG de la fig. 15, donde se ha tenido en cuenta la ubicación de unas operaciones en un mismo ciclo, y el caso de operaciones multiciclo, según los requerimientos de tiempo de cada operación, adaptado de [13].

El enlace de almacenamiento asigna variables a unidades de almacenamiento (registros, ficheros de registros, unidades de memoria), excepto las variables del exterior, que no se asignan a estas unidades. Este enlace se basa en hallar particiones de variables compatibles, es decir, que no estén activas simultáneamente, pues en este caso no se pueden enviar a la misma unidad de almacenamiento.

Este enlace se realiza en tres pasos, que comentamos tomando como ejemplo la planificación mostrada en la figura III.19.

1. En primer lugar se realiza el esquema de duración de la actividad de las variables, es decir, un diagrama, figura III.20-a, que recoja cuánto tiempo y en qué períodos del mismo está activa cada variable. A y B no son consideradas porque son variables externas (puestas en juego desde el exterior), y éstas no se asignan a las unidades de almacenamiento, como ya hemos comentado más arriba. Por otro lado, E es un simple hilo no necesita registro, puesto que no está en la frontera de estados, es decir, no está en el paso de un estado a otro, como pasa con C o D entre los estados 1 y 2, sino que "trabaja" siempre en el mismo estado y por ello no es necesario su almacenamiento, por lo que puede implementarse con un hilo de conexión.
2. Obtener el grafo de compatibilidad según se muestra en la figura III.20-b. Cada nudo representa una variable, y los arcos unen variables de vida mutuamente excluyentes, es decir, cuyos períodos de duración no coinciden en ningún momento. Así, C y D son escritas en el estado 1 y leídas en el 2; E es escrita y leída en el estado 2; y F es escrita en el estado 2 y leída en los estados 3 y 4.
3. Por último, obtener las particiones de nudos mutuamente excluyentes (llamadas *cliques*), indicadas en la figura III.20-c.

Cada partición obtenida se puede enlazar a un mismo recurso *hardware*. Hemos obtenido dos particiones, por lo que tenemos dos soluciones al problema, cada una asignando las variables a dos registros. La solución 1 asigna al registro 1 las variables C y F, y al registro 2 la variable D, mientras que la solución 2 asigna D y F al registro 1 y C al registro 2.

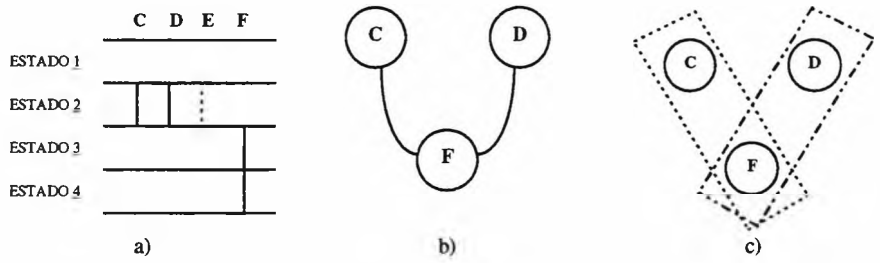
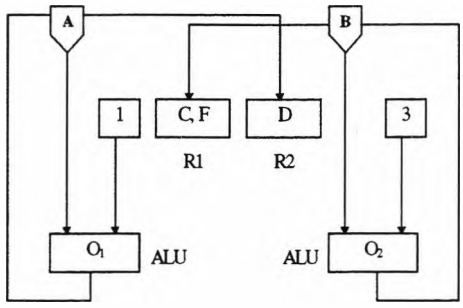


Figura III.20. Fases para obtener particiones. a) Diagrama de longevidad de las variables, b) grafo de compatibilidad, c) particiones o *cliques*.

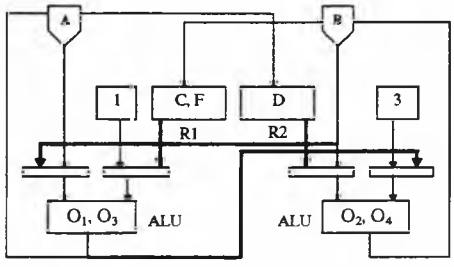
En resumen, lo que hemos hecho en estas tres fases es realizar, una vez obtenidos los ciclos que ocupa la actividad de cada variable, un grafo de compatibilidad, sobre el cual se definen subgrafos completos (*cliques*), en los que cada par de nudos están unidos por una rama. Los *cliques* establecen particiones mutuamente excluyentes (soluciones que descartan a las demás), elegiremos la partición o solución que mejor satisfaga nuestras necesidades.

El enlace a unidades funcionales asigna operaciones en cada estado, a las unidades funcionales seleccionadas. Esto se realiza considerando uno por uno todos los estados. Tenemos un ejemplo en la figura III.21. En (a) tenemos el diseño parcial, justo tras enlazar las operaciones del estado 1. Una vez enlazadas las operaciones del estado 2 tenemos dos posibles soluciones (b) y (c). En la primera, adicionalmente a la circuitería implementada tras el primer ciclo de operaciones, se necesitan cuatro multiplexores de dos entradas u ocho dispositivos triestado. La segunda solución sólo requiere de tres multiplexores de dos entradas o seis dispositivos triestado. Esta última alternativa es claramente mejor que la primera solución propuesta.

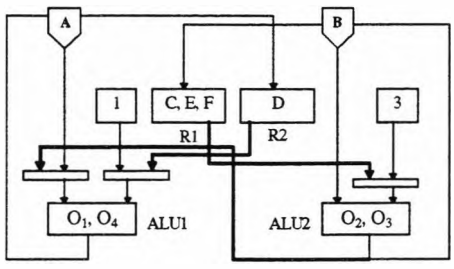
Existen otros algoritmos que permiten almacenar variables en estructuras regulares, del tipo de ficheros de registros o memorias de N puertos. Si existieran variables de conjunto (*array* variables) en nuestro diseño, se enlazarían a módulos de memoria de igual dimensión que estas variables de conjunto, aunque algunos algoritmos realizan *clusters* de variables de conjunto sobre un único módulo de memoria.



a)



b)



c)

Figura III.21. Enlace a unidades funcionales: a) Resultado tras el estado 1, b) y c) dos alternativas correspondientes a las dos particiones obtenidas en la fig. 20. Adaptado de [13].

5. SÍNTESIS DE TESTABILIDAD

La evolución seguida por las técnicas de síntesis tienen su impacto al considerar los aspectos del test de un circuito como una tarea más a tener presente en el

planteamiento de un diseño ya desde la concepción del mismo. Ésta ha sido la tendencia manifestada desde comienzo de la década de los noventa, donde la incorporación de recursos para el test cada vez se presenta desde una fase más temprana del desarrollo de un sistema y a menudo entra en el propio conjunto de especificaciones iniciales.

Para algunos, la mejor medida de la calidad de un producto es la detección de fallos no sólo antes de la entrada en servicio sino lo antes posible, porque está ampliamente aceptada la regla de que el coste de un fallo se multiplica por diez al pasar de la fase de diseño al prototipo, tabla III.2, y se sigue afectando por este factor en los sucesivos pasos a producción, encapsulado y puesta en servicio.

REPERCUSIÓN COMERCIAL DE UN FALLO	
Nivel	**Coste en unidades monetarias arbitrarias**
Diseño	1
Prototipo	10
Producción	100
Encapsulado	1000
En servicio	10000

Tabla III.2. Repercusión relativa de la detección de un fallo según la fase en la que se ha detectado.

Hay que tener en cuenta de que un C.I. puede cumplir perfectamente con sus especificaciones, pero no dar los resultados esperados. Esto podría deberse a un fallo en la propia fase de definición y/o captura de especificaciones, por lo que no deberíamos considerar en tal caso, que fuese un fallo del C.I. Más bien se trataría de un fallo que comenzó al principio del diseño y se fue deslizando hasta el final del desarrollo. Éste es el motivo de que haya que estratificar el test en diferentes niveles para no correr el riesgo de que un fallo en el nivel de especificación funcional llegue a la fase de implementación física, donde el coste puede no ser tolerable.

Con el continuo crecimiento de la complejidad de los sistemas VLSI, el test es un factor más crítico, por ello se hace necesario encontrar un equilibrio entre el coste del *hardware* adicional y disminución del coste total al abaratar el procedimiento de test [31], figura III.22. Sin embargo, la concepción jerárquica de los sistemas plantea

requisitos distintos para el test al recorrer los distintos niveles que van desde los componentes a las tarjetas de circuito impreso y de aquí a las unidades o subsistemas y finalmente al sistema completo. Por ello se han desarrollado técnicas en cada nivel [31] y hay un esfuerzo en la actualidad de elaborar procedimientos de test multinivel mediante las normas de la familia 1149.x que va elaborando el IEEE para el campo digital, el analógico y el modo de señales mezcladas analógico/digital.

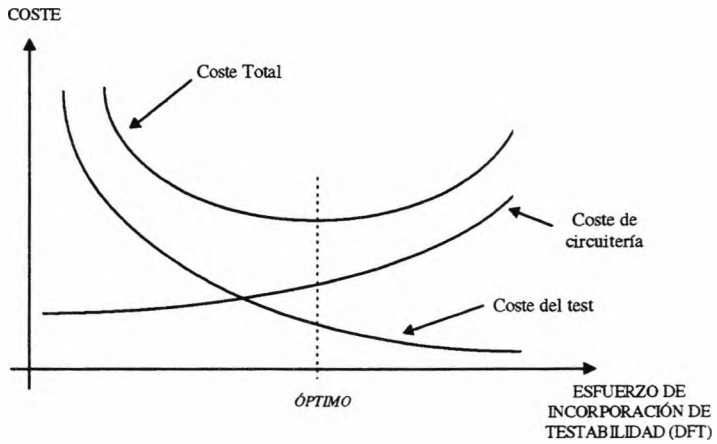


Figura III.22. Coste de la circuitería adicional, en relación al coste total del test.

En el nivel de componentes, el test se encarga de verificar si un circuito integrado cumple sus especificaciones y ha de responder a cuestiones básicas como, ¿funciona el diseño?, ¿esconde algún problema potencial?, ¿tiene completa la funcionalidad esperada?. Las alternativas al test a este nivel pueden venir desde el exterior del *chip*, mediante técnicas de generación de patrones de test (TPG) o conjunto de estímulos y respuesta esperada. Pero pueden ser planteadas desde el interior del propio circuito y esto es lo que ha dado lugar a una disciplina que desde hace algún tiempo se le llama diseño para la testabilidad (DFT), mediante alguna de las técnicas conocidas como técnicas *ad hoc*, técnicas estructuradas y técnicas de autotestado interno (BIST) [23,1].

Para llevar a cabo un diseño con mayor testabilidad se aplican las técnicas de síntesis de alto nivel desde que se realiza la descripción de comportamiento del circuito

añadiendo circuitería específica de test mediante alguna de las técnicas antes mencionadas, para lo cual existen herramientas CAD en el mercado [2]. De esta forma al comienzo del ciclo de diseño, figura III.23, se añade la testabilidad como una restricción más del circuito aumentando el conjunto de las más convencionales como área, consumo y velocidad. El flujo de trabajo seria el siguiente.

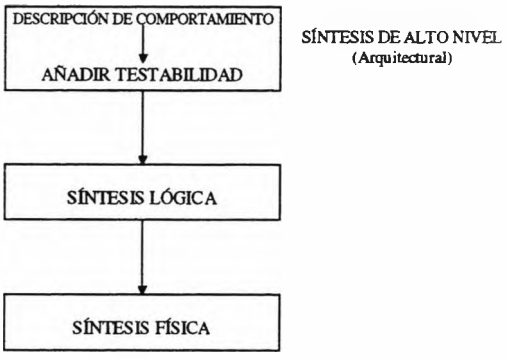


Figura III.23. La incorporación de recursos para facilitar el test se adelanta cada vez más en el ciclo de diseño.

Aumentar la testabilidad significa ganar en facilidad de realización del test, desde el punto de vista del coste y por tanto diseñar para la testabilidad es realizar un esfuerzo deliberado que asegure la testabilidad del circuito mejorando factores como la controlabilidad y observabilidad, que indican la habilidad para manejar el flujo de señal de un circuito y la medida del margen en que se puede monitorizar la actividad de una señal. A los anteriores objetivos contribuye fuertemente el grado de particionado del sistema, es decir la facilidad de reducir un circuito complejo a un conjunto de subcircuitos mínimamente interdependientes y más fácilmente testable.

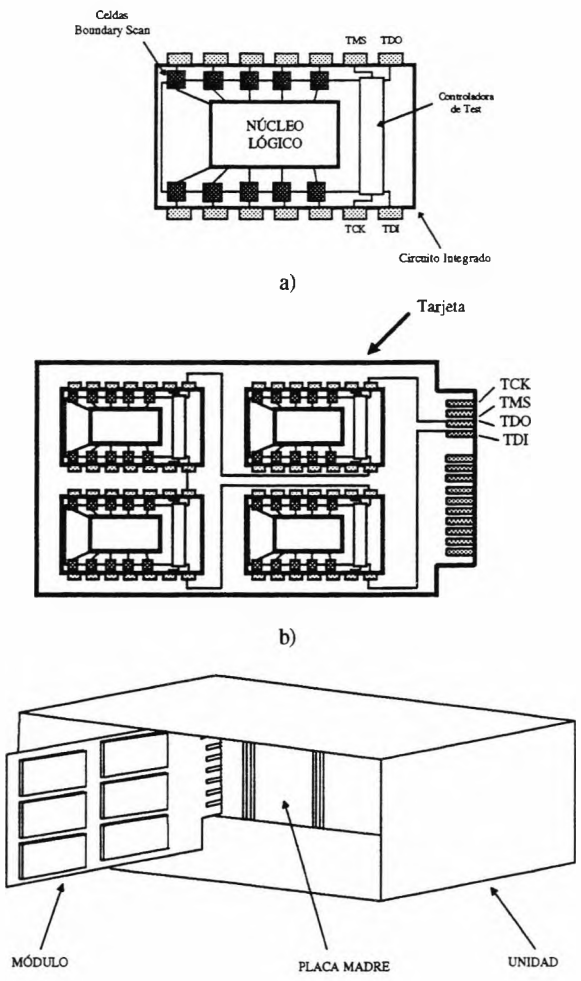


Figura III.24. Un equipo o unidad electrónica consta de varias tarjetas, y en ellas se disponen: a) Componentes adaptados a la norma de rastreo periférico, que facilitan su test, el de la tarjeta ,caso b) y por extensión todo el equipo, caso c).

En el momento actual los elementos más complejos que se realizan son los *chips* multimódulos (MCM). Para ellos las técnicas de BIST y rastreo periférico (*Boundary Scan*) son las soluciones por las que se ha inclinado la industria de los semiconductores. Ambas corresponden a la categoría de técnicas estructuradas referidas previamente.

En el plano de las tarjetas de circuitos impresos, tradicionalmente se realizaba su test mediante conectores periféricos de inserción o mediante medidas eléctricas en la denominada cama de pinchos, pero la situación actual de circuitos impreso multicapas con componentes en ambas caras a menudo conectados mediante técnica de montaje superficial, ha obligado a adoptar un bus test en la tarjeta con pines dedicados a esta misión.

La tarjeta que adoptan la técnica de test de rastreo periférico, incorpora componentes que considerados aisladamente han de cumplir la norma, así hacen posible una mayor controlabilidad y observabilidad al poder enlazarse formando una cadena entre ellos que recorre todos los elementos de la placa. Esta es la idea de las normas de la familia 1149.x, figura III.24, que además se están extendiendo al bus de sistema de varias placas y a los campos de señales mezcladas analógico-digitales.

6. PROSPECTIVA EN SÍNTESIS.

La mayor parte del esfuerzo realizado para automatizar la traslación y optimización (síntesis) de una descripción funcional del comportamiento [18] de un sistema electrónico a la representación geométrica e implementación en VLSI se ha descompuesto en pasos, en cierto grado solapados [21], que en la figura III.25 se representan en forma de círculos concéntricas, entre las que distinguimos tres técnicas específicas de la automatización y síntesis de los diseños digitales (CAD), la síntesis de alto nivel, la síntesis lógica y la compilación de silicio. Las tres en conjunto realizan el ambicioso objetivo de convertir un algoritmo en una implementación física.

En la síntesis de alto nivel, como hemos visto se traslada la descripción del comportamiento y su algoritmo en una descripción microarquitectural en el nivel RTL. La síntesis lógica traslada la descripción RTL a una descripción Booleana, y la compilación de silicio se encarga de la síntesis física, y la convierte en una descripción del *layout* o en un fichero de configuración de dispositivos programables.

Antes de ver la evolución y perspectivas de futuro de las herramientas CAD de síntesis, vamos a ver qué tipos de clasificaciones podemos realizar de estas herramientas:

- Según su origen: si es un producto comercial o universitario.
- Según nivel y campo de aplicación: si es de un nivel o multinivel de abstracción, y si su uso está restringido a un campo especializado (por ejemplo, al campo de los DSP).
- Según la calidad del diseño, medido con respecto a área, prestaciones y testabilidad, y según su calidad de trabajo en tareas concretas.
- Según su facilidad de acceso a datos internos del entorno y de incorporación de aplicaciones.

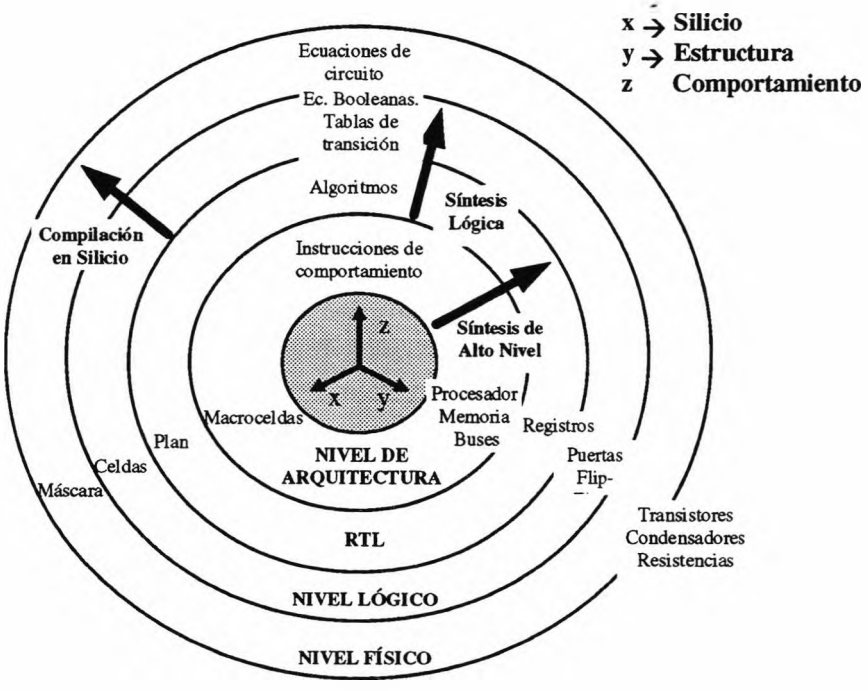


Figura III.25. Solapamiento entre niveles de síntesis.

A continuación mostramos dos tablas que incluyen algunas herramientas de síntesis, que existen en la actualidad. La tabla III.3, de herramientas comerciales cuyo mercado de 1991 a 1995 pasó de 150 a 650 millones de dólares, y la tabla III.4, de herramientas obtenidas fruto de la investigación, ya sea universitaria o bien programas de CAD privados propiedad de empresas como IBM o Bell Labs, que en términos generales no se comercializan y son de uso interno.

HERRAMIENTAS CAD COMERCIALES		
Fabricante	**Sistema**	**Características principales**
Cadence Design Systems	SYNERGY	Síntesis desde VHDL y Verilog. Síntesis lógica y optimización. Enlaces de librerías.
Compass	ASIC SYNTHESIZER	Síntesis separada de data path y control desde VHDL, Verilog y entradas gráficas. Compartición de recursos. Síntesis lógica y optimización. Enlaces de librerías.
Dazix/Intergraph	ARCHSYN	Síntesis desde VHDL y Verilog. Compartición de recursos. Síntesis lógica y optimización. Enlaces de librerías.
Exemplar Logic	CORE	Síntesis desde VHDL y Verilog. Optimización lógica y enlace para FPGAs.
Mentor Graphics	AUTOLOGIC	Síntesis desde VHDL y M. Síntesis lógica y optimización. Enlaces de librerías.
Synopsys	HDL/DESIGN COMPILER DESIGN WARE	Síntesis desde VHDL y Verilog. Compartición de recursos. Síntesis lógica y optimización. Enlaces de librerías.
Viewlogic	SILOSYN VIEW SYNTHESIS	Síntesis desde VHDL. Compartición de recursos. Síntesis de control para bucles. Síntesis lógica y optimización. Enlaces de librerías.

Tabla III.3. Algunas herramientas comerciales de síntesis con indicación del tipo de síntesis y lenguajes HDL que aceptan.

Muy probablemente, la próxima generación de herramientas incorpore la captura de especificaciónes para llegar a la síntesis de arquitectura a partir de descripciones HDL, con procesos de optimización frente a un conjunto de parámetros. Las técnicas de verificación formal se encargarán de comprobar la equivalencia entre dos sistemas descritos mediante HDL y la mayoría de las arquitecturas se sintetizarán

con un camino de datos y su correspondiente bloque de control como ya vimos, forzando en el camino de datos el paralelismo, o la complejidad en la sección de control, según el caso.

En la actualidad se desarrollan herramientas y métodos especializadas en ciertos campos de aplicación muy específicos, como puede ser el prototipaje rápido de procesadores [22] y el campo de los procesadores digitales de señal, DSP. No es fácil encontrar herramientas adecuadas para cualquier propósito. Tal especialización nos lleva a conseguir en la síntesis de arquitectura una reducción del tiempo de diseño de hasta un orden de magnitud. El problema reside en la dificultad de integración de estas herramientas en los entornos de diseño comerciales [3]. La búsqueda de soluciones a este problema corre a cargo de las universidades de forma mayoritaria, habiéndose conseguido gracias a estos esfuerzos algunos "compiladores de comportamiento". Éstos permiten explorar el espacio de diseño de los circuitos de forma automática para conocer las especificaciones área-prestaciones y obtener las curvas de compromiso.

HERRAMIENTAS CAD UNIVERSITARIAS			
Organización	**Sistema**	**Entrada**	**Aplicaciones***
AT&T	BRIDGE CHARM	HDL2	SAO
CMU	SAW	Verilog,ISPS	SAO
IBM	HIS	VHDL	SAO
IMAG	ASYL	FSM, networks	SLO
IMEC	CATHEDRAL IV	Silage	SAO, MG
INPG	SOLAR	Silage, VHDL	SAO, SS
Philips	PYRAMID PHIDEO	Silage	SAO, MG
Princeton University	PUBSS	FSM	SAO, SLO
Stanford University	OLYMPUS	*HardwareC*	SAO, SLO
U.C.Berkeley	SIS	FSM, networks, BDS	SLO
U.C.Irvine	SIERA	VHDL,Verilog	ASO, SS
U.C.Boulder	BOLD	FSM, networks	SLO
U.C.Irvine	VSS	VHDL	SAO
U.Karlsruhe/Siemens	CADDY CALLAS	DSL, VHDL	SAO
USC	ADAM	SLIDE, DDS	SAO

* SAO: Síntesis de Arquitectura y Optimización.
SLO: Síntesis Lógica, Optimización y enlace de recursos.
SS : Síntesis de Sistema
GM : Generación de Módulos.

Tabla III.4. Algunas herramientas no comercializadas, fruto de desarrollo en centros de investigación y para uso interno de casas comerciales.

Los "compiladores de comportamiento" optimizan el diseño basándose en restricciones especificadas previamente, y llevan a cabo detallados análisis de estas restricciones, informan de conflictos entre estas restricciones antes de llevar a cabo la síntesis y muestran el estado tras la síntesis de acuerdo con las restricciones impuestas (área, prestaciones, retardos).

Para evaluar el éxito de una herramienta CAD de síntesis nos remitimos al uso que encuentra dicha herramienta en distintas aplicaciones. Los algoritmos que prevalecen son los que, siendo más simples, resuelven una mayor cantidad de problemas. Además de estas puntualizaciones, el éxito de una herramienta queda determinada por su interfaz con el usuario, su base de datos y su adaptabilidad a las necesidades del usuario. Con respecto a esto existen actualmente ciertas tendencias, como la adopción de HDL's como herramientas de diseño, el uso de formatos de diseño comerciales o la introducción de técnicas de verificación formal, entre otras.

Debido a que estas herramientas todavía son "jóvenes", aún quedan muchos problemas por resolver, como por ejemplo:

- La posibilidad de realizar una síntesis eficiente de circuitos segmentados y de memorias jerarquizadas.
- La interactividad, porque la comunidad de diseño no acepta diseños automáticos, la herramienta debe ser abierta a la interactividad para que el diseñador pueda controlar el proceso, tomar decisiones e introducir reglas.
- Conseguir una conexión entre síntesis de alto nivel y síntesis geométrica. El diseñador debe conocer las implicaciones para el *layout* de las decisiones arquitecturales tomadas en la síntesis de alto nivel, por ello las herramientas de síntesis de alto nivel que permitan la interacción con las herramientas lógicas y de *layout* y den una estimación exacta, mejorarán la calidad de los diseños sintetizados.
- Desarrollar nuevos modelos y métodos de síntesis para circuitos mixtos y circuitos con partes síncronas y asíncronas.

Aunque muchos de estos escollos aún no han podido superarse, se han propuesto y resuelto otros muchos retos en la investigación y el desarrollo de estas herramientas, sobre todo por parte de la comunidad investigadora universitaria (como

es el caso de CHOP [19], SIERA [30,29] o AMICAL), y goza en este campo de una mayor iniciativa sobre productos comerciales ya maduros, que en 1996 se han aproximado a un volumen de ventas próximo a los mil millones de pesetas con cuatro grandes casas, Synopsys, Cadence, Mentor Graphics y Viewlogic, acaparando el 79% del mercado.

La alternativa del codiseño *hardware/software* [34,25] persigue la obtención de sistemas empotrados [14] donde un equipo *hardware* adaptado al problema en cuestión sea mejorado mediante la programación del *software* de sus módulos. Además, resulta posible actualizar el sistema de forma más fácil gracias a la parte *software* del mismo, y a la facilidad de desarrollo de prototipos *software* que acelerarían el desarrollo del producto [10]. No obstante son múltiples los retos planteados como por ejemplo:

- Desarrollar herramientas para particiones, para acoplo *hardware/software* y para captura de especificaciones.
- Disponer de modelos abstractos, que sean mejores que los modelos actuales y más fáciles de mantener, además de desarrollar lenguajes que permitan expresarlos.
- Extender las técnicas de verificación formal al campo de estos modelos abstractos.
- Disponer de evaluadores de coste y prestaciones.

A pocos sorprenderá que estos retos se sobrepasen con éxito, gracias a la colaboración entre ingenieros de *software* y de diseño electrónico. El recorrido a través de esta monografía es una invitación, hacia un campo donde la innovación tecnológica es incesante, y nos anticipa lo que podrá ser una realidad industrial en un tiempo breve.

Los especialistas en diseño electrónico saben que siempre tienen una revolución pendiente, han visto como numerosos avances del campo digital se han aplicado a las técnicas analógicas, y observan ahora la llegada revolucionaria de los microsistemas, cuya naturaleza física no exclusivamente eléctrica requerirá de equipos de trabajo multidisciplinares para el desarrollo de las ideas.

REFERENCIAS BIBLIOGRÁFICAS

[1] Abramovici, M., Breuer, M.A., Freidman, A.D. *Digital systems testing and testable design.* Computer Science Press 90, ISBN 0716781794

[2] Aitken, R.C. "An overview of test synthesys tools". *IEEE Design and test of Computers*, Vol 12, n° 2, Summer 1995, pp 8-15.

[3] Barnes, T., Harrison, D., Newton, A.R., Spickelmier, R.L. *Electronic CAD frameworks.* Kluwer 92, ISBN 0792392523.

[4] Bhasker, J. *A VHDL primer.* Prentice Hall 95, ISBN 0131814478.

[5] Birmingham, V.P., Gupta, A.P. Siewiorek, D.P. *Automating the Design of Computer System: The Micom Project.* Boston, MA: Jones and Barlett, Feb. 1992.

[6] Brown, S., Rose, J. "FPGA and CPLD architectures: A tutorial". *IEEE Design and test of Computers*, Vol 13, N° 2, Summer 1996, pp 42-57.

[7] Camposano, R., Saunders, L.F., Tabet, R.M. "VHDL as input for high-level synthesis". *IEEE Design and test of Computers*, Vol 8, N° 2, March 1991, pp 43-49.

[8] Carlson, S., Girczyc, E. "Understanding synthesys begins with knowing the terminology". *EDN* Sept. 3, vol.1992, pp. 125-131.

[9] Chiodo, N., Giusto, P., Hsieh, H., Jurecrka, A., Lavagno, L., Sangiovanni-Vicentelli, A. *Synthesis of Mixed Software-Hardware Implementation from CFSM Specifications.* Electron. Res.Lab. Univ. of California, Berkeley, CA, Memo. UCB/ERL M93/49. Jun. 1993.

[10] Dung, L.R., Madisetti, V.K. "Conceptual prototyping of scalabe embedded DSP systems". *IEEE Design and test of Computers*, Vol 13, N° 3, Fall 1996, pp 54-65.

[11] Gajski, D.D., Dutt, N.,Wu, A., Lin, S. *High-level synthesis.* Kluwer 92, ISBN 0792391942.

[12] Gajski,D.D., Narayan, S., Ramachandran, L., Vahid, F., Fung, P. "System design methodologies: Aiming at the 100 h design cycle". *IEEE Tran. on VLSI systems*, Vol.4, n° 1, March 96, pp. 70-82.

[13] Gajski, D.D., Ramachandran, L. "Introduction to high-level synthesis". *IEEE Design and test of Computers*, Vol 11, n° 4, winter 1994, pp 44-54.

[14] Gajski, D.D., Vahid, F. "Specification and design of embedded hardware software systems". *IEEE Design and test of Computers*, Vol 12, n° 1, Spring 1995, pp 53-67.

[15] Gajski, D.D., Vahid, F., Narayan, S., Gong, J. *Specification and design of embedded systems.* Prentice Hall 94, ISBN 0131507311.

[16] Gong, J., Gajski, D.D., Bakshi, S. "Model Refinement for Hardware-Software Codesign". *IEEE Proceedings of the European Design and Test Conference*, pp. 270-274, Paris, Mar.1996.

[17] Hosticka, B.J., Brockherde, W., Klinke, R., Kokozinski, R. "Design methodology for analog monolithic circuits". *IEEE Tran. on Circuits Syst.* I., Fundam. Theory Appl. Vol.41, n° 5, May 94, pp. 387-394.

[18] Knapp, D.W. *Behavioral synthesis.* Prentice Hall 96, ISBN 0135692520.

[19] Ku ̧ük ̧akar, K., Parker, A. "A methodology and design tools to support system-level VLSI design". *IEEE Tran. on VLSI systems*, Vol.3, n° 3, Sept. 95, pp. 355-369.

[20] Lee, E.A., Messerschmitt, D.G. "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing". *IEEE Trans. Comp.*, Vol. 36, N° 1, pp. 24-35. Jan. 1987.

[21] Madisetti, V.K. *VLSI digital signal processors.* Butterworth Hinemann 95, ISBN 0750694068.

[22] Madisetti, V.K. "Rapid digital system prototyping: current practice, future challenge". *IEEE Design and test of Computers*, Vol 13, nº 3, Fall 1996, pp 12-22.

[23] McCluskey, E.J. *Logic design principles with emphasis on testable semicustom circuits*. Prentice Hall 86 ISBN 0135397847.

[24] Michel, P.,Lauther, U., Duzy, P. *The synthesis approach to digital system design*. Kluwer 92, ISBN 0792391993.

[25] Micheli, G. De. "Computer-aided hardware-software codesign". *IEEE Micro*, Vol 14, nº 4, Aug. 94, pp. 10-16.

[26] Micheli, G.De. *Synthesis and optimization of digital circuits*. McGraw Hill 94, ISBN 0070163332.

[27] Navabi, Z. *VHDL Analysis and modeling of digital systems*. McGraw Hill 93, ISBN 0071127321.

[28] Oldfield, J.V., Dorf, R.C. *Field programmable gate arrays*. J. Wiley 95, ISBN 0471556653.

[29] Srivastava, M., Brodersen, R.W. "SIERA: A unified framework for rapid-protyping of system-level hardware and software". *IEEE Tran. on CAD of Integrated cicuits and systems*, Vol.14, nº 6, Jun 95, pp. 676-693.

[30] Srivastava, M., Richards, B.C., Brodersen, R.W. "System level hardware module generation". *IEEE Tran. on VLSI systems*, Vol.3, nº 1, March 95, pp. 20-35.

[31] Turino, J. *Design to test*, 2nd. Ed. Van Nostrand Reinhold 90, ISBN 0442001703.

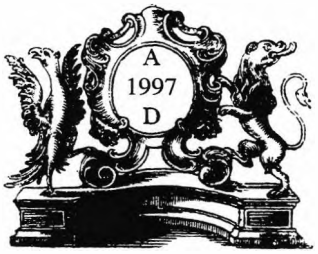
[32] Vahid, F., Narayan, S., Gajski, D.D. "SpecCharts: A VHDL front-end for embedded systems". *IEEE Tran. on CAD of Integrated cicuits and systems*, Vol.14, nº 6, Jun 95, pp. 694-706.

[33] Walker, R.A., Chaudhuri, S. "Introduction to the scheduling problem". *IEEE Design and test of Computers*, Vol 12, nº 2, Summer 1995, pp 60-69.

[34] Wolf, W.H. "Hardware-Software co-design of embedded systems". *Proc. of the IEEE*, Vol. 82, nº 7, Jul 94, pp. 967-989.

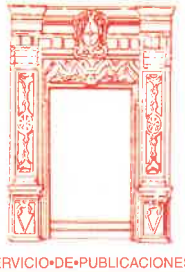
[35] Yu, T. "Create optimal simulation libraries using VHDL". *EDN* May 13, Vol. 1993, pp. 133-140.

Diego Gómez Vela, Doctor en CC. Físicas. Pedro Galindo Riaño, Doctor en Informática. Carmen García López, Lcda. en CC. Físicas. Son en la actualidad profesores de los siguientes departamentos de la Universidad de Cádiz: - Ingeniería de Sistemas y Automática, Tecnología Electrónica y Electrónica, - Lenguajes y Sistemas Informáticos - Ingeniería Eléctrica. Su actividad investigadora en el Grupo de Diseño de Circuitos Microelectrónicos de esta Universidad, se centra en la aplicación de las técnicas de síntesis de alto nivel y test de circuitos en diversas tecnologías microelectrónicas. Este grupo ha participado en numerosos cursos de formación, proyectos y publicaciones.



Terminose de imprimir este libro
en el Servicio de Autoedición e Impresión,
el día 3 de mayo,
festividad de la Invención de la Santa Cruz,
celebrada ya desde el lejano siglo VII en nuestra Hispania
y efeméride de la liberación de Cádiz
del yugo sarraceno.

Diego Gómez Vela, Doctor en Ciencias Físicas, Pedro Galindo Riaño, Doctor en Informática, y Carmen García López, Licenciada en Ciencias Físicas, son en la actualidad profesores de los departamentos de Ingeniería de Sistemas y Automática, Tecnología Electrónica y Electrónica y Lenguajes y Sistemas Informáticos-Ingeniería Eléctrica de la Universidad de Cádiz. Su actividad investigadora en el grupo de Diseño de Circuitos MIcroelectrónicos de esta Universidad, se centra en la aplicación de las técnicas de síntesis de alto nivel y test de circuitos en diversas tecnologías microelectrónicas. Este grupo ha participado en numerosos cursos de formación, proyectos y publicaciones.



SERVICIO•DE•PUBLICACIONES
UNIVERSIDAD•DE•CADIZ
1997

ISBN 84-7786-428-4

9 788477 864288