TEAPAN: UNA HERRAMIENTA PARA EL DISEÑO DE ARQUITECTURAS A ALTO NIVEL[a]

D. Peñalosa[1], C.J. Jiménez[2], M. Valencia[2].
[1] IES Ruiz Gijón de Utrera, dionisiodavid@terra.es
[2] Univ. Sevilla – Instituto de Microelectrónica de Sevilla, cjesus@imse.cnm.es, manolov@dte.us.es
Tel: +34 955 05 66 66 Fax: +34 955 05 66 86

ABSTRACT

La existencia de diferentes arquitecturas de procesadores cada vez más complejas hace que la construcción de estos sea costosa utilizando las herramientas tradicionales de desarrollo. Se hace necesario pues, disponer de herramientas capaces de especificar las distintas arquitecturas en un lenguaje de alto nivel, para que luego puedan estas ser simuladas. En esta comunicación se presenta una herramienta de ese tipo. Para ello, a partir de un estudio de las particularidades más importantes de los principales procesadores comerciales más recientes, se extraen los bloques más relevantes de estas arquitecturas. Estos bloques se integran en una herramienta para la especificación de arquitecturas a alto nivel. Finalmente, se especifica con la herramienta un computador sencillo, para posteriormente simularlo y comprobar así la validez de la misma.

1. INTRODUCCION

En los últimos años las arquitecturas de los microprocesadores han ido aumentando en complejidad y diversidad. Tanto es así, que cualquier innovación o mejora de las arquitecturas actuales se hace siempre a costa de un alto tiempo de rediseño y/o de coste de implementación, sin saber a ciencia cierta si la mejora va a sutir efecto o no. Esto plantea la necesidad de elaborar herramientas capaces de especificar primero y simular después distintas arquitecturas a alto nivel.

En esta comunicación se presenta a TEAPAN (Traductor de Especificaciones de Arquitectura de Procesadores a Alto Nivel), una herramienta que cumple estas características. Para su construcción se han analizado algunos de los procesadores comerciales más utilizados. A partir de este análisis se han extraido una serie de componentes, conformando una librería de bloques, con los que pueden construirse procesadores con distintas arquitecturas. Posteriormente, se ha construido una herramienta capaz de combinar estos bloques para así especificar y simular de forma flexible y sencilla cualquier tipo de arquitectura.

Para corroborar la facilidad de uso de la herramienta se ha especificado un computador simple, y luego se ha simulado con MODELSIM (de Mentor Graphics) para comprobar su validez.

La organización de esta comunicación es la siguiente: en el siguiente apartado se presenta un análisis de las características más relevantes de los procesadores actuales, en concreto, analizando seis de las arquitecturas de procesadores más empleadas. Para cada una de estas arquitecturas se extraen los principales bloques funcionales que la forman. En el apartado tres se detallan los bloques funcionales que forman la librería de componentes. En el apartado cuatro, cinco, y seis se describe la herramienta en sí, mostrando tanto su funcionamiento como la naturaleza e implementación de los componentes. En el apartado 7 se muestra en un ejemplo la especificación y posterior simulación de un computador pipeline sumador de vectores. Finalmente se extraen algunas conclusiones.

2. ESTUDIO DE ARQUITECTURAS PIPELINE

Se han analizado seis procesadores ampliamente utilizados comercialmente. De este análisis se han excluido los elementos que optimizan los accesos a memoria (caché, memoria virtual, etc.). Los microprocesadores analizados son: SPARCv3, Alfa21164, Petium III Xeon, Pentium Itanium, Athlon AMD, y PowerPC 601.

Las arquitecturas de estos microprocesadores se caracterizan por procesar varias instrucciones en paralelo, mediante el empleo de estructuras Pipeline/Superpipeline. Esto se consigue dividiendo el procesamiento de cada instrucción en una serie de etapas, de forma que cuando una instrucción "salga" de una etapa, otra instrucción pueda "entrar" en ella como si de una cadena de producción se tratara.

Además de este tipo de característica, estos microprocesadores poseen los siguientes elementos comunes [1]:

- Fichero de Registro: Agrupa a todos los registros de propósito general. Funciona como si fuera una memoria multipuerta, de forma que se permita la lectura de todos los operandos de una instrucción en un ciclo de reloj.

[a]Este artículo ha sido parcialmente financiado por el proyecto META (TEC 2004-00840/MIC)

- Cola de Instrucciones: Consiste en un buffer que contiene una serie de instrucciones dispuestas en el mismo orden llegada a éste. De esta forma, la tarea de búsqueda de instrucción consiste en ir llenando este buffer e ir tomando las de que están en la salida de la cola.
- Cola de Lectura/Escritura: De forma parecida a la "Cola de Instrucciones", también se suelen emplear unos buffers para la lectura y escritura de datos en la memoria, que almacenan tanto la dirección como el dato que se desea escribir y/o el dato que se espera leer.
- Unidades Funcionales: Se emplean en la fase de ejecución una vez se haya decodificado la instrucción y obtenido sus operandos. Estos elementos son bloques encargados de realizar de forma óptima una serie de operaciones concretas, como operaciones con reales, operaciones con enteros, etc.
- Lógica de saltos: Controla cuál va a ser la dirección de acceso a la próxima instrucción. Adquiere gran importancia en estos procesadores, por lo que se suele tratarse en un bloque aparte.

Los bloques descritos son comunes en microprocesadores que ejecutan instrucciones en paralelo. El hecho de ejecutar una instrucción antes de finalizar la que le precede, acarrea una serie de inconvenientes que hacen que este paralelismo no pueda llevarse a cabo en determinadas circunstancias. La forma de resolver o minimizar estos inconvenientes, es lo que hace que la arquitectura de un determinado microprocesador sea distinta y se comporte mejor que la de otro.

Los conflictos que hacen que dos o más instrucciones no puedan ejecutarse al mismo tiempo se categorizan de la siguiente forma [1]:

- Conflictos debido a la Estructura. Estos conflictos son originados por una falta de recursos físicos, como podría ser el caso en el que dos instrucciones necesiten un mismo sumador, o bien el caso en el que una instrucción esté en una etapa de acceso a memoria y esta se retrase debido a fallos de caché (impidiendo que otra instrucción "entre" en dicha etapa).
- Conflictos debido a Instrucciones de Control. Estos conflictos son originados por aquellas instrucciones que modifican el transcurso lineal de un programa, como son las instrucciones de salto, de llamada a subrutinas o las excepciones. El problema ocurre cuando el microprocesador notifica que ha de dar un salto en la memoria para tomar una instrucción y ya ha empezado a ejecutar a la que le sigue a la actual.
- Conflictos debido a los Datos Implicados en las Instrucciones. Estos conflictos surgen cuando existe una dependencia de datos entre dos instrucciones consecutivas. Estos conflictos son tan complejos que a su vez se subdividen en tres nuevos tipos: RAW, WAR y WAV.
- Conflictos RAW (Read After Write). Suceden cuando se procesa una instrucción que lee de un registro después de haberse procesado una que escribe en este mismo registro. Se pudiera dar el caso que la instrucción de escritura se retrasara y la de lectura leyera el valor antiguo del registro.
- Conflictos WAR (Write After Read). Suceden cuando se procesa una instrucción que escribe en un registro después de haberse procesado otra que lee del mismo registro. Se pudiera dar el caso que la instrucción de lectura se retrasara, invirtiéndose así el orden de ejecución entre ambas.
- Conflictos WAW (Write After Write). Suceden cuando se procesa una instrucción que escribe un registro después de haberse procesado otra que escribe en ese mismo registro. Se pudiera dar el caso que la primera instrucción se retrasara, invirtiéndose así el orden de ejecución entre ambas.

La forma más fácil de resolver cualquiera de estos conflictos es detener la cadena de procesamiento a partir de la instrucción que está provocando el conflicto. Los procesadores actuales utilizan una serie de técnicas para resolver estos conflictos sin necesidad de parar la cadena o deteniéndola durante los menos ciclos posibles.

Entre todas estas, merece la pena destacar la técnica de Tomasulo [1], ya que es muy empleada en los computadores recientes y supone un cambio grande dentro de la arquitectura pipeline. Esta técnica reduce el número de conflictos y el número de ciclos de detención de pipeline haciendo una planificación dinámica de instrucciones. Esto se consigue mediante un aumento de coste en el hardware (en la unidad de procesado).

La disposición de la unidad de procesado típica en los microprocesadores que utilizan esta técnica es la siguiente: La caché de instrucciones se conecta con una cola de instrucciones y, de aquí, parten éstas hacia unas estaciones de reserva. En estas estaciones se "anota" el tipo operación de la instrucción, y los valores de sus operandos. Si el valor del operando estuviera siendo procesado por otra instrucción anterior a la actual, se anotaría la estación de reserva asociada a la unidad funcional que producirá el valor del operando. Con esto se eliminan los conflictos RAW.

Las estaciones de reserva, a su vez, se conectan con las unidades funcionales. Estas producen resultados que van dirigidos hacia los registros de propósito general y a las propias estaciones de reserva, eliminando así los conflictos WAR.

Los registros de propósito general están dotados de unas etiquetas que indican cuál es la estación de reserva que proporcionará su valor. Esta técnica elimina los conflictos WAW y recibe le nombre de "Renaming".

Una vez introducidos todos estos conceptos preliminares, pasamos al estudio de algunos de los microprocesadores más conocidos.

1. SPARC V3 [2]

Esta es la arquitectura empleada en los microprocesadores de los ordenadores de Sun Microsystem. Fue una de las pioneras en utilizar un conjunto reducido de instrucciones (RISC) y fraccionar el procesamiento de cada una de ellas para realizar pipeline. Los microprocesadores que emplean esta arquitectura están formados por los siguientes componentes: una unidad funcional de entero, una unidad de punto flotante, un buffer de escritura de 4 registros, un controlador para buses esclavos y una unidad de control de memoria.

Para minimizar los conflictos debidos a instrucciones de control del programa utiliza la técnica del "Salto Retrasado" [1]. Esta técnica consiste en cambiar de situación de una de las instrucciones (libre de dependencias) que vengan antes de la de salto, para ponerla justo detrás de esta. Con esta técnica se gana un ciclo cuando surgen este tipo de conflictos.

2. Alpha 21164 de Digital [3, 4]

Esta arquitectura es la empleada en los procesadores de los ordenadores de Digital. En los procesadores diseñados con esta arquitectura pueden distinguirse los siguientes bloques: una unidad de predicción de saltos, una unidad para el manejo de la memoria virtual, una unidad de decodificación de instrucciones, una cola para las instrucciones enteras de 20 entradas, una cola para las instrucciones de punto flotante de 15 entradas, dos unidades funcionales de entero, dos ficheros de registros de entero (80 registros cada uno), dos unidades funcionales de punto flotante, un fichero de registro de punto flotante (72 registros) y un subsistema de memoria de altas prestaciones (que incluye una cola para lectura de datos en memoria y otra para escritura).

Para minimizar los conflictos RAW, utiliza una técnica llamada "Bypasses" [1], la cual permite utilizar el resultado de una unidad funcional sin necesidad de que éste se escriba en uno de los registros de la máquina. Otra técnica que se utiliza para minimizar también los conflictos RAW y los WAW es la llamada técnica del "Marcador" [1]. Esta técnica consiste en ir anotando en un registro interno las unidades que van a generar un resultado provechoso para otra instrucción y llevar en una lista los registros a los que irán destinados los resultados de las operaciones. De esta forma se puede utilizar el resultado de algunas instrucciones, sin necesidad de que éste se escriba en un registro intermedio.

Los conflictos debidos a instrucciones de control del programa, se minimizan utilizando una "Lógica de Predicción de Saltos" [1]. Para ello, el resultado de los saltos que se van tomando se almacena en una memoria de 2Kbit, y esta información es utilizada para predecir un nuevo salto (en caso de realizar una predicción falsa, se deberá restaurar los registros de la máquina afectados).

3. P6 de Intel [5-7]

Esta es la arquitectura utilizada en los microprocesadores Pentium Pro, Pentium II, Pentium III, Celeron, y Pentium III Xeon. Esta arquitectura, en una primera etapa, traduce instrucciones en microoperaciones que son ejecutadas en una cadena de pipeline (como las instrucciones RISC) compuesta por un total de 14 etapas. Internamente, la arquitectura P6 está formada por una unidad de enteros, otra de punto flotante, una unidad de saltos, una unidad para el direccionamiento de memoria, y una unidad para el acceso de memoria.

Para reducir los conflictos de datos que puedan originarse (ya sea RAW, WAR, o WAW) emplea la técnica de Tomasulo y para minimizar los conflictos debidos a instrucciones de control del programa utiliza un predictor de saltos de 512 entradas.

4. IA-64 [5]

Esta arquitectura es la utilizada en los procesadores Itanium de Intel. Está formada por una serie de registros generales, los cuales pueden ser accedidos en forma de pila para optimizar las llamadas a subrutinas. Contiene también dos unidades de entero, dos unidades para acceso a memoria, dos de punto flotante y tres para la ejecución de bifurcaciones en el programa.

Esta arquitectura utiliza una cadena de pipeline de 10 etapas, encargadas de realizar fundamentalmente cuatro tareas: "Terminal Frontal" donde se toman las instrucciones y se predicen los saltos, "Deliberación de Instrucción" donde se distribuyen las instrucciones hacia sus respectivas unidades funcionales, aquí se aplica la técnica de "Renaming" para reducir los conflictos WAW, "Deliberación de Operandos" donde se accede al fichero de registro y se emplean la técnicas de "bypasses" y del "marcador" para minimizar los conflictos RAW y WAW y por último la "Ejecución", donde se realizan las funciones específicas de cada instrucción, se tratan las excepciones y se escriben los resultados en los registros.

Además de las técnicas descritas para reducir los conflictos, esta arquitectura utiliza las técnicas de "especulación de instrucciones" para minimizar la latencia en accesos a memoria, "instrucciones con predicado" para eliminar las instrucciones de control del programa (y así los conflictos originadas por éstos) y "predicciones de salto" [1].

5. Athlon AMD [8]

Esta arquitectura, empleada en los procesadores Athlon de AMD, está compuesta por un bloque encargado de la planificación y distribución de las instrucciones por las diversas unidades funcionales (denominado "Front End"), por una unidad de entero formada a su vez por tres "ALU's" y tres unidades de direccionamiento que pueden funcionar en paralelo. Dispone también de una unidad de punto flotante, que a su vez está formada por una unidad de

suma en punto flotante, una unidad de multiplicación en punto flotante y una unidad de almacenamiento en punto flotante, todas ellas con posibilidad de funcionar también en paralelo.
Para minimizar los conflictos propios de datos, Athlon utiliza la técnica de Tomasulo y para reducir los conflictos debidos a instrucciones de control del programa, utiliza predicción de saltos.

6. PowerPC 601 [9]

Esta es la arquitectura que emplean los procesadores PowerPC de Motorola. Es una arquitectura superpipeline (puede ejecutar hasta 3 instrucciones en pipeline) que está formada por tres unidades funcionales de entero, una unidad para la administración de memoria y una unidad de punto flotante. Posee una cadena de Pipeline que varía de entre 7 y 10 fases.
Para reducir cualquiera de los conflictos de datos, emplea la técnica de Tomasulo y para reducir los conflictos debidos a instrucciones de control del programa, utiliza predicción de saltos.
En la tabla 1 se muestra un cuadro resumen de las diversas características de las arquitecturas presentadas.

Computador	Grado Pipeline	Técnicas Empleadas
SPARC V3	5	Salto Retrasado
Alpha 21164	6-8	Bypasses Marcador Predicción de Saltos
Intel P6 (Xeon)	14	Tomasulo Predicción de Saltos
Pentium Itanium	10	Bypasses Marcador Especulación Predicados Predicción de Saltos Planificación Estática
Athlon AMD	10-15	Tomasulo Predicción de Saltos
PowerPC 601	7-10	Tomasulo Predicción de Saltos

Tabla 1. Resumen comparativo de las arquitecturas presentadas

3. EXTRACCIÓN DE COMPONENTES

El análisis de las arquitecturas de algunos de los procesadores comerciales más empleados ha permitido su partición en bloques y la creación de una librería de componentes con los que construir cualquiera de las arquitecturas.

Estos componentes han sido clasificados en tres categorías. En la primera de ellas se introducen aquellos que son necesarios para ejecutar instrucciones en una arquitectura pipeline. En una segunda categoría, se incluyen otros componentes que son opcionales en este tipo de arquitectura. Por último, en una tercera categoría, se incluyen los componentes funcionales encargados de operar con los datos y también los componentes de control.

Los componentes de Arquitectura Pipeline son los siguientes:

- COLAINSTR(...): Cola de Instrucciones. Este componente aparece en la etapa de búsqueda de instrucción y consta de un array de registros donde se van introduciendo las instrucciones, de tal forma que la primera en llegar será la primera en salir. Se puede indicar tanto el número de registros que lo forman, como la anchura de los mismos.
- ESTACIONRESERVA(...): Registros adicionales utilizados en el algoritmo de Tomasulo para eliminar conflictos de datos. Se puede indicar el número de operandos que ha de tener y su anchura.
- FICHREG(...): Conjunto de registros con estructura de RAM. Suelen ser de propósito general. Se puede indicar tanto la cantidad como la anchura de los mismos.
- BUFFERDIR(...): Conjunto de registros utilizados para los accesos a memoria. Se puede indicar su tamaño y anchura.
- BUFFERDAT(...): Igual que el componente anterior, pero para los datos.
- ETIREG(...): Etiquetas de Registros. Conjunto de registros que se utilizan de forma complementaria al fichero de registros y que sirven para que éstos sepan de que unidad funcional han de cargar el resultado. Se emplea para reducir los conflictos de datos. Se puede indicar tanto el tamaño como la anchura (proporcional al número de estaciones de reserva).

En los Componentes Opcionales, entre otros, se dispone de los siguientes:

- REGISTRO(...): Registro para el almacenamiento de datos. Aparte de la anchura del mismo, se puede indicar la funcionalidad de éste, es decir, si va a ser un registro con carga, con incremento, decremento, puesta a cero, puesta a uno, desplazamiento, etc.
- ROM(...): Esta función junto con un fichero de texto que contenga información con ceros y unos formarán una memoria de sólo lectura. Se puede establecer su tamaño.
- RAM(...): Memoria de Acceso aleatorio. Se definen el tamaño y la anchura de la misma.
- ALUE(...): Unidad Aritmético-Lógica Entera.

- BUSHI(...): Dispositivo que pone en alta impedancia la salida de otro. Se utiliza para compartir un bus. Se puede considerar como una señal de "Output Enable".
- BUSCE(...): Dispositivo que pone a cero la salida de un determinado componente.
- CERO(...): Detector de valores igual a cero.
- NCERO(...): Detector de valores distintos de cero.
- MUX21(...),MUX41(...): Multiplexores de dos/cuatro entradas y una salida.
- NOT(...), AND(...), OR(...), XOR(...), NAND(...), NOR(…):NOT y otras puertas lógicas de dos entradas.

Los Componentes de Unidades Funcionales y de Control tienen especificada la interfaz pero no su contenido, para lo cual el usuario dispondrá de una serie de recursos que le proporcionará la herramienta:

- UCONTROL(): Unidad de Control del Computador. La forma de especificar unidades de control se verá con más detalle en la siguiente sección.
- USALTO(): Unidad que calcula la siguiente instrucción que ha de venir en el programa. No se encuentran disponibles los recursos para definirlas.
- UFUNCIONALX(): Unidad de propósito específico. Al igual que la Unidad de Saltos, tampoco se encuentran desarrollados los recursos para especificarlos.

4. GENERACIÓN DE VHDL A TRAVÉS DE LOS COMPONENTES

Para poder especificar sistemas complejos utilizando estos componentes, se ha desarrollado una herramienta llamada TEAPAN (Traductor de Especificaciones de Arquitecturas de Procesadores a Alto Nivel). El objetivo de nuestra herramienta es generar el código VHDL de un Computador libre de errores para que se pueda simular y sintetizar. Para lograr este objetivo se dispondrán los componentes descritos en el apartado anterior dentro de una especificación, y se interconectarán mediante sus parámetros [10]. Nuestra herramienta tomará esta especificación y generará el código VHDL requerido. Este procedimiento se puede apreciar en la figura 1:

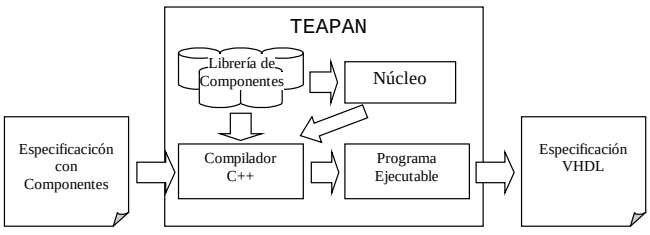

Figura 1: Especificación con TEAPAN

En esta figura, se puede apreciar el proceso que sigue TEAPAN para transformar la especificación realizada con componentes en una especificación VHDL. Para comprender la figura es necesario saber que cada componente está encapsulado mediante una función del Lenguaje de Programación C++. De esta forma, una Especificación con Componentes no es más que un programa en C++, en donde aparece la función "main" y en su interior sólo hay una secuencia de llamadas a funciones que representan a los diversos componentes.

TEAPAN integra un Compilador de C++ (figura 1), el cual toma la especificación, y con una librería de componentes y una serie de objetos de C++ (cuya funcionalidad la veremos en el siguiente apartado) que llamaremos "Núcleo", genera un programa ejecutable. De forma transparente al usuario, TEAPAN ejecuta dicho programa y este generará la especificación VHDL deseada.

El código VHDL generado estará formado por una única "Entidad", y el cuerpo de esta estará formado por el código VHDL "plano" que refleja todas las funcionalidades que se han querido plasmar con los componentes.

La relación entre el código VHDL y la interconexión de componentes es la siguiente: cada componente genera la porción de código VHDL correspondiente a la funcionalidad de este dentro del cuerpo de la "Entidad" definida, y el Núcleo se encarga dar una forma compacta y adecuada a la suma de todas las especificaciones generadas por los componentes (generación de la interfaz, declaración de señales, etc).

Una vez explicado cómo se genera una especificación VHDL, vamos a centrarnos ahora en cómo se configuran e interconectan estos componentes para formar una arquitectura determinada.

La función C que representa a cada componente recibe tres tipos de parámetros: identificador del componente, parámetros configuradores y parámetros de interconexión. Mediante los parámetros configuradores, podremos concretizar aspectos propios de cada componente tales como la anchura de los buses que procesa, capacidades de almacenamiento, etc. Mediante los parámetros de interconexión, indicaremos de qué buses tomaremos una determinada información y sobre qué buses la verteremos.

Cada componente, además de su propio código VHDL característico, genera unas determinadas señales de control, tanto de entradas (para poder indicar cuándo se desea activar alguna funcionalidad del mismo), como de salida (banderas de estado). El nombre de esta señal estará formado por el identificador del componente más un mnemotécnico que haga referencia a la funcionalidad de la misma.

En la figura 2 se puede apreciar el ejemplo del código generado por un componente configurado para generar un registro (PC) contador de 8 bits, capaz de iniciarse a cero, **REGISTRO("PC",8,"IC","","BusDir")** :

```
process(PCCL,PCIC)
begin
  if PCCL = '1' then
    PC <= "00000000";
  elsif PCIC = '1' and PCIC'event then
    PC <= CONV_STD_LOGIC_VECTOR
(CONV_INTEGER(PC) + 1,4);
  end if;
end process;
BusDir <= PC;
```

Figura 2: Código VHDL generado por REGISTRO.

Este componente tiene como parámetro identificador "PC"; presenta dos parámetros configuradores: "8", que indica la anchura del registro; e "IC" que indica que el registro es de "Incremento" (I) y con capacidad de ponerse a "Cero" (C). El primer parámetro de interconexión viene en blanco, ya que este componente no presenta ninguna entrada de datos; y el segundo es "BusDir" indicando el bus por el que se desea verter la información del registro. Este componente generará dos señales de control: PCCL, la cual en flanco de subida pone a cero el registro; y PCIC, la cual en flanco de subida incrementa el registro en una unidad.
Si deseáramos interconectar la salida de este registro con la entrada de otro registro (que llamaremos IR), bastaría con indicar "BusDir" en el parámetro de entrada de datos del registro:

REGISTRO("IR",8,"L","BusDir","BusDatos").

Compartiendo los parámetros de los componentes de esta forma, podremos crear documentos de Especificación de Componentes y especificar cualquier tipo de arquitectura.

5. COMPONENTES DE UNIDADES FUNCIONALES Y DE CONTROL.

Como se comentó en el apartado 3, debido al carácter específico de los Componentes de Unidades Funcionales y de Control, estos no vienen especificados pero la herramienta ofrece al usuario una serie de recursos para poder especificar estos componentes. En este apartado se explicarán en qué consisten estos recursos para especificar tanto una unidad de control como una unidad funcional.

5.1. Especificando las Unidades de Control.

La unidad de control es la porción de hardware encargada de activar las señales de control de la unidad de procesado, para así poder ejecutar las instrucciones. Esta unidad es más compleja en los procesadores con arquitectura tradicional Von Newman, que en los Pipeline, pero debido al carácter general que se desea que tenga la herramienta a la hora de especificar computadores, se ha incluido una serie de recursos para facilitar la tarea del diseño de este tipo de circuitos.
La herramienta implementará a la unidad de control mediante máquinas de estados, ya que es un método suficientemente extendido [11][12], y presenta un coste en hardware relativamente bajo.
Las máquinas de estados se implementarán disponiendo de forma secuencial en una especificación las siguientes funciones del lenguaje C++:

- MAQUINA(): Con esta función se indica que se desea construir una máquina. En esta función se ha de indicar una señal de reloj (para la sincronización de los estados), una señal para volver al estado inicial (reset), y un vector con el código de la instrucción (para facilitar la labor de decodificación).
- ESTINI(): Con esta función se definirá el estado inicial.
- ESTADO(): Con esta función se define un estado cualquiera.
- SI(): Esta función sirve para pasar de un estado a otro. Se indicará una señal que se desea testear y el estado al que se quiere ir en caso de estar esta señal en alto. También se podrán testear vectores.
- SIINSTR(): Esta función sirve para la decodificación de las instrucciones, de forma que si el código de la instrucción que se indica en los parámetros coincidiera con el del vector indicado en la función MAQUINA, se pasaría a un estado determinado.
- ACTIVA(): Indicará las señales que se desean activar en un estado.
- DEFINSTR(): Asocia un mnemotécnico a un código de una instrucción.

Combinando estas funciones dentro de una especificación, se podrá definir una unidad de control como se puede apreciar en la figura 3:

```
MAQUINA(1,"Clk","Rst","IR(15 downto 13)");
ESTINI("E0");
SI("-","E1");
ACTIVA("pcIC");
ESTADO("E1");
SIINSTR("001","E2");
SIINSTR("010","E3");
ESTADO("E2");
SI("-","E0");
ESTADO("E3");
SI("compZ","E2");
SI("-","E1");
```

Figura 3: Ejemplo de Unidad de Control.

En esta figura se puede apreciar cómo se define la máquina de estados número 1 y se le indica la señal de reloj (Clk) con la que se sincronizan los estados y la señal de reset

(Rst) con la que se podrá ir al estado inicial "E0". Dentro de este estado se activará la señal "pcIC", y se pasará de forma incondicional ("-") al estado "E1" en el siguiente ciclo de reloj. En el estado "E1" se comprobará si el código de instrucción dado en el vector IR(15-13) coincide con "001" o con "010", en cuyo caso, pasará al estado "E2" o "E3" respectivamente. Del estado "E3" se pasará al estado "E2" en caso de estar activa la señal "compZ", o al estado "E1" en cualquier otro caso.
Así pues, combinando funciones como estas, se podrá pasar de un estado a otro y activar señales en los estados.

5.2 Especificando las Unidades Funcionales.

Por Unidades Funcionales [1] se comprenderá aquellos circuitos orientados a la realización de una tarea concreta, que es utilizada por la CPU. Ejemplos de Unidades Funcionales conocidas son la ALU (realiza operaciones aritméticas), Multiplicadores, Divisores, etc.
Con nuestra herramienta vamos a implementar dos tipos de Unidades Funcionales: las Unidades Funcionales que hacen operaciones aritméticas (devuelven el resultado de aplicar una expresión a los parámetros de entrada); y las unidades funcionales que emiten ondas cuadradas en función de sus parámetros.
Una misma Unidad Funcional puede combinar ambos tipos y se especificará mediante un fichero de especificación como el mostrado en la figura 4:

```
Síncrono:
Clk
Especificación:
GFEN onda 4 DUR
          2 DUR
FREQ expr NOTA*1440+5
```

Figura 4: Especificación de una Unidad Funcional

Al insertar en nuestra especificación una función del tipo:

UF("UFSon","DUR,NOTA","FGEN,FREQ");

y escribir en un fichero de texto llamado "UFSon.txt" una especificación como la de la figura 4, se generará un código VHDL en donde se verterá en la señal FREQ el resultado de calcular NOTA*1440+5; y se generará también un código VHDL que emita por la señal FGEN una onda cuadrada que alterne "DUR" periodos de una señal de 4 ciclos y otra de 2 (figura 5).

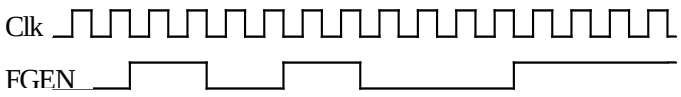

Figura 5: señal FGEN para DUR = 2

6. GENERACIÓN DE LA ENTIDAD GLOBAL

La porción de código VHDL generada por cada componente obvia aspectos tales como la declaración de señales, el chequeo de determinadas interconexiones sin sentido o la generación de la interfaz de entrada/salida. Todas estas tareas son realizadas por una parte integrada en la herramienta denominada "Núcleo" (figura 1). El "Núcleo", es transparente al usuario, y recibe informaciones de los componentes para poder llevar a cabo todas sus funciones.
Así pues las principales funciones del núcleo, son las siguientes:

- Declaración de señales. Cada vez que aparezca un bus de datos en un componente o el componente genere una señal para que el sistema pueda controlarlo, se almacenará dicha señal para su posterior declaración.
- Cuenta de Componentes. Cada componente ha de indicar al núcleo que ha sido insertado en una especificación para que pueda referenciar este en caso de cometer un error en la especificación.
- Asignación de Señales. El Núcleo ha de arbitrar las asignaciones de varias señales simples en una compleja, así como las asignaciones de una señal compleja en varias simples. En los dos casos se ha de validar que la señal simple sea menor que el resto que queda por asignar en la señal compleja; llevar la cuenta de las posiciones asignadas en la señal compleja, para que el usuario pueda asignarle varias simples sin tener que indicar la posición de forma explícita; y que, en el caso de indicar el usuario explícitamente la dirección a asignar, esta asignación se realice de forma correcta.
- Tratamiento de señales. El núcleo deberá de conocer la naturaleza de la señal que hay detrás de un identificador pasado por parámetros, si es un vector o no; si un parámetro contiene un valor constante o inmediato; y desglosar en identificador – límite superior – límite inferior del subvector, en caso de ser la señal una porción de vector.
- Generación de especificaciones. El núcleo será el que recoja todas las especificaciones generadas por los componentes, la información relativa a las declaraciones de señales que este utiliza y generar un código en VHDL libre de errores.
- Generar Informe de Errores. Otra de las funcionalidades del núcleo es la de generar un informe sobre los errores semánticos cometidos a la hora de especificar un circuito en alto nivel. Cuando se detecte un error, se volcará en un fichero de texto de errores, la denominación del error cometido, el identificador si procede del elemento que ha causado el error, y el número de orden del componente donde se ha detectado dicho error.

De esta forma, TEAPAN toma una especificación y da un informe sobre los posibles errores semánticos cometidos; simplifica la especificación de las señales, creándolas de forma automática en función de la naturaleza de los componentes a las que se conectan; y generando una especificación única y plana en VHDL libre de errores.

7. EJEMPLO DE ESPECIFICACIÓN.

Con el fin de demostrar la capacidad de especificar computadores que tiene nuestra herramienta, en esta sección vamos a mostrar con un ejemplo cómo sería la especificación de un computador pipeline sencillo. Para ello, primeramente vamos a definir la arquitectura; después vamos a mostrar cómo sería la especificación con nuestra herramienta; a continuación mostraremos unos fragmentos de la especificación generada en VHDL (no la mostraremos completa debido a su extensión); y, por último, lo simularemos para comprobar que funciona conforme a lo esperado.

El computador que vamos a especificar va a tener una arquitectura con dos etapas de Pipeline, en la primera tomará un valor de ocho bits de una ROM, y sumará los dos nibbles (palabras de 4 bits) que forman dicho valor. En la segunda etapa de Pipeline se almacenará el resultado dentro de una memoria RAM.

La arquitectura que hace esto se puede apreciar en la figura 6:

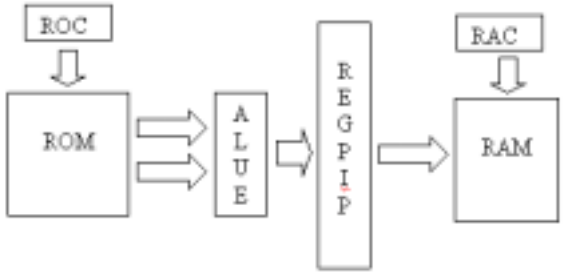

Figura 6: Arquitectura del computador Sumador.

En esta figura, se pueden apreciar los seis componentes más importantes que forman nuestro computador: una ROM que contiene los números a sumar; un registro contador (ROC) que apunta al siguiente par de nibbles a sumar; una ALU que realiza dicha suma; un registro de Pipeline (REGPIP), una RAM que almacena el resultado y un registro contador (RAC) que apunta al siguiente valor donde se almacenará el resultado en la RAM.

La especificación detallada del computador dada con TEAPAN, se puede apreciar en la figura 7:

```
ABRE("CSumador");
ENTRADA("Clk");
ENTRADA("Rst");
SALIDA("RomSal");
SALIDA("RamEnt");
REGISTRO("ROC",2,"IC","","RomDir");
ROM("sumandos",8,4,"RomDat");
ALUE("alu",4,"RomDat(7 downto 4)","RomDat(3 downto 0)","SalAlu");
REGPIP(0,4,"SalAlu","RamDat");
REGISTRO("RAC",2,"IP","","RamDir");
RAM("sumas",4,4,"RamDat");
COMPONER("RamDir","sumasDIR");
COMPONER("RomDir","sumandosDIR");
ASIGNA("RomDat","RomSal");
ASIGNA("RamDat","RamEnt");
AND("Clk","RP0LD");
AND("0","RP0ST");
AND("0","RP0BB");
AND("0","RP0CL");
AND("Clk","RACIC");
AND("Rst","RACPR");
AND("Clk","ROCIC");
AND("Rst","ROCCL");
AND("Clk","sumasW");
AND("0","aluCT(0)");
AND("0","aluCT(1)");
CIERRA();
```

Figura 7: Especificación TEAPAN

En esta figura, primeramente aparece el nombre de la especificación con la directiva ABRE(…); después se ha especificado las señales que se desea que pertenezca a la interfaz de entrada (Reloj y Reset), y de salida (Salida de la ROM, y Entrada de la RAM). En negrita se han señalado aquellos componentes que aparecen en la figura 6 y que son los más relevantes de la arquitectura. La última parte de la especificación está formada por una serie de componentes que interconectan a las diferentes señales de control de estos dispositivos para lograr un funcionamiento correcto.

Al compilar este código con un Lenguaje de C++ y ejecutar el programa resultante, obtendremos la especificación VHDL lista para simular. A modo de ejemplo, y debido a la gran extensión de la especificación VHDL, mostramos solamente la interfaz del sistema que se genera con esta especificación, y la porción de código que genera el componente RAM (figura 8).

```
ENTITY CSumador IS
PORT(
    RamEnt : out std_logic_vector(3 downto 0);
    RomSal : out std_logic_vector(7 downto 0);
    Rst : in std_logic;
    Clk : in std_logic);
END CSumador;
```

```
ARCHITECTURE funcional OF CSumador IS
                    ...
RamDir <= RAC;
process(sumasW)
begin
    if sumasW = '1' and sumasW'event then
        if sumasDIR(0) = '0' and sumasDIR(1) = '0' then
                sumas0 <= RamDat;
        elsif sumasDIR(0) = '1' and sumasDIR(1) = '0' then
                sumas1 <= RamDat;
        elsif sumasDIR(0) = '0' and sumasDIR(1) = '1' then
                sumas2 <= RamDat;
        elsif sumasDIR(0) = '1' and sumasDIR(1) = '1' then
                sumas3 <= RamDat;
        end if;
    end if;
end process;
with sumasDIR select
sumasint <=
    sumas0 when "00",
    sumas1 when "01",
    sumas2 when "10",
    sumas3 when others;
RamDat <= sumasint when sumasOE='1' else "ZZZZ";
                    ...
```

Figura 8: Interfaz y componente RAM en VHDL

De esta figura, es importante destacar el contraste entre la simplicidad mostrada en la especificación de TEAPAN, en la que con cuatro componentes especificamos la interfaz sin necesidad de indicar la anchura de las señales, frente a la especificación VHDL más detallada y compleja. Con el componente RAM, se puede comprobar esto mismo pero de una forma más exagerada, pues con tan sólo una línea de especificación: RAM("sumas",4,4,"RamDat"), se genera una gran cantidad de líneas detalladas en VHDL (sin contar con las señales que se declaran de forma automática).

Para comprobar que este código es válido, podemos simularlo con cualquier herramienta de simulación de VHDL. En nuestro caso hemos escogido MODELSIM de Mentor Graphics, con la cual, al introducirle un patrón de entrada con varios pulsos de reloj y un pulso inicial en la señal de reset, hemos obtenido el siguiente resultado:

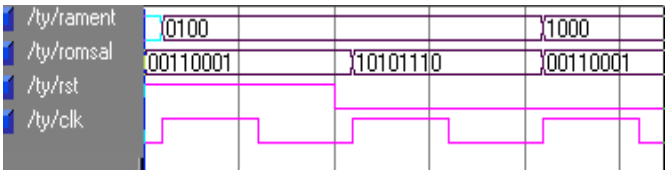

Figura 9: Resultado de la simulación del computador sumador

En la Figura 9 se muestran cuatro señales: “rament”, que se corresponde con el dato que se almacenará en la RAM; “romsal”, que se corresponde con el dato que se lee de la ROM; la señal “rst” que inicializa al sistema; y la señal “clk”, que es la señal de reloj.

Tal y como se puede apreciar en la figura, en un primer ciclo aparece el dato de ocho bits cuyos nibbles se desean sumar (1010 1110), y en un segundo ciclo, aparece la suma a almacenar en la RAM (1000), por lo que se puede comprobar que el funcionamiento de la arquitectura es correcto.

8. CONCLUSIONES

En esta comunicación se ha presentado una herramienta que permite diseñar arquitecturas de procesadores de forma sencilla.

Para desarrollar esta herramienta, en primer lugar se ha hecho un estudio de los microprocesadores actuales más importantes. En base a esto se ha generado una librería de bloques con los cuales pueden construirse procesadores de distintas arquitecturas. Los hemos clasificado en función de la importancia de estos en la Arquitectura Pipeline (“Componentes de Arquitectura Pipeline” frente “Componentes Opcionales”); y en función de la complejidad de los mismos (“Componentes de Unidades Funcionales y de Control”).

Una vez realizada esa librería de bloques, se ha construido una herramienta en la cual se implementan estos componentes, y se ofrece unas serie de recursos para que el usuario pueda implementar los componentes pertenecientes a la categoría “Unidades Funcionales y Control”. TEAPAN incluye también una serie de mecanismos para interconectar los componentes de forma fácil y generar algunos informes de errores de diseño.

Finalmente se ha especificado y simulado un computador pipeline de ejemplo, para así demostrar la validez de nuestra herramienta en el diseño de este tipo de arquitecturas.

9. REFERENCIAS

[1] Paterson, Hennesy, "Architecture of Computers", McGraw Hill, Second Ed. 1995.

[2] Tadpole Technology, "Sparc Book 3 Series: Technical Reference Manual". Ed 1997.

[3] John H. Edmonson, Paul I. Rubinfeld, others, "Internal Organization of the Alpha 21164, a 300-MHz 64-bit Quad-issue CMOS RISC Microprocessor", Ed Digital Equipment Corporation 1995.

[4] Ilhyum Kim, Donghyun Baik, "Mapping DSP Algorithms to General-Purpose Out-of-Order Processor", Ed ECE734 Project Report 2002.

[5] Paterson, Hennesy, "Architecture of Computers", McGraw Hill, Third Ed. 2003.

[6] David Klepacki, "IA32 Architecture: Basic Arquitecture and Programing Performance". Ed. IBM Thomas J. Watson Research Center, 2001.
[7] Intel Corporation, "Can a server give your business a competitive edge?", Ed Intel Literature Center 2002.
[8] AMD, "AMD Athlon Processor x86 Code Optimization Guide", Ed Advanced Micro Devices, Inc 2002.
[9] Motorola, IBM, "Power PC 601 RISC Microprocessor User's Manual", Ed Motorola Inc. 1995, International Business Machines Corp. 1991-1995.
[10] D. Peñalosa, C.J. Jiménez, M.Valencia, “Descripción de Arquitecturas de Procesadores a Alto Nivel con HEAPAN”, Iberchip, 2005. , págs 42-45.
[11] Jan Gray, “Building a RISC system in a FPGA”, Circuit Cellar, March 2000, págs 26-32,April 2000 págs 1-7.
[12] C.N. Liu, J.Y. Jou, “Efficent Coverage analisys metric for HDL design validation”, IEE, January 2001, págs 1 –6.