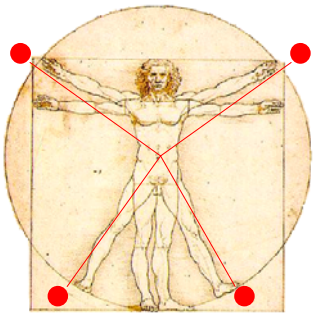
TECNOLOGÍ@ y *DESARROLLO*

Revista de Ciencia, Tecnología y Medio Ambiente

VOLUMEN II. AÑO 2004

SEPARATA



NUEVAS TENDENCIAS EN EL DISEÑO ELECTRÓNICO DIGITAL: CODISEÑO HARDWARE/SOFTWARE

Basil M. Al-Hadithi, Juan Suardíaz Muro

UNIVERSIDAD ALFONSO X EL SABIO
Escuela Politécnica Superior

Villanueva de la Cañada (Madrid)

ISSN: 1696-8085
Editor: Julio Merino García tecnologia@uax.es

NUEVAS TENDENCIAS EN EL DISEÑO ELECTRÓNICO DIGITAL: CODISEÑO HARDWARE/SOFTWARE

Basil M. Al-Hadithi[a], Juan Suardíaz Muro[b]

[a)]Dr Ing. Industrial,
Departamento de Electrónica y Sistemas, Escuela Politécnica Superior, Universidad Alfonso X el Sabio.
Avda. De la Universidad nº1, Villanueva de la Cañada, 28691 Madrid. España. Tlf.:918105035, email: bmal@uax.es

[b)]Dr Ing. Industrial,
Departamento de Tecnología Electrónica, Escuela Técnica Superior de Ingenieros Industriales,
Universidad Politécnica de Cartagena (Murcia).
Campus Moralla del Mar, 30202, Cartagena. España. Tlf.:968325380, email: Juan.Suardiaz@upct.es

RESUMEN:
Este artículo presenta un análisis de una de las principales tendencias que existe en la actualidad en lo referente al diseño electrónico digital: el codiseño. Se parte de una pequeña introducción en la que se muestra cuál ha sido la evolución que ha presentado el diseño electrónico, para posteriormente detallar lo que constituye la base del codiseño, mostrando sus principales ventajas e inconvenientes, así como las principales herramientas disponibles en el mercado para trabajar con esta metodología. Finalmente, se hace hincapié en las herramientas basadas en lenguajes tipo C/C++ y en el entorno MatLab/Simulink, utilizado actualmente por los arriba firmantes.

PALABRAS CLAVE: Diseño Electrónico Digital, Codiseño, FPGAs, Lenguajes de Descipción de Hardware.

ABSTRACT:
This paper introduces one of the most helpful trends of the current digital design process: codesign. In order to achieve this, a previous introduction regarding the evolution of the different design technologies and methodologies is presented. After this, codesign is introduced as well as the most used tools available in the electronic market. Finally, C/C++ and MatLab/Simulink EDA tools are enfatized as one of the most atractive solutions for today's hardware designing.

KEY-WORDS: Digital Electronic Design, Codesign, Field Programmable Gate Arrays (FPGAs), Hardware Description Languages.

1. Introducción

Pocos campos de la ciencia han avanzado de una forma tan vertiginosa como lo ha hecho la Electrónica. Desde sus comienzos teóricos a principios de siglo hasta los últimos sistemas integrados se ha recorrido un largo camino, marcado por numerosos hitos fundamentales, entre los que cabe destacar los siguientes:

- El desarrollo del primer diodo (de válvula) en 1904.
- La construcción del primer transistor en 1947, realizado por los premios Nobel John Bardeen, William Shockley y Walter H Brattain de los Bell Telephone Laboratories.
- La consecución del primer circuito integrado (CI) se construyó en 1958, inventado independientemente por Jack Kilby (Texas Instruments) y por Robert Noyce (Fairchild Semiconductors).

Actualmente se fabrican circuitos que llegan a superar la integración de alrededor de 100 millones de transistores, siguiendo la denominada ley de Moore (fig. 1), ley experimental que afirma que "*el número de transistores en un chip se dobla cada dos años; al igual que su velocidad de funcionamiento*". A pesar de lo impresionante de esta ley, se espera que la tecnología electrónica madure y no crezca al ritmo actual en 2020 debido a que los físicos vaticinan un límite a la ley de Moore en los 25nm (100 veces la anchura del átomo), punto en el que los efectos cuánticos de la materia tendrán efectos a considerar sobre los dispositivos electrónicos.

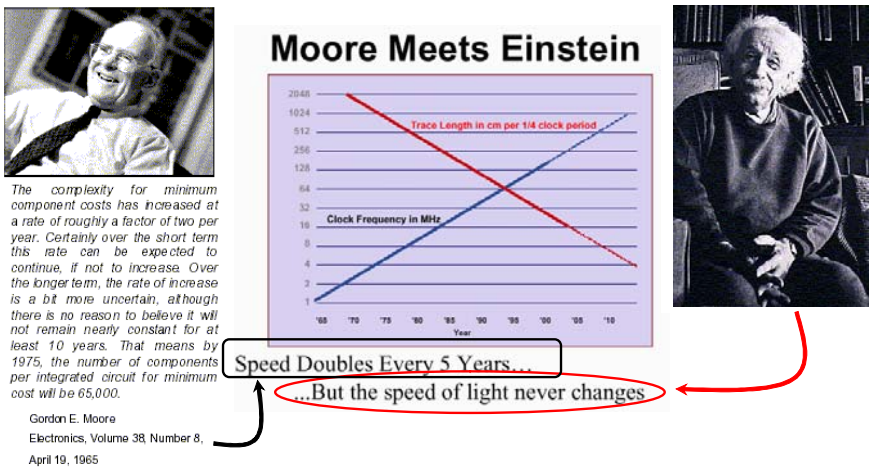


Fig. 1 Ley de Moore y su límite cuántico

Toda esta evolución en la tecnología electrónica, ha dejado su impronta en las tendencias del diseño electrónico, pudiendo considerar que se han producido lo que algunos autores (Boemo, 2003) consideran diferentes revoluciones, siendo sobre todo más evidente en lo que al diseño digital se refiere:

Una primera revolución la constituyó la *estandarización de los componentes*, acelerada por descubrimientos tales como la radio (1930), la televisión (1944) o las guerras mundiales (2ª Guerra Mundial 1939-1945). Otro factor también crítico lo constituyó la *miniaturización*, que evolucionó por el PCB (1945), el transistor (1947), PCB+transistor (1952), los circuitos integrados SSI (1965), MSI (1970), LSI (1975), etc.

Como consecuencia de lo anterior surge una filosofía de diseño modular, basada en circuitos integrados estándar, la cual permite el desarrollo productivo de sistemas relativamente complejos de una forma sencilla, flexibilizando su reproducibilidad y su mantenimiento. Sin embargo, cuando se trata de abordar con esta filosofía el diseño de sistemas más complejos, pronto comienzan a aparecer problemas asociados a la velocidad, el consumo o incluso a la confidencialidad, ya que si un diseño electrónico se encuentra basado en dispositivos estándar es presa fácil de todos aquéllos que quieran copiarlo, siendo esto un enorme inconveniente en un mercado tan competitivo como es el electrónico.

Ante la problemática anterior, un nuevo hito marca un cambio en la forma de diseño: los sistemas microprogramables. En 1969, Busicon encarga a una pequeña compañía de 10 empleados llamada Integrated Electronics (conocida hoy como Intel) un chipset para una computadora. Tratan de utilizar la potencia de la integración con la finalidad de hacer un micro-computador en un chip y luego “personalizarlo” mediante un programa almacenado en memoria. Como resultado surge en Febrero de 1971 el Intel 4004, ofreciendo a los diseñadores un circuito que permite construir sistemas lo suficientemente complejos mediante una simple programación, pero que es a la vez lo bastante estándar como para poder asegurar la flexibilidad y modularidad comentada anteriormente. Su impacto fue tan grande que pronto los fabricantes desarrollaron versiones económicas, denominadas microcontroladores, orientados al desarrollo e implantación de controladores industriales o incluso versiones más potentes en lo que a habilidades de cálculo se refiere, originando los denominados DSP (*Digital Signal Processor*). Sin embargo, una vez más los requerimientos de diseños cada vez más y más complejos pronto volvieron a colocar una barrera en lo que a velocidad y prestaciones de estos elementos se refiere a la hora de desarrollar sistemas que demandasen una elevada velocidad de trabajo, y lo que es peor, la alta confidenciabilidad ofrecida inicialmente pronto se vio que presentaba sus lagunas.

Muchos diseñadores entonces volvieron a pensar en los denominados dispositivos ASIC (*Application Specific Integrated Circuit*) como la única solución viable ante tal problemática. Sin embargo, el elevado costo de desarrollo asociado a estos sistemas, tanto en esfuerzo de diseño como de gastos asociados a ingeniería no recurrente, los convertía en una solución poco atractiva para muchos diseñadores.

Fue entonces cuando partiendo de la idea que tuvo Texas Instruments con las PAL en los 70, al difundir puertas lógicas que se puedan unir con la red de interconexiones programables, Xilinx difunde bloque lógicos que puedan realizar cualquier función lógica (Mux+LUT+flip-flops) con una red de interconexiones programables y surge así en 1985 la primera FPGA.

Las FPGAs son circuitos de aplicación específica (ASIC) de alta densidad programables por el usuario en un tiempo reducido y sin la necesidad de verificación de sus componentes, tarea ya realizada por el fabricante al tratarse de un producto estándar. Se las considera como un derivado de los Gate Array: FPGA = *Field Programmable Gate Array* ⇔ Gate Array Programable en Campo. Aunque es menos usado, también se les conoce como LCAs (*Logic Cell Array*), denominación registrada por Xilinx. La tabla 1 muestra una comparativa entre este tipo de dispositivos y los circuitos ASIC de los que derivan.

La evolución de las FPGAs hasta la fecha he seguido tres vías diferentes (fig. 2):

a.- Tecnológica

- Geometrías cada vez más pequeñas usando transistores más pequeños y rápidos, acompañada de costes cada vez menores (por área).

b.- Estructural

- Orientación al diseño de sistemas: generadores de acarreos, memorias y multiplicadores embebidos.
- Interconexiones jerárquicas, control de impedancias E/S.

c.- Metodológica

- Disponibilidad de módulos sintetizables cada vez más complejos.
- Orientada hacia el diseño modular y en equipo.

	Gate Array	FPGA
Tipo de producto	Específico	Estándar
Impacto en la producción	Retrasos en la producción	Rapidez de distribución en mercado
Programación	Sólo en proceso de fabricación.	Por el usuario (reprogramable)
Simulación	Complicada	Fácil
Verificaciones en fases previas al diseño final.	Imposible	Posible
Cambios en el diseño	Muy costosos	Posible en cualquier momento.
Comprobación del dispositivo	Específica para cada diseño	Comprobada por el fabricante

Tabla 1 Comparativa FPGAs/Gate Arrays

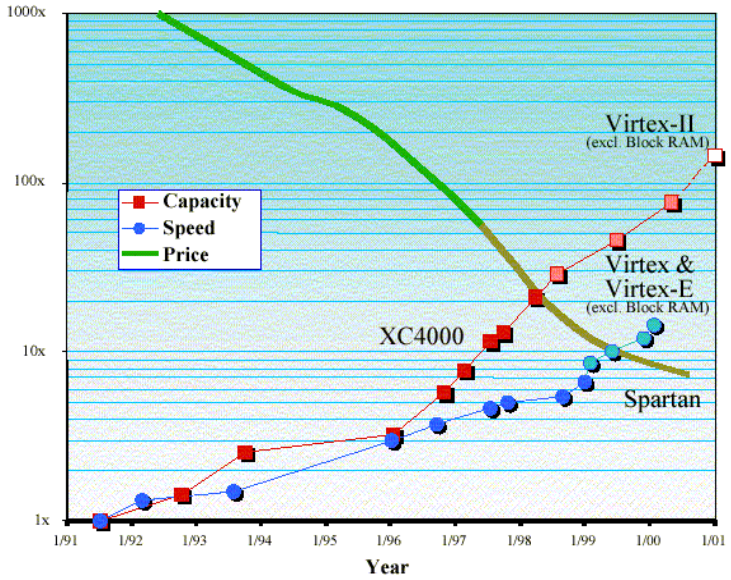


Fig. 2 Evolución de las FPGAS de Xilinx

Actualmente las FPGAs basadas en SRAM constituyen el grueso de los dispositivos de implantación de arquitecturas reconfigurables (familias XC y VIRTEX de Xilinx).

Se puede decir que, por lo general, la arquitectura de un dispositivo reconfigurable puede ser descrita como una red de celdas lógicas comunicadas a través de una red de celdas de interconexión. Cada conexión programable está controlada mediante una celda de memoria SRAM, por lo que establecer una ruta entre dos componentes del dispositivo, o determinar la funcionalidad de un bloque de lógica, puede hacerse fácilmente escribiendo unos cuantos bits en las correspondientes celdas SRAM. Cualquier proceso que necesite varias operaciones puede multiplexarse temporalmente sobre el mismo circuito consiguiéndose una eficiencia máxima del hardware.

Otra serie de ventajas no menos importantes son:

- Alto rendimiento: gran cantidad de registros, trabajo a alta frecuencia 100MHz.
- Alta densidad y capacidad: amplia gama de densidades de integración.
- Memoria integrada en varios niveles: generalmente estos dispositivos incluyen cierta cantidad de memoria RAM que puede emplearse para el almacenamiento temporal de coeficientes o la construcción de pequeñas FIFOs, y en un segundo nivel bloques completos de RAM para crear grandes FIFOs, memorias cachés o buffers de vídeo.
- Reprogramabilidad en el sistema: sin necesidad de extraerlo del sistema en el que está inmerso. Conexión a una memoria EPROM donde se guardan las distintas configuraciones o reconfiguración a través de microprocesador de propósito general.
- Reconfiguración parcial: ciertos dispositivos pueden ser parcialmente reconfigurados sin perder funcionalidad de otras partes del dispositivo Con esto se consiguen tiempos de reconfiguración aún menores y mayor flexibilidad.
- Ciclos de diseño relativamente cortos y baratos: el creciente desarrollo de herramientas de diseño junto con el grado de madurez alcanzado por los lenguajes de descripción hardware de alto nivel permiten diseñar, implementar, simular y verificar en poco tiempo.

A todas estas ventajas hay que añadir unos precios por pieza no mucho más elevados que los tradicionales ASIC, existiendo soluciones ya cableadas con precios muy similares a estos para grandes tiradas. Esto ha hecho que cada vez más diseñadores las consideren como alternativa de implementación, tal y como refleja la figura 3, donde se muestra una comparativa de demanda entre dispositivos ASIC y dispositivos lógicos programables complejos (CPLDs y FPGAs).

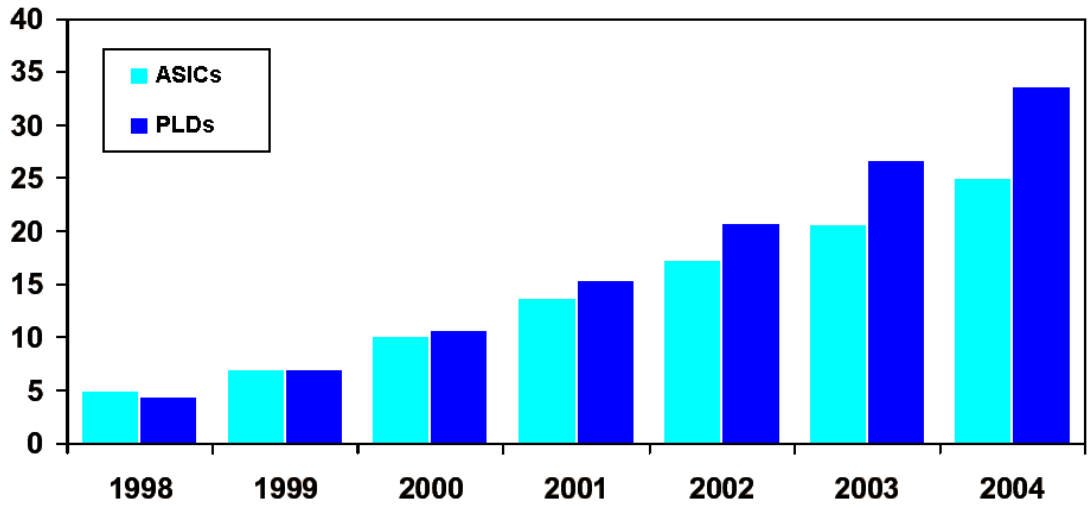


Fig. 3 Demanda de ASICs y PLDs valorada en millones de dólares.

2. Evolución del diseño electrónico

El desarrollo de las nuevas tecnologías de fabricación de circuitos integrados durante las últimas décadas ha propiciado a su vez la aparición de herramientas software y hardware de diseño y simulación de dispositivos electrónicos cada vez más potentes y avanzadas. Éstas, a su vez, han posibilitado la creación de nuevos dispositivos y circuitos electrónicos de una complejidad y funcionalidad con un crecimiento prácticamente exponencial. A su vez, esta complejidad creciente en los diseños ha originado que las metodologías de diseño sufran también una evolución (Pardo, 1997).

En los inicios, todo el diseño de un nuevo circuito integrado se realizaba a mano, transistor a transistor, indicando sus dimensiones, su ubicación en el plano base y su conexionado con el resto de componentes del circuito. A esto se le conoce como diseño '*Bottom-Up*' (ascendente). Según este tipo de metodología, se ha de comenzar el diseño a partir de los elementos más pequeños para conseguir que la suma de sus efectos realice la aplicación deseada.

Esta metodología no implica una dependencia jerárquica funcional de los elementos del circuito, por lo que se trata de una técnica poco eficiente cuando se pretende hacer

funcionar un sistema compuesto por miles de componentes de bajo nivel. Además, el sistema no puede comprobarse (simularse) hasta que no está totalmente terminado, y un gran número de decisiones criticas a nivel de tecnología de fabricación deben tomarse al comienzo del diseño. Es por ello que, cuando los proyectos son complejos o de gran envergadura, esta forma de trabajo conlleva un alto costo en la producción, ya que los fallos y errores durante el diseño son frecuentes debido a la naturaleza, sensibilidad a los cambios y/o ajustes de los elementos más pequeños.

Hasta este momento, sólo existían herramientas de simulación de circuitos eléctricos, tipo SPICE, que ayudaban en el diseño a nivel de transistor. Conforme los procesos tecnológicos de fabricación de CI (circuitos integrados) se hacían más complejos, para poder asumir mayores densidades de integración, las técnicas de diseño "manual" de CI se fueron quedando obsoletas. Era prácticamente imposible manejar la complejidad que requerían los nuevos circuitos, en número de transistores y en los bancos de pruebas (test-benches) necesarios para comprobar una correcta funcionalidad.

La consolidación en la década de los noventa de los lenguajes de descripción de hardware (HDLs, *Hardware Description Languages*) ha supuesto, por otro lado, la implantación progresiva de las denominadas metodologías de diseño '*Top-Down*' (descendente) que, en contraposición a la metodología ascendente (Bottom-Up), permiten la descripción del sistema al más alto nivel. En el diseño Top-down la dependencia de la implementación final es prácticamente inexistente en las etapas de definición funcional, y se irá concretando en las sucesivas fases de diseño, hasta llegar a la síntesis de nuestro sistema sobre cualquiera de las tecnologías existentes (fulI-custom, ASICs, FPGAs, CPLDs,...).

En este caso, podemos concentrar nuestro esfuerzo en la descripción del sistema a nivel funcional y comportamental, valorando la adecuación de distintas arquitecturas, antes de abordar el diseño detallado a nivel físico. Esto es posible porque las herramientas de CAD que soportan los HDLs proporcionan entornos de simulación con distintos niveles de precisión: desde una simulación únicamente funcional, antes de tener sintetizado el circuito sobre ninguna tecnología en concreto, que nos permite comprobar el correcto comportamiento de la especificación de nuestro circuito, hasta una simulación temporal, teniendo en cuenta retardos concretos sobre los componentes electrónicos de base y en el rutado de éstos, para una implementación en concreto.

2.1.- Los lenguajes de descripción de hardware

Los lenguajes de descripción de hardware (HDLs, *Hardware Description Languages*) son utilizados para describir la arquitectura y comportamiento de un sistema electrónico, los cuales fueron desarrollados para trabajar con diseños complejos, como es caso de los PLDs y de hay su importancia en este proyecto.

Comparando un HDL con los lenguajes para el desarrollo de software vemos que, en un lenguaje de este tipo, un programa que se encuentra en un lenguaje de alto nivel necesita ser ensamblado a código máquina para poder ser interpretado por el procesador. De igual manera, el objetivo de un HDL es describir un circuito mediante un conjunto de instrucciones de alto nivel de abstracción para que el programa de síntesis genere (ensamble) un circuito que pueda ser implementado físicamente.

La forma más común de describir un circuito es mediante la utilización de esquemas, que son una representación gráfica de lo que se pretende realizar. Con la aparición de herramientas EDA (*Electronic Design Automation*) cada vez más complejas, que integran en el mismo marco de trabajo las herramientas de descripción, síntesis, simulación y realización, apareció la necesidad de disponer de un método de descripción de circuitos, que permitiera el intercambio de información entre las diferentes herramientas que componen el ciclo de diseño.

En principio se utilizó un lenguaje de descripción que permitía, mediante sentencias simples, describir completamente un circuito. A estos lenguajes se les llamó Netlist puesto que eran simplemente eso, un conjunto de instrucciones que indicaban las interconexiones entre los componentes de un diseño. A partir de estos lenguajes simples, que ya eran auténticos lenguajes de descripción hardware, se descubrió el interés que podría tener el describir circuitos utilizando un lenguaje en vez de usar esquemas. Sin embargo, se siguieron utilizando esquemas puesto que desde el punto de vista del ser humano son mucho más sencillos de entender, aunque un lenguaje siempre permite una edición más rápida y sencilla.

Conforme las herramientas de diseño se volvieron más sofisticadas, y la posibilidad de desarrollar circuitos digitales mediante dispositivos programables era más viable, apareció la necesidad de poder describir los circuitos mediante un lenguaje de alto nivel de abstracción. No desde un punto de vista *estructural*, sino desde el punto de vista *funcional*.

Este nivel de abstracción se había alcanzado ya con las herramientas de simulación, ya que, para poder simular partes de un sistema, era necesario disponer de modelos que describieran el funcionamiento de bloques del circuito o de cada componente si fuera necesario. Estos lenguajes estaban sobre todo orientados a la simulación, por lo que poco importaba que el nivel de abstracción fuera tan alto que no fuera sencilla una realización o síntesis a partir de dicho modelo. Con la aparición de técnicas para la síntesis de circuitos a partir de lenguajes de alto nivel de abstracción, se comenzaron a utilizar los lenguajes de simulación para sintetizar circuitos. Que si bien alcanzan un altísimo nivel de abstracción, su orientación era básicamente la de simular, por lo que los resultados de una simulación a partir de descripciones con estos lenguajes no eran siempre las más optimas.

Además, los lenguajes de descripción de hardware al formar parte de las herramientas EDA permiten el trabajo en equipo. Así, al estructurar el desarrollo del proyecto, cada integrante del equipo de diseño puede trabajar en subproyectos antes de integrar todas las partes del sistema. En el caso concreto del VHDL, esta modulación se lleva un paso más allá con la posibilidad de reutilización del código, mediante comandos tales como *generate* o *generic*, que permiten generar módulos de diferentes dimensión cambiando pocas líneas de código.

Hoy en día los HDLs están ampliamente difundidos y estandarizados bajo la IEEE, por lo que su aprendizaje, más que aconsejable, es una necesidad en la pequeña y gran industria. Sin embargo, debido la numerosa cantidad de HDLs existentes (VHDL, Verilog, ABEL, SystemC...), el diseñador no debe limitarse dominar uno sólo de ellos, sino aprender las características comunes de ellos y en que radican sus diferencias (Van den Bout, 1999).

Como se puede apreciar en el siguiente gráfico (fig. 4), el rango de etapas del proceso de fabricación que abarcan los HDLs es bastante amplio.

Una metodología de diseño que utiliza un HDL posee varias ventajas sobre la metodología tradicional de diseño a nivel compuerta, entre las que cabe destacar las siguientes:

- Es posible verificar el funcionamiento del sistema dentro del proceso de diseño sin necesidad de implementar el circuito.
- Las simulaciones del diseño, antes de que éste sea implementado mediante compuertas, permiten probar la arquitectura del sistema para tomar decisiones en cuanto a cambios en el diseño.

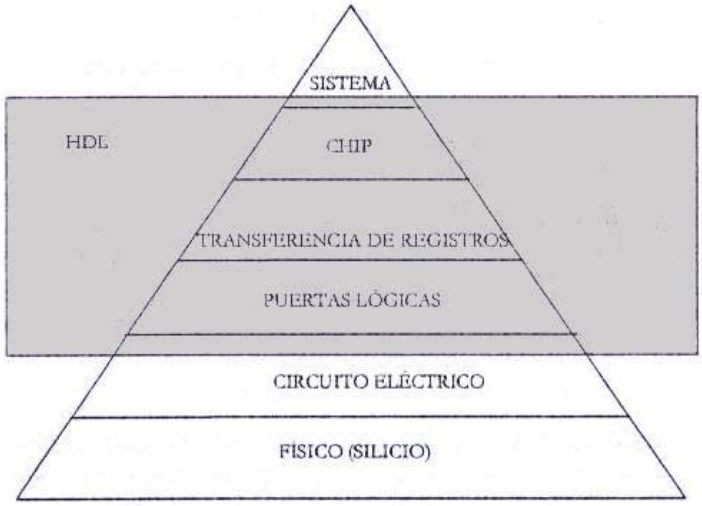


Fig. 4 Estratos abarcados por los HDLs

- Las herramientas de síntesis tienen la capacidad de convertir una descripción hecha en un HDL a compuertas lógicas y, además, optimizar dicha descripción de acuerdo a la tecnología utilizada.
- La competencia entre los programadores de los diferentes sintetizadores del mercado (Synplify, LeonardoSpectrum...), que ponen su mayor empeño en conseguir que su producto sea el mejor y optimice al máximo lo descrito en el HDL, hace que las herramientas mejoren constantemente.
- Esta metodología elimina el antiguo método tedioso de diseño mediante compuertas, reduce el tiempo de diseño y la cantidad de errores producidos por el armado del circuito.
- Las herramientas de síntesis pueden transformar automáticamente un circuito obtenido mediante la síntesis de un código en algún HDL, a un circuito pequeño y rápido. Además, es posible aplicar ciertas características al circuito dentro de la descripción para afinar detalles (retardos, simplificación de compuertas etc.) en la arquitectura del circuito y que estas características se obtengan en la síntesis de la descripción.
- Las descripciones en un HDL proporcionan documentación de la funcionalidad de un diseño independientemente de la tecnología utilizada, al contrario que el trabajo en esquemáticos, en los cuales, cada símbolo tiene que estar previamente definido para el dispositivo con el que se trabaja.
- Un circuito hecho mediante una descripción en un HDL puede ser utilizado en cualquier tipo de dispositivo programable capaz de soportar la densidad del

diseño. Es decir, no es necesario adecuar el circuito a cada dispositivo porque las herramientas de síntesis se encargan de ello.

- Una descripción realizada en un HDL es más fácil de leer y comprender que los Netlist o circuitos esquemáticos.

2.2.- El lenguaje VHDL

Las siglas VHDL vienen de VHSIC (*Very High Speed Integrated Circuit*) *Hardware Description Language*. El VHDL es un lenguaje de descripción y modelado, diseñado para describir (en una forma que los humanos y las máquinas puedan leer y entender) la funcionalidad y la organización de sistemas hardware digitales, placas de circuitos, y componentes. Otros lenguajes de descripción hardware comúnmente empleados son el ABEL y el VERILOG. El ABEL permite la síntesis lógica (FSMs, funciones lógicas) pero en comparación con el VHDL, la sintaxis es demasiado limitada para la descripción de comportamiento (en el apartado VHDL describe estructura y comportamiento se explicara a que se refiere cuando se dice comportamiento). El VERILOG fue desarrollado inicialmente por Cadence Design System y esta orientado a la descripción de circuitos a nivel funcional, lógico y de conmutador. Su sintaxis es concisa en comparación con el VHDL y comparten una mayor difusión.

El VHDL fue desarrollado como lenguaje para modelado y simulación lógica dirigida por eventos de sistemas digitales, y actualmente se utiliza también para la síntesis automática de circuitos.

VHDL es un lenguaje con una sintaxis amplia y flexible que permite el modelado estructural, en flujo de datos y de comportamiento hardware. El lenguaje permite el modelado preciso, en distintos estilos, del comportamiento de un sistema digital conocido y el desarrollo de modelos de simulación.

Uno de los objetivos del lenguaje VHDL es el modelado. Modelado es el desarrollo de un modelo para simulación de un circuito o sistema previamente implementado cuyo comportamiento, por tanto, se conoce. El objetivo del modelado es la simulación.

Otro de los usos de este lenguaje es la síntesis automática de circuitos. En el proceso de síntesis, se parte de una especificación de entrada con un determinado nivel de abstracción, y se llega a una implementación más detallada, menos abstracta. Por tanto,

la síntesis es una tarea vertical entre niveles de abstracción, del nivel más alto en la jerarquía de diseño se va hacia el más bajo nivel de la jerarquía.

El VHDL es un lenguaje que fue diseñado inicialmente para ser usado en el modelado de sistemas digitales. Es por esta razón que su utilización en síntesis no es inmediata, aunque lo cierto es que la sofisticación de las actuales herramientas de síntesis es tal que permiten implementar diseños especificados en un alto nivel de abstracción.

La síntesis a partir de VHDL constituye hoy en día una de las principales aplicaciones del lenguaje con una gran demanda de uso. Las herramientas de síntesis basadas en el lenguaje permiten en la actualidad ganancias importantes en la productividad de diseño.

Algunas ventajas del uso de VHDL para la descripción hardware son:

- VHDL permite diseñar, modelar, y comprobar un sistema desde un alto nivel de abstracción bajando hasta el nivel de definición estructural de puertas.
- Circuitos descritos utilizando VHDL, siguiendo unas guías para síntesis, pueden ser utilizados por herramientas de síntesis para crear implementaciones de diseños a nivel de puertas.
- Al estar basado en un estándar (IEEE Std 1076-1987) los ingenieros de toda la industria de diseño pueden usar este lenguaje para minimizar errores de comunicación y problemas de compatibilidad.
- VHDL permite diseño Top_Down, esto es, permite describir (modelado) el comportamiento de los bloques de alto nivel, analizándolos (simulación), y refinar la funcionalidad de alto nivel requerida antes de llegar a niveles más bajos de abstracción de la implementación del diseño.
- Modularidad: VHDL permite dividir o descomponer un diseño hardware y su descripción VHDL en unidades más pequeñas.

Uno de los motivos que ha hecho que el se haya difundido tanto es que permite conjugar las dos metodologías de diseño previamente comentadas. Por un lado se puede describir indicando los diferentes componentes que lo forman, de esta manera se tiene especificado un circuito y se sabe como funciona. Esta es la forma habitual en que se ha venido describiendo circuitos y las herramientas utilizadas para ello han sido las de captura de esquemas y las descripciones netlist. La segunda forma consiste en describir un circuito indicando lo que hace o cómo funciona, es decir, describiendo su comportamiento. Naturalmente esta forma de describir un circuito es mucho mejor para un diseñador puesto que lo que realmente interesa es el funcionamiento del circuito más

que sus componentes. En consecuencia, el mayor atractivo del lenguaje VHDL radica en que va a permitir los dos tipos de descripciones de circuito:

- Estructural: VHDL puede ser usado como un lenguaje de Netlist normal y corriente donde se especifican por un lado los componentes del sistema y por otro sus interconexiones.
- Comportamental: VHDL también se puede utilizar para la descripción comportamental o funcional el circuito. Esto es lo que lo distingue de un lenguaje de Netlist. Sin necesidad de conocer la estructura interna de un circuito es posible describirlo explicando su funcionalidad. Esto es especialmente útil en simulación ya que permite simular un sistema sin conocer su estructura interna. Este tipo de descripción se está volviendo cada día más importante porque las actuales herramientas de síntesis permiten la creación automática de circuitos a partir de una descripción de su funcionamiento.

3. El futuro del diseño digital: el codiseño

Tal y como se puede apreciar en la figura 5, la complejidad de los sistemas electrónicos digitales con el devenir de los años presenta una evolución creciente en complejidad, prestaciones y densidad.

Gartner (Gartner, 2000) vaticina que en el 2004 la mayor parte de los diseños electrónicos digitales necesitarán de una fase previa, en la que serán descritos usando un lenguaje de alto nivel (HLL, *High Level Language*).

La cada vez más imperiosa demanda del mercado, obliga a los fabricantes de herramientas de diseño electrónico (herramientas EDA, *Electronic Design Automation*) a dotarlas de capacidades de desarrollo que posibiliten la transformación de una idea inicial en un diseño final, de forma rápida, eficiente, fiable y, por supuesto, con un bajo coste asociado.

Tales restricciones de diseño han hecho evolucionar la forma clásica de desarrollo de sistemas, y en especial los digitales, en la que lo habitual era un ciclo de diseño basado en prototipos que se iban mejorando hasta conseguir un producto final. Sin embargo, actualmente el esfuerzo se concentra en el nivel conceptual, verificando previamente la funcionalidad del sistema sin necesidad de llegar a un nivel de detalle excesivo.

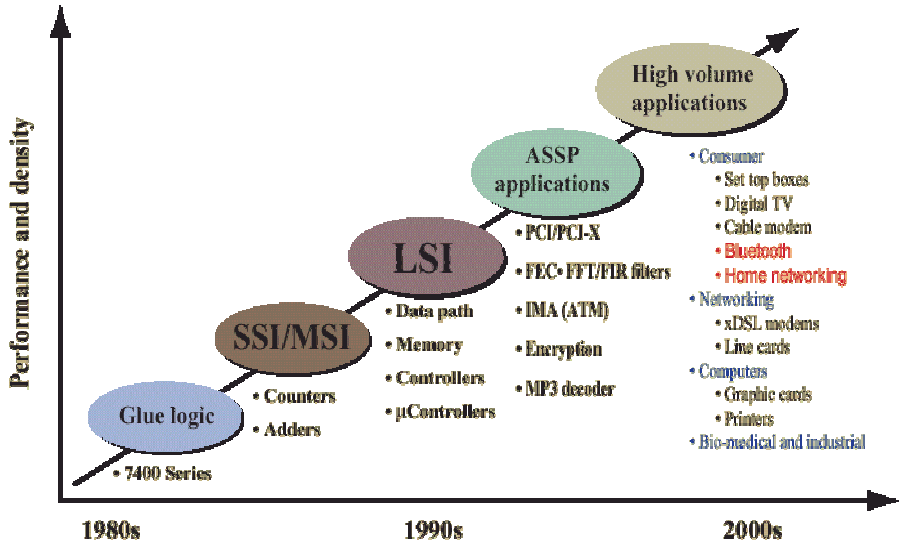


Fig. 5 Crecimiento de la densidad y complejidad de los diseños electrónicos.

El uso de lenguajes de descripción de hardware (HDL, *Hardware Description Language*) frente a una descripción mediante esquemáticos, permitió que los diseñadores pudieran recuperar el control a lo largo del proceso de creación de sistemas digitales complejos, ayudados por herramientas que permitían un diseño basado en una complejidad organizada de forma jerárquica y modular. Hoy en día, el empleo de técnicas de modelización a alto nivel, unido al concepto de Diseño a Nivel de Sistemas (SLD, *System Level Design*) – frente al cada vez más clásico diseño a nivel de transferencia de registros (RTL, *Register Transfer Level*) –, permite el empleo de técnicas de "*prototipado*" rápido de sistemas, gracias a la creación de bibliotecas y reutilización de componentes hardware/software.

La mayor parte de los diseños que se llevan a cabo en la actualidad implican enfrentarse a un equivalente de millones de puertas. Esto hace que el lograr a la primera un sistema que cumpla las especificaciones, sea una tarea difícil y costosa si no se lleva a cabo una cuidadosa planificación de tareas. La metodología de diseño SLD, en la que los diseños se implementan usando plataformas hardware y software ya validadas, permite obtener productos finales en los que los cambios y actualizaciones se llevan a cabo sin un excesivo trabajo adicional.

La cada vez más frecuente aplicación de los HLL en el proceso de diseño hace que las fronteras entre lo que se puede considerar desarrollo de software y de hardware se difuminen. Los métodos tradicionales aplicados en el diseño de sistemas, requerían especificaciones diferentes para los elementos hardware y software. El modo de trabajo más común consistía en un primer desarrollo de una especificación (muchas veces incompleta), que se describía en un lenguaje "no-formal" y se mandaba a los ingenieros de software y de hardware. Esto requería un *particionado* del sistema "a-priori", esto es, una decisión de las tareas que se pensaban implementar usando software y aquéllas otras que se desarrollarían mediante hardware.

Los principales inconvenientes que aparecen asociados a este tipo de metodología de diseño pueden resumirse en:

- Una carencia de representaciones unificadas tanto del hardware como del software, lo que complica el proceso de verificación del sistema en su totalidad y ocasiona la aparición de incompatibilidades a lo largo de la frontera hardware/software.
- Un proceso previo de *particionado* que, a pesar de poder estar fundamentado en la experiencia de los mejores profesionales, es susceptible de conducir a diseños diferentes a los que podrían considerarse como óptimos (diseños sub-óptimos).
- Falta de un flujo de diseño bien definido, lo que enmaraña los procesos de revisión de especificaciones y, a la larga, afecta al proceso productivo y en especial, al tan temido tiempo de diseño (*time-to-market*).

Una nueva técnica de diseño ha surgido para tratar de solventar estos inconvenientes: el *codiseño*. Su objetivo principal es "*el desarrollo de sistemas que no sólo cumplan con las especificaciones iniciales de funcionamiento, sino que además, el diseño tanto del hardware como del software, como la decisión de qué se hace en hardware o en software se tome utilizando unos criterios objetivos de optimización*" (Gomez, 1998)

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 el coste monetario, sino también en tiempo de diseño (*time-to-market*), factor clave en los mercados actuales. 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 a fin de lograr una optimización de costes. Por último, además se deben considerar factores de tipo técnico, como la fiabilidad del sistema obtenido en términos de control de la calidad, así como la eficiencia obtenida para el objetivo para el que el sistema se encuentra diseñado.

El hecho de utilizar una metodología de diseño que no optimice los factores mencionados, 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. Es por ello que la aplicación de técnicas de codiseño a los procesos de desarrollo de sistemas supone una clara ventaja sobre los procesos de diseño tradicionales, posicionando en un puesto de privilegio en el mercado a aquéllas empresas que las apliquen.

4. Perspectiva del mercado de herramientas de codiseño

A lo largo de la última década, los diseñadores han tomado conciencia de la potencia que ofrecen las técnicas de codiseño, al posibilitar trabajar en un nivel de abstracción que se puede considerar independiente de la plataforma que sustentará el diseño final. Esto posibilita la creación de módulos de diseño que no presentan las restricciones asociadas a la elección de una determinada tecnología, sistema o plataforma para su implementación.

En la actualidad, el codiseño es un tema en auge que ha motivado la aparición en distintos ambientes de herramientas destinadas a facilitar la tarea de los diseñadores, tal y como se puede deducir de la tabla adjunta, en la que se muestra a modo de resumen los sistemas de desarrollo basados en esta metodología más usados por los diseñadores hoy en día.

Herramienta	Creador	Descripción
Proyectos Académicos:		
Ptolemy	Univ. Berkeley (1990)	Herramienta para el desarrollo a nivel de sistema, que permite trabajar con modelos mixtos. Esto significa que es posible conectar un modelo a nivel hardware con otro modelo de un sistema llevado a cabo a alto nivel
Polis	Univ. Berkeley (1990)	Se trata de paquete software, que se añade a Ptolemy, centrado en una representación basada en máquinas de estado finitas, originando la denominada Co-design Finite State Machine (CFSM).
Cosyma	Univ. Braunshweig	Se trata de una herramienta que particiona automáticamente un conjunto de tareas sobre hardware y software, en base a una arquitectura consistente en un procesador, una memoria y componentes hardware. El algoritmo de particionado parte de una implementación del sistema basada solamente en software y redirecciona partes del sistema hacia el hardware, a fin de cumplir con las restricciones temporales pedidas.
Castle	German National Research Center for Information Technology.	Opera sobre una representación en grano fino del sistema, usualmente realizada en C (aunque también soporta VHDL y Verilog), usando un formato de representación especial para las operaciones del sistema. Es capaz de generar código VHDL sintetizable, listo para ser descargado sobre los componentes hardware.

Herramienta	Creador	Descripción
Lycos		Herramienta que posibilita una modelización de forma abstracta, tanto de hardware como de software. Para tal fin, emplea un subconjunto del VHDL y otro de C.
Roseta		Lenguaje para el diseño a nivel de sistema.
Comet	Univ. Cincinnati.	Entorno de desarrollo de técnicas de codiseño basado en VHDL y C, que usa un fichero especial de reglas con el fin de conectar ambas descripciones y lograr una descripción del sistema completo.
Cosmos	Univ. Grenoble	Herramienta para la síntesis a nivel de sistema, orientada al diseño y síntesis de sistemas mixtos hardware/software complejos.
Chinook	Univ. Washington	Herramienta CAD que facilita tanto la especificación, como la síntesis de sistemas.
Cobra		Herramienta que posibilita el análisis y verificación de propiedades del diseño, como la funcionalidad y consumo de potencia y tiempos.
Esterel	Univ. París, CNRS e INRIA	Esterel es tanto un lenguaje de programación, dedicado a programar sistemas reactivos, como un compilador que traduce estos programas en una especie de máquinas de estado finitas.
CodeSign	ETH Zurcí	Herramienta que, de forma gráfica, permite una modelización con características de un entorno orientado a objetos, basado en redes de Petri temporales.
ASP	Univ. Queensland	Herramienta automática de codiseño, orientada para la implementación de algoritmos con fuertes resticciones temporales. Identifica e implementa sobre hardware los segmentos de código que limitan la respuesta temporal del sistema.
Herramientas comerciales:		
VIOOL	ASC	Esta herramienta permite convertir VHDL en código C++, creando un "hardware virtual" que es posible comunicar con los posibles componentes software del diseño.
Foresight	Un Thena Systems, Inc	Ofrece un lenguaje de diseño a nivel de sistema muy fácil de entender, integrándolo en un IDE para el diseño y análisis de sistemas. Contiene un simulador y módulos capaces de generar código C y VHDL.
Cosmos-Galaxy	Omniview	Sistema de desarrollo que permite una especificación del hardware y del software en VHDL. Una vez decidida una estrategia de diseño, Galaxy es capaz de sintetizar el sistema en componentes y/o esquemáticos.
N2C	Coware	Permite realizar una especificación del sistema en C/C++, la cual puede ser implementada sobre diferentes arquitecturas, mediante el uso de una colección de modelos.
Eagle, COSSAP	Synopsys.	Esta herramienta proporciona un conjunto de modelos de "procesador virtual", encargados de ejecutar software que se enlaza con una simulación del hardware,
SES-Workbench	SES	Herramienta para el diseño de sistemas y simulación a alto nivel, basada en una serie de modelos de procesadores disponibles y de componentes compartidos por diversos fabricantes.
Gaio	Gaio Technology	Herramienta para el diseño de sistemas basados en microcontroladores.
VCC	Cadence	Entorno de desarrollo de sistemas, basado en componentes virtuales.
CAE Plus	Archgen	Herramienta que permite modelar hardware mediante C, a fin de posibilitar su simulación.
SystemGenerator	Xilinx & MathWorks	Herramienta para el diseño de sistemas que usen FGAs de Xilinx, basada en Matlab y Simulink.
System-C	Synopsys	Leguaje basado en C/C++ que permite describir los diseños a nivel de sistema.

Tabla 2 Entornos de codiseño más utilizados en la actualidad.

5. Lenguajes de alto nivel basados en C

El C (Kernighan, 1988) es un lenguaje de programación con 20 años de historia, que domina los entornos de desarrollo de sistemas empotrados, aplicaciones software y sistemas operativos.

El uso de este lenguaje para la descripción de componentes hardware permite aprovechar una serie de elementos ya validados, como pueden ser incontables bibliotecas de algoritmos, que los diseñadores pueden tomar como punto de partida para implementarlos sobre hardware. A esto hay que añadir la facilidad y rapidez con que es posible implementar en este lenguaje entornos de prueba (*test-benches*) y modelos hardware, consiguiendo así acelerar el ciclo productivo.

Así pues, añadiendo una serie de extensiones a lo que era el lenguaje clásico, ha sido posible generar una serie de lenguajes de alto nivel (HLL) que de una forma sencilla permiten describir sistemas complejos, como Handel-C (Celoxica, 2002), Spec-C (STOC, 2002) o SystemC (SystemC, 2004), los cuales permiten describir el comportamiento de un determinado elemento hardware de la misma forma que un programador describe el comportamiento de un procesador que ejecute el software que programa

4.1. Handel-C

Handel-C es un lenguaje idóneo para el diseño de lógica síncrona y es particularmente adecuado para la implementación de algoritmos complejos sobre FPGAs (Celoxica, 2002), ya que:

- Posibilita la generación automática de máquinas de estado a partir de una descripción algorítmica del circuito, en términos de bloques de código paralelos y secuenciales.
- El código que se ejecuta tras un grupo de bloques de código de ejecución en paralelo sólo comenzará a ejecutarse si todos los procesos del bloque paralelo anterior han terminado de ejecutarse. Para ello, usa un modelo de concurrencia basado en CSP (Communicating Sequential Processes); de forma que consigue mediante una ejecución concurrente el efecto de una ejecución en paralelo real.
- Posee canales para la comunicación entre bloques de código, así como mecanismos de sincronización entre éstos.
- Permite modelar retrasos temporales de las señales.
- Permite la generación automática de relojes, señales de habilitación y/o reset.

Además de trabajar a alto nivel, es posible controlar el funcionamiento a bajo nivel de los diseños:

- El lenguaje lleva implícito señales temporales muy precisas.
- Es posible implementar lógica combinacional usando elementos de tipo bus, puerto e incluso señal.
- Permite diseñar las zonas en las que sea necesario un nivel de detalle equivalente al que ofrece Verilog, manteniendo una perspectiva global de alto nivel.

4.2. SystemC

En 1999, varias compañías dedicadas al desarrollo de software y sistemas empotrados anunciaron la "Open SystemC Initiative" (OSCI), que suponía la posibilidad de disponer de una plataforma de modelado de libre distribución, basada en C++ y denominada SystemC.

Se trata de un lenguaje desarrollado para la modelización a nivel de sistema y el diseño hardware (SystemC, 2004) (Synopsys, 2000a) (Swan, 2001), que resulta muy útil para describir tanto los componentes hardware como software de un sistema.

En lo que respecta al hardware, dispone de un conjunto de clases que permiten modelar componentes hardware, que es posible simular gracias al kernel de simulación que también posee.

Además, existen herramientas tales como Cocentric SystemC Compiler (Synopsys) (Synopsys, 2000b) (Synopsys, 2000c) y el compilador N2C de CoWare (Coware, 2004), que permiten sintetizar componentes hardware a partir de una descripción en SystemC.

Por otro lado, al ser su base el lenguaje de programación C++, es posible utilizarlo también para modelar los componentes software del sistema, los cuales se comunicarán de alguna forma con los elementos hardware.

A nivel de sistema, permite trabajar con concurrencia, comunicaciones de tipo asíncrono, análisis de tiempos y con cálculos controlados bien por datos o bien por eventos.

Por consiguiente, el principal objetivo de este lenguaje es posibilitar la modelización, de una forma sencilla, de las arquitecturas que constituyen los sistemas, sus interfases y hasta los algoritmos software que se ejecutarán sobre ellos. Esto hace que, además de su faceta de lenguaje, pueda ser considerado como una metodología que permite crear modelos efectivos de elementos tales como:

- Algoritmos software.
- Arquitecturas hardware.
- Interfaces de Sistemas en Chip (SoC, *System on Chip*).
- Arquitecturas de sistemas.
- Diseños a nivel de sistema.

Actualmente este lenguaje está siendo adoptado como herramienta de diseño de empresas como Fujitsu, Matshushita, Sony o STMicroelectronics, permitiéndoles acortar la duración de los ciclos de diseño, gracias a las ventajas que aporta al flujo de diseño, como el poder ser empleado como plataforma de simulación mucho más veloz que las de co-verificación basadas en HDLs, el permitir verificar arquitecturas desde las fases iniciales del diseño, comprobando y evaluando las posibles particiones que pueden llevarse a cabo o las diferentes arquitecturas con las que se podría implementar un diseño determinado. Además, hay que añadir la posibilidad que ofrece de poder revisar un trabajo o comenzar uno nuevo reutilizando componentes ya diseñados.

Al trabajar con el lenguaje C en todos los niveles, ya no es necesario el uso de programas como Verilog Program Language Interface (PLI) o VHDL Foreign Language Interface (FLI), empleados por otras herramientas de codiseño para enlazar los modelos o posibles bancos de prueba (*test-benches*) escritos en C con las simulaciones HDL, lo cual constituía un cuello de botella en la fase de verificación funcional (fig. 6).

En este caso, es posible realizar verificaciones en C a diferentes niveles, que pueden mezclarse para obtener la simulación de un sistema completo. A modo de ejemplo, sirva destacar que los laboratorios de Fujitsu han realizado pruebas donde se observa que las simulaciones basadas en entornos C suelen ser del orden de 50 a 1000 veces más rápidas que aquéllas realizadas a nivel RTL (Hardee, 2002).

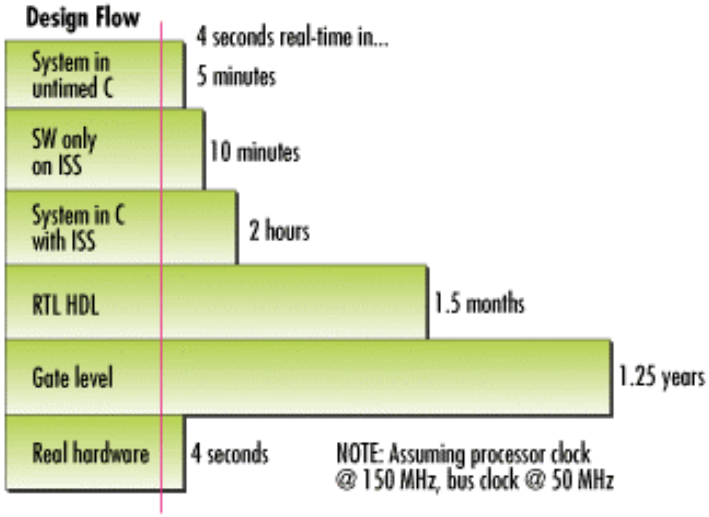


Fig. 6 Comparativas de tiempos asociados a simulaciones de un sistema funcionando en tiempo real.

6. Entornos de desarrollo basados en modelos Matlab/Simulink

La empresa Mathworks oferta la herramienta Matlab/Simulink, que es la base que sustenta otras herramientas de codiseño bastante atractivas.

5.1. MatLab

MatLab es un programa de cálculo numérico y visualización de datos. Aunque generalmente se ha empleado por los ingenieros de control, posee una extraordinaria versatilidad y capacidad para resolver problemas en matemática aplicada, física, química, ingeniería, finanzas y muchas otras aplicaciones.

Su base la constituye el cálculo matricial e integra análisis numérico, cálculo matricial, procesamiento de señales y visualización gráfica (figura 7). Todo ello en un entorno completo, donde los problemas y sus soluciones se expresan del mismo modo en que se escribían tradicionalmente, sin necesidad de hacer uso de la programación típica de los lenguajes FORTRAN, BASIC o C.

MatLab fue originariamente desarrollado en lenguaje FORTRAN para ser usado en computadoras mainframe. Fue el resultado de los proyectos Linpack y Eispack, desarrollados en el Argonne National Laboratory.

El nombre de MatLab proviene de la contracción de los términos MATrix LABoratory y fue inicialmente concebido para proporcionar un acceso sencillo a las bibliotecas Linpack y Eispack, dos de las más importantes en lo que a computación y cálculo matricial se refiere. A lo largo de los años, fue complementado e implementado en lenguaje C. Actualmente la licencia de MatLab es propiedad de MathWorks Inc.

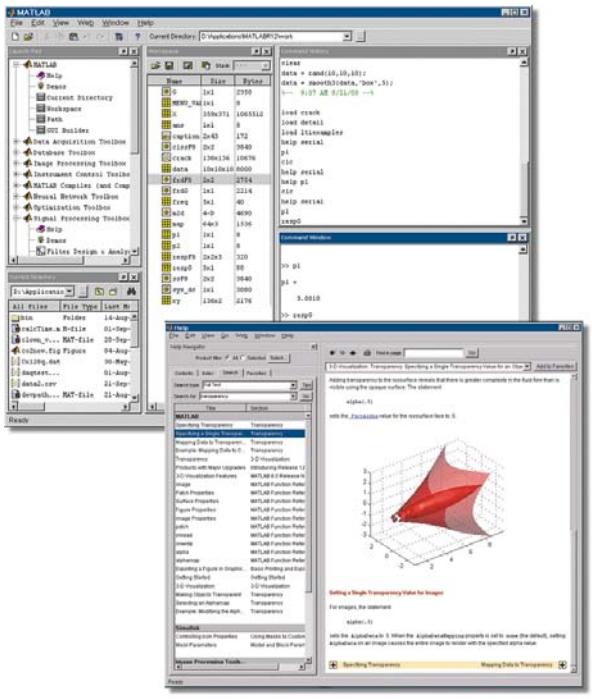


Fig. 7 Entorno MatLab.

Hoy en día, MatLab goza de un alto nivel de implantación en escuelas y centros universitarios, así como en departamentos de investigación y desarrollo de muchas compañías nacionales e internacionales. En entornos universitarios, por ejemplo, se ha convertido en una herramienta básica, tanto para los profesionales e investigadores de centros docentes, como una importante herramienta para la docencia de temas tales como sistemas automáticos, álgebra lineal, procesamiento digital de imágenes, etc.

En el mundo industrial, está siendo utilizado como herramienta de investigación para la resolución de problemas complejos planteados en la realización y aplicación de modelos matemáticos en ingeniería.

Además, dispone de un amplio abanico de programa de apoyo especializados, denominados "*toolboxes*", que extienden significativamente el número de funciones incorporadas en el programa principal. Estos "*toolboxes*" cubren prácticamente casi todas las áreas principales en el mundo de la ingeniería y la simulación.

MatLab se encuentra disponible para un amplio número de plataformas: estaciones de trabajo SUN, Apollo, VAXstation y HP, VAX, MicroVAX, Gould, Apple Macintosh y PC AT compatibles 80486 o superiores; operando bajo sistemas operativos UNIX, Macintosh y Windows.

5.2. Simulink

Simulink es un paquete de software para modelar, simular y analizar sistemas dinámicos. Soporta sistemas lineales y no lineales, modelados en tiempo continuo, muestreados o un híbrido de los dos. Los sistemas pueden ser también multifrecuencia, es decir, tienen diferentes partes que se muestrean o actualizan con diferentes velocidades.

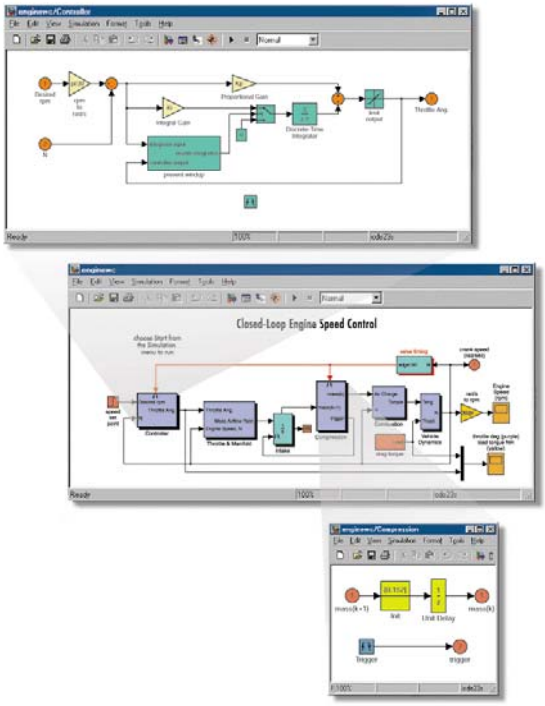


Fig. 8 Entorno Simulink

Para modelar, Simulink proporciona una interfaz de usuario gráfica (GUI) para construir los modelos como diagramas de bloques, utilizando operaciones como el ratón del tipo pulsar y arrastrar (figura 8). Con esta interfaz, los modelos se representan de forma similar a como se dibujan con el lápiz y el papel o como representan la mayor parte de los libros de texto. Esto supone un cambio radical respecto a los paquetes de simulación previos, que requieren una formulación de las ecuaciones diferenciales o en diferencias en forma de lenguaje o programa. Simulink incluye una amplia biblioteca de sumideros, fuentes, componentes lineales y no lineales y conectores.

Además, es posible generar modelos jerárquicos usando una metodología descendente y ascendente. Se puede visualizar el sistema en un nivel superior, desde donde mediante un doble clic sobre los modelos se puede ir descendiendo a través de los niveles a fin de ver con más detalle el modelo. Esto proporciona una comprensión de cómo se organiza un modelo y cómo interactúan sus partes.

Después de definir un modelo, se puede simular e interactuar con dicha simulación a través de los diferentes menús que ofrece el programa. Utilizando bloques específicos (*Scopes*) y otros bloques de visualización se pueden ir visualizando los resultados de la simulación mientras ésta se está ejecutando. Además, es posible cambiar los parámetros y ver de forma inmediata lo que sucede tras el cambio; pudiendo transferir los resultados de la simulación al espacio de trabajo del programa MatLab, para su posterior post-procesamiento y visualización.

5.3. CodeSimulink

CodeSimulink permite que el diseñador describa los sistemas mediante bloques tipo Simulink, a los cuales se les añade unas propiedades adicionales, a fin de describir el hardware o software que los constituyen. De esta forma, un ingeniero de desarrollo puede describir de una forma relativamente sencilla sistemas compuestos de:

- Un sistema, normalmente electromecánico, que ha de ser controlado.
- Un entorno exterior, que generará órdenes y eventos sobre el sistema anterior.
- Una interfaz gráfica que ha de ejecutarse sobre un PC o estación de trabajo.

Gracias al uso de esta herramienta, es posible desarrollar de una forma rápida todo el software y hardware necesarios que constituyen un sistema, reduciendo drásticamente la duración de la fase de desarrollo. Para ello, el programa consta del siguiente conjunto de bloques (blocks) que, a modo de ladrillos, permiten construir el sistema:

- *Bloques externos (external blocks)*: se emplean para simular partes electromecánicas de los sistemas.
- *Bloques hardware (HW blocks)*: usados para implementar la zona hardware, que habitualmente es sobre la que se implementan los componentes que demandan un funcionamiento a elevada velocidad.
- Bloques software (*SW blocks*): usados para implementar tareas que habitualmente no tienen fuertes restricciones temporales.
- Interfaz de usuario (*user interface*): que permite una comunicación entre el sistema y el usuario final.

Una vez que se ha descrito todo el sistema en términos de bloques, es posible realizar una simulación muy precisa del funcionamiento del mismo. Los resultados de tal simulación pueden usarse para ajustar los posibles parámetros del diseño, de tal forma que es posible una comunicación total entre el diseñador y la herramienta, a fin de lograr el diseño óptimo.

Una vez que mediante las simulaciones se ha comprobado que la operatividad del sistema satisface todos y cada uno de los requerimientos solicitados en la fase de especificaciones, se genera la implementación del diseño final. Aquí, lo que es la interfaz de usuario queda inalterada, cambiando lo que son las rutinas que existen bajo ella. Los bloques hardware y software se traducen a lenguajes VHDL y C respectivamente, y a continuación se compilan, generando los archivos de configuración que constituirán el sistema.

5.4. Xilinx System Generator

System Generator posibilita simulaciones reales de diseños basados en FPGAs mediante modelos Simulink. Además, genera de forma automática, partiendo de un modelo realizado en Simulink, una implementación basada en lenguajes de descripción de hardware (HDL). Para ello, el sistema convierte un conjunto de bloques adicionales, que extienden el programa Simulink, a una serie de '*cores*' optimizados por Xilinx.

Las principales características de esta herramienta se muestran a continuación:

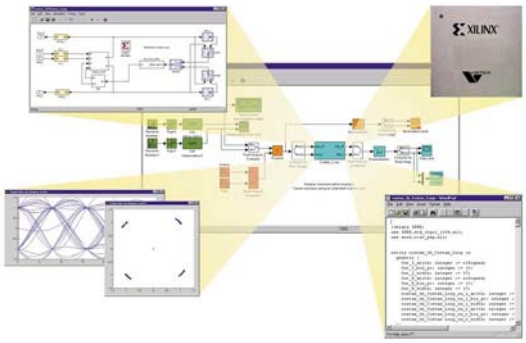


Fig. 9 Xilinx System Generator

- Permite representar a nivel de sistema diseños basados en FPGAs. Para ello facilita una interfaz gráfica muy sencilla de usar y un conjunto de bibliotecas que encapsulan las funciones más usuales.
- Permite una generación automática de código. Bien VHDL sintetizable, obtenido a partir del modelo Simulink realizado con los bloques suministrados, bien una serie de ficheros script de ModelSim, bien una serie de bancos de pruebas (*test-benches*) en VHDL.
- Flujo de diseño optimizado, gracias a la conversión de los modelos a cores optimizados de Xilinx, llevada a cabo por el sistema CORE Generator.
- Ofrece una surtida biblioteca de modelos, mediante bloques parametrizables asociados a funciones aritméticas, lógicas o incluso DSPs.
- A la par de la implementación del sistema, el programa genera también de forma automática, una serie de bancos de prueba (*test-benches*) y vectores de datos.

7. Conclusiones

A lo largo del presente artículo se muestra cómo las técnicas de codiseño permiten reducir drásticamente las fases de desarrollo de sistemas complejos. Asimismo, se ha dado una perspectiva de las diferentes herramientas basadas en esta metodología, que actualmente se hallan presentes en el mercado.

Se ha analizado cómo aquéllas herramientas basadas en el lenguaje C o que permiten una modelización a nivel de sistema y es posible comunicar con lenguaje C (figura 10), como las basadas en el Simulink, presentan una cierta ventaja respecto al resto.

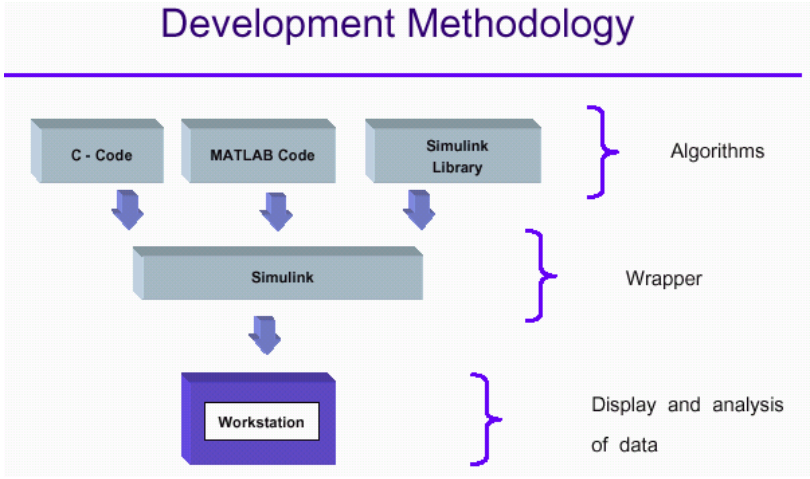


Fig. 10 Comunicación Entornos Matlab/Entornos C

8. Bibliografía

Boemo, E. (2003). “Apuntes del curso intensivo FPGAs de Xilinx”. Universidad Autónoma de Madrid.

Celoxica, (2002). “Handel-C Language Reference Manual v.3.1”. Celoxica.

Coware, (2004). http://www.coware.com

Gartner, (2000). “Electronic Design Automation Worlwide 2000”. DoD VHDL Project.

Gómez, D., Galindo P. (1998). “Tendencias en síntesis de sistemas electrónicos”. Servicio de publicaciones de la Universidad de Cádiz, 1998.

Hardee P., (2002). “SystemC: a realistic SoC debug strategy”. EETimes, CMP Media.

Kernighan B. W., Ritchie D. M.(1988) . “The C programming language”. Prentice Hall, Inc.

Pardo Carpio, F. (1997). “VHDL Lenguaje para descripción y modelado de circuitos”. Universidad de Valencia.

STOC, (2002). “Spec-C Language Reference Manual v. 1.0”, STOC.

SystemC, (2004). http://www.systemc.org

Synopsys, (2000a). Synopsys Inc, CoWare Inc, and Frontier Design Inc. SystemC, version 1.0 User’s Guide.

Synopsys, (2000b). Synopsys Inc, CoCentric SystemC Compiler, Behavioral Modeling Guide.

Synopsys, (2000c). Synopsys Inc, CoCentric SystemC Compiler, User Guide.

Swan, (2001). S. Swan et al. Functional Specification for SystemC 2.0.

Van den Bout, D. (1999). “The practical Xilinx designer lab book”. Ed Prentice Hall.