5 Propuesta software

5.1 Herramientas de diseño, control y depuración

En el mercado actual existen numerosas herramientas para diseño digital en FPGAs, pero las principales son las que pasamos a analizar a continuación.

5.1.1 Xilinx ISE

Esta es la herramienta por excelencia de Xilinx para el diseño, simulación, síntesis e implementación de dispositivos sobre FPGAs de la familia Xilinx. Permite la descripción de circuitos mediante los llamados lenguajes de descripción hardware (VHDL o Verilog), y simulaciones de distintos niveles. Incluye una potente herramienta que facilita la integración de los IP (*Intelectual Property*). Los IP son bloques que ya han sido diseñados y que se pueden utilizar o bien de forma gratuita o bien adquiriendo la licencia, dependiendo del bloque. Este kit de desarrollo soporta todas las FPGAs de las distintas familias de Xilinx. Mediante la herramienta iMPACT se puede crear y descargar el *bitstream,* así como en otros dispositivos, memorias, CPLD…

5.1.2 EDK

EDK (*Embedded Development Kit*) es una herramienta de Xilinx especialmente ideada para el diseño de sistemas empotrados. Dentro de este *kit* se incluye todo lo necesario para el desarrollo de plataformas basadas en FPGAs de Xilinx, incluyendo los procesadores PowerPC y Microblaze. Incluye soporte para línea de comandos y *Graphical IDE* para desarrollar y depurar plataformas HW/SW para aplicaciones empotradas. Gracias al uso del asistente se puede configurar de forma rápida el procesador empotrado, los buses y los periféricos.

En cuanto a las herramientas para el desarrollo software incluye el compilador y depurador GNU C/C++, la utilidad Data2MEM para la carga y actualización del *bitstream*, *Base System Builder,* soporte para diferentes placas y *Platform Studio SDK* (*Software Development Kit*). Además contiene un paquete BSP (*Board Support Package*) que soporta numerosas *third parties* de diferentes fabricantes como *Mentor, Wind River, Green Hill, LynuxWorks* y otros líderes de la industria de sistemas empotrados. Esta herramienta de desarrollo no es libre y es necesario adquirir una licencia para su uso.

5.1.3 Quartus II

Es una suite de Altera que permite el diseño de circuitos digitales mediantes los lenguajes de descripción hardware (HDL). Es el equivalente de Altera al software de xilinx ISE. Soporta todas las familias de FPGAs y CPLDs de Altera.

Quartus permite al diseñador crear sus diseños, realizar análisis temporales, examinar diagramas RTL y cargar el diseño en el dispositivo que corresponda.

5.1.4 SOPC Builder

Se trata de una herramienta exclusiva de Altera que permite crear sistemas y evaluar sistemas empotrados. Todas las versiones de Quartus II integran este software. Permite incluir de forma automática los IP de Altera (*Intelectual Properties*). También facilita la labor de conexión con procesadores externos mediante el uso de Gigabit Ethernet, SerialRapidIO, PCI o PCI Express.

Con esta herramienta se pueden crear procesadores empotrados e incluirlos fácilmente en el sistema que se esté diseñando.

5.1.5 Selección del HDL para realizar los diseños

Existen varios HDLs que pueden utilizarse para realizar la descripción de un diseño. Los dos lenguajes más extendidos y mejor soportados son Verilog y VHDL, aunque existen muchos otros como AHDL (Altera HDL, lenguaje propietario de Altera) o SystemC. Ambos lenguajes son bastante potentes, y el hardware modelado con uno de ellos puede ser también modelado con el otro. La elección del HDL no se basa, por tanto, en la capacidad técnica, sino en otros aspectos como las herramientas software de diseño disponibles. Debido a que el VHDL está ampliamente extendido, y el software Xilinx ISE Webpack es de uso común en muchas instituciones universitarias y proyectos de gran importancia, es la mejor alternativa para los objetivos planteados.

El lenguaje VHDL está definido en el estándar IEEE 1076. Algunas herramientas propietarias muy utilizadas han implementado características que violan este estándar.

5.2 Xilinx ISE Webpack

Las versiones de ISE (Integrated Software Environment) de Xilinx conforman un entorno integrado de desarrollo (IDE: Integrated Development Environment) que permite recorrer fácilmente las diferentes fases del proceso de desarrollo de un circuito

lógico, desde el diseño hasta la implementación sobre una arquitectura reconfigurable. ISE se encarga de llamar de manera automática a las diferentes utilidades y herramientas suministradas dependiendo de la etapa de diseño en la que nos encontremos. Todos estos programas disponen de una completa ayuda y manuales correspondientes, que se pueden encontrar en la página web de Xilinx. ISE se autodenomina Navegador de Proyecto (Project Navigator) y está orientado a facilitar el proceso de desarrollo.

5.2.1 Flujo de diseño

En esta sección se presenta el flujo de diseño completo para lógica reconfigurable de Xilinx. Podemos distinguir las siguientes etapas:

1. *Introducción del diseño.* Modelado de un circuito lógico (HDL, esquemático, EDIF, máquina de estados, etc.).
2. *Imposición de restricciones.* Asignación de pines, limitaciones temporales.
3. *Síntesis del diseño.* Traducción a puertas lógicas.
4. *Implementación del diseño.* Generación del programa que configura un determinado dispositivo destino (FPGA o CPLD).
5. *Configuración del dispositivo.* Carga del programa en el dispositivo destino.

La figura siguiente ilustra lo dicho indicando las diferentes fases de desarrollo según la enumeración anterior. Dicha enumeración no significa que necesariamente haya que seguir ese orden de principio a fin, más bien es un proceso en el que se avanza y retrocede iterativamente hasta alcanzar el objetivo.

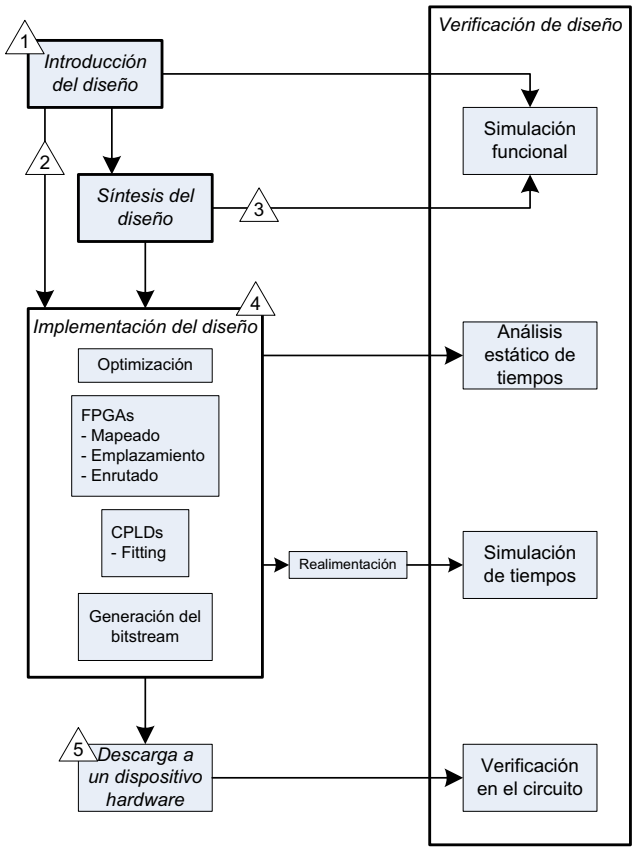

Imagen 9. Flujo de diseño Xilinx simplificado

Cada una de las etapas del flujo de diseño puede realizarse dentro del entorno integrado o bien utilizar herramientas de terceros. Algunas de esas herramientas también son integrables en ISE si están convenientemente instaladas. Por ejemplo, para la síntesis de circuitos, la herramienta XST (Xilinx Synthesis Technology) está integrada en el paquete ISE, mientras que otras herramientas como *Precision Synthesis* o *Synplify* son programas que se adquieren de forma independiente.

Como se mencionó en el apartado anterior, la herramienta Xilinx ISE está orientada al flujo de diseño, de manera que se facilite en lo posible el proceso de desarrollo. Podemos observar este hecho en el árbol de procesos que en todo momento tenemos presente en la parte izquierda de la ventana del Project Navigator, como se muestra en la siguiente figura.

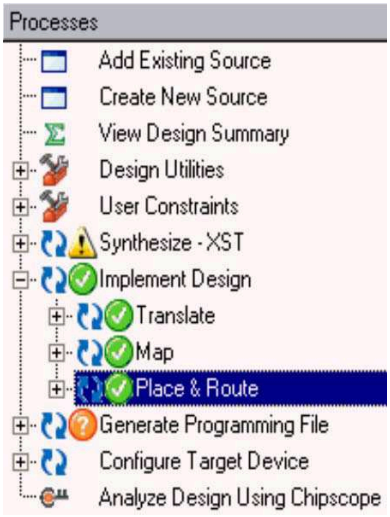

Imagen 10. Árbol de procesos del Xilinx ISE Webpack

Conforme se van realizando las diferentes etapas, aparecerán marcas junto a cada una de ellas, indicando su estado, si han sido completadas completamente, si han finalizado con errores o *warnings*, o bien si necesitan ser ejecutadas de nuevo debido a que están desincronizadas con la versión del archivo fuente, por ejemplo.

5.2.2 Arranque y empleo de la aplicación ISE WebPack

El ISE WebPack v10.1 es una edición libre, descargable de la página web de Xillinx. Como hemos dicho, este pack integra numerosas herramientas destinadas a la realización de nuestro diseño, pasando por todas las etapas del flujo de diseño. Tras la ventana de bienvenida aparece la ventana principal del entorno, en la que se distinguen cuatro partes:

- Panel *Sources*, situado en la parte superior izquierda de la ventana. Aquí se muestra el nombre del proyecto, el dispositivo especificado, y los documentos de usuario, ficheros fuente, y demás ficheros (por ejemplo, *test bench*) asociados con dicho proyecto. Los ficheros fuente aparecen jerárquicamente agrupados, de manera análoga a como están emplazados en el diseño.

- Panel Processes, situado justo debajo del anterior. Este panel representa el flujo de diseño con las acciones que se pueden realizar en función del objeto seleccionado en el panel de ficheros fuente. Al hacer doble clic en cualquier elemento del árbol, se ejecutan los procesos necesarios para llevar a cabo ese paso.

    - Añadir una fuente existente
    - Crear nueva fuente
    - Ver informe de diseño

  - Utilidades de diseño
  - Restricciones de usuario. Acceso a la edición de restricciones de tiempo y localización.
  - Síntesis. Desde aquí, se puede comprobar la sintaxis, realizar la síntesis, ver el esquemático y los informes de síntesis.
  - Implementar diseño. Herramientas de implementación e informes de flujo de diseño.
  - Generar archivo de programación. Da acceso a las herramientas de configuración y a la generación del *bitstream* o fichero a cargar en la FPGA.

- Panel *Transcript*, situado en la parte inferior. Su pestaña principal, *Console*, muestra errores, *warnings* y mensajes de información, la salida de texto generada por las diferentes utilidades que van siendo llamadas en el proceso de diseño.

- El panel principal, *Workspace*, ocupa la mayor parte de la ventana del *Project Navigator*, y puede mostrar las siguientes ventanas:

  - Informe de diseño (*Design Summary*), donde podremos ver información de alto nivel del proyecto, como información general, un informe de utilización del dispositivo, datos de rendimiento, información de restricciones y enlaces a todos los informes individuales generados en el proceso de diseño.

  - Editor de texto configurable, con el ISE Text Editor por defecto, que puede editar archivos fuente y otros documentos de texto.

  - Simulador ISE / Editor de formas de onda, que se usa para crear y simular bancos de prueba (*test bench*). En este último se pueden introducir estímulos gráficamente y la respuesta esperada.

  - Editor de esquemáticos, puede crear y ver gráficamente diseños lógicos.

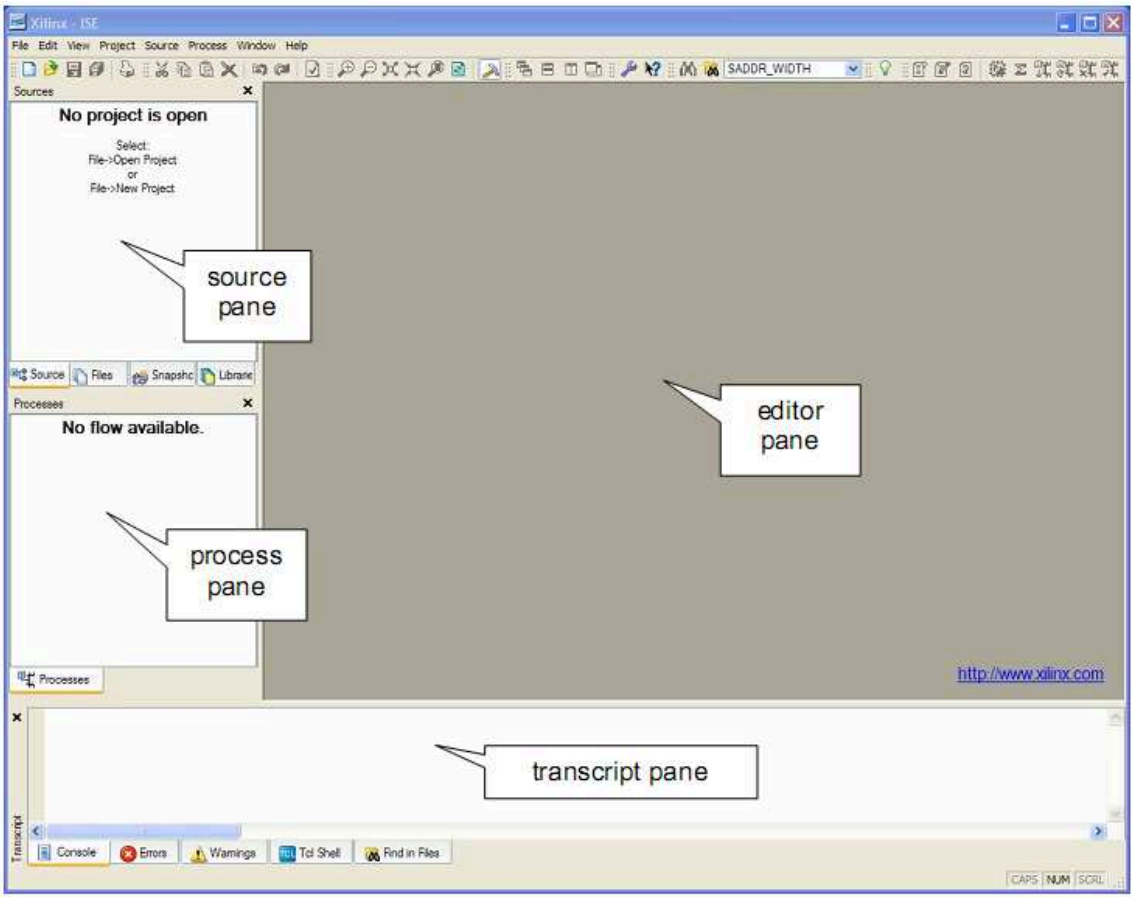

Imagen 9. Ventana principal de Xilinx ISE Webpack

Cada vez que abrimos la aplicación se carga el último proyecto tratado mostrando el estado correspondiente a las actuaciones realizadas en cada una de las ventanas.

5.2.3 Creación de un proyecto

En el entorno de desarrollo ISE un proyecto es un conjunto de unidades de diseño -modelos en lenguajes de descripción de *hardware*, esquemáticos, etc.-, unas restricciones (*constraints*) de implantación física y una arquitectura de dispositivo reconfigurable. Además de esto, el proyecto integrará en el entorno algunas herramientas de simulación y síntesis que seleccionará el usuario de entre las nativas de Xilinx y las proporcionadas por terceros.

Para crear un proyecto, se debe hacer lo siguiente:

1. Iniciar la aplicación ISE WebPack.

2. Seleccionar la opción *File / New Project…*. Aparecerá la ventana de dialogo del asistente para nuevo proyecto. En esta ventana se ingresa el nombre del proyecto, la ruta del nuevo proyecto, y tipo de fichero fuente de entre los disponibles (en el que seleccionaremos lenguaje de descripción de *hardware* (HDL)).

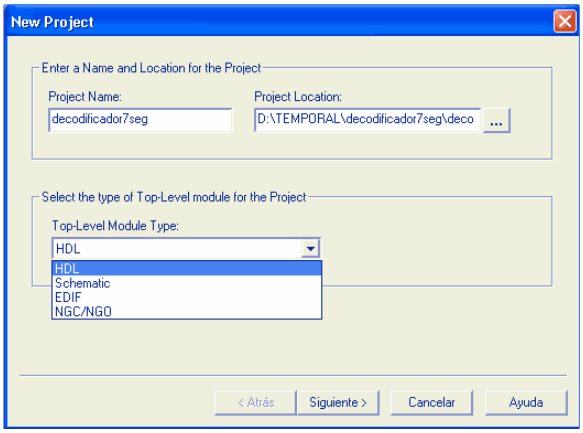

Imagen 11. Primera ventana de dialogo para la creación de un proyecto.

3. En el siguiente paso del asistente, aparece la ventana de diálogo de la figura siguiente en la que se determina el dispositivo reconfigurable destino, su encapsulado y grado de velocidad. Las herramientas de síntesis (*XST* de *Xilinx*) y simulación (*ISE Simulator*) aparecen automáticamente si están instaladas en nuestro ordenador. También se selecciona el lenguaje HDL concreto que se va a utilizar en los ficheros fuente.

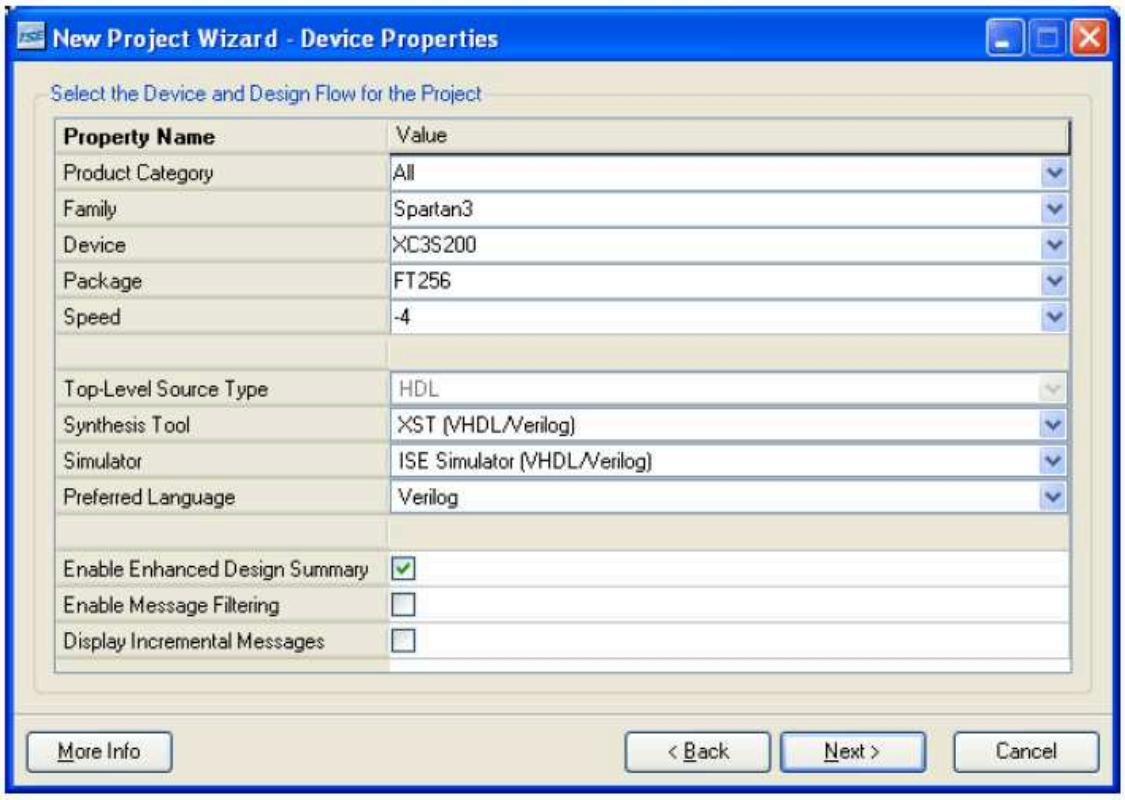

Imagen 12. Ventana de diálogo Device Properties en la creación de un proyecto.

4. A continuación, se creará el fichero HDL de nivel superior del diseño. Hacemos clic en *New Source*, y seleccionamos *VHDL module* como origen. Escribimos el nombre del fichero, asegurándonos de que *Add to project* está marcado.

5. En el siguiente paso, se ha de rellenar la información sobre los puertos en la ventana de definición de módulo que se muestra en la figura siguiente:

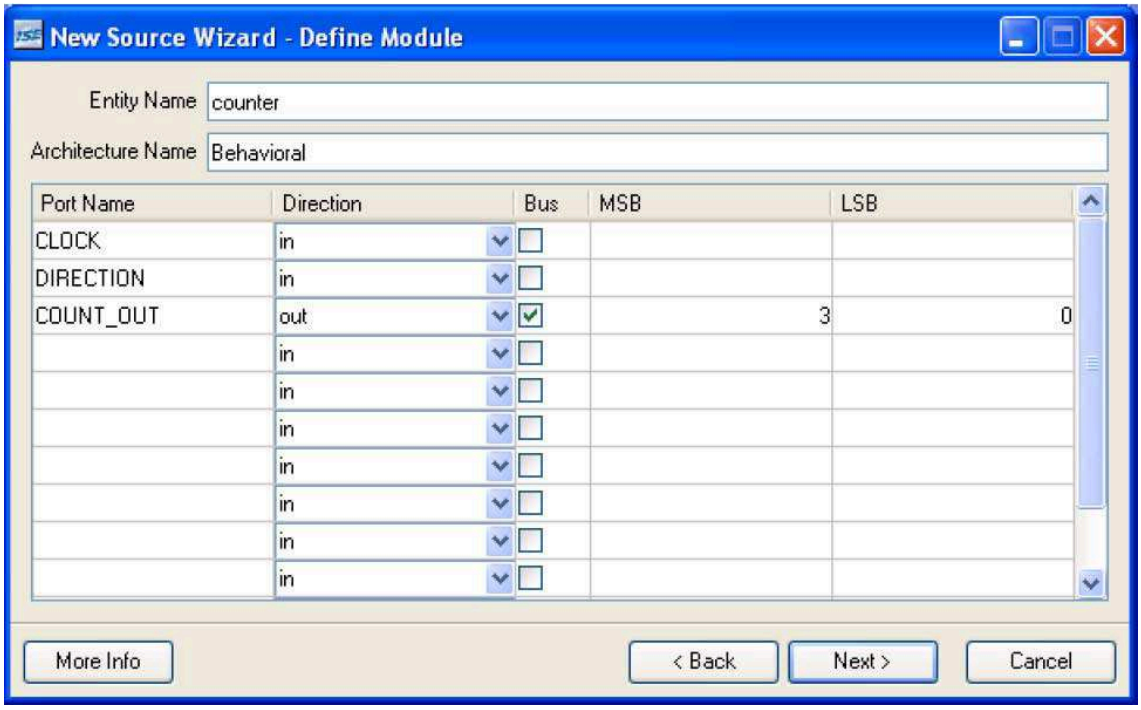

Imagen 13. Definición de los puertos de un módulo.

6. Si finalizamos el asistente, aparecen en el *Workspace* el archivo fuente que contiene el par entidad / arquitectura, y el módulo con el nombre elegido se muestra en la pestaña *Source*, como se ve en la figura inferior:

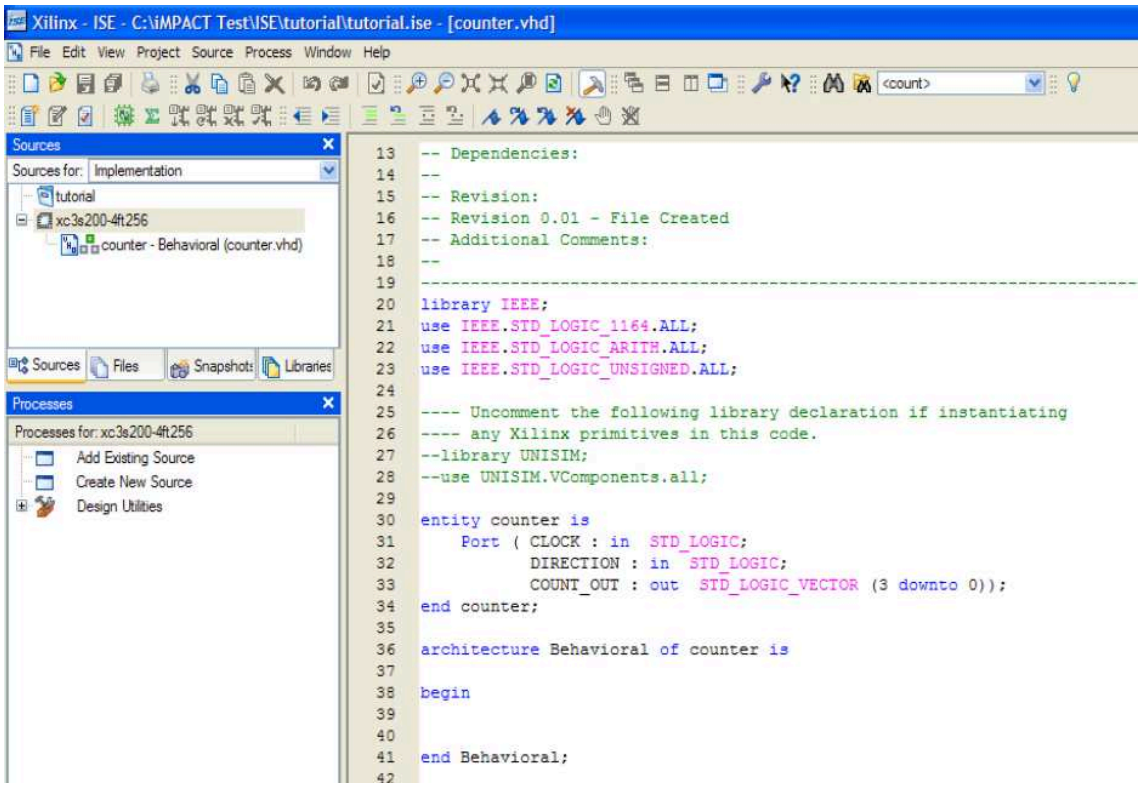

Imagen 14. Proyecto creado en ISE.

7. A continuación, hay que añadir la descripción del comportamiento del módulo a diseñar, en el apartado *Behavioral* del fichero .vhd, añadiendo las señales y sub-módulos necesarios de la misma forma.

8. Una vez terminados los ficheros fuente, se ha de comprobar la sintaxis del diseño para encontrar errores y fallos tipográficos. Para ello, con la opción *Implementation* seleccionada del menú desplegable *Sources*, seleccionamos el módulo correspondiente, y hacemos doble clic en el proceso *Synthesize-XST/Check Syntax* de la parte de *Processes*. Los errores se pueden comprobar en la pestaña *Console* de el panel *Transcript*; dichos errores han de ser corregidos para que el diseño se pueda simular o sintetizar.

La creación del proyecto tiene las siguientes consecuencias:

1. Se crea el fichero de extensión *.npl* de configuración del proyecto en el directorio del proyecto.

2. Aparecen los ficheros fuente en el panel de fuentes. Además de los ficheros de modelos y restricciones tenemos una entrada para el dispositivo reconfigurable. Con el menú contextual podemos crear un nuevo fichero fuente, añadir uno existente, abrir uno presente con su editor específico, eliminar fuentes o consultar las propiedades de cada elemento.

3. Aparece en el panel de procesos el flujo de diseño específico del dispositivo destino. Cada una de las entradas del mismo pueden ejecutarse. Si el resultado es correcto se marcará con el símbolo ✓ mientras que si genera errores se indicará con el símbolo ×.

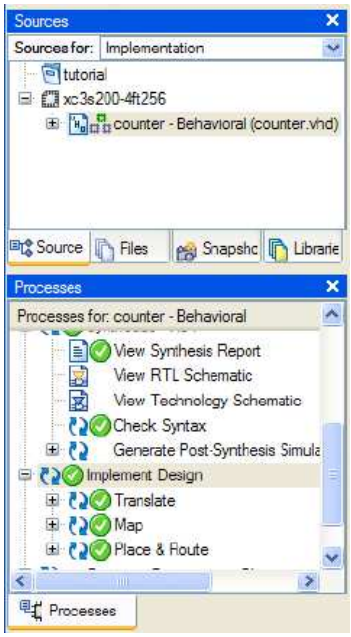

Imagen 15. Ventanas de fuentes y procesos.

5.2.4 Verificar la funcionalidad usando simulación (Behavioral Simulation)

A la hora de realizar un diseño VHDL, el ISE Webpack nos permite realizar simulaciones, tanto de bloques concretos como del diseño completo, para poder verificar su funcionamiento antes de cargar el diseño en la placa. De esta forma, se puede ver el valor de una señal en cualquier momento, permitiendo de esta manera ver en detalle el comportamiento de nuestro circuito de cara a correcciones y posibles mejoras.

Para realizar una simulación del comportamiento de nuestro circuito, hemos de realizar un *test bench* o banco de pruebas, en el cual se indican explícitamente las entradas del sistema a lo largo del tiempo.

Para realizar una forma de onda del *test bench*, hacemos clic en *Project / New Source*, donde elegiremos *Test Bench WaveForm*, a continuación asociaremos al fichero fuente correspondiente y finalizaremos el asistente. A continuación, se han de configurar diferentes aspectos relacionados con el reloj y la temporización en el cuadro *Initial Timing and Clock Wizard*, que se puede ver en la siguiente figura:

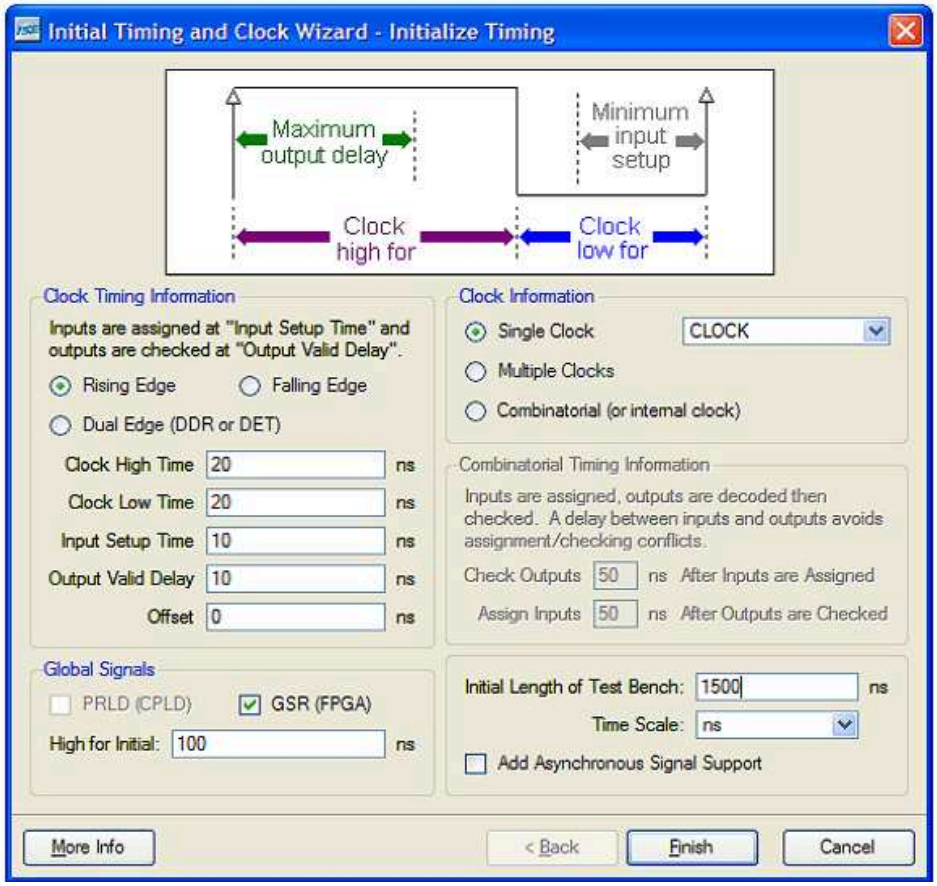

Imagen 16. Configuración de reloj y temporización inicial

Una vez finalizado este asistente, podemos ver que el archivo de forma de onda se ha añadido automáticamente al proyecto al seleccionar la opción *Behavioural Simulation* en el panel *Sources*, como muestra la siguiente figura:

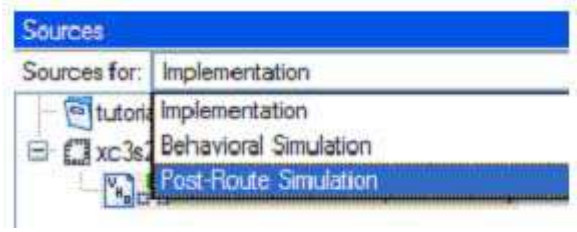

Imagen 17. Seleccionar *Behavioral Simulation*

Haciendo doble clic en el archivo de forma de onda, podremos modificar su contenido en la parte derecha de la pantalla del ISE Webpack, donde se procederá a especificar los valores que tomarán las entradas a lo largo del tiempo y que estimularán a nuestro diseño.

Una vez concluida la edición de dicho fichero, y tras guardar los cambios, podemos pasar finalmente a simular la funcionalidad del diseño. Para ello, con el archivo de forma de onda seleccionado en el panel *Sources*, haremos doble clic en el proceso *Xilinx ISE Simulator process / Simulate Behavioral Model*, dentro de la pestaña *Processes*. En este momento, se abre el ISE Simulator y se ejecuta la simulación hasta el final del *test bench*. Una vez finalizada, podremos ver los resultados de la simulación en la pestaña *Simulation*, mostrando una ventana de similares características a la que se puede ver en la siguiente figura:

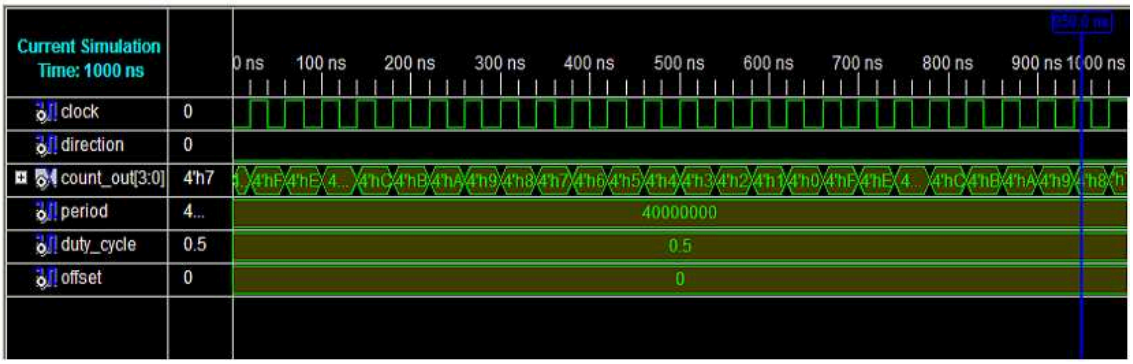

Imagen 18. Ventana de simulación

En esta ventana se muestra el tiempo en el eje horizontal, y las diferentes señales (en principio las entradas y salidas generales) en vertical; se pueden configurar ambos aspectos, representando las señales que queramos y en los periodos de tiempo que nos resulten útiles. También disponemos en la ventana de simulación de numerosas herramientas, tales como zoom o detección de flancos, para comprobar que el funcionamiento de nuestro diseño sea el esperado, o poder descubrir el origen de posibles fallos en este.

5.2.5 Crear restricciones de temporización

La temporización se especifica introduciendo restricciones que guían en el proceso de ubicación y rutado del diseño. Se recomienda introducir restricciones globales. La restricción de periodo de reloj especifica la frecuencia de reloj en la cual nuestro diseño debe operar dentro de la FPGA. Las restricciones de *offset* especifican cuándo se han de esperar datos válidos en las entradas de la FPGA y cuándo va a haber datos válidos en las salidas de esta.

Para establecer las restricciones del diseño, procedemos de la siguiente forma:

En el panel *Sources*, seleccionamos *Implementation* de la lista desplegable; a continuación, seleccionamos nuestro fichero fuente HDL. Hacemos doble clic en *User Constraints / Create Timing Constraints*; ISE Webpack ejecuta los pasos de síntesis y traducción y automáticamente crea un fichero .ucf, que se añadirá al proyecto cuando hagamos clic en *Yes* en el siguiente cuadro de diálogo que nos aparece:

Imagen 19. Confirmación de añadir fichero .ucf al proyecto

En el cuadro de diálogo *Timing Constraints*, rellenaremos los campos correspondientes a las restricciones que queramos añadir. Una vez hecho esto, seleccionando *Constraint Type / Timing Constraints* en la pestaña *Timing Constraints*, podemos ver las restricciones de tiempo recién creadas de manera similar a como se muestra en la siguiente figura.

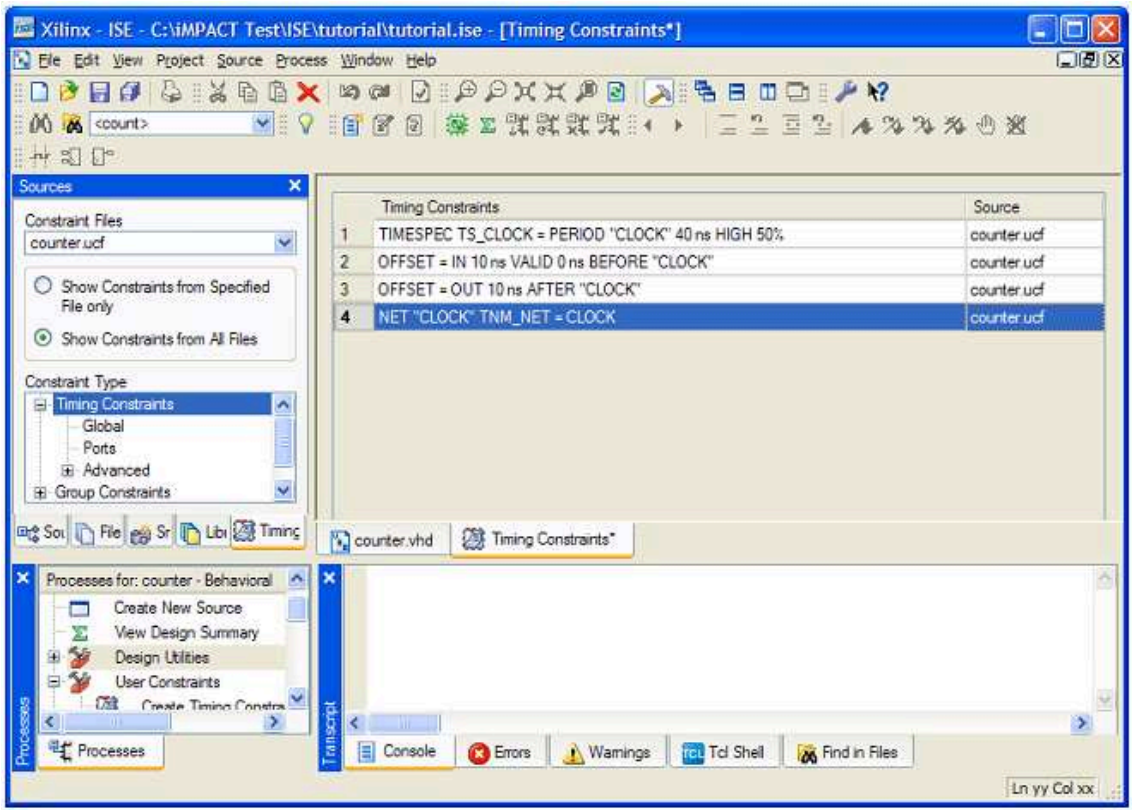

Imagen 20. Restricciones temporales

Finalmente, guardamos las restricciones temporales y cerramos el *Constraints Editor*. Por último, ya podemos pasar a la última fase, que es la de implementación del diseño.

5.2.6 Implementación del diseño

Seleccionamos el fichero fuente en el panel *Sources*. Podemos ver el resumen del diseño, haciendo clic en el proceso *View Design Summary* de la pestaña de procesos. Para que comience la implementación del diseño, hacemos doble clic en el proceso *Implement Design* en la pestaña *Processes*. Vemos que se van marcando con un símbolo ✓ de color verde si los diferentes pasos se ejecutan correctamente sin errores o *warnings*.

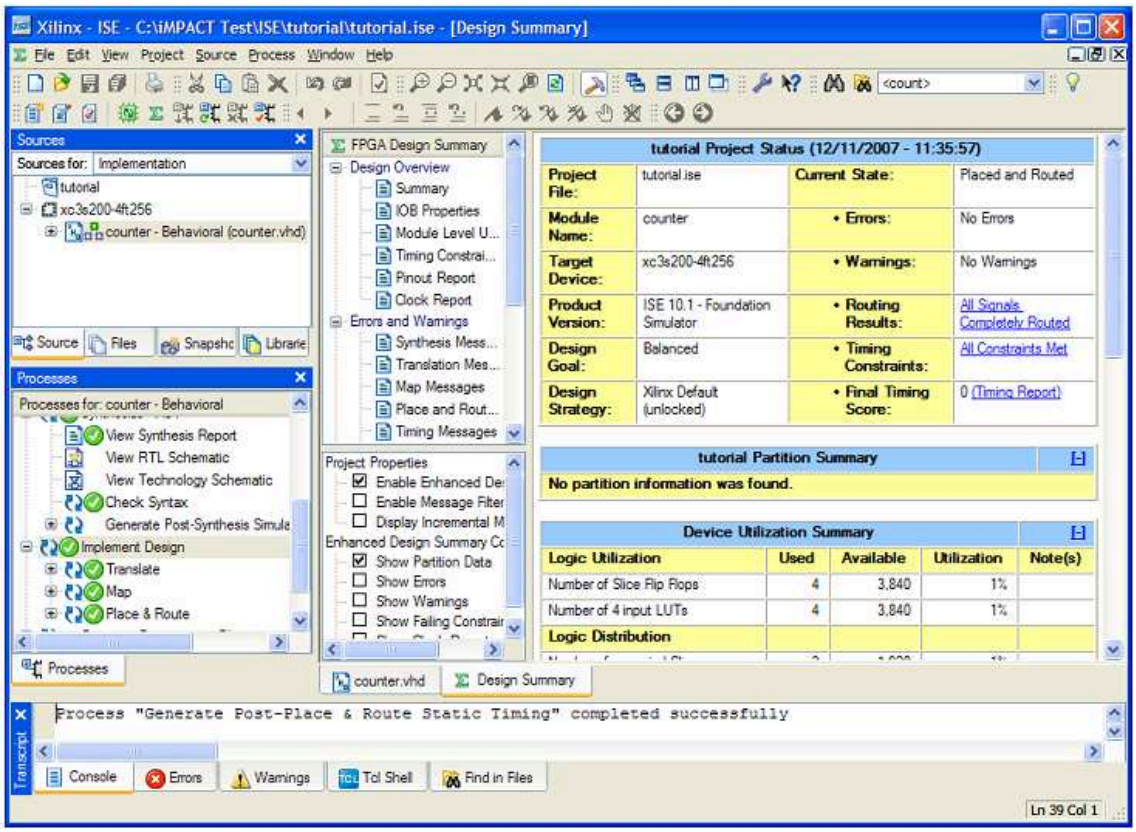

Imagen 21. Informe de diseño al finalizar la implementación

En la tabla *Performance Summary*, cerca del final del *Design Summary*, hacemos clic en el link *All Constraints Met* en el campo *Timing constraints*, donde podremos ver si nuestro diseño cumple las restricciones temporales establecidas.

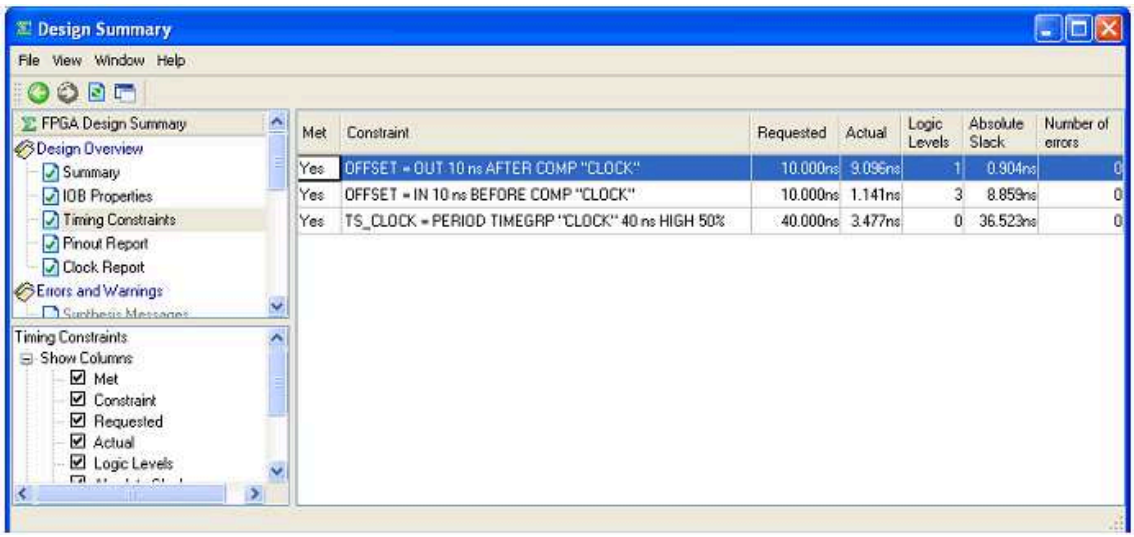

Imagen 22. Informe de restricciones cumplidas

5.2.7 Asignación de restricciones de ubicación de pines

También se pueden establecer restricciones para la ubicación de pines, de manera que, por ejemplo, podamos ubicar las entradas y salidas de nuestro diseño a pines a nuestra elección. Para asociar puertos del diseño con los pines, hay que hacer lo siguiente: en el panel *Processes*, ejecutar el proceso *User Constraints / Floorplan Area/IO/Logic – Post Synthesis* haciendo doble clic en él; se abrirá el Xilinx PACE (Pinout and Area Constraints Editor). Seleccionando la pestaña *Package View*, podemos establecer una localización para cada pin en la columna *Loc*.

El editor gráfico genera automáticamente un fichero de asignación de pines en modo texto de extensión *.ucf*. También es posible editar directamente este archivo.

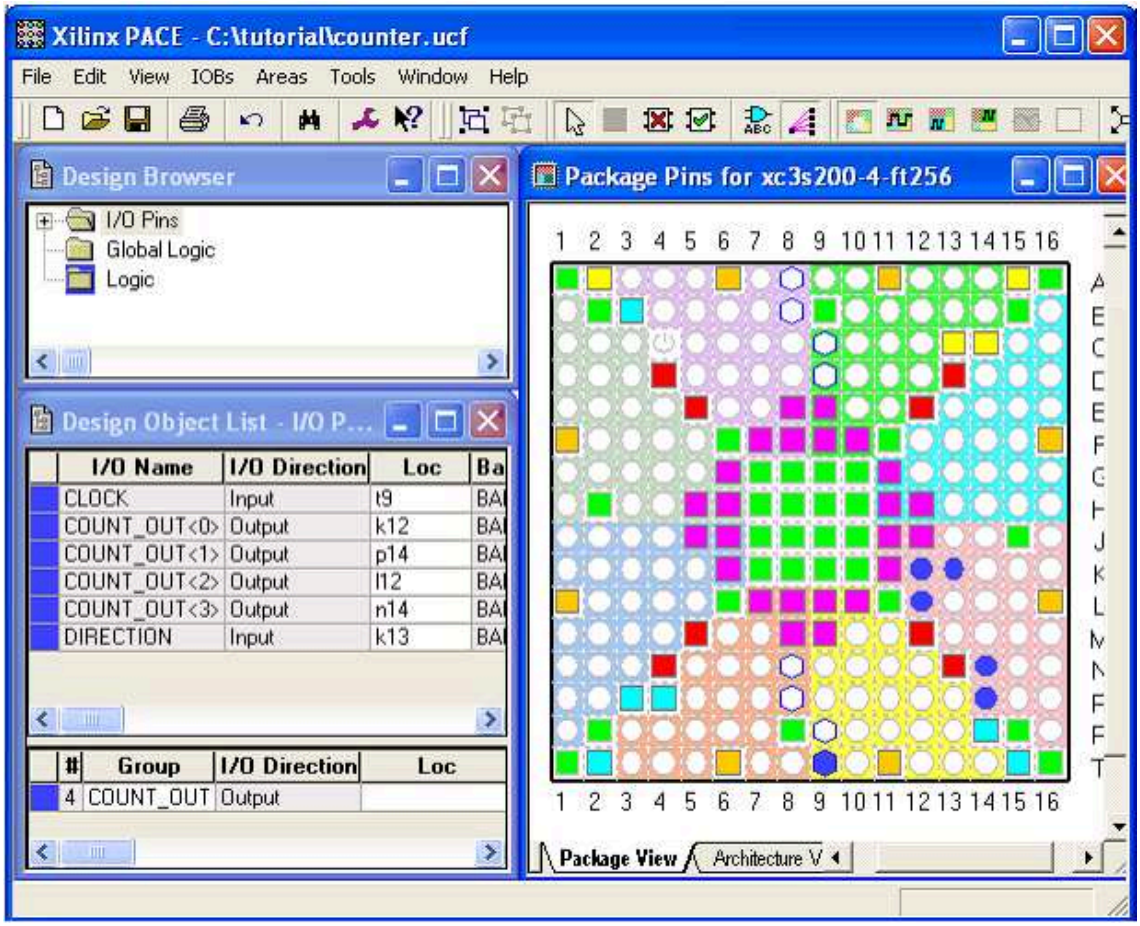

Imagen 23. Ubicación de pines

Conforme vayamos asignando las ubicaciones de los pines, se irán mostrando en azul en el esquemático anexo.

Al guardar, se nos preguntará la herramienta de síntesis que estamos usando, seleccionaremos *XST default*. Una vez en la ventana principal del ISE, nos daremos

cuenta que los procesos de *Implement Design* tienen un signo de interrogación naranja junto a ellos, indicando que no están fuera de fecha con uno o más ficheros de diseño; esto es debido a que el archivo *.ucf* ha sido modificado. En este punto, podemos reimplementar el diseño y verificar que los puertos del diseño se han enrutado hacia los pines especificados anteriormente.

Para llevar a cabo esta acción, hacemos doble clic en el proceso *Implement Design* del panel de procesos. A continuación, abrimos el Informe de Diseño haciendo doble clic en el proceso *View Design Summary* en el panel de procesos. Seleccionamos *Pinout Report* en el panel *FPGA Design Summary* de la izquierda, y ordenamos los resultados por el nombre de la señal, haciendo clic en la columna *Signal name*. Podemos comprobar que las señales están rutadas hacia los pines correctos, especificados en las restricciones anteriormente indicadas.

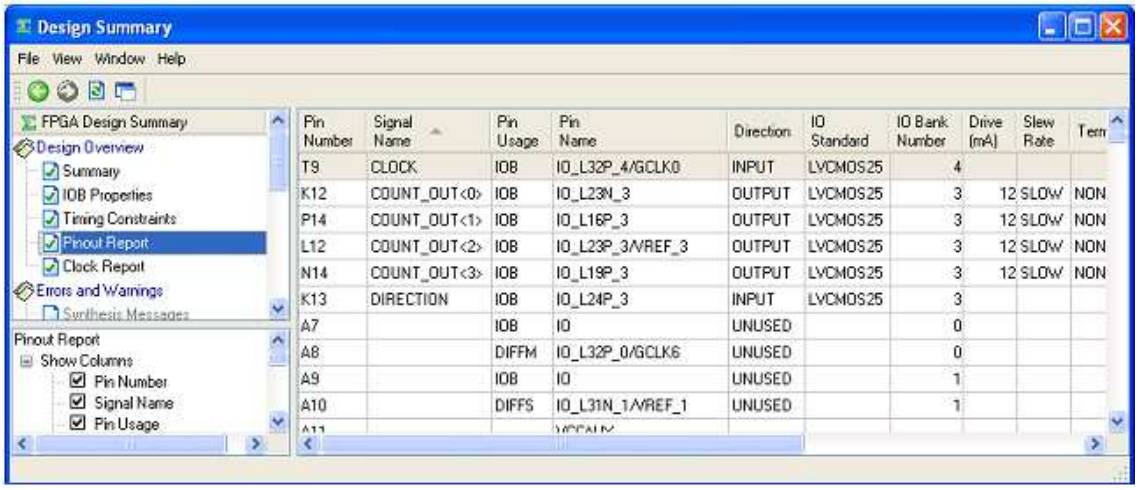

Imagen 24. Ventana de informe de asignación de pines

5.2.8 Carga del programa de configuración en la FPGA

La fase previa de implementación consiste en la creación del fichero de configuración. En este último paso, dicho fichero de configuración se carga en el dispositivo programable. Una vez cargado, le hará trabajar conforme a las especificaciones de nuestro diseño.

Para realizar esta carga del fichero de configuración, procederemos de la siguiente manera: en el panel *Sources*, seleccionar *Implementation* y a continuación, el fichero fuente de nuestro diseño. En el panel *Processes*, hacer doble clic en el proceso *Configure Target Device*. Entonces, se abre la utilidad iMPACT, mostrándose el cuadro de diálogo de configuración de dispositivos, donde seleccionaremos la manera en que vamos a cargar nuestra implementación en la FPGA. Seleccionaremos la opción *Configure devices using Boundary-Scan (JTAG)*, asegurándonos de que está

seleccionada la opción *Automatically connect to a cable and identify Boundary-Scan Caín.*

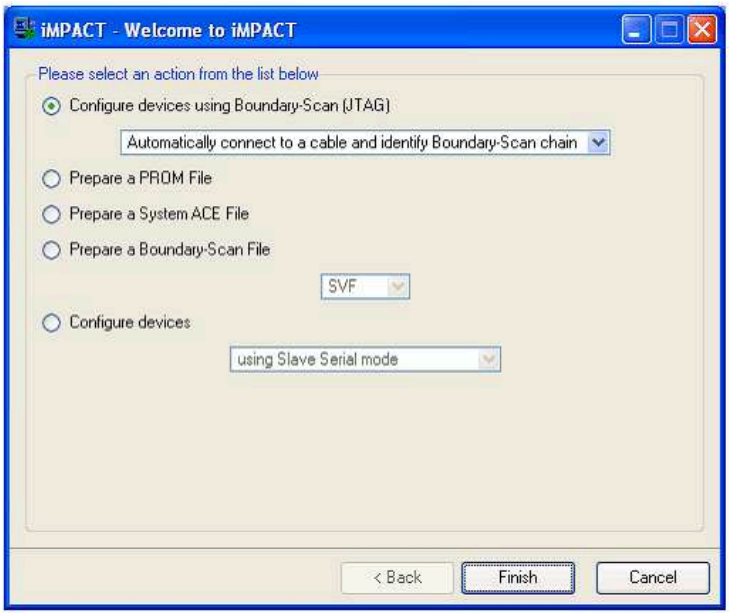

Imagen 25. Cuadro de diálogo de configuración de dispositivos de iMPACT.

Por último, hacemos clic en finalizar para salir del cuadro de diálogo; puede aparecernos un mensaje diciendo que se han detectado dos dispositivos; en ese caso, hacemos clic en *OK* para continuar. Los dispositivos conectados al cable JTAG se detectan y se muestran en la ventana de iMPACT.

Aparece entonces el cuadro de diálogo *Assign New Configuration File*. Para asignar el archivo de configuración (en el caso de una FPGA, se genera un archivo *.bit*) al dispositivo conectado al cable JTAG, seleccionaremos el archivo *.bit* y haremos clic en *Open*.

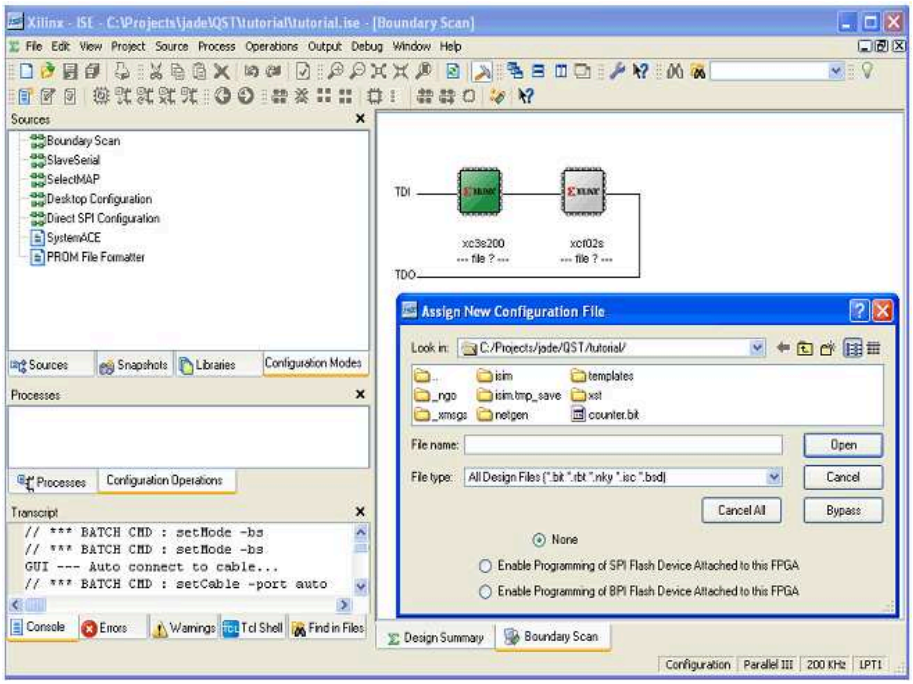

Imagen 26. Cuadro de diálogo de configuración de dispositivos de iMPACT.

En este cuadro, seleccionamos *Bypass* para descartar cualquier dispositivo restante. Hacemos clic con el botón derecho en la imagen del dispositivo, seleccionamos *Program...* en el menú emergente, con lo que aparece el cuadro de diálogo de *Programming Properties*, donde haremos clic en *OK* para programar finalmente el dispositivo, cargando el diseño en la FPGA. Cuando haya finalizado la programación, aparecerá un mensaje *Program Succeeded*, indicando que el proceso de carga en la FPGA ha finalizado correctamente.

5.3 Codiseño hardware/software

Tradicionalmente, se suelen desarrollar y luego analizar los requerimientos del sistema para determinar el nivel de complejidad de tecnología necesario para satisfacer estas necesidades. Por tanto, los equipos de trabajo de hardware y de software desarrollan independientemente sus diseños, combinándolos más tarde en una etapa avanzada del ciclo de desarrollo para la prueba del primer prototipo. Esto tiende a crear una plataforma hardware generalizada y software específico, y no optimiza el proceso en su conjunto.

El codiseño hardware/software busca mover la funcionalidad específica del software al hardware, para lo cual se beneficia de la habilidad del procesamiento paralelo de tareas que tiene el hardware, frente a la naturaleza secuencial de la ejecución de software. Esto tiende a crear una distribución balanceada de las especificaciones de la aplicación entre el hardware y el software, reduciendo la complejidad del software.

La existencia de un gestor de hilos software aporta los servicios y las estructuras de datos necesarias para conocer el estado actual de cada uno de los hilos de software del sistema. Este gestor coordina el acceso a dichos servicios para asegurar un correcto funcionamiento. Incorpora una cola de procesos listos para ejecutarse y un simple mecanismo FIFO de planificación. Además, aporta una interfaz para separar el módulo planificador para implementar otros algoritmos de planificación adicionales.

5.3.1 Introducción

Los sistemas empotrados juegan un papel muy importante en nuestra vida cotidiana, lo vemos en telecomunicaciones, instrumentación médica, automóviles, sistemas multimedia, etc., extendiéndose día a día y resolviendo problemáticas cada vez más complejas.

Para el desarrollo de los mismos, es necesario tener en cuenta los aspectos de hardware y de software desde que se elabora la idea. El área de codiseño hardware / software es la que se encarga de la planificación del sistema, tratando de desarrollar los métodos y herramientas que pueden usarse para mejorar la calidad de la aplicación final.

Todo proyecto de este tipo, comienza con una descripción del comportamiento que da origen a la especificación y a su simulación funcional, obteniendo luego una síntesis del código a utilizar. Existen distintas herramientas para resolver este problema, pero para su utilización debemos conocer qué tipo de modelo computacional maneja y su posibilidad de adaptación a la problemática planteada.

De las herramientas más utilizadas en esta área, PeaCE, Ptolemy y Simulink, se puede determinar que Ptolemy es una buena herramienta para el modelado y la simulación de sistemas pero su capacidad en la síntesis de código está muy restringida por la implementación del sistema; puede producir código C desde una representación de flujo de datos (DF) pero la calidad de código no es satisfactoria. Apenas podría producir un código sintetizable VHDL, pero no resultaría un código totalmente optimizado. No hay modo de sintetizar arquitectura desde Ptolemy.

Para solucionar todos estos problemas, se creó el proyecto PeaCE que es una extensión de Ptolemy en el ambiente de codiseño. Como está construido sobre la base de Ptolemy, básicamente PeaCE hereda todas las características positivas de Ptolemy, como la integración de diversos modelos de computación, capacidades poderosas de simulación especialmente para aplicaciones de Procesamiento de Señales Digitales (DSP), mejorando las restantes.

Simulink es un paquete de software que permite modelar, simular y analizar sistemas dinámicos, esto es sistemas cuyas salidas y estados internos cambien con el tiempo. Es un entorno gráfico, donde se crea un modelo en bloques del sistema, utilizando librerías

de bloques estándar y un editor que permite interconectar los bloques del sistema. El modelo representa gráficamente las relaciones matemáticas dependientes del tiempo a través de las entradas, estados y salidas del sistema. Con el agregado de paquetes adicionales, permite simular máquinas de estados finitos con jerarquía, *flowcharts*; generar código C para diferentes plataformas o código ADA, a partir del modelo del sistema.

Además de poder simular en tiempo real, permite implementar modelos en tiempo continuo, en tiempo discreto, mixto (discreto y continuo), máquinas de estado finito (con jerarquía y concurrencia) y diagramas de flujo. La gran ventaja que posee sobre PeaCE es que al estar mucho mas difundido tiene una gran cantidad de Librerías. La desventaja que posee es que está orientado a control, en cambio PeaCE posee modelos de computación de control y de flujo de datos.

En conclusión se adoptará el Entorno PeaCE como mejor solución a los requerimientos antes nombrados, analizando sus principales virtudes en el desarrollo de este apartado.

5.3.2 Codiseño hw/sw para sistemas empotrados

Como el desarrollo de tecnología de implementación avanza a grandes pasos, el diseño de sistemas empotrados multimedia se ve perjudicado por la complejidad creciente del sistema y la necesidad de un rápido desarrollo del diseño. La práctica convencional en el diseño de sistemas empotrados multimedia, como se muestra en la figura siguiente, realiza el diseño del algoritmo, diseño de hardware, y diseño de software separadamente y secuencialmente. Después de varias experiencias de diseño y de aplicar algunas técnicas que optimizan el desempeño, se decidió que una vez seleccionada la arquitectura de hardware, se debe determinar qué partes de las funciones del sistema son implementadas con qué componentes, para luego realizar el desarrollo de software optimizado y el prototipado de hardware concurrentemente. Después de hacer el prototipo de hardware, se programan los procesadores con los códigos de software desarrollados y se verifican.

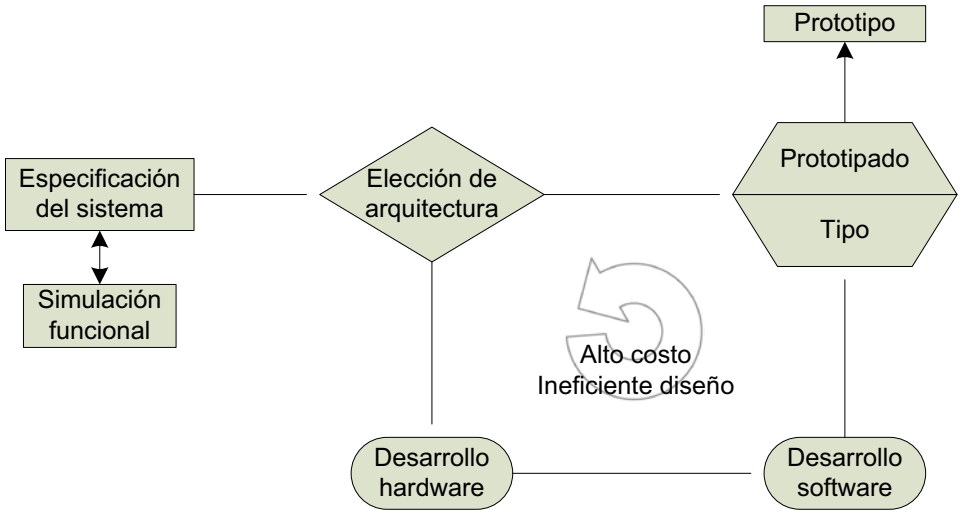

Imagen 27. Flujo de diseño convencional

Ya que la esperada tasa de crecimiento en la productividad de los diseños, aplicando los métodos de diseño convencionales, está muy por debajo de la complejidad del sistema, el codiseño hardware/software (HW/SW) ha surgido como una nueva metodología de diseño a nivel sistema, la cual no separa el diseño hardware y el diseño software. Una vez seleccionada la arquitectura de hardware, se explora la gran variedad de diseños factibles y se evalúa cada uno, estimando el desempeño esperado, luego se consideran todas las demás decisiones del diseño como particionamiento hardware/software e implementación software.

Un desarrollo sistemático del espacio de diseño necesita separar especificación de funciones y arquitecturas. El mapeo es la obtención a partir del modelo computacional de un modelo físico. El mapeo explícito consiste en planificar parte de la especificación funcional en la arquitectura de los bloques construidos.

Luego se debe estimar el rendimiento esperado y en base a esto, realizar más iteraciones de especificación de arquitectura y mapeo hasta que se logre una arquitectura óptima. A continuación, se hace una verificación exacta de rendimiento antes de la implementación final del hardware, en caso de que el costo de implementación del hardware sea muy alto (por ejemplo para la implementación del Sistema en Chip). De esta forma, el codiseño hardware/software incluye soluciones a problemas diversos de diseño incluyendo la especificación del sistema, particionamiento hardware/software, estimación de desempeño, coverificación hardware/software, y síntesis del sistema. Un ambiente de codiseño es una herramienta de software que facilita la solución de esos problemas de diseño.

Se identifican los siguientes cuatro puntos como los temas indispensables en la metodología de codiseño HW/SW:

- La síntesis a nivel especificación

- Exploración sistemática del espacio de diseño
- Coverificación y cosimulación HW/SW
- Síntesis de código automática desde modelos a nivel sistema

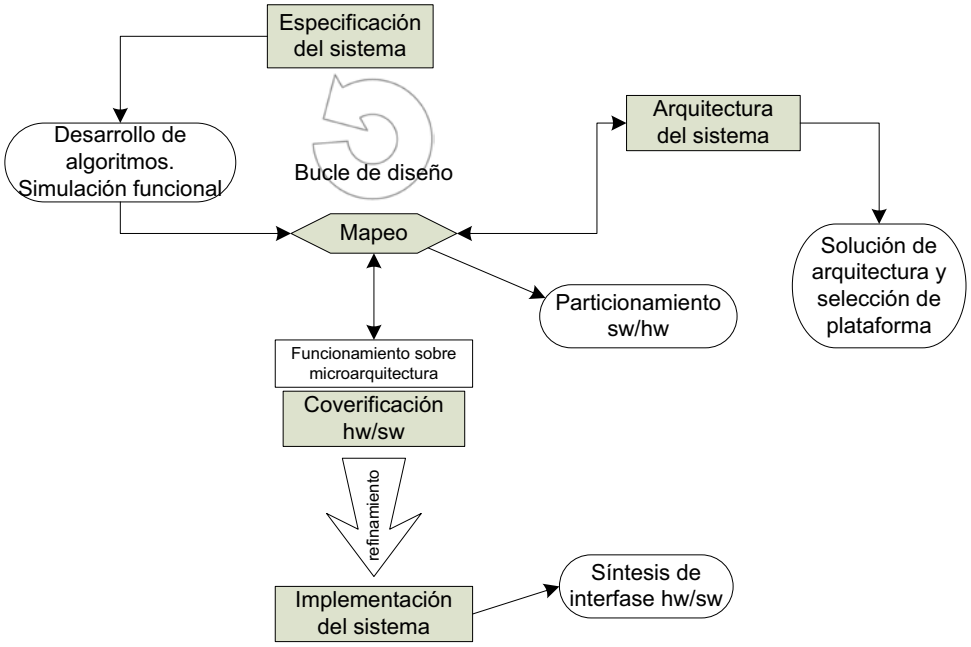

Imagen 28. Codiseño HW/SW separando funciones y especificación de arquitecturas.

Entre estos, las herramientas de codiseño comerciales le dan prioridad a la cosimulación / coverificación HW/SW, ya que la necesidad mas urgente es tener un ambiente virtual para prototipado que permita desarrollar el software antes que se haga el prototipo del hardware. La cosimulación hardware/software permite a los diseñadores evaluar o verificar el sistema bajo diseño antes que se fabrique. La cosimulación se puede usar en las diferentes etapas de diseño con propósitos diferentes; puede usarse para evaluar el desempeño del sistema, verificar la precisión funcional y la temporización del sistema. La práctica actual de codiseño HW/SW se basa en el empleo de estas herramientas.

Estas herramientas de codiseño tienen los siguientes problemas que PeaCE pretende solucionar:

1. Cada paso de diseño está aislado, si bien esa integración de herramientas de diseño es problemática. PeaCE provee flujo de codiseño libre de irregularidades, desde simulación funcional hasta síntesis de sistema.

2. El flujo de diseño puede ser pesado, dependiendo de la herramienta de diseño seleccionada. PeaCE posee una estructura configurable para la cual la tercera parte de la herramienta de diseño puede ser fácilmente integrada.

3. La exploración del espacio de diseño debe hacerse manualmente usando una herramienta de simulación TLM (Transaction level Modules -nivel de interacción entre los módulos). PeaCE incluye una estructura interactiva DSE (Design Space exploration - Exploración del Espacio de Diseño) que consta de particionamiento HW/SW y optimización de la arquitectura de comunicación.

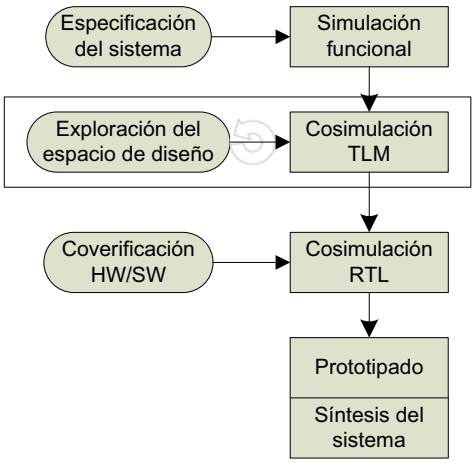

Imagen 29. Práctica actual de codiseño

El flujo de codiseño en PeaCE es reconfigurable. Cada paso de diseño es modularizado a fin de que los diversos algoritmos u otras herramientas CAD puedan ser integradas con un mínimo esfuerzo de diseño para traducir los archivos *.xml* (los archivos de extensión *.xml* indican las entradas y salidas de los pasos de diseño, planificación, simulación). La secuencia de pasos de diseño se asocia con la conexión de diseño en la interfaz del usuario.

El flujo de diseño comienza con la especificación del comportamiento del sistema con modelos formales: modelo *dataflow* para computación y un modelo de Máquinas de Estados Finitos (FSM) para el módulo de control del sistema. En el más alto nivel, se usa un modelo de tarea para modelar la interacción entre las tareas. La interfase gráfica del usuario en PeaCE, llamada Hae, permite al usuario dibujar un diagrama en bloques jerárquico reutilizando tantos bloques predefinidos como sea posible. Tal reutilización reduce el tiempo de diseño. Ya que cada diagrama en bloques se dibuja con una regla de ejecución bien definida, el diseño puede entenderse con facilidad sin ayuda del diseñador. Esto facilita el mantenimiento del diseño y posibilita a la gente trabajar conjuntamente sin confusiones. A continuación se detallan los pasos de diseño:

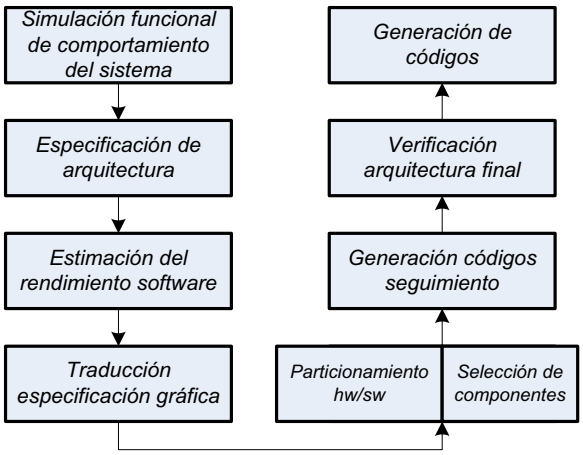

Imagen 30.Pasos del codiseño con PeaCE

Paso 1: Simulación funcional de comportamiento del sistema. Esta simulación se puede integrar con otras herramientas, como por ejemplo MATLAB u Octave. Desde la especificación se genera un código C, se compila y se corre en la máquina del usuario por simulación. Esta capacidad es comparable con Simulink a fin de que el usuario pueda reemplazarlo con PeaCE. La diferencia principal entre PeaCE y las otras herramientas, es que no es un motor de simulación pero es una herramienta de generación de código equivalente a la simulación funcional. Hace simulación que se puede depurar fácil y rápidamente.

Paso 2: Se especifican las posibles arquitecturas a ser exploradas. En este paso, listamos todos los elementos de procesamiento y los núcleos procesadores, que están disponibles en la librería de diseños.

Paso 3: Se estima el rendimiento de software de cada bloque funcional en cada núcleo procesador examinando si no está disponible en la librería de diseños. En este paso, se usa un ISS (simulador de un conjunto de instrucciones) del núcleo procesador. El usuario puede saltearse este paso si todas las estimaciones de rendimiento de bloques funcionales ya están registradas en la librería.

Paso 4: Este paso traduce la especificación gráfica a tres conjuntos de archivos de texto, la cuál describe la topología de la gráfica, la información de desempeño del bloque, y las restricciones de temporización de tareas. Tal interpretación modulariza a PeaCE de manera que el siguiente paso de diseño no dependa de su núcleo y de la interfase del usuario.

Paso 5: En este paso se realiza particionamiento HW/SW y la selección de componentes al mismo tiempo. La salida está escrita en un archivo de texto, lo cual describe el mapeo y la planificación de bloques funcionales encima de cada elemento procesador. Si

bien PeaCE provee herramientas de particionamiento HW/SW, el usuario puede usar otra herramienta gracias al mecanismo de la interfaz de archivo.

Paso 6: Después del particionamiento, genera códigos particionados y cosimulados y el sistema para generar el seguimiento de memoria de los elementos de procesamiento. PeaCE provee una herramienta de cosimulación basada en la técnica virtual de sincronización, la cual es rápida y exacta en tiempo. En este paso, se asume que la arquitectura de comunicación todavía no está determinada. Usando el seguimiento de memoria e información del planificador, exploramos la arquitectura de bus. Más adelante se explorarán otras arquitecturas de comunicación. La arquitectura final de bus se registra en un archivo de texto, archi.xml.

Paso 7: Después de determinar la arquitectura de bus, se verifica la arquitectura final antes de la síntesis. Este paso realiza cosimulación HW/SW en tiempo real para coverificación. Podemos usar CVE (coverificación del ambiente) (no posee irregularidades) en este paso o la herramienta de cosimulación usada en el paso 6, pero usando modelado exacto de arquitectura del bus.

Paso 8: Este paso es para generar los códigos para los elementos procesadores en una placa prototipada. Para un núcleo procesador, generamos un código C y un código VHDL para la implementación de hardware FPGA.

En los pasos 6 a 8, se deben generar códigos de acuerdo a la decisión de particionamiento hecha en el paso 5. Quiere decir que se deberán generar los códigos de la interfase entre elementos de procesamiento y el código envolvente para la herramienta de simulación (paso 6 y paso 7) o para la placa de prototipado (paso 8). Si se quiere usar una herramienta de simulación o una placa de prototipado diferente, entonces se debe crear un nuevo objeto Target (arquitectura destino) y nuevos bloques de la interfaz.

5.4 Sistemas operativos

Un sistema operativo es el interfaz entre el programa de usuario y el hardware del ordenador que está detrás. También gestiona la ejecución de los programas de usuario, de tal forma que varios programas puedan ejecutarse simultáneamente, accediendo al mismo hardware. Todo el que utiliza un ordenador se encuentra con un sistema operativo, ya sea Windows, Mac, Linux, DOS o UNIX.

Un sistema operativo, o más concretamente el núcleo del *kernel* del sistema operativo, reside siempre en memoria y aporta las interfaces entre el programa de usuario y el hardware del ordenador. Esto se muestra esquemáticamente en la figura . El nombre *kernel* (almendra, en inglés) proviene de la analogía con una nuez, ya que *kernel* es el

corazón de la nuez, y en el dominio de los ordenadores, el *kernel* es el corazón del sistema operativo. Continuando con la analogía, las diversas capas protectoras alrededor del *kernel* que proporcionan autorización e interacción con el usuario se llaman *shells* (cáscara).

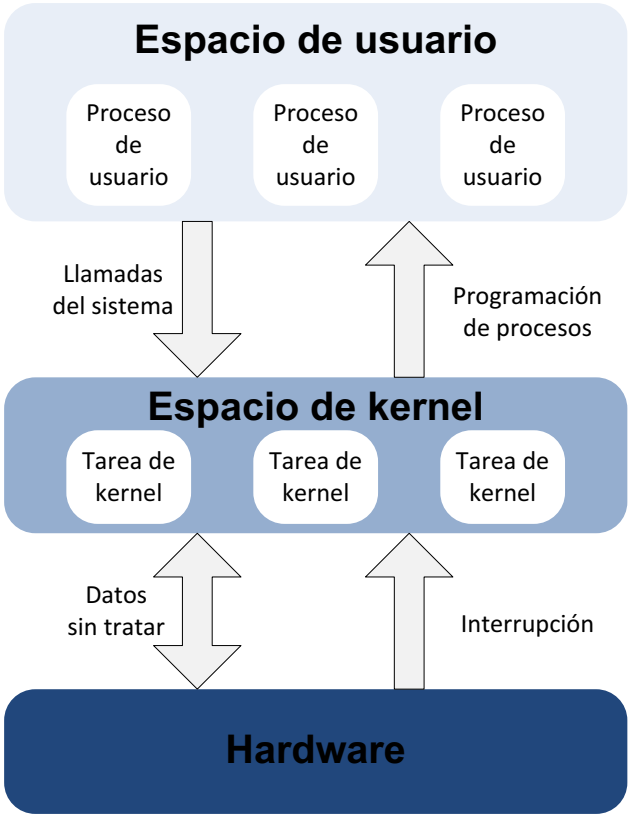

Imagen 31. Arquitectura de un Sistema Operativo

La memoria física del ordenador se divide en espacio de usuario y espacio de *kernel*, que está reservado para el código del *kernel*. El *kernel* de un sistema operativo multitarea puede gestionar múltiples programas ejecutándose simultáneamente en el espacio de usuario, de manera que cada programa piense que tiene un uso completo de todos los recursos hardware del computador y, salvo por mensajes intencionadamente enviados entre programas, cada uno piensa que tiene su propio espacio de memoria y que es el único programa en ejecución.

La comunicación entre los programas en el espacio de usuario y el código del *kernel* se consigue a través de llamadas de sistema al código del *kernel*. Normalmente, estas llamadas al sistema son para acceder a recursos físicos compartidos, tales como unidades de disco, puertos serie/paralelo, interfaces de red, teclados, ratones, pantallas y dispositivos de audio/vídeo. Un aspecto unificador de los sistemas Linux/Unix es que todos los recursos físicos se muestran a los programas de usuario como archivos, y están

controlados con las mismas llamadas al sistema, tales como *open*(), *close*(), *read*() o *write*().

Toda la actividad de entrada/salida se controla con el código del *kernel*, de tal forma que los programas del espacio de usuario no tienen que preocuparse con los detalles de la compartición de recursos físicos comunes. Los drivers específicos de cada dispositivo en el *kernel* se encargan de gestionar esos detalles. Un sistema operativo está diseñado para ejecutarse en un hardware específico, y aísla a los programas de usuario de lo concerniente al hardware, permitiendo portabilidad a otros sistemas del código de la aplicación del espacio de usuario.

5.4.1 Funciones del sistema operativo

Las principales funciones del *kernel* del sistema operativo son la gestión de memoria, la planificación de procesos, el interfaz con el hardware, la gestión de archivos y la comunicación con dispositivos y redes externos.

5.4.1.1 Gestión de memoria

La gestión de memoria es el mecanismo que aporte el sistema operativo para reservar la memoria requerida por un proceso y liberarla cuando el proceso termina. Otro requisito es asegurar que la memoria que ha sido previamente reservada y que el proceso ya no requiera se libera y está disponible para que la utilicen otros procesos. Este último requisito se denomina recolección de basura.

El código del *kernel* utiliza la CPU y las características hardware de gestión de memoria para mover de forma transparente las instrucciones y los datos entre las memorias caché, física y virtual, intentando predecir a qué datos e instrucciones se accederá y tenerlos disponibles en caché o al menos en RAM cuando se necesiten.

5.4.1.2 Protección de memoria

Un requerimiento básico para un sistema operativo de propósito general es que cada proceso de usuario opera en su propio espacio de memoria. Ningún proceso de usuario debería poder escribir en la memoria asignada a otro proceso de usuario. Esta protección de la memoria del espacio de usuario se refuerza con la MMU de la CPU. Para la mayoría de las aplicaciones empotradas, la CPU no dispone de MMU y, por tanto, los procesos de espacio de usuario se han de gestionar cuidadosamente, de manera que la memoria controlada por otros procesos no se corrompa.

Para el caso especial del propio código del *kernel*, todas las tareas en el *kernel* comparten el mismo espacio de memoria. Se asume, por tanto, que los programadores

con experiencia que escriban el código del *kernel* ponen mucho cuidado en no escribir en zonas de memoria de la que no son responsables.

Estas reglas de protección de memoria también son aplicables a los sistemas operativos en tiempo real. Si el sistema operativo en tiempo real puede soportar procesos en tiempo real en el espacio de usuario, y la CPU dispone de MMU, entonces los procesos se ejecutan en sus propios espacios de memoria protegidos.

5.4.1.3 Programación y gestión de tareas

El código del *kernel* es responsable de programar la ejecución de los programas del espacio de usuario. La programación se hace de una manera en la que los usuarios no son conscientes de que la ejecución del código de programa será pausada durante unos milisegundos, para permitir la ejecución de otros códigos de programas de usuario. Para la programación, se puede dar a cada programa de usuario un porcentaje fijo de tiempo de CPU, o bien se pueden dar diferentes prioridades a los programas de usuario, de manera que los programas prioritarios tengan mayor porcentaje de tiempo de CPU.

Cuando un programa de mayor prioridad se ejecuta, pausando al de menor prioridad, entonces se dice que el primero tiene preferencia sobre el segundo. Este cambio debe realizarse de forma rápida y determinista, porque es la clave de la eficiencia de un sistema operativo en tiempo real.

El estándar POSIX define un hilo para ser un flujo de control dentro de un proceso. Todos los hilos comparten el mismo espacio de memoria dentro del proceso.

En dicho estándar, un proceso se define como un único espacio de memoria con uno o más hilos ejecutándose en ese espacio. Cada uno de los procesos ejecutándose bajo el sistema operativo tiene su propio espacio de memoria. Se acepta generalmente que los procesos se ejecutan más bien en el espacio de usuario que en el espacio del *kernel*.

No existe una definición común para lo que constituye una tarea, aunque aquí el término se usará para describir los componentes de código ejecutable que se ejecutan en el espacio del *kernel*.

Una idea fundamental en la programación de tareas es dividir el tiempo de ejecución en trozos de igual tamaño. Esta división temporal se puede implementar en el espacio de usuario mediante la ejecución de una llamada al sistema *sleep*() cuando el proceso ha terminado el componente cíclico de su ejecución. El *kernel* devuelve control al proceso de usuario durante, o después de la expiración del tiempo solicitado en la llamada *sleep*(). Devolviendo el control cierto tiempo después de la expiración de la llamada *sleep*(), un sistema operativo puede entonces solicitar que se cumpla POSIX y de esa manera introducir *jitter* en la ejecución de tareas repetitivas.

5.4.1.4 Interfaces Hardware

Los sistemas operativos disponen de ciertos programas de bajo nivel denominados controladores de dispositivos, que manejan el hardware. Los controladores se pueden escribir para que los llame directamente el código del *kernel*, o pueden operar independientemente, interrumpiendo al código del *kernel* cuando ocurre algo que necesita atención. El primer modo se denomina sondeo, mientras que el segundo se llama conducido por interrupción.

La mayoría de los sistemas operativos trabajan con controladores conducidos por interrupción. Los datos originados en fuentes externas al sistema pueden llegar en cualquier momento. El dispositivo de interfaz (teclado, ratón, puerto serie o paralelo, interfaz de red) indica la llegada de los datos enviando señales a la CPU a nivel hardware. Esta señalización se realiza mediante los pines de interrupción de los que dispone la CPU. Los sistemas operativos generalmente asienten a una interrupción interrumpiendo la ejecución del proceso actual en la CPU, y ejecutando la rutina del servicio de interrupción (ISR) que lee los datos de entrada del dispositivo interfaz. Cuando termina el ISR, el control se devuelve al proceso original que fue interrumpido. Estos procesos de interrupción son una parte integral del *kernel* y se diseñan para conseguir un mínimo procesamiento, normalmente solo leyendo los datos de entrada y escribiéndolos en el espacio de almacenamiento del *kernel*, antes de volver a la ejecución normal.

Se pueden asignar diferentes prioridades a las interrupciones, de manera que los procesos que necesiten rápida respuesta, tales como lecturas de disco, puedan interrumpir a procesos con menor prioridad tales como lecturas de puerto serie. Otro posible uso de las prioridades es el enmascaramiento, que consigue impedir que se detecten las interrupciones de nivel inferior a una prioridad dada.

El tiempo entre un dispositivo interfaz solicitando un servicio, subiendo una bandera de interrupción, y el tiempo en el que la CPU comienza a procesar el proceso de interrupción se denomina latencia de interrupción.

Otra función importante del código del *kernel* es la de sincronizar el acceso compartido a los recursos hardware del espacio de usuario y del espacio del *kernel*. El método tradicional consiste en habilitar una especie de bloqueo software en el *kernel* cuando un programa del espacio de usuario está usando un recurso físico, y deshabilitar ese bloqueo cuando este programa ha terminado de trabajar con el recurso. Todos los otros programas del espacio de usuario tienen prohibido (por el código del *kernel*) el acceso al recurso físico mientras está bloqueado. Un tipo específico de bloqueo es en el que el programa entra en un pequeño bucle hasta que el dispositivo hardware requerido queda disponible. Esta forma maximiza la respuesta para el programa de usuario.

5.4.1.5 Gestión de ficheros

Las aplicaciones en tiempo real pueden tener ciertas necesidades de gestión de ficheros. Un ejemplo de esto son los sistemas de adquisición de datos. En estas aplicaciones, las tareas de gestión de ficheros se ejecutarán en menor prioridad, mientras que las aplicaciones en tiempo real se ejecutarán con mayor prioridad, procrastinando las tareas de gestión de ficheros según se requiera.

5.4.2 Funcionalidad requerida para el tiempo real

Los sistemas operativos en tiempo real (RTOS) son aquellos que pueden proporcionar cierto nivel de servicio en un tiempo de respuesta limitado. Estos sistemas pueden dar una respuesta en un tiempo menor que un intervalo de tiempo designado. Dicho intervalo puede ser largo en términos de tiempo de computación (por ejemplo, del orden de segundos) o puede ser corto (del orden de microsegundos). Por ejemplo, un sistema de control en tiempo real para una planta química o de comida, puede que solo muestree un sensor y calcule un comando de control una vez por segundo. Por otra parte, para una respuesta suave, a un motor por pasos se le debe dar servicio cada pocos microsegundos. Un denominado sistema operativo en tiempo real duro es aquel que no supera ningún límite marcado de tiempo, mientras que un sistema en tiempo real suave puede tolerar superar algunos límites marcados.

5.4.2.1 Sistemas operativos empotrados vs tiempo real

Los programas empotrados son aquellos que son fijos y son parte integral de un dispositivo. Por ejemplo, un ordenador de mano, un sistema de respuesta telefónica, y el control por ordenador para el motor de un coche tienen todos programas fijos que se arrancan en cuanto se les aporta energía eléctrica; se denominan aplicaciones empotradas. Dependiendo del tiempo de respuesta requerido, los sistemas operativos para aplicaciones empotradas pueden o no ser considerados sistemas operativos en tiempo real. Dar servicio a interacciones humanas no requiere en general un rendimiento en tiempo real, pero controlar una máquina, experimento científico o sistema de armas sí lo requiere.

5.4.2.2 Medidas de rendimiento para sistemas operativos en tiempo real

La característica fundamental de un sistema operativo en tiempo real es cómo de sensible es ante eventos internos y externos. Estos eventos incluyen interrupciones hardware externas, señales software internas e interrupciones internas del temporizador. Una medida de esa sensibilidad es la latencia, el tiempo entre que ocurre un evento y se ejecuta la primera instrucción del código de la interrupción. Una segunda medida es el *jitter*, la variación en el periodo de eventos de periodo nominalmente constante. Para

poder ofrecer baja latencia y bajo *jitter*, el sistema operativo debe asegurar que cualquier tarea del *kernel* será aplazada por la tarea en tiempo real.

5.4.2.3 Extensiones POSIX para aplicaciones en tiempo real

Las extensiones de tiempo real de POSIX aportan otro punto de vista en la funcionalidad adicional que se requiere en un sistema operativo en tiempo real. Estas extensiones de tiempo real añaden colas de mensajes (para comunicación entre tareas), memoria compartida, semáforos contadores (necesarios para sincronizar accesos a la memoria compartida), programación de ejecución basada en prioridades, extensiones para señales en tiempo real y temporizadores de mayor resolución. Los temporizadores pueden general intervalos de tiempo con al menos una resolución de un microsegundo.

5.5 Sistemas operativos de fuentes abiertas

Se empleará software de fuentes abiertas para ser ejecutado en el dispositivo, ya que presenta, entre otras, las siguientes ventajas:

- La *capacidad de aprender* observando el código fuente de las aplicaciones y los *cores*.
- La *posibilidad de adaptar* a gusto según las necesidades particulares de cada interesado.
- La *oportunidad de mejorar* el código y brindar esas mejoras al resto de la comunidad.
- *Bajo costo*. Los productos propietarios de este rubro suelen tener altos costos de licencias, lo que limita y restringe su aplicación en forma masiva en proyectos e instituciones de bajos recursos y en países en desarrollo.

5.5.1 Linux estándar para aplicaciones en tiempo real suave

Es posible utilizar Linux estándar para aplicaciones de control en tiempo real suave cuando el tiempo de muestreo es relativamente largo, por ejemplo de centenares de milisegundos, y la aplicación puede tolerar algunas pérdidas de programación de temporización. Un ejemplo es la representación de video, donde las pérdidas en el tiempo se traducen en un pequeño número de cuadros perdidos, pero el vídeo continúa. Otro ejemplo es para un flujo de audio, donde las pérdidas significan pérdidas de muestras de audio, generalmente escuchadas como *clics*. El ejemplo más interesante en este caso es el de adquisición y muestra de datos, donde las pérdidas se traducen en un pequeño conjunto de puntos de datos que nunca se muestran por pantalla. Si la pérdida

de datos se puede tolerar, entonces se puede utilizar un sistema estándar de Linux para este propósito. Si no es el caso, entonces se requiere una implementación en tiempo real duro.

5.5.2 Aplicaciones empotradas

Los sistemas en tiempo real pueden ser también sistemas empotrados. Las aplicaciones empotradas normalmente se ejecutan en pequeños dispositivos físicos, muchas veces sin teclados, ratones o monitores, y habitualmente sin dispositivos como discos duros, CD-ROMs, que no pueden soportar entornos hostiles. Se puede configurar Linux de manera que pueda ejecutarse sin ninguno de estos dispositivos, y existen numerosas populares distribuciones de Linux que vienen preconfiguradas e incluyen herramientas útiles para la personalización. Las personalizaciones más habituales comprenden la adición de memoria de almacenamiento flash que reemplaza a los discos duros rotatorios, y reemplazar el sistema de gráficos de Windows que consume mucha memoria y disco. De hecho, muchos productos comerciales ejecutan Linux internamente y no aparentan ser ordenadores de escritorio: grabadores de vídeo en televisión, PDAs, consolas de videojuegos… Muchas distribuciones de Linux reducen considerablemente las necesidades del sistema, tanto en tiempo de carga como en consumo de recursos.

En las aplicaciones empotradas, el tiempo de carga de varios minutos para un Linux estándar supondría un problema en general. Por ejemplo, un consumidor esperaría que un aparato de televisión por cable se pudiera usar en cuanto se encendiera. Los tiempos de carga por debajo de 10 segundos son posibles con un sistema de archivos basado en memoria y, en el caso de un IA32 PC, reemplazando la BIOS con LinuxBIOS, que contiene solo la mínima funcionalidad que Linux requiere en el periodo de carga.

La plataforma habitualmente empleada para un Linux estándar es un escritorio basado en Intel o un ordenador portátil. Pero últimamente se está haciendo habitual para los fabricantes de dispositivos portátiles, de salón, o aplicaciones dedicadas el incorporar Linux estándar como el sistema operativo para esas aplicaciones empotradas. Estas aplicaciones pueden no necesitar un rendimiento en tiempo real, y la latencia y el jitter del Linux estándar no suponen una limitación para estas aplicaciones. Por otra parte, puede existir hardware empotrado que requiera un rendimiento determinista en tiempo real y, en esos casos se empleará una de las distribuciones de Linux en tiempo real para la aplicación empotradas. Por tanto, Linux empotrado puede ejecutarse sobre un hardware desde máquinas de escritorio hasta relojes de pulsera con vídeo, pero el Linux empotrado no tiene por qué ser necesariamente en tiempo real.

5.5.3 Modificaciones para hacer de Linux un sistema en tiempo real

El procesamiento de instrucciones en el *kernel* estándar se divide en dos mitades, que se denominan tareas de la mitad superior e inferior, respectivamente. La tarea de la mitad inferior es el manejador de interrupciones que lee datos del dispositivo físico y los guarda en un búfer de memoria. La tarea superior lee del búfer de memoria y pasa los datos a un búfer accesible por el *kernel*. En el *kernel* estándar, sin los parches que hacen que una tarea más preferente pause a otra, todas las interrupciones están deshabilitadas mientas la tarea de la mitad inferior está ejecutándose. Esto significa que puede existir un retraso arbitrario (latencia) antes de que una segunda interrupción pueda ser atendida.

El *kernel* del Linux estándar aporta una resolución de temporizador por defecto de 10 milisegundos. La resolución del temporizador puede ser mejorada recompilando el *kernel* estándar utilizando un divisor de tiempo diferente, pero incluso con la latencia mejorada de un milisegundo de la versión 2.5.4, no tiene mucho sentido incrementar la resolución del temporizador.

Todas las variantes del Linux en tiempo real han introducido modificaciones al nivel del *kernel*. El resultado de estas modificaciones es reducir tanto la latencia de interrupción como el *jitter* entre interrupciones periódicas en el rango de los milisegundos, permitiendo una respuesta más rápida a eventos externos y una resolución de temporización mayor.

5.5.3.1 Gestión de las prioridades en el kernel estándar

La metodología es modificar el *kernel* del Linux estándar para asegurarse que los procesos del *kernel* de mayor prioridad puedan pausar a los de menor prioridad y obtener acceso a los recursos necesarios. Esto implica cambiar el controlador estándar de dispositivos manejador de interrupciones, de manera que los procesos de mayor prioridad no se bloqueen durante cantidades arbitrarias de tiempo esperando a que un manejador de interrupciones de prioridad inferior complete su tarea.

Comenzando por la versión 2.5.4 de Linux, el *kernel* estándar incorpora una lógica de preferencia de tareas mostrada en la siguiente figura. La latencia y el *jitter* resultantes son de alrededor de un milisegundo en procesadores Pentium con relojes de CPU a una frecuencia de centenares de MHz. Esta mejora en el rendimiento es satisfactoria para aplicaciones en el espacio de usuario que soportan audio y vídeo, pero es claramente no adecuada para ejecutar interrupciones periódicas en periodos de muestreo por debajo del milisegundo.

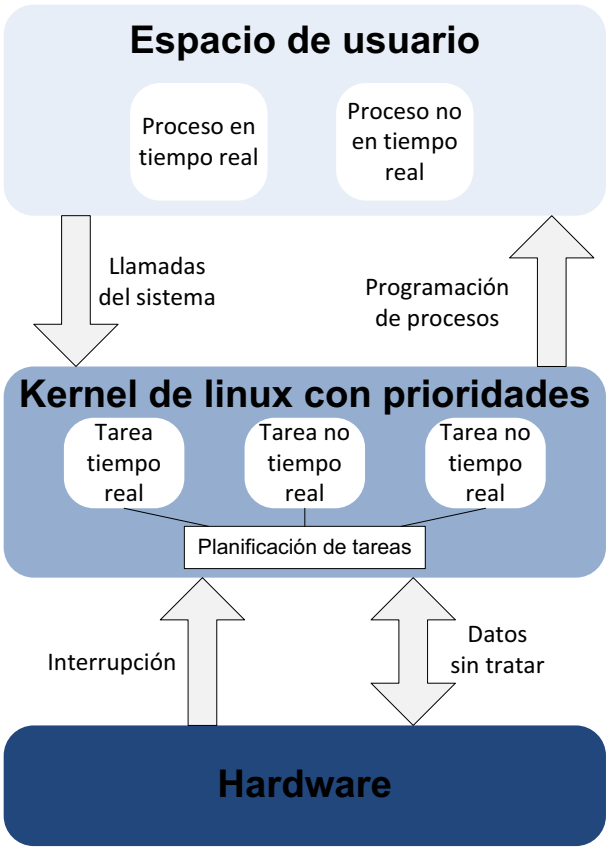

Imagen 32. Arquitectura de *kernel* que gestiona prioridades

Nótese que en la figura anterior la tarea de tiempo real es una de tantas, controlada por el programador del *kernel*. Esta tarea es la referencia del *kernel* para el proceso en tiempo real el espacio de usuario. El programador puede ejecutar esta tarea de mayor prioridad pausando cualquier tarea de menor prioridad en el espacio del *kernel* y de esta manera el proceso en el espacio de usuario puede conseguir la latencia de un milisegundo que la versión 2.5.4 hacía posible.

Las versiones en tiempo real de Linux han madurado hasta el punto de que ahora pueden ser consideradas unos candidatos viables para las aplicaciones en tiempo real. En los últimos años, diversos grupos de investigación en universidades, industria, y laboratorios del gobierno han logrando avances significativos en las aplicaciones en tiempo real. Las modificaciones en tiempo real son estables y están probadas, y las empresas comerciales ofrecen servicios de formación y soporte.

5.5.4 Conclusiones

El *kernel* estándar en versiones de desarrollo recientes muestra una latencia (tiempo entre que aparece y se comienza a procesar una interrupción) y *jitter* (variación de tiempos en eventos periódicos) del orden de un milisegundo. Las versiones en tiempo real de Linux tienen latencia y *jitter* del orden de unos pocos microsegundos en procesadores que funcionan a centenares de MHz. La siguiente tabla muestra que el rendimiento es comparale a los *kernel*s comerciales en tiempo real.

Sistema operativo	**Aplicación**	**Latencia/Jitter**
S.O. estándar	No tiempo real	100 µs - 100 ms
Linux estándar	Tiempo real suave	1 ms
Linux IEEE 1003.1d	Tiempo real duro	10 - 100 µs
Linux Micro-*Kernel*	Tiempo real duro	1 - 10 µs
RTOS *Kernel*	Tiempo real duro	1 - 10 µs

Tabla 3: Comparación de rendimiento de Linux

Existen diversas áreas en las cuales un sistema Linux en tiempo real se queda corto en funcionalidad cuando se compara con un sistema Linux estándar y con ofertas comerciales de RTOS. El usuario potencial debe tener en cuenta que no solo tiene que compilar el *kernel* con parches procedentes de otra fuente, sino que también tendrá que tratar directamente con asuntos de bajo nivel en la interfaz con el hardware y la gestión de memoria. Por otra parte, la formación y los servicios de consulta para Linux en tiempo real están disponibles comercialmente hoy en día, y el grado de desarrollo y las herramientas de soporte se consideran sustanciales y en continuo crecimiento. La pregunta a hacerse es si tener un sistema operativo gratis y de fuentes abiertas para una aplicación en tiempo real merece el esfuerzo extra para implementar esta aplicación. El entusiasmo de los que han cambiado a un sistema Linux en tiempo real es una forma de probar que muchos encuentran interesante este cambio.