UNIVERSIDAD DE CANTABRIA
DEPARTAMENTO DE ELECTRONICA

PSAL : ESTUDIO, ANALISIS E IMPLEMENTACION DE ALGORITMOS DE SINTESIS DE ALTO NIVEL

MEMORIA
Presentada para optar al Grado de
DOCTOR EN CIENCIAS FISICAS
por el Licenciado en Ciencias,

Pablo Pedro Sánchez Espeso

EL DIRECTOR

D. Eugenio Villar Bonet
Profesor Titular de Electrónica
Departamento de Electrónica
Universidad de Cantabria

Santander, Marzo de 1991

CAPITULO 4

PSAL2: SISTEMA DE SINTESIS DE ALTO NIVEL

I) Introducción

Una vez concluida la implementación de PSAL1 se procedio a su análisis y evaluación. Las conclusiones de este trabajo fueron reflejadas en el desarrollo de un nuevo sistema, PSAL2 (Programa de Síntesis ALgorítmica, versión 2) , que supera muchas de las limitaciones de su antecesor.

Dos son los condicionantes que determinan el tipo de sistema que PSAL1 va a sintetizar. Por un lado, las metodologías de síntesis del programa, basadas en un determinado estilo de diseño. Por otro, los lenguajes utilizados para describir el sistema antes y despues del proceso de síntesis. En este sentido, el programa PSAL1 solo sintetizará correctamente aquellos circuitos que admitan una descripción algorítmica en ISPS (lenguaje orientado a procesadores de conjunto de instrucciones). Además, las estruturas de datos de PSAL1 estan especialmente orientadas a dicho lenguaje, lo que conlleva ciertas dificultades durante el proceso de síntesis. Con objeto de salvar este obstaculo, PSAL2 fue desarrollado de forma independiente del lenguaje de descripción de hardware utilizado para describir inicialmente el sistema, disponiendo de un lenguaje propio, que permite acceder al diseño o introducir nuevos datos en cualquier fase del proceso de síntesis. Como consecuencia, las primitivas de dicho lenguaje estan fuertemente relacionadas con la representación intermedia, reflejando en gran medidad su estructura y mostrando toda la información importante que dicha representación intermedia contiene. Además, al poder ser expresado el estado del sistema en cualquier momento, el lenguaje diseñado se convierte en una poderosa herramienta de ayuda en el análisis y evaluación de los algoritmos utilizados por el sistema en el proceso de síntesis, dado que permite acceder a la información de la base de datos cuando el usuario desee, pudiendo continuar despues el proceso de síntesis. En la implementación actual, además de esta representación, puede utilizarse el lenguaje estandar multinivel VHDL para describir inicialmente el sistema a diseñar.

La segunda causa que limitaba la capacidad de PSAL1 erán las metodologías de síntesis.En PSAL2 se ha optado por no utilizar una metodología prefijada. Ello nos ha llevado a sustituir el esquema vertical de PSAL1 (figura 3.1), en la cual los diferentes procesos de síntesis eran ejecutados secuencialmente de forma rígida, por una estructura horizontal (figura 4.1) , en donde los algoritmos actuan sobre la base de datos, sin un orden predeterminado.

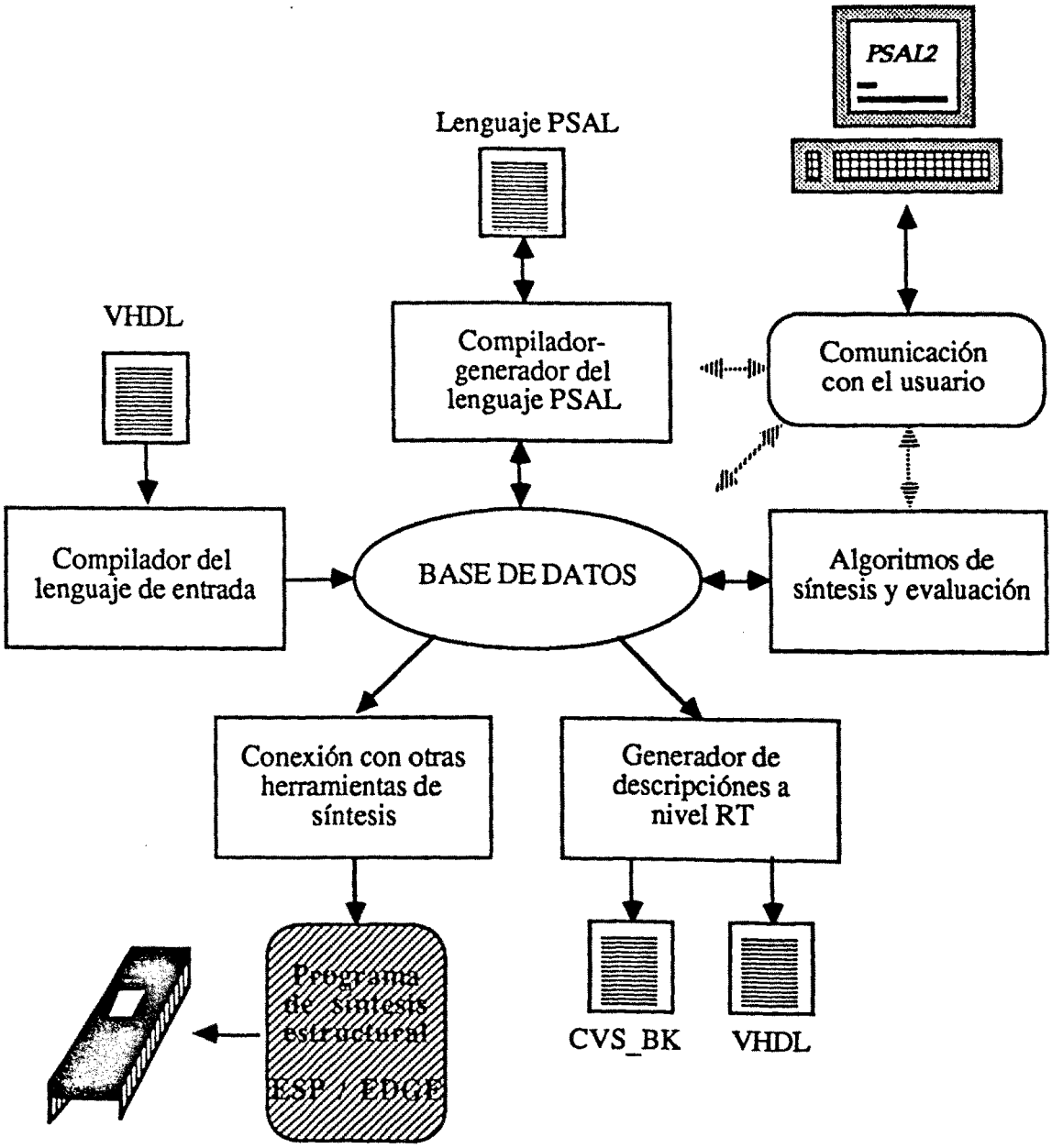


Figura 4.1

Ello permite adaptar el sistema a varias metodologías de diseño con facilidad. Con objeto de alcanzar este objetivo, la representación intermedia o base de datos almacena una doble visión (estructural y algorítmica) del sistema a diseñar. Gracias a ello, no es preciso dar un significado estructural a las primitivas algorítmicas, lo que confiere al sistema una mayor flexibilidad y tolerancia en cuanto a las metodologías de diseño a utilizar y arquitecturas a implementar.

Otra de las ventajas que ofrece el sistema es su conexión con otras herramientas CAD. En particular, la conexión entre PSAL2 y el sistema ESP (programa de síntesis estructural actualmente operativo en el grupo), permite disponer de un sistema de diseño completo desde el nivel algorítmico al lógico, nivel al cual se produce la conexión con las herramientas de diseño de circuitos integrados del Grupo de Microelectrónica de la Universidad de Cantabria (en la actualidad el sistema EDGE de CADENCE). Esta estructura permite continuar el proceso de diseño iniciado por PSAL2 hasta el layout final. Otro esfuerzo realizado en esta dirección ha sido la implementación de un módulo que permite expresar el resultado de la síntesis en el lenguaje CVS_BK [CVS88].

En el capítulo anterior se indicaban los 5 condicionantes del proceso de síntesis impuestos por PSAL1. De ellos solo el diseño de sistemas síncronos sigue siendo una restricción en PSAL2. Todas las demás han sido superadas. Por ejemplo, el nuevo sistema no esta restringido al esquema clásico de una unidad de control y una unidad de datos. PSAL2 admite sistemas con varias unidades de control, lo que le confiere una gran flexibilidad sin aumentar su complejidad.

El nuevo sistema permite situaciones no contempladas en el anterior como encadenamiento de operaciones, asignación de variables a terminales o unidades operacionales con mas de dos operandos.

La estructura básica de PSAL2 aparece en la figura 4.1. Como se puede observar, presenta una estructura horizontal, muy diferente a la que presentaba su antecesor PSAL1 (figura 3.1). En ella se pueden diferenciar varias partes:

a) Comunicación con el usuario.
Dado que el sistema tiene una estructra muy flexible es necesario un módulo que permita al usuario especificar todos los parámetros que el sistema precisa para realizar la sintesis. Este entorno permite además obtener información sobre el estado actual del proceso. El "interface" de usuario dispone (en su versión "sunview") de menús que permiten al usurario introducir las ordenes de una forma cómoda.

b) Compilador del lenguaje de entrada.

Como ya se ha comentado existen dos lenguajes que permiten introducir la descripción inicial en el sistema : VHDL y PSAL. PSAL2 utiliza un subconjunto del lenguaje multinivel estandar VHDL para describir el sistema a sintetizar a nivel algorítmico. La definición de dicho subconjunto fue realizada en el capítulo 2. El módulo encargado de la compilación de este subconjunto (VHDL2PSAL) no genera directamente la nueva base de datos, sino la descripción del sistema en lenguaje PSAL. Antes de que dicha descripción sea introducida en PSAL2 se hace necesario un preprocesado de ésta. El objetivo de esta tarea es reemplazar los símbolos introducidos por el compilador (VHDL2PSAL) por sus valores, ya que estos no eran conocidos cuando aquel escribió los simbolos (proceso similar al realizado por el comando "link" con los lenguajes software). Para alcanzar este objetivo se utiliza el preprocesador del lenguaje C, cpp, el cual hace uso de un fichero generado por VHDL2PSAL en el cual se define el valor de los símbolos utilizando la directiva "#define".

c) Compilador-generador del lenguaje PSAL2.

Uno de los puntos débiles de PSAL1 era la falta de un lenguaje que permitiese representar el diseño en cualquier fase del proceso de síntesis. El lenguaje PSAL es un medio que el usuario puede emplear para conocer en cualquier instante el estado exacto del proceso de síntesis. Además permite introducir representaciones en la base de datos lo que facilita de forma notable la evaluación de los diferentes algoritmos.

d) Base de datos.

Durante el proceso de síntesis los datos son almacenados utilizando un cierto formato intermedio. En nuestro sistema, dicha representación se caracteriza por almacenar una doble visión del sistema a diseñar (estructural y algorítmica), lo que aumenta su flexibilidad y capacidad de modelado.

e) Algoritmos de síntesis y evaluación.

Frente a la estructura vertical de PSAL1 (figura 3.1), PSAL2 presenta una estructura horizontal. Ello supone que el sistema no impone (por su construcción) una determinada secuencia de tareas de síntesis, dando por lo tanto gran libertad, en cuanto a elección de metodología de síntesis se refiere, al usuario. PSAL2 dispone de algoritmos de análisis de variables, asignación de operaciones y localización del "data path" (que serán descritos mas adelante), pudiendo el usuario escoger en cada caso el algoritmo que desee. De esta forma el sistema pone a disposición del usuario tres algoritmos de asignación de operaciones: "Forced directed", "List scheduling" y localización global. La elección de uno u otro algoritmo dependerá del tipo de restricciones que el usuario quiera introducir, por ejemplo, si desea principalmente

reducir el tiempo de ejecución puede utilizar el primer o el último algoritmo, pero si desea introducir restricciones que limiten el coste final del sistema debe de utilizar los otros dos. El último algoritmo, a diferencia de los anteriores, realiza una asignación teniendo en cuenta las operaciones de control, por lo que suele producir implementaciones más rápidas que los anteriores.

Los algoritmos de síntesis del data-path que utiliza PSAL2, pueden ser clasificados en dos grupos:

- Algoritmos especializados. Cada una de estas metodologias solo resuelven, en PSAL2, una de las subtareas de la síntesis del data path. Por ejemplo, se utiliza un algoritmo heuristico de partición de grafos en cliques para realizar la síntesis de registros y una técnica iterativo-constructiva para asignar las operaciones a unidades operacionales.

- Algoritmo general. PSAL2 utiliza una metodología basada en probabilidad para realizar cualquiera de las tareas de síntesis del data path. Entre las ventajas de esta nueva técnica, destacan la posibilidad de evaluar el coste del sistema en cualquier momento del proceso de síntesis lo que permite dirigir ésta con objeto de minimizar el coste final del sistema.

f) Módulos encargados de expresar el resultado del proceso de síntesis a nivel estructural.

Existe un módulo encargado de representar el sistema resultante a nivel de transferencias de registros, de forma que pueda ser utilizada por programas de síntesis estructural. PSAL2 dispone de módulos que le permiten expresar el resultado de la síntesis en los lenguajes VHDL y CVS_BK. La elección de estos lenguajes es una consecuencia del proceso historico que ha tenido este trabajo, como ya se comentó en la Introducción. Además se ha implementado un módulo que permite almacenar todo la información que utiliza la herramienta en la base de datos estandar del sistema operativo UNIX, DBM. Esta base de datos es una forma de conectar el sistema PSAL2 con otras herramientas, pudiendose utilizar mas adelante como medio de almacenar resultados parciales del procesos de síntesis y de esta forma permitir una mas comoda exploración del espacio de diseño.

g) Conexión con otras herramientas de síntesis.

El sistema PSAL2 dispone de un módulo que le permite expresar el resultado de la síntesis en un formato aceptado por la herramienta de diseño a nivel estructural ESP, actualmente operativa en el grupo. Con ello, el sistema se integra en el conjunto de

herramientas de diseño de circuitos integrados actualmente en uso en el grupo. Este entorno utiliza PSAL2 como herramienta de síntesis a nivel algorítmico, ESP a nivel de transferencia de regitros, siendo realizada la síntesis desde el nivel lógico al layout por EDGE (desarrollado por CADENCE).

Una novedad que incorpora PSAL2 respecto de su antecesor es la de estar implementado sobre el sistema operativo UNIX. Las anterior versión lo estaban sobre VMS. Gracias a ello, el sistema puede utilizar algunas de las multiples herramientas y librerias estandard que dicho sistema operativo ofrece, con las consiguientes ventajas en cuanto a simplicidad y verificación del codigo programado. El programa esta escrito en lenguaje C y consta de unas 30.000 lineas de código.

II) Interface de usuario.

PSAL2 es un sistema caracterizado por su flexibilidad. Por esta razón se ha desarrollado un "interface" de usuario que permite un facil dialogo entre éste y el sistema. Este interface esta adaptado a tres tipos de entornos gráficos:

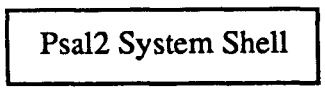
- Sunview.
- Terminal VT100.
- Entorno diferente de los anteriormente señalados.

Desde un entorno Sunview (sistema de ventanas estandar en SUN), es posible llamar al sistema desde el menu general ("rootmenu"). Al escoger la opción "PSAL2" de dicho menu, aparece una ventana en la cual se esta ejecutando el sistema. Desde esta ventana se puede controlar u obtener cualquier información sobre el proceso de síntesis bien a traves de comandos escritos mediante el teclado, bien mediante la utilización de la opción "Extras" del menu de dicha ventana. Dicha opción abre una nuevo menu en el cual aparecen todos los comandos de PSAL2, permitiendo el usuario una manejo más cómodo del sistema. Dicha ventana dispone además de un icono especial ("PSAL2icon"), que la identifica cuando no esta abierta. El entorno Sunview dispone además de un comando que no tienen los otros entornos : "PSALlogo". La ejecución de este comando genera una ventana con el nombre e identificación del sistema PSAL2.

Para usuarios que no deseen utilizar el entorno Sunview, se han previsto dos entornos alternativos. El primero supone que el terminal sobre el que trabaja el usuario es compatible VT100. En este interface el sistema hace uso de algunos códigos especiales, como por ejemplo, modificadores del tamaño de un caracter, caracteres especiales o parpadeo. Una vez determinado que el terminal es VT100 compatible, el interface muestra la carátula de

presentación de PSAL2, para a continuación pasar al estado normal de trabajo. En el caso de que el usuario no disponga de un terminal de estas caracteristicas, existe una opción que impide al sistema utilizar los caracteres especiales antes mencionados.

Despues de mostrar una carátula de presentación, el programa simula una "shell" UNIX, desde la cual se pueden dar instrucciones al programa o recibir información de éste. En la figura 4.2 puede verse un aspecto de la consola cuando el usuario trabaja con el sistema.




```
Psal>
Psal>set file ../example/exam3
Psal>set input psal2
Psal>set output cvs_bk
Psal>sh

   STATUS DEL SISTEMA

data :
      data base : off

input :
      file : ../example/exam3.psal2
      type : psal2
      lis. : standard output (OFF)

output :
      file : ../example/exam3.cvs
      type : cvs_bk

Psal>load
```

Figura 4.2

Este entorno admite diversos tipos de comandos, cuya descripción será realizada a continuación:

- **"allocation"**. Este comando inicia la ejecución de los algorítmos de síntesis del "data-path".

- **"comment"**. La línea que se inicia con este comando es ignorada por el sistema. Permite introducir comentarios en los ficheros que mas tarde serán leidos con el comando **source**.

- **"data"**. Realiza un análisis del flujo de variables.

- **"default"**. Permite definir los parámetros por defecto que despues utilizará el sistema. La ejecución de este comando introduce al usuario en una nueva "shell", con comandos específicos. El "prompt" se modifica, apareciendo la indicación "default", con objeto de indicar el tipo de "shell" con la que se trabaja. En ella se admiten los siguientes comandos:

  - **"set" [elemento dato]**. Esta primitiva permite asignar un valor ("dato") a un parámetro ("elemento"). En el caso de no especificarse el elemento, el sistema preguntará al usuario el valor de todos los parámetros no definidos. En la actualidad se pueden definir los tipos de módulos asociados a los puertos del sistema y de las memorias.

  - **"show" [elemento]**. Muestra el valor del parámetro "elemento". Si dicho parámetro no se especifica, se muestran todos los definidos.

  - **"quit"** . Este comando finaliza la ejecución de esta "shell" especial. El "promt" de la "shell" vuelve a tener su aspecto original, pudiendose volvar a utilizar los comandos generales.

- **"edit" [fichero]**. Edición del fichero "fichero" (si es especificado). Se utiliza el editor de texto "vi".

- **"exit"**. Finaliza la ejecución de PSAL2.

- **"help"**. Comando de ayuda. Muestra una breve descripción de todos los comandos aceptados por PSAL2.

- **"load"**. Esta primitiva permite introducir una nueva descripción en la base de datos. El lenguaje en el cual esta descrito el sistema es especificado mediante el comando "set input", y el nombre del fichero que lo contiene con "set file".

- **"more" file.** Este comando es equivalente a la primitiva "more" del sistema operativo UNIX. Permite conocer el contenido del fichero "file".

- **"scheduling".** Ejecución de los algoritmos de localización de operaciones. En la implementación actual se puede escoger entre 3 algoritmos:

  - "Forced directed"
  - "List scheduling".
  - Localización global.

- **"set" opcion valor.** Es este un comando muy versatil que permite especificar varias opciones relacionadas con el interface de usuario y los lenguajes de entrada y salida. Las opciones y valores permitidos son:

  - **"file" name.** Especifica el nombre del fichero sin extensión, en el que se almacena la descripción inicial del sistema y donde se escribirá el resultado de la síntesis. Cuando se pueda utilizar el mismo lenguaje para representar el sistema a nivel algorítmico y RT (casos de PSAL y VHDL), el sistema añade al nombre del fichero la terminación "_s", dado que en otro caso los ficheros de entrada y salida tendrian el mismo nombre y extensión, produciendose un borrado de la descripción de entrada cada vez que se genere una salida.

  - **" flashing on | off.** Señala si el título de la "shell" debe de estar o no en modo "flash". Este comando solo tiene sentido en un terminal VT100.

  - **"input" tipo.** Especifica el tipo de lenguaje utilizado para representar inicialmente al sistema. Los tipos reconocidos son: PSAL y VHDL.

  - **"listing " on | off.** Señala al sistema si la información generada por los compiladores de los lenguajes de entrada al ejecutar el comando "load", debe de ser enviada a la salida estandar (opción "off") o a un fichero (opcion "on").

  - **"output" tipo.** Especifica el lenguaje en el cual será descrito el sistema resultante del proceso de síntesis. Los tipos reconocidos son:

    - PSAL
    - VHDL.
    - CVS_BK
    - ESP
    - DBM.

- **"prompt" new_prompt** . Toma "new_prompt" como nuevo prompt del sistema.

- **"show"**. Muestra información sobre los tipos de lenguajes seleccionados, asi como si ya ha sido cargada la base de datos.

- **"source" file**. Permite ejecutar comandos desde el fichero "file". Este comando fue introducido con objeto de que todas las definiciones de los valores por defecto de los parámetros utilizados por PSAL2, pudiesen ser leidas desde un fichero en vez de escritas en la "shell" correspondiente.

- **"unix"** . Este comando detiene momentaneamente la ejecución de PSAL2, dando paso a una "cshell" del sistema operativo UNIX. Esto permite realizar operaciones en el sistema operativo sin necesidad de salir del entorno PSAL2. Para volver al sistema de síntesis, hay que utilizar el comando UNIX, "exit".

- **"write"**. El sistema resultante del proceso de síntesis será descrito en el lenguaje a nivel de transferencias de registros, seleccionado anteriormente con "set output", tras la invocación a este comando.

III) El lenguaje PSAL2

Algunas de las restricciones que tenía PSAL1 eran producidas por la dependencia entre el tipo de lenguaje utilizado para describir el sistema a nivel algorítmico y el formato intermedio utilizado por el sistema. Con el fin de superar este problema, PSAL2 fue concebido de forma independiente del lenguaje de entrada. Ello obligó a definir un formato que permitiese introducir y extraer información en la base de datos. La necesidad de este formato ya había empezado a dislumbrarse en PSAL1.

El lenguaje PSAL nació con el fin de cubrir esta necesidad. Esta representación no es un lenguaje algorítmico clásico, más bien es un lenguaje similar al empleado en las bases de datos. En PSAL2 sucede lo contrario que en PSAL1: no existe un lenguaje que condicione la base de datos (como pasaba con el primer sistema y el lenguaje ISPS) sino que hay un lenguaje que refleja los datos contenidos en el formato intermedio asi como su estructura. Esta representación presenta una notable ventaja : permite representar el diseño en cualquier fase del proceso de síntesis. Gracias a esta propiedad es posible analizar, introducir o modificar los datos contenidos en el programa de una forma sencilla.

La descripción se inicia con una sentencia "system", tras la cual se especifica al nombre del sistema a diseñar. La descipción del sistema aparece entre los "token" "begin" y "end", tal y como se puede observar en la regla "Psal_input":

Psal_input ::= **SYSTEM** [name] := **BEGIN** definitions **END**

La definición del sistema en este lenguaje se divide en dos partes, las cuales reflejan las dos vistas del sistema (estructural y algorítmica), contenidas de la base de datos:

definitions ::= [**DEFARQ** := **BEGIN** arq_def **END**]
DEFBEH := **BEGIN** beh_def **END**

La definición de la arquitectura es opcional, ya que no tiene sentido en descripciones a nivel algorítmico. En ella se declaran todos los módulos hardware que forman el sistema, asi como sus interconexiones. Su definición formal aparece en la regla "arq_def" :

arq_def :: = **DEFMOD** := **BEGIN** mod_list **END**,
DEFPOR := **BEGIN** por_list **END**,
DEFWIR := **BEGIN** wir_list **END**

Como se puede observar la definición de la arquitectura esta formada por 3 subsistemas:

1) Definición de módulos.
2) Declaración de puertos.
3) Definición de interconexiones.

Los módulos son los elementos hardware que forman el sistema digital. En la definición de uno de estos elementos (regla mod_list) se indica el nombre y número del módulo asi como su tipo ("code") tal y como se ve en la siguiente regla :

mod_list ::= { **CREAMOD** number , [**CALLED** name ,] code, flag ,
((in_number : { port_number_list }) | (**#p** in_number : { module_number_list })),
((out_number : { port_number_list }) | (**#p** out_number : {module_number_list })),
bit_size [, word_size] (([, **#o** oper_number : { code class bit_size control_code
[delay nutil] | }) | (, **#R** word_number : { number }) | (, **#m** module_number :
{number })) [, #d delay_number] [, #c : number] ; }

Un módulo puede tener puertos por entradas (como en el caso de un registro) u otros módulos (como en el caso de las memorias, cuyos puertos son en realidad terminales o regsitros modelados con otros módulos). En la regla "mod_list" aparece la opción

"in_number : ", cuya misión es modelar los puertos del módulo. En el caso de que éste tenga sus entradas representadas por módulos, se utiliza la segunda opción: "**#p** in_number:". Lo mismo pasa con las salidas: en el caso de que sean modeladas con puertos se utliza la opción "out_number: ", y si son módulos se emplea "**#p** out_number :".

Para cada módulo, se especifica además, el número de bits y palabras (solo en memorias) del elemento. Un módulo puede modelar unidades operacionales (opción "**#o**"), ROMs (con la opción "**#R**" se puede especificar su contenido) o macro-bloques (opción "**#m**"). Otros elementos, como memorias, registros o terminales no tienen opción específica. Se señala igualmente el retraso maximo del módulo ("**#d**"), lo que puede ser utilizado por herramientas de análisis del sistema para determinar caminos críticos y retrasos en los elementos del sistema. El hecho de que esta información se almacene en la base de datos confiere al sistema una gran flexibilidad ya que separa los conceptos de operación y retraso, permitiendo utilizar valores diferentes para la misma operación (en función, por ejemplo del número de bits o de la estructura elegida para implementar la operación). En la base de datos se almacena igualmente información sobre el coste del módulo, lo que se refleja en la opción "#c" de la regla "mod_list". El objetivo es similar al descrito anteriormente: dar coste a los módulos particulares, en vez de dar costes generales a las operaciones.

Como se acaba de señalar, existen módulos especiales, denominados macro-bloques, que agrupan diversos módulos e interconexiones. Estos módulos representan funciones combinacionales complejas. Además, al no existir limitaciones en el número de entradas y salidas, se pueden definir todo tipo de elementos. El lenguaje dispone además de módulos de tipo "caja negra", cuya funcionalidad es desconocida, y únicamente se conocen sus entradas y salidas.

Los puertos representan las entradas y salidas de un módulo. Su definición aparece en la regla "por_list":

```
port_list ::= { CREAPOR number, flag, left_bit, righ_bit, module_numbe,
                input_port_number, out_port_number [ , #w left_word righ_word ]
                [ , #d delay_number ] ; }
```

En ellos se indican las anchuras en bits de dicho elemento de interconexión asi como el módulo al cual estan asociados. En los puertos de los módulos asociados a las direcciones de las memorias aparecen los rangos de palabras a los que accede.

En la definición de las interconexiones ("wir_list") se señala como se conectan los puertos señalando además el rango de bits que intervienen en la conexión, asi como el coste y el

retraso de la misma.

wir_list ::= { **CREAWIR** number, left_bit, righ_bit, port1_number, left_bit, righ_bit,
port2_number [, #d delay_number] [, #c number] ; }

En la segunda parte de la descripción, declaración del comportamiento , se diferencian cuatro partes:

1) Definición de subrutinas.
2) Declaración de variables.
3) Definición de links (opcional).
4) Descripción del comportamiento en forma de sentencias.

Su definición formal aparece en la regla "beh_def":

beh_def ::= **DEFENT** := **BEGIN** ent_list **END** ,
DEFVAR := **BEGIN** var_list **END** ,
[**DEFLIN** := **BEGIN** lin_list **END** ,]
DEFSEN := **BEGIN** sen_list **END**

La descripción del comportamiento en PSAL2 es una descripción jerárquica en la cual las sentencias se agrupan en funciones o subrutinas. En la declaración de éstas, se señala su nombre, función en la que es definida asi como las sentencias que contiene, tal y como aparece a la regla "ent_list":

ent_list ::= { **CREAENT** number, [**CALLED** name ,] flag , contex_entity_number
[, **#i** first_sentence_number] ; }

En la declaración de variables se indica el nombre (o el valor en las constantes), tipo, número de bits y función en la que es definida. En las memorias se indica además, el número de palabras. Se admiten variables con multiples indices (arrays con multiples dimensiones, definidos mediante el "input_port_number"). Además se señala el módulo en el cual esta localizada la variable asi como el desplazamiento en bits y en palabras. La razón es que en el caso de que una variable sea localizada en una memoria (por ejemplo en un "register file"). Es preciso señalar la dirección (o lo que es lo mismo el desplazamiento con respecto a la primera dirección), en la cual se localiza. Algo parecido pasa con los bits: PSAL2 no exige que todas las variables asociadas a un registro tengan que tener un bit

común, ya que el parámetro bit_offset permite escoger los bits del registro que almacenarán el dato contenido en la variable. En la regla "var_list", aparece igualmente la definición de los segmentos de la variable (en la opción "#s"). Este concepto ya fue introducido en PSAL1, con objeto de analizar descripciones en las cuales se realiza un uso arbitrario de rangos de bits de la variables. Los valores que aparecen en la definición señalan el limite superior e inferior del segmento asi como su posición en los vectores de análisis del flujo de datos. Por otro lado, la opción "#v", permite señalar que variables modelan los puertos de las memorias.

var_list ::= { **CREAVAR** number,code, flag [(**CALLED** name) | (**VALUE** number),] left_bit, righ_bit, input_port_number : { left_bit righ_bit | } [,#w { left_word righ_word | }] [, #c contex_entity_number] [, mdule_number , bit_offset , word_offset , port_number] [, #v number { var_number }] [, #s number : { max_number min_number pos_number | }, use_number,def_number]; }

En la representación de un sistema en forma de grafo de flujo de datos, los nudos representan operaciones y los arcos variables. En el sistema PSAL2 las operaciones se modelan como sentencias y las variables poseen una definición propia. Se precisa un elemento que modele los arcos del grafo de flujo de datos, es decir, que represente las transferencias entre datos y operaciones. Este elemento es el link. En él se señala la variable y operación (o viceversa) que une. Además se indica la interconexión asociada a dicha transferencia. Su definición formal aparece en la regla "lin_list":

lin_list ::= { **CREALIN** number, flag, (variable_number | module_number), input_port, left_bit, righ_bit, (variable_number | module_number), out_port, left_bit, righ_bit [, delay [, { wire_list }]] ; }

Por último se describe el comportamiento del sistema en forma de sentencias. Se señala no solo la operación que se realiza sino tambien los datos (variables) y links empleados, asi como el módulo en el que estan localizadas y la función en la que son definidas. La descripción formal aparece en la regla "sen_list". Se puede observar que una sentencia puede tener entrada de datos o de control (caracterizada por el prefijo "ante"). Lo mismo ocurre con la salida.

sen_list ::= { **CREASEN** number, code, flag, (input_number: { variable_number left_bit righ_bit [link_number] | }) | (**ante** numbe : { sentence_number }), (out_number: {variable_number left_bit righ_bit [link_number] | }) | (**suc** number: { case max_index min_index sentence_number | })[,#m module_number][,absolute_state, relative_state, delay, control_signal_value] [,#f last_sentence_number] [,#c contex_entity_number] ; }

En un proximo apartado, se explicará con más detalle los tipos de operaciones asi como las conexiones existentes entre la parte estructural y la algorítmica. En la figura 4.3 se muestra un fragmento de una descripción utilizando este lenguaje.

```
system ejemplo1:=
begin

defarq:=
begin

 defmod:=
 begin
      creamod 1,called mem_x,3,98304,#p 1:9,#p 1:6,16,256;
      creamod 2,called mem_y,3,98304,#p 1:8,#p 1:7,16,256;
      creamod 3,called sub,4,0,2: 5 6,1:7,16,#o 1:66 0 16 .1|;
      creamod 4,called adder,4,0,2:8 9,1:10,32,#o 1:65 0 32 .1|;
      creamod 5,called r,1,0,1:11,1:12,32;
      creamod 6,called dat_bus_x,1,1032,1:13,1:13,16,#m 1:1;
      creamod 7,called dat_bus_y,1,1032,1:16,1:16,16,#m 1:2;
      creamod 8,called address_y,1,2056,1:19,0:,8,#m 1:2;
      creamod 9,called address_x,1,2056,1:17,0:,8,#m 1:1;
      creamod 10,called j,1,0,1:22,1:21,8;
      creamod 11,called cons_index,10,4,0:,1:23,8,#r 1:1;
      creamod 12,called cons_reset,10,4,0:,1:24,32,#r 1:0;
      creamod 13,called control,12,0,1:25,0:,0;
      creamod 14,called inc,4,0,2:26 27,1:29,8,#o 1:65 0 16 .1|;
```

Figura 4.3

Utilizando las herramientas YACC y LEX se ha implementado un Parser que permite compilar dicha descripción, generando la base de datos necesaria para el proceso de síntesis. Se dispone además de un módulo capaz de generar la descripción PSAL2 a partir de los datos contenidos en el formato intermedio. El Parser cuenta con casi 3.000 lineas de código (incluyendo las rutinas semánticas), y esta escrito en el lenguaje utilizado por la herramienta YACC.

IV) La Base de Datos.

El formato intermedio era, en nuestra opinión, uno de los puntos más restrictivos de PSAL1. El antiguo formato estaba demasiado ligado a ISPS y no soportaba nuevos algoritmos o nuevos esquemas de diseño con facilidad. La creación de un nuevo formato intermedio fue una de las primera conclusiones que se extrajo del análisis de PSAL1. El nuevo formato debía estar implementado sobre el sistema operativo UNIX (con objeto de poder utilizar muchas de las ventajas que dicho entorno ofrece) siendo al mismo tiempo más flexible y versatil que el antiguo.

Además de definir un nuevo formato, se ha implementado toda una librería de funciones que permiten el manejo, a bajo nivel, de dicha estructura. Tareas tales como la creación de nuevos elementos, su eliminación, inserción o extracción de una lista , el redimensionado, copiado o reinicializado son realizadas por dichas funciones. Ello permite un mejor manejo de la base de datos.

Como ya ha sido comentado, el formato intermedio del sistema PSAL2 se recoge una doble visión del sistema a diseñar. Por una parte se guarda la descripción del comportamiento del sistema, por otra la estructura que la implementa. Ambas descripciones estan interelacionadas.

Los algoritmos de síntesis tienen como misión no solo señalar los recursos hardware que se precisan, sino tambien las relaciones que se establecen entre dichos recursos y la descripción del comportamiento. De esta forma, los algoritmos de asignación de variables en registros, por ejemplo, no solo deben señalar el número de estos, sino tambien que variables son asignadas a cada registro (relación variable-registro). En la figura 4.4 se muestra gráficamente dicha estructura.

Los elementos que forman la vista algorítmica (operaciones y transferencias) se relacionan con los recursos hardware que lo implementan (OUs, registros e interconexiones). Como se puede observar, a la operación "a = i + j" se le asocia la unidad operacional en la cual se realiza dicha operación, además del estado en el cual se ejecuta (que no aparece en la figura). Por otro lado, en la vista estructural existe un elemento, la unidad de control, a la cual se asocian todas las acciones de control. Las transferencias de datos (en el ejemplo la transferencia del dato "a" desde la operación ya comentada a la de control) se relacionan con las interconexiones que unen los puertos de los modulos de la vista estructural. Estos "puertos", representados por triangulos en la figura, representan los "pines" del módulo. Su caracter direccional queda de manifiesto por la dirección que señalan. Además el almacenamiento de un dato por parte de la variable "a" es modelado a nivel estructural como un registro, relacionado con aquella. Con este esquema, la síntesis de alto nivel tiene por

objetivo determinar el número y tipo de los elementos que aparecen en la vista estructural asi como su relación con la vista algorítmica.

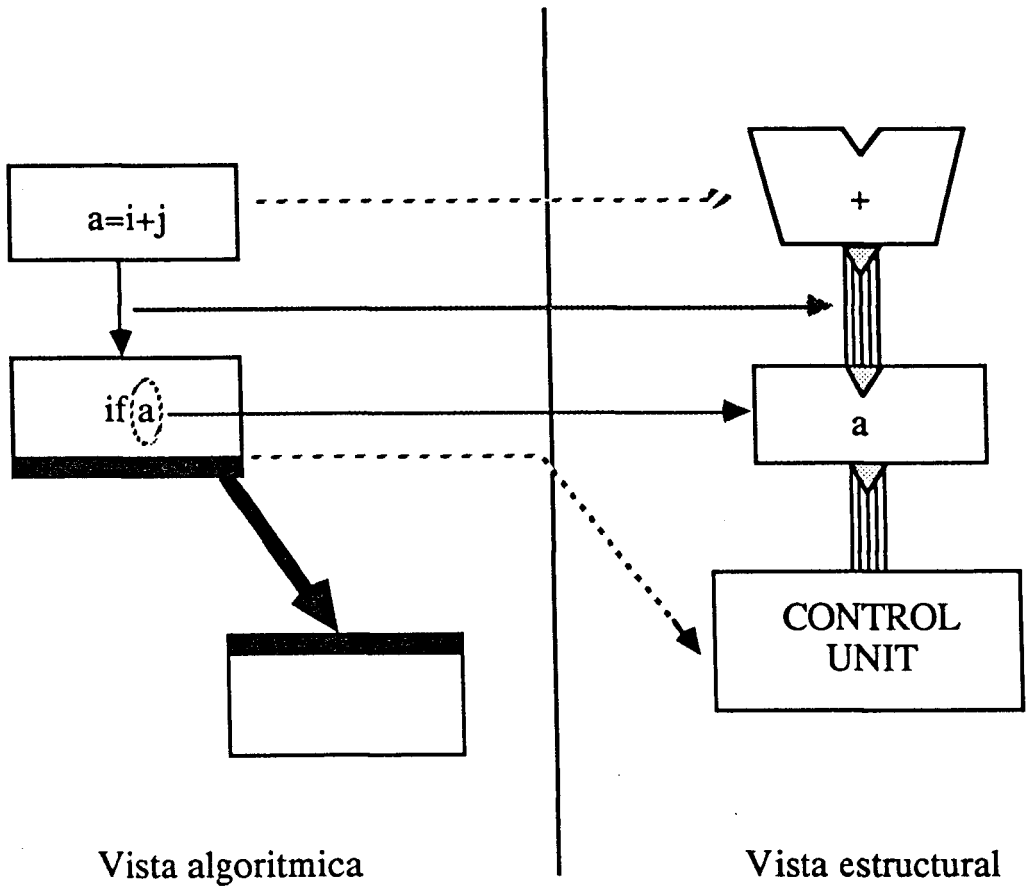


Figura 4.4

La vista estructural esta compuesta por 3 listas:

1º) Lista de módulos.
2º) Lista de puertos.
3º) Lista de conexiones.

La primera de ellas esta formada por los elementos que modelan los módulos, elementos hardware del diseño. Existen, en la implementación actual, 12 tipos de módulos:

1º) Registro.
2º) Terminal.
3º) memoria.
4º) Unidad operacional.
5º) Bus.
6º) Multiplexor.
7º) Caja negra.
8º) ROM.
9º) Macro-bloque.
10º) Constante algorítmica.
11º) Memoria algorítmica.
12º) Controlador.

Junto a elementos cuya representación hardware es evidente (registros, terminales, etc.), aparecen otros cuyo significado es el siguiente: una "caja negra" es un elemento, cuyo comportamiento es desconocido y del cual solo se conocen sus entradas y salidas. Gracias a esta primitiva se pueden introducir en el diseño bloques ya implementados.

En ocasiones existen módulos en los cuales se realizan funciones complejas. En principio en un módulo solo se puede realizar operaciones básicas (una suma por ejemplo). Cuando se desea realizar operaciones más complejas (calcular la suma de las diferencias entre varios datos, por ejemplo), es preciso unir varios módulos, de forma que cada uno de ellos realice una operación básica. Un macro-bloque es un tipo especial de elemento, que agrupa a varios módulos. El primero se interpreta como un único módulo, que realiza una función compleja. Dicha función será implementada como un único circuito por las herramientas de síntesis estructural (no varios conectados). Dichas herramientas de síntesis estructural utilizarán algoritmos de minimización lógica con objeto de optimizar la implementación del módulo. Al final del proceso, el macro-bloque se habrá transformado en un conjunto de puertas lógicas en el cual no tiene porque haberse conservado la estructura de módulos inicial.

Aparecen 2 tipos de módulos, a los cuales se les añade el calificativo de "algorítmico". Ello es debido a que no modelan exactamente un elemento hardware, sino un comportamiento. Una constante algorítmica, por ejemplo, será traducida a nivel estructural como una constante estructural, sin señalar como se implementa. Serán las herramientas de síntesis lógica las que decidan como implementarla. Dos son las formas de implementar una constante en PSAL2: implementarla en una ROM de constantes o, en el caso de constante algorítmica, dejando su implementación en manos de la herramienta de síntesis estructural.

La memoria algorítmica es un elemento sin significado hardware definido. Se utiliza para

mantener la coherencia interna de la base de datos (es el módulo en el cual se localizan memorias multi-índice). Las herramientas de síntesis estructural podrían interpretarla como una memoria clásica, en la cual el bus de direcciones esta dividido en tantas partes como dimensiones posee la memoria. La concatenación de los valores asociados a cada dimensión proporciona la palabra de la memoria a la cual se desea acceder.

Con objeto de modelar la unidad de control se ha introducido un módulo denominado controlador. PSAL2 no introduce restricciones en cuanto al número de unidades de control. Las acciones que se realicen en el sistema son asociadas a controladores, y no existe ningún limite en cuanto a su número.

A cada módulo se le asocian diversas propiedades tales como ser puerto de acceso de memoria o al sistema, no estar implementado en el chip, estar contenido en un macro-bloque, etc. Se señalan igualmente, las entradas y salidas que posee, asi como la anchura en bits (en el caso de registros, terminales, etc) y en palabras (en el caso de memorias) del elemento.
Al igual que ocurría en el leguaje de PSAL2, las entradas y salidas de los módulos son asociadas a puertos, los cuales se almacenan en la lista de puertos, cuya descripción será realizada mas adelante.

En la base de datos se señala igualmente las operaciones que se realizan en cada módulo. Para ello a cada elemento se le asocia una lista con las diferentes operaciones que en él se ejecutan, señalando además el número de bits de la operación asi como el número de veces que dicha operación se utiliza.

La base de datos contiene información específica de ciertos tipos de elementos. Asi, en el caso de las ROMs, se señala las constantes que estan localizadas en cada palabra y en los macro-bloques se indican los bloques que en él estan contenidos.

Además de la lista de módulos, existe una lista de puertos,es decir, de conexiones de los módulos. Básicamente, en cada elemento se indica el número de bits del puerto asi como el módulo al cual esta asociado y su tipo (entrada, salida, etc).

La lista de "wires" contiene todas las interconexiones que se realizan en el sistema. Cada elemento tiene asociados los dos puertos que conecta, asi como la anchura en bits de la interconexión.

En cuanto a las descripción del comportamiento, esta se encuentra formada por 4 listas:

1º) Funciones o subrutinas.
2º) Variables.
3º) "Links".
4º) Sentencias.

La descripción del comportamiento es una representación jerárquica del sistema a diseñar, en la cual se pueden diferenciar varias funciones. Estas juegan en la descripción un papel similar a las subrutinas en un lenguaje de programación de alto nivel, aunque a diferencia de éstas, no admiten parámetros. En los elementos de la lista de funciones se señala el nombre de cada una, la función en la cual esta definida y cuales son las instrucciones que agrupa.

Los elementos encargados de almacenar los datos en la descripción del comportamiento se encuentran agrupados en la lista de variables. En ella se señala en nombre, tipo y rango de biys de cada variable. En la implementación actual han sido definidos 10 tipos de variables:

1º) Dato.
2º) Constante.
3º) Indice.
4º) Memoria muti-dimensión.
5º) Temporal.
6º) Terminal del sistema.
7º) Terminal de datos de memoria.
8º) Terminal de direcciones.
9º) Conexión de caja negra.
10º) Memoria (en el sentido hardware).

Un índice es una variable utilizada únicamente como contador de un lazo. Dependiendo del proceso de síntesis, será implementado en el data path o en la unidad de control. Una variable temporal es un elemento que el sistema utiliza con el fin de establecer dependencias entre operaciones sin que ello conlleve la existencia de un elemento hardware que lo implemente. Esta primitiva es utilizada, por ejemplo, en algunos tipos de cajas negras, y tiene por objeto establecer una dependencia entre la operación que inicia la ejecución de dicho elemento y aquellas que utilizan su resultado, con objeto de evitar que el sistema trate de utilizar el resultado de la caja negra, antes de que este se haya producido.

Por otro lado, se utilizan dos primitivas para modelar memorias. La razón es simple: existen dos tipos de matrices a modelar. El primer tipo agrupa a las memorias tradicionales, en las cuales es preciso tener en cuenta las variables que se utilizan para acceder a los datos o direcciones de dicho elemento. Son matrices unidimensionales. En el segundo grupo se

encuadran los arrays con multiples dimensiones, similares a las matrices en los lenguajes software de alto nivel. Dichos elementos no tienen variables asignadas a cada una de sus dimensiones y permiten a PSAL2 utilizar algorítmos de síntesis especializados en un tipo de arquitectura (por ejemplo, algoritmos de síntesis de arrays sistólicos). En la variable se indica además el módulo hardware en el cual estan localizados. Otros datos que se incluyen son los relativos a los segmentos que se definen en la variable (concepto heredado de PSAL1).

Otra lista de la representación del comportamiento almacena los "links". Como ya se ha comentado, este elemento modela las transferencias de datos en la descripción algorítmica. En un "link" se señala cual es la variable origen y cual la operación destino, asi como las interconexiones mediante las cuales dicha transferencia es implementada.

El último componente de la descripción del comportamiento es la lista de sentencias. En cada elemento de ella, se señala, en primer lugar, el tipo de operación que se realiza, asi como el número de entradas y salidas que posee. En PSAL2 no existe la limitación que existía en PSAL1 en cuanto al número de entradas y salidas, lo cual permite modelar nuevos tipos de operaciones.

En la implementación actual, se pueden realizar las mismas operaciones que se realizaban en PSAL1, con la excepción de las construcciones de control. Construcciones como LEAVE, RESUME y TERMINATE, típicas de ISPS y de PSAL1, no existen en PSAL2. En este sistema la ejecución de un subrutina puede detenerse mediante el comando RETURN, equivalente a su homologo en los lenguajes software de alto nivel (C, Pascal,etc.). No es posible, por lo tanto, detener la ejecución de una subrutina y continuar la ejecución en una función diferente de la que la llamó (como ocurría en PSAL1).

Además se han introducido diversos comandos con el fin de modelar los lazos de tipo "for" y "while". Hay que destacar que existen elementos en la base de datos que identifican la condición del lazo, o el lugar en el cual se realiza la incrementación (en un lazo "for"), lo cual facilita de forma notable el análisis de dichas estructuras.

En una descripción del comportamiento existen dos tipos de dependencias:

1º) Dependencias de datos.
2º) Dependencias de control.

Ambos son modeladas en los elementos de la lista de sentencias. Es este uno de los aspectos en el cual más difieren PSAL1 y PSAL2. En el primero existían dos tipos de elementos encargados de modelar ambos tipos de dependencias. La experiencia demostró,

que aunque dicha filosofía en ciertos casos simplificaba el problema, en general obligaba a realizar la misma tarea dos veces, sobre elementos distintos.
Al tener elementos iguales, el sistema gana en homogeneidad y facilidad de manejo. Además se ha establecido una via, dentro de la lista de sentencias, que permite tratar los elementos de PSAL2 como si fuesen de PSAL1, lo que permite extender algunos algoritmos del segundo al primero.

Al modelar en cada elemento los dos tipos de dependencias, las sentencias en PSAL2 se clasifican en cuatro tipos, dependiendo de las dependencias que modelen:

1º) Sentencias cuyos datos y resultados son variables (dependencia de datos a la entrada y la salida, como en el caso de una suma).

2º) Sentencias con dependencia de entrada de datos, y con dependencia de salida de control (por ejemplo una sentencia de tipo "if").

3º) Sentencias con dependencia de control en las entradas y con datos por resultado (eg: el inicio de una subrutina).

4º) Sentencias con dependencias de control en las entradas y en las salidas (eg: la primera sentencia de un lazo).

Con el fin de permitir un mejor acceso a los datos contenidos en la lista de sentencias, existen 3 vias de navegación en la base de datos:

1º) La ruta estandard, que permite recorrer todos los elementos, sin diferenciarlos.

2º) Via de navegación de control, la cual visita todos aquellos nudos que tienen asociadas dependencias de control. Esta ruta permite ver la base de datos de PSAL2 como si estuviese formada por un conjunto de bloques básicos.

3º) Ruta de ejecución. Cada nudo con dependencia de control asociado posee una serie de nudos sucesores, lo que permite conocer el orden en el cual se ejecutan las instrucciones. Esta via permite recorrer la base de datos, de la forma en que lo haría un simulador.

Cada sentencia, además del tipo de operación, posee asignados elementos tales como:

- Variables que utiliza como datos.
- Links que modelan las transferencias.
- Módulo en el cual esta localizada.
- Unidad de control que ejecuta dicha instrucción.
- En el caso de sentencias con control asociado, los vectores de dependencia de datos, similares a los utilizados en PSAL1.

Los algoritmos de síntesis deben por lo tanto asignar cada operación a un estado (y por extensión a una unidad de control) y a un módulo hardware, cada variable a un módulo de tipo registro o terminal, y cada "link" a una interconexión.
Con objeto de mostrar la flexibilidad y versatilidad de la base de datos se ha introducido una descripción algorítmica en la base de datos. En ella se calcula la expresión:

$$r=\sum_{i=0}^{255} (x[i] - y[i])$$

En vez de mostrar el contenido de la base de datos, mostraremos la descripción a nivel de transferencias de registros generada por el sistema (realizada en el lenguaje CVS_BK) a partir de los datos contenidos en el formato intermedio. En todos los casos, la descripción algorítmica es la misma, no ha cambiado. Lo único que se ha modificado han sido las relaciones entre dicha representación y la descripción de la arquitectura, asi como la asignación de estados.

El primer ejemplo muestra lo que el sistema habría obtenido, si hubiese estado regido por las mismas restricciones que PSAL1 (no se permite encadenamiento y el acceso a memoria se realiza atraves de un bus de direcciones):

```
module ejemplo1 (
      in    $psal_reset_terminal
      );
      clock $psal_clock
            default;
      ram
            mem_x[255..0;15..0];
            mem_y[255..0;15..0];
      node
            address_x[7..0];
            address_y[7..0];
      register
            $psal_reset;
            r[31..0];
            t3[31..0];
            t2[15..0];
            t1[15..0];
            j[7..0];
```

```
    constant
        $psal_const_11[7..0].=1;
        $psal_const_15[7..0].=1;
        $psal_const_12[31..0].=0;
    state
        PSAL_ejemplo_state_list = (   PSAL_ejemplo_state_1,  PSAL_ejemplo_state_2,
                                      PSAL_ejemplo_state_3,  PSAL_ejemplo_state_4   );
  begin
whenever ( $psal_reset or $psal_reset_terminal ) goto PSAL_ejemplo_state_1;
    sequence
    PSAL_ejemplo_state_1:
        if ($psal_reset and $psal_reset_terminal ) then $psal_reset:='0 endif;
        r:=$psal_const_12;
        j:=$psal_const_11;   goto PSAL_ejemplo_state_2
    PSAL_ejemplo_state_2:
        if  not(equ( j))  then
        address_x.= j;
        address_y.= j;
        t1:=mem_x[ address_x] { t1 <- dat_bus_x } ;
        t2:=mem_y[ address_y] { t2 <- dat_bus_y } ;
          goto PSAL_ejemplo_state_3
        else    $psal_reset:='1;    goto PSAL_ejemplo_state_1    endif
    PSAL_ejemplo_state_3:
        t3:= t1[15]. t1[15]. t1[15]. t1[15]. t1[15]. t1[15]. t1[15].
 t1[15]. t1[15]. t1[15]. t1[15]. t1[15]. t1[15]. t1[15]. t1[15]. t1[15]. t1 -  t2[15]. t2[15]. t2[15]. t2[15].
t2[15]. t2[15]. t2[15]. t2[15]. t2[15]. t2[15]. t2[15]. t2[15]. t2[15]. t2[15]. t2[15]. t2[15]. t2[15]. t2;
          goto PSAL_ejemplo_state_4
    PSAL_ejemplo_state_4:
        r:= t3 +  r;
        j:= j + $psal_const_15;
        goto PSAL_ejemplo_state_2
end.
```

En el lenguaje CVS_BK (y en la mayoría de los lenguajes de descripción de harware a nivel RT) es preciso que las dos partes de una asignación tengan el mismo número de bits. Esta regla tambien se aplica a los operandos de expresiones aritmético/lógicas. PSAL2 es muy flexible en este aspecto ya que algunos lenguajes a nivel algorítmico, como por ejemplo ISPS, no presentan esta restricción. Por esta razón, a la hora de representar el sistema a nivel RT, el sistema reajusta de forma automática el tamaño de los operandos y del resultado. En el ejemplo, y dado que las operaciones se realizan en complemento-2, este ajuste implica una extensión del signo de los operandos mediente una concatenación (en CVS_BK esta operación se representa como '.').

En próximo ejemplo, se ha supuesto que el acceso a memoria se produce atraves de un registro de direcciones. La base de datos de PSAL2 es muy flexible en este aspecto: cada memoria podria tener el tipo de acceso que se quisiera. Además se permite encadenar operaciones, por lo que ciertas variables (t1, t2 y t3) son localizadas en terminales en vez de registros.

```
module ejemplo1 (  in  $psal_reset_terminal      );
     clock $psal_clock
           default;
     ram
           mem_x[255..0;15..0];
           mem_y[255..0;15..0];
     node
           t1[15..0];
           t3[31..0];
           t2[15..0];
     register
           $psal_reset;
           r[31..0];
           j[7..0];
          address_x[7..0];
          address_y[7..0];
     constant
           $psal_const_11[7..0].=1;
           $psal_const_15[7..0].=1;
           $psal_const_12[31..0].=0;
     state
        PSAL_ejemplo_state_list = ( PSAL_ejemplo_state_1, PSAL_ejemplo_state_2,
                                    PSAL_ejemplo_state_3  );
     begin
     whenever ( $psal_reset or $psal_reset_terminal ) goto PSAL_ejemplo_state_1;
      sequence
      PSAL_ejemplo_state_1:
           if ($psal_reset and $psal_reset_terminal ) then $psal_reset:='0 endif;
           r:=$psal_const_12;
           j:=$psal_const_11;   goto PSAL_ejemplo_state_2
      PSAL_ejemplo_state_2:
           if  not(equ( j))  then
           address_x:= j;
           address_y:= j;  goto PSAL_ejemplo_state_3
            else
           $psal_reset:='1;   goto PSAL_ejemplo_state_1
           endif
      PSAL_ejemplo_state_3:
           t1.=mem_x[ address_x] { t1 <- dat_bus_x } ;
           t2.=mem_y[ address_y] { t2 <- dat_bus_y } ;
            t3.= t1[15]. t1[15]. t1[15]. t1[15]. t1[15]. t1[15]. t1[15]. t1[15]
. t1[15]. t1[15]. t1[15]. t1[15]. t1[15]. t1[15]. t1[15]. t1[15]. t1 -  t2[15]. t2[15]. t2[15]. t2[15]. t2[15].
t2[15]. t2[15]. t2[15]. t2[15]. t2[15]. t2[15]. t2[15]. t2[15]. t2[15]. t2[15]. t2[15]. t2;
           r:= t3 +  r;
           j:= j + $psal_const_15;
           goto PSAL_ejemplo_state_2
end.
```

Por último se muestra una descripción en la cual varias variables (t1,t2 y t3) han sido localizadas en una memoria. Lo mas interesante de esta descripción es que solo se ha modificado la relación entre la descripción algorítmica y la estructural, sin modificar la estructura de la primera. En la representeción CVS_BK no se señala la OU en la cual las operaciones son realizadas, debido a que en dicho lenguaje todas las operaciones de PSAL2 son primitivas y no hace falta indicar en donde se realizan.

```
module ejemplo1 (    in    $psal_reset_terminal    );
      clock $psal_clock   default;
      ram
            mem_x[255..0;15..0];
            register_file[1..0;31..0];
            mem_y[255..0;15..0];
      node
            address_x[7..0];
            address_y[7..0];
      register
            $psal_reset;
            r[31..0];
            j[7..0];
      constant
            $psal_const_11[7..0].=1;
            $psal_const_15[7..0].=1;
            $psal_const_12[31..0].=0;
      state
            PSAL_ejemplo_state_list = (
                  PSAL_ejemplo_state_1,
                  PSAL_ejemplo_state_2,
                  PSAL_ejemplo_state_3,
                  PSAL_ejemplo_state_4,
                  PSAL_ejemplo_state_5   );
      begin
      whenever ( $psal_reset or $psal_reset_terminal ) goto PSAL_ejemplo_state_1;
      sequence
      PSAL_ejemplo_state_1:
            if ($psal_reset and $psal_reset_terminal ) then $psal_reset:='0 endif;
            r:=$psal_const_12;
            j:=$psal_const_11;
            goto PSAL_ejemplo_state_2
      PSAL_ejemplo_state_2:
            if  not(equ( j))   then
            address_x.= j;
            address_y.= j;
           register_file[0]:=register_file[0;31..16]. (mem_x[ address_x] {register_file <- dat_bus_x});
             goto PSAL_ejemplo_state_3
            else
            $psal_reset:='1; goto PSAL_ejemplo_state_1;    endif
      PSAL_ejemplo_state_3:
           register_file[0]:=(mem_y[address_y]{register_file <- dat_bus_y}). register_file[0;15..0] ;
             goto PSAL_ejemplo_state_4
      PSAL_ejemplo_state_4:
           register_file[1]:= register_file[ 0 ;15]. register_file[0;15]. register_file[0;15]. register_file[ 0 ;15].
register_file[ 0 ;15]. register_file[ 0;15]. register_file[ 0 ;15]. register_file[ 0 ;15]. register_file[ 0 ;15].
register_file[ 0 ;15]. register_file[ 0 ;15]. register_file[ 0 ;15]. register_file[ 0 ;15].register_file[ 0 ;15].
register_file[ 0 ;15]. register_file[ 0 ;15]. register_file[0 ;15..0]  -  register_file[ 0 ;31]. register_file[ 0;31].
register_file[ 0 ;31]. register_file[0;31]. register_file[ 0 ;31]. register_file[ 0 ;31]. register_file[ 0 ;31].
register_file[ 0 ;31]. register_file[ 0 ;31]. register_file[ 0 ;31]. register_file[ 0 ;31]. register_file[ 0 ;31].
register_file[ 0 ;31]. register_file[ 0;31..16];
             goto PSAL_ejemplo_state_5
      PSAL_ejemplo_state_5:
            r:= register_file[ 1 ;31..0] +  r;
            j:= j + $psal_const_15;
            goto PSAL_ejemplo_state_2
end.
```

Con el fin de poder manejar la base de datos descrita anteriormente, fue creada una librería formada por 175 funciones. Esta librería permite realizar tareas de bajo nivel sobre la base de datos, tales como:

- Crear un elemento.
- Inicializarlo.
- Borrarle.
- Redimensionarle.
- Insertarle en la lista (antes o después de un elemento).
- Extraerle de la lista.
- Copiarle.
- Buscar un elemento.
- Mover un elemento.
- Intercambiar dos elementos.

Estas operaciones no solo se realizan sobre elementos individuales, tambien se realizan sobre arrays de elementos. Además, dependiendo de ciertos parámetros de la librería, la gestión de la memoria (creación o destrucción de elementos) es realizada por estas funciones, lo cual reduce las peticiones de memoria al sistema operativo, con la consiguiente ventaja en cuanto a tiempo de ejecución y optimización de la memoria utilizada.

El nombre de las funciones fue elegido de forma que defina facilmente su función. Dicho nombre esta formado por 3 elementos:

- Prefijo indicativo de que se trata de una función de libreria. El prefijo escogido fue : "s_".
- Los primeros caracteres indican el objetivo de la función. Por ejemplo, los caracteres "mk", indican que es una función que crea elementos.
- Los últimos caracteres señalan el objeto sobre el que actua. Asi , "sen" señala que el objeto es un elemento de tipo sentencia.

Por lo tanto, la funcion "s_mksen", será la encargada de crear elementos del tipo sentencia.

Además de las funciones de manejo de la base de datos, se incluyeron en la libreria las funciones de manejo de memoria virtual (calloc, realloc, malloc y free). La razón es consecuencia de la evolución histórica de PSAL2. Este sistema fue inicialmente desarrollado en una DECStation, siendo posteriormente adaptado a un entorno SUN. Al cambiar de maquina, las funciones antes mencionadas presentaron un comportamiento diferente al previsto, por lo que se opto por definirlas en la libreria como CALLOC, REALLOC,

MALLOC y FREE. La correspondencia entre las nuevas funciones y las antiguas depende del sistema, pero el comportamineto de las nuevas, desde el punto de vista de PSAL2, es siempre el mismo.

V) Algoritmos utilizados en PSAL2.

El sistema utiliza algunos algoritmos que proporcionaron buenos resultados en PSAL1, como el algoritmo de análisis de variables. Esta técnica tuvo que sufrir importantes modificaciones para adaparse al nuevo formato y estructura del sistema. La filosofia, sin embargo es similar a la expuesta en el capítulo 3. Una diferencia importante entre PSAL1 y PSAL2 en lo referente a este algoritmo, es que en este último sistema puede realizarse el análisis de variables cuantas veces se desee y en cualquier instante del proceso de síntesis. Esta propiedad es de gran ayuda cuando se desea mantener la coherencia del análisis despues de optimizaciones que lleven aparejadas movimiento de operaciones fuera de un bloque básico.

De la misma forma, PSAL2 utiliza un algoritmo de sustitución "inline" de subrutinas similar al que emplea PSAL1.

Por otra parte, PSAL2 incorpora nuevos algoritmos de localización de operaciones y sintesis del "data-path", cuya descripción será realizada a continuación.

1º) Algoritmos de localización de operaciones en estados.

1.1) Introducción

PSAL2 incorpora, a diferencia de PSAL1, varios algoritmos de "scheduling", pudiendo el usuario seleccionar el que desee en cada caso. Los algoritmos implementados son:

1.- Localización de operaciones dirigido por fuerza
2.- "List scheduling"
3.- Localización global

Los dos primeros se encuentran plenamente operativos. El último se encuentra en una fase avanzada de implementación, por lo que ya podemos ofrecer algunos interesantes resultados.

Las tareas a realizar durante el proceso de asignación de estados tienen lugar, en PSAL2,

según la siguiente secuencia:

1.- Determinación de las dependencias
2.- Ejecución del algoritmo de síntesis
3.- Asignación de estados
4.- Determinación de operaciones encadenadas: definición de terminales
5.- Reanálisis del flujo de datos.

Dado que el sistema dispone de mas de un algoritmo de localización, se hizo necesario el desarrollo de un módulo que se encarge de comunicarse con el usuario y de, en función de sus preferencias, seleccionar el algoritmo y ejecutar la secuencia antes descrita. Esta tarea es realizada por el módulo "SCHEDULING".

En la primera tarea se determinan las dependencias entre sentencias a partir de la información resultante del análisis del flujo de datos. Este proceso es un prerequisito para poder realizar la asignación de operaciones, por lo que si el usuario aún no lo ha hecho, el sistema lo realiza. Otro prerequisito es que los puertos del sistema y de las memorias esten definidos.

A continuación se ejecuta el algoritmo de asignación. Los dos primeros algoritmos implementados ("forced directed" y "list scheduling"), trabajan sobre bloques básicos, por lo que "SCHEDULING", selecciona las operaciones que se encuentran entre un nudo con entrada de control y otro con control de salida y despues llama al algoritmo correspondiente. El último algoritmo implementado (localización global), realiza la asignación de operaciones incluso con la presencia de operaciones de control. Por esta razón trabaja con entidades en vez de con bloques básicas.

Una vez el algoritmo de "scheduling" a sido ejecutado, se produce la asignación de operaciones a estados determinada por aquel para, a continuación, realizar un análisis de las operaciones encadenadas. PSAL2 puede localizar en un mismo estado la operación que produce un dato y la operación que lo consume. En este caso, se dice que ambas operaciones están encadenadas. Este hecho obliga a que el dato definido y consumido en el mismo estado sea asignado a un terminal y, en el caso de que dicho dato sea usado posteriormente, debe definirse una nueva variable que modele el encadenamiento y esté asignada a un terminal, y una nueva operación, que carge el valor de la nueva variable en la antigua. Ello es debido a que en un ciclo de reloj, todas las acciones se ejecutan de forma concurrente y no secuencial como en la descripción inicial. Además debe sustituirse la variable antigua por la nueva en las operaciones encadenadas. El módulo encargado de esta tarea, MK-CHAIN, realiza un análisis del papel de cada variable en la descripción resultante de la asignación de estados, detectando variables con el comportamiento antes reseñado, y

generando terminales y nuevas operaciones donde sea preciso.

A continuación se mostrará con un ejemplo como funciona. Dada la siguiente asignación de estados:

```
estado i:    a := b and c;                    1
             h := a + m;                      2
                 ...
estado j:    f := a - f     3
```

Es posible observar, como la variable "a" definida en la operación 1 es usada en la operación 2, en el mismo estado, y en la operación 3, en distinto estado. Es necesario, por lo tanto, definir una nueva variable (tmp) que será asignada a un terminal y modelará el papel de "a" en el estado i. La descripción quedará:

```
estado i:    tmp := b and c;                  1
             a := tmp ;
             h := tmp + m;                    2
                 ...
estado j:    f := a - f                       3
```

El algoritmo implementado tiene también en cuenta la posible existencia en el estado de caminos condicionales. Por ejemplo:

```
Estado i:
    ...
  case h is
    when "00"
        acc: = aac + 1;
    when "01"
        acc: = acc and ir;
        acc: = acc or mask;
  end case;
    ...
```

Se transformará en:

```
Estado i:
    ...
  case h is
    when "00"
          acc: = acc + 1;
    when "01":
          imp: = acc and ir;
          acc: = imp or mask;
  end case;
    ...
```

Una vez ha sido realizado el análisis de operaciones encadenadas se repite el análisis del flujo de datos, ya que el proceso anterior, puede haber modificado alguna operación o introducido nuevas variables. Las técnicas de localización de operaciones, implementadas en el sistema PSAL2, permiten:

- Encadenamiento
- Multiciclo
- Pipeline

Por esta razón, el sistema dispone de unas tablas en las que se indica el tiempo que una operación necesita para ejecutarse (expresado en número de estados), y el tiempo que la unidad operacional que la implementa no puede ser utilizada para procesar nuevos datos (lo cual permite modelar pipeline). De esta forma, si queremos modelar las multiplicaciones mediante multiplicadores pipeline que necesiten tres estados para completar la multiplicación, y que durante el 1º estado no pueden recibir nuevos datos, sólo debemos señalar en la tabla que dicha operación tiene una "profundidad" de 3 estados y este "activa" 1 estado. Más adelante veremos como se tienen en cuenta estos datos en los algoritmos. Además en dichas tablas se indica si la operación debe de ser tenida en cuenta a la hora de contabilizar el número de unidades operacionales, así como el número máximo de operaciones de ese tipo localizables en un estado (se puede, por tanto, obligar al sistema a que como máximo localice una multiplicación en un estado, por ejemplo).

1.2) Modificaciones introducidas en los algoritmos de localización dirigidos por fuerza.

La ténica "force directed", explicada en el capítulo 1 ha sufrido varias modificaciones, con dos objetivos:

a) Permitir el encadenamiento, multicicle y pipeline.
b) Optimizar la asignación de estados a los nudos de control.

El primer objetivo afecta a las técnicas ASAP y ALAP, así como al algoritmo de calculo del grafo de distribución (DG). El segundo afecta principalmente a la técnica ASAP.

El algoritmo de localización dirigido por fuerza puede ser aplicado únicamente a secuencias de operaciones sin sentencias de control. Por ello, la asignación de estados a las operaciones de control es uno de los puntos débiles de esta metodología, pudiendo producir estados extra innecesarios. Al existir operaciones que precisan varios ciclos de reloj para terminar su ejecución se puden producir incluso violaciones de la dependencia de datos. Para resolverlo se definen los vectores de inercia de las variables. Se dice que una variable tiene una inercia "i", cuando se necesitan al menos "i" ciclos de reloj para que su valor haya sido actualizado. Estos vectores de inercia son asociados a todas las operaciones de control. El algoritmo aplica dentro de cada bloque básico la metodología expuesta en el capítulo 1, y los bloques básicos se unen utilizando la técnica ASAP y los vectores de inercia antes definidos. La metodología se puede definir de la siguiente forma:

1.- Para cada bloque básico:
  1.1.- Recorrer la lista de variables
    1.1.1.- Para cada antecedente del bloque básico:
      1.1.1.1.- Seleccionar el mayor valor de inercia asociado a la variable.
  1.2.- Aplicar algoritmo de localización
  1.3.- Determinar el vector de inercia del nudo fin del bloque básico.
2.- Fin del algoritmo de localización.

Un paso previo a la localización es la realización del análisis de dependencias el cual determina las relaciones que deben de verificarse en dicho proceso. Entre una operación A y otra B, se establece una dependencia si :

1º) Si en A se define el dato que se crea en B
2º) Si en A se usa un valor que es redefinido en B.

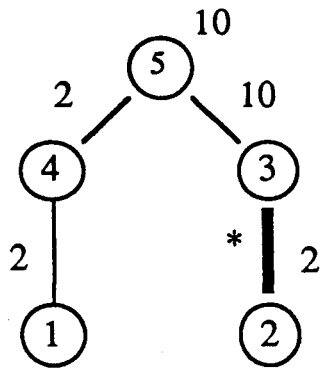
A las operaciones "A" que unifican las condiciones anteriores se las denomina "padres" de la operación "B". De la misma forma, se dice que "B" es un hijo de "A", ya que el último es padre del primero.

Existe otro tipo de dependencia, que aunque fué estudiado no ha sido implementado en la versión actual: la dependencia que modela el tiempo real de ejecución de una operación. En los algoritmos implementados se trabaja con dependencias medidas en ciclos de reloj. Sin embargo, en diseños físicos se trabaja con tiempos de retraso y periodos de reloj. Para un periodo de reloj dado es fácil determinar el número de ciclos asociados a una operación,

salvo en el caso de operaciones encadenadas. En el modelo propuesto no hay límite en tiempo al encadenamiento de operaciones, cosa que no sucede en hardware. La solución estudiada es muy sencilla. En primer lugar se determinan las dependencias tal y como se hace ahora, y más tarde se realiza un análisis de las dependencias mediante un algoritmo de búsqueda en grafos, cuyo nudo raiz es el nudo que actualmente se esta sometiendo a análisis y las ramas que de él salen modelan las dependencias con sus padres. El resto del grafo esta formado por sus padres y los padres de sus padres. Cada rama tiene asociado el tiempo necesario para ejecutar la operación. La profundidad del grafo puede limitarse introduciendo la premisa de no expander mas que aquellas ramas cuyo tiempo de ejecución es menor que un ciclo de reloj. A continuación y utilizando cualquier algoritmo de búsqueda en grafos (por ejemplo una búsqueda en profundidad) se determinan los caminos que precisan más de un periodo para ejecutarse. En ese caso se introducen dependencias adicionales, para modelar ese hecho. En la figura se puede ver un ejemplo. A partir de la descripción se genera el grafo de dependencias, a cuyos arcos se les asocia el tiempo que la instrucción anterior precisa para ejecutarse, tal y como puede observarse en la figura 4.5.

```
          ...
1         a := b and c ;          and = 2 nseg.
2         d := h and k ;          +   = 10 nseg.
3         f := d + b ;            -   = 10 nseg.
4         m := a or c ;           or  = 2 nseg.
5         t := f -m ;
               ...                periodo de reloj = 20 nseg.
```

Descripción Relación de tiempos



Grafo de dependencias temporales

figura 4.5

En el grafo de dependencias, el arco "*" no puede ser escogido, porque el coste en tiempo del camino, sería 10 + 10 + 2 = 22 > 20, mayor por lo tanto que la restricción (periodo de

reloj). Se precisa por lo tanto introducir una nueva restricción: 2 es padre de 5 con "tiempo de ejecución equivalente" de un ciclo de reloj. Con esta nueva restricción se impide que las operaciones 5,3 y 2 puedan formar parte de un mismo estado, dado que precisan mas de un ciclo de reloj para ejecutarse simultaneamente.

Antes de pasar a describir la estructura del algoritmo ASAP utilizado, debemos señalar, que en algoritmo de "list scheduling", se suelen producir condiciones como "no localizar la operación A antes del estado S". Al tener en cuenta estas situaciones, aparece en el algoritmo en una condición de "aplazamiento" y una profundidad de retraso (en el ejemplo S). El algoritmo ASAP utilizado en PSAL2, puede sintetizarse en los siguientes pasos:

1.- for i = 0 hasta que i > = número de operaciones
  1.1.- Si la operación i no está asignada
    1.1.1.- Si la operación i no tiene padres y no tiene condición de aplazamiento:
      1.1.1.1.- Determinar el valor máximo de la inercia de las variables que utiliza como datos.
      1.1.1.2.- Asignar como "profundidad ASAP" dicho máximo valor.
    1.1.2.- Si no se cumple la condición 1.1.1
      1.1.2.1.- Marcar el nudo como no pre-asignado
      1.1.2.2.- Determinar la inercia máxima de las variables utilizadas como datos.
      1.1.2.3.- Si dicha inercia no es 0 y no hay condición de aplazamiento.
      1.1.2.3.1.- Introducir condición de aplazamineto
      1.1.2.3.2.- Asignar la máxima inercia como profundidad de aplazamiento.
    1.1.3.- Determinar la máxima profundidad asignada a una operación pre-asignada.
  1.2.- Si el nudo no está preasignado.
    1.2.1.- Si el nudo tiene condición de aplazamiento:
      mínimo = profundidad de aplazamiento.
    en otro caso
      mínimo = 0
    1.2.2.- for j = 0 hasta que j < número de padres de i
      1.2.2.1.- Determinar la suma de la profundidad del padre j más el tiempo que dicha operación necesita para ejecutarse.
      1.2.2.2.- Si dicha profundidad es mayor que "mínimo", asignar a esta variable dicho valor.
    1.2.3.- Asignar como profundidad ASAP de la operación "i" el valor de la variable mínimo
    1.2.4.- Recalcular la profundidad máxima.
2.- Fin del algoritmo.

En este algoritmo se supone que las operaciones están ordenadas de forma que las definiciones se encuentren antes que los usos de una variable. Esta ordenación causa-efecto es la que siempre se mantiene en la base de datos del sistema PSAL2. Por otro lado, se puede observar, que en el algorítmo se han introducido modificaciones importantes respecto al "directed forced" original [PaKn89a] (puntos 1.1.1.1., 1.1.2.2. y 1.2.2.1) para poder utilizar las restricciones reseñadas anteriormente. Gracias a estos puntos las operaciones encadenadas, con pipeline o multicicle puden ser implementadas al tiempo que el flujo de control es optimizado con la utiliziación de vectores de dependencia.

Estas condiciones afectan además al algoritmo de determinación del grafo de distribución (en PSAL2, el módulo DG). En la secuencia a analizar pueden existir operaciones que necesiten "m" ciclos de reloj para proporcionar un resultado, y que durante "n" ciclos la unidad operacional en la cual ha sido localizada no se pueda utilizar para realizar otra operación. De esta forma, la operación producto de la figura 4.6 necesita 3 ciclos de reloj para generar un resultado, pero solo es "activa" durante los 2 primeros, por lo cual las 2 multiplicaciones podrían localizarse en la misma unidad operacional.

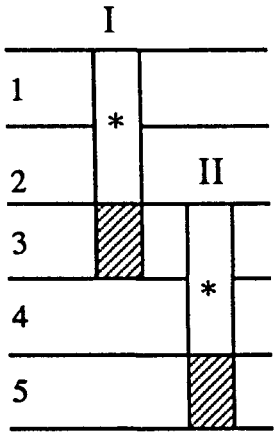


figura 4.6

En el algoritmo dirigido por fuerza, la probabilidad de una operación de ser localiza en una paso de control o "cstep" es:

prob = 1/ (ALAP depth - ASAP depth)

En el caso de operaciones que precisan más de un ciclo dicha probabilidad debe de multiplicarse por un factor que modela el solapamiento entre las etapas de dicha operación. De esta forma, si la operación I tuviese una profundidad ASAP de 1 y una ALAP de 5, los

factores a aplicar en cada cstep serían:

cstep	factor
1	1
2	2
3	2
4	2
5	2
6	1

Puede observarse:

1.- Que la profundidad ALAP se "extiende" en tantas unidades como ciclos de reloj este activa la operación menos 1
2.- El factor máximo es el número de ciclos que la operación está activa.

3.- En las proximidades de los límites, el factor tiene un comportamietno diferente, debido a que el solapamiento es distinto.

4.- El factor depende del número de ciclos que la operación esté activa (2 en el ejemplo), y es independiente del número de ciclos necesarios paa completarla (3 en el ejemplo).

El DG se calcula sumando las probabilidades asociadas a cada "cstep" de cada tipo de operación, teniendo encuenta los factores antes mencionados.

Por último vamos a comentar brevemente la función utilizada en el algoritmo "list scheduling" para resolver los conflictos en las asignaciones. En este tipo de algoritmos las operaciones se asignan a pasos de control de forma secuencial, paso de control a paso de control. Supongamos que en el paso "i", existen "m" operaciones candidatas a ser localizadas (es decir, que su profundidad ASAP sea "i"). De ellas sólo "n" tienen "fuerza mínima" en "i". Entonces, las "m - n" operaciones restantes son rechazadas, no asignadas, en el paso "i" (su profundidad de aplazamiento será "i + 1"), ya que su localización en "i" no es la óptima. De las "n" operaciones resultantes, solo "k" son localizables en unidades operacionales. Por lo tanto el resto, "n - k", son automáticamente localizadas en el paso "i", ya que sobre ellas no pesa ninguna restricción adicional. Las "k" operaciones restantes serán localizadas en "i", si "k" es menor que el número máximo de unidades operacionales especificado por el usuario. En otro caso, se irán localizando por orden de menor fuerza hasta que hayan sido localizadas "l" operaciones, siendo "l" el límite en el número de

unidades operacionales. El resto, "k - 1", son aplazadas al paso "i" al paso "i + 1". El algoritmo comprueba además que no se viole el número máximo de operaciones de un tipo en un estado. Si durante la asignación de operaciones la próxima candidata a ser localizada violase dicha restricción, dicha operación seria desplazada del paso "i" al "i + 1".

1.3) Algoritmo de localización global.

El punto débil de los algoritmos descritos en el punto anterior es que manejan muy bien secuencias de operaciones, en los cuales no existe ninguna operación de control. El algoritmo propuesto maneja secuencias que incluyen control, con la única excepción de lazos. En PSAL2 se estudio el problema de los lazos y se han propusto diversas alternativas para "plegarlos" (loop folding). En el apartado siguiente se mostrará una de las alternativas analizadas.

El algoritmo de localización global es un algoritmo de localización múltiple, es decir, la técnica puede decidir localizar una operación en diferentes caminos de control que se ejecutan de forma condicional. No se produce por lo tanto una asignación de una operación a un estado, sino que en principio, una operción puede ser asignada a múltiples estados.

El algoritmo estudiado es del tipo "list scheduling", utilizando la movilidad como criterio de asignación. La utilización de modelos más complejos, como fuerza, es compleja, debido al tipo de descripción con el que se trabaja. Esta representación esta formada por nudos en los cuales se realizan operaciones de datos, nudos que rompen el flujo de control, el camino principal, en caminos condicionales y, por último, nudos en los cuales los caminos condicionales se vuelven a unir en el original. Con este tipo de descripción dos aspectos son claves:

1.- Determinación de las dependencias.
2.- Representación de las profundidades de una operación.

Como ya hemos comentado, una de las ventajas de PSAL2 sobre su antecesor PSAL1 reside en poseer una estructura interna muy flexible, en la cual práctiamente no hay diferencias entre los nudos de datos y de control. Con ello, el concepto de "padre" definido en el apartado anterior puede ser extendido, definiéndose padres de datos (aquellas operaciones que generan el dato que se utilia en la operación que estamos analizando, en su mismo camino condicional) y padres de control, que marcan el punto de control en donde se especifica el camino en el cual se encuentra la operación. Veámoslo con un ejemplo:

```
a := b + 1 ;                    0
h: = h + 1 ;                    1
        ...
if a                            2
        then                    3
                ...
                h: = h + m;     4
        else                    5
                ...
end if ;                        6
a:= a + 1;                      7
m: = h + b;                     8
```

Conviene señalar que la operacion "if" (2) que divide el flujo de control en 2 caminos se ha dividido en 4 sentencias de control:

1.- Operación 2 que indica la condición de ejecución de 2 caminos condicionados.

2.- Operación 3 que indica el inicio de un camino condicional.

3.- Operación 5 que indica el inicio de otro camino condicional.

4.- La operación 6 que señala la unión de los caminos condicionales en el principal.

En este ejemplo se establecen las siguientes dependencias:

1.- La operación 3 tiene por padre de control a 2, ya que de él recibe el flujo de control.

2.- La operación 4 tiene por padres 1 (padre de datos) y 3 (padre de control).

3.- La operación 5 tiene por padre de control a 2, por la razón expuesta en la primera observación.

4.- La operación 7 tiene por padre de datos a la operación 0. Esta operación podría localizarse en los caminos condicionales del "if" sin ningún problema, ya que están dentro de su "marco temporal". Este sería un caso de asignación múltiple.

5.- La operación 8 depende de los datos de 4 y del control de 5. La razón es que la

operación 2 rompe el flujo de control en dos caminos, de forma que en uno de ellos, se redefine "h". La operación 8 dependerá por lo tanto de 4 (valor de h en el camino "then") y de 5, (valor en el camino "else"; se corresponde con el valor anterior de "h").

6.- La operación 6 tiene por padres a 2 y a los finales de los caminos condicionales "then" y "else". Estos finales, tienen por padres las opeaciones que definen un nuevo valor en el camino condicional que cierran.

Las dependencias asi definidas, definen todas las restricciones impuestas por la descripción del sistema a diseñar. Asi, por ejemplo, la asignación condicional que las variables tienen en una sentencia "if" son modeladas mediante las dependencias del punto 6 anteriormente reseñado.

Con este esquema, uno de los problemas a resolver es como modelar la profundidad de una operación. En los algoritmos del apartado anterior, se puede modelar dicha profundidad con un valor entero ya que solo existe un camino. En este algoritmo no solo hay que especificar la profundidad del nudo, sino también el camino en el que se encuentra. Se ha decidido expresar la profundidad en un camino condicional de forma relativa al punto de ruptura que lo produce. A dicha sentencia, se asocia la profundidad "0" en todos los caminos que de él parten. Además de este valor se especifican el punto de ruptura y el camino, dando lugar a una terna que permite expresar cualquier profundidad en el esquema propuesto.

Otro problema a resolver fue que profundidad asignar al nudo que marca el fin de los caminos condicionales introducidos en el punto de ruptura. La dificultad nace del hecho de que cada camino condicional tiene, en principio, profundidad máxima diferente. Se ha decidido definir como profundidad de la ruptura del control, la mínima profundidad máxima de los caminos condicionales que define. A efectos del nudo fin de ruptura, y de otros nudos (como en el ejemplo la operación 7). La sentencia condicional se comporta como una secuencia de profundidad igual a la profundidad de la ruptura, es decir, se modela el "if" como si se ejecutase el camino más rápido, el que genera menos profundidad. La utilización de profundidades relativas al punto de ruptura permite utilizar profundidades diferentes en los distintos caminos condicionales sin que el hecho de que todos los caminos terminen en la misma operación (el "end if" en el ejemplo) introduzca las contradicciones que se producirían al utilizar caminos absolutos. Dado que a su vez el punto de ruptura posee una terna ("ruptura - camino - profundidad") que señala su posición en el camino principal, es posible conocer la posición "absoluta" de cualquier par de operaciones, aunque no estén en el mismo camino condicional. Dicha posición absoluta determina si una operación se ejecuta antes, depues, o en un camino condicional paralelo a otra.

Con estas estructuras y dependencias, los algoritmos tradicionales sufren modificaciones significativas. Así, por ejemplo, la forma más sencilla de modelar la técnica ASAP tradicional podría ser:

1. for i = 0 hasta que i < = número de operaciones
   1.1.- mínimo = 0
   1.2.- for j = 0 hasta que j < = número de padres
      1.2.1.- Si la profundidad del padre j es mayor que mínimo
         1.2.1.1.- Asignar a mínimo la profundidad del padre j.
   1.3.- Asignar como profundidad de la operación i, mínimo.

La nueva técnica modifica este esquema introduciendo la posibilidad de asignación multiple de una operación. El esquema equivalente a anterior con la nueva filosofia podría ser:

1.- Desde que i = 0 hasta que i = número de operaciones
   1.1.- Desde que j = 0 hasta que j = número de padres
      1.1.1.- Desde que k = 0 hasta que k = número de elementos en la lista de asignaciones de la operación i.
         1.1.1.1.- Comparar la posición del padre j y del elemento k de la lista de profundidad:

            - si el padre tiene menor profundidad que el elemento, marcar al padre como no insertable.
            - si el padre tiene mayor profundidad que el elemento: eliminar dico elemento de la lista.
            - si padre y elemento están en caminos condicionados (esto es, nunca se ejecuta en el mismo camino). No hacer nada.

      1.1.2.- Si el padre j es insertable, entonces insertar el padre j en la cola de asignaciones de la lista de la operación i.

2.- fin del algoritmo.

Como se puede observar, en vez de hablar de "profundidad" se habla de "lista de profundidad", y en vez de estudiar el caso de "mayor profundidad", es necesario estudiar los 3 casos (mayor, menor y condicional). El algoritmo implementado es, en realidad, un poco más complicado, ya que la "profundidad del padre" también es una lista. En el ejemplo

se puede observar el resultado de la aplicación de la técnica ASAP es una descripción. Se ha supuesto que las operaciones "+" y "-" necesitan 1 paso de control para ejecutarse, mientras el resto se pueden ejecutar en menos de un ciclo de reloj, pudiéndose producir entonces encadenamientos entre ellos:

```
entity p is

port (
        a: in bit-vector (0 to 15);
        b: out bit-vector (0 to 15);
        );
    end;

architecture p1 of p is
    begin
    process
        variable c: bit-vector (0 to 15);
        variable d: bit-vector (0 to 15);
        variable e: bit-vector (0 to 15);
        variable f: bit-vector (0 to 15),
        begin
            c: = a + f
            b <= c + e;
            if f = e then
                d: = d + e
                f: = d and e;
            else
                d: = d - e;
            end if;
            c: = c + 1
            e: = f + d;
            d: = d + e
        end process;
        end;
```

El algoritmo anteriormente comentado, produce el siguiente resultado (cada conjunto indica las acciones que se realizan en cada estado):

```
S1:    c: = a + f;
       tmp: = f = e;
       if tmp then
           d: = d + e;
           siguiente estado: = S2;
       else
           d: = d - e;
           siguiente estado: = S3;
       end if
```

```
S2:   b = c + e;
      tmp 2 : = d and e;
      c: = c + 1;
      e = tmp 2 + d;
      f: = tmp 2;
      siguiente estado: = S4

S3:   b = c + e ;
      c: = c + 1;
      e: = f + d;
      siguiente estado : = S4

S4: = d: = d + e;
```

Es importante destacar que, en cualquier caso, solo se necesitan 3 ciclos de reloj para ejecutar la descripción (S1, S2 y S4 ó S1, S3 y S4). Se puede observar igualmente la aparición de terminales (tmp e tmp2) que modelan el encadenamiento de la operación de comparación y el "if", así como la definición y uso de la variable "f" en el estado S3. Se observa igualmente que seha producido una asignación multiple de las operaciones "c:=c+1" y "e:=f+d" a los estados S2 y S3.

El aspecto que tendría el resultado si la operación de comparación "igual a" ('=') precisase, al igual que '+' y '-' un estado para ejecutarse, aparece a continuación:

```
S1:   c: = a + f;
      tmp: = f = e;
      siguiente estado: = S2

S2 :  b = c+ e;
      c: = c + 1
      if tmp then
         d: = d + e;
         siguiente estado: = S3;
      else
         d: = d - e;
         siguiente estado := S4;

S3:   e: = f + d;
      siguiente estado S5;

S4:   tmp2: = d and e;
      e: = tmp 2 + d;
      f: = tmp 2;
      siguiente estado S5;

S5:   d: = d + e;
```

En este caso se obtiene el mismo número de estados que con un algoritmo tradicional

(independientemente de tiempo de ejecución de la operación "igual a" (=)). Existe una diferencia importante: en esta descripción todos los caminos tienen 4 estados, mientras que con un algoritmo tradicional hay caminos con 5 estados. Observar también cómo la operación "c: = c+ 1" es localizada antes del "if". Aunque originalmente esté depues. Esto es consecuencia del algoritmo, que trata las operaciones de control como operaciones normales.

1.4.) Estudio de la asignación de operaciones en lazos.

Los lazos suelen ser uno de las contrucciones en donde más tiempo de ejecución se emplea. Es por esta razón por lo que se han estudiado diversas técnicas para reducir el tiempo de ejecución en los lazos. El objetivo de estas técnicas es "plegar" el lazo, reduciendo tanto el número de iteraciones como la "profundidad" de cada iteración. Las técnicas de plegado tratan de utilizar valores producidos en otras iteraciones con objeto de obtener los objetivos propustos. Sobre las técnicas de desarrollado de lazos ("loop unrolling") expuestas en el capítulo 1, estas técnicas ("loop folding") tienen la ventaja de no aumentar el número de estados, al tiempo que su aplicación no esta restringida a lazos finitos.

El problema de extender el algoritmo expuesto en el apartado anterior al caso de lazos reside en el hecho de que en un lazo aparecen dependencias en las cuales la iteración en la cual se realizan las operaciones es diferente y por lo tanto se pueden presentar situaciones en las cuales un hijo sea a su vez padre de su padre (obviamente con una iteración de diferencia), como se ve en el ejemplo:

```
while ... loop
      a: = b+ 1;                    1
      b: = a + m;                   2
end loop;
```

En este ejemplo la operación 1 es padre de 2, y esta (en la iteración anterior) es padre de 1. En esta situación los algoritmos comentados anteriormente no son aplicables ya que la profundidad del padre de la operación 1 (operación 2 en la iteración anterior) no es conocida de antemano. La solución estudiada realiza una asignación de operaciones en lazos de forma jerárquica empezando por el lazo más interno. En cada lazo se realiza, en primer lugar una asignación de operaciones con uno de los algoritmos comentados anterioremnte. En esta asignación sólo se tienen en cuenta las dependencias con "profundidad de iteración 0" , es decir, las dependencias entre operaciones de la misma iteración. Si se desea introducir pipeline y multiciclo, hay que definir un vector de inercia en el comienzo del lazo.

Inicialmente la inercia de todas las variables es 0 pero despues de la asignación, habrá variables que tengan cierta inercia (debido a que al reiniciarse la ejecución del lazo, aún no han sido producidas por la operación que las genera). Dicha inercia será considerada en una nueva asignación. El proceso se repite hasta que todas las operaciones han sido asignadas.

El segundo paso del algoritmo es el plegado. Si el número de iteraciones que se realizan en el lazo en estudio es suficientemente alto, se puede demostrar que cada operación aparece una y sólo una vez en el lazo plegado. Suponiendo que el lazo es de este tipo, el algoritmo de plegamiento se pude ver como un algoritmo de "list schudeling" en el cual en cada paso se pliegan las operaciones de un estado. La mejor forma de ver este algoritmo es con un ejemplo. Dada la descripción de la figura:

```
while    loop

   1:    a: = b + 1
   2:    g: = h + 1
   3:    c: = a + d;
   4:    b: = a + 1;
   5:    d: = c + 1
   6:    e: = d + f;
   7:    f: = d - f;
end loop;
```

Se establecen las dependencias existes entre operaciones. En dichas dependencias se señala, no solo el padre, sino tambien la diferencia de iteraciones entre operaciones :

operación			iteración
1	depende de	4	1
2	depende de	-	-
3	depende de	1	0
		5	1
4	depende de	1	0
5	depende de	3	0
6	depende de	5	0
		7	1
7	depende de	5	0
		7	1

Un valor de iteración 1 significa la iteración anterior, 0 la actual. Aquellas operaciones que no tienen dependencias como 2, representan operaciones cuyo resultado no cambia en el lazo y por lo tanto pueden ser extraídas. Aquellas operaciones que no tienen hijos en el lazo (como la operación 6 de la que nadie depende), pueden extraerse del lazo porque su resultado no es utilizado en él. En este caso dicha operación será ejecutada despues del lazo, en el otro antes.

Supongamos que realizamos una preasignación utilizando la técnica ASAP. El resultado sería (considerando sólo dependencias con iteración 0):

estado	operación
1	1
2	3, 4
3	5
4	7

A continuación se utiliza una adaptación de la técnica "list scheduling", en el cual se consideran todas las dependencias, para realizar el "scheduling" final. Para enrollar el lazo se ha utilizado la técnica ASAP, aunque se podrían utilizar otras técnicas. Así:

1º Iteración. Análisis del estado 1:
La operación 1 depende de la 4 con 1 iteración menos. Dado que la operación 4 esta localizada en el estado 2, la operación 1_{ite1} sería localizable en los pasos de control 3 y 4. Si se realiza una asignación de tipo ASAP ("as soon as possible") será localizada en el estado 3. Luego la descripción resultante será :

estado	operación
2	$3_0, 4_0$
3	$5_0, 1_1$
4	7_0

2º Iteración. Análisis del paso 2:
La operación 3 depende de las operaciones :

- 1 con iteración 0, luego sería localizable en el paso 4, con el mismo índice de iteración que 1, es decir, 1.
- 5 con iteración 1, luego sería localizable en el paso 4, con índice de iteración 1.

Luego la operación 3 será localizada en el estado 4 con índice de iteración 1. En el caso

de existir diferentes puntos de asignación con diferente iteración se sigue el criterio de menor valor de iteración con mayor profundidad de asignación para localizar la operación. Repitiendo el proceso para la otra operación de este estado, la operación 4, se obtiene:

estado	operación
3	50, 11
4	70, 31, 41

3 º.- Iteración. Análisis del paso 3.

La operación 5 depende de la operación 3 con indice de iteración 0. Por la tanto la operación 5 podría ser localizada en el estado 5, que esta fuera del marco estudiado. Por esta razón no se enrolla, se deja en donde esta, ya que es equivalente localizarla en el primer estado de la iteración 0 que en el primero de la iteración siguiente. Lo mismo ocurre con la operación 4.

4º.- Iteración. Análisis del paso 4.

El análisis es similar al realizado en el caso anterior. El resultado del plegamiento, por lo tanto, es:

estado	operación
1	50, 11
2	70, 31, 41

Por lo tanto, la descripción resultante del plegamiento será:

```
g:=h+1;
a: = b + 1;          *
b: = a + 1;          *
while ... loop
   d: = c + 1;
   a: = b + 1;
   f: = d - f;
   d: = a + d;
   b: = a + 1;
end loop
d: = c + 1;          **
f: = d - f;          **
e: = d + f;
```

Se puede observar:

-Existen operaciones que se adelantan al lazo (marcadas con *) que modelan las operaciones que deben realizarse antes de la primera iteración debido al plegamiento.

- Igualmente quedan operaciones sin realizar, que deben ser extraídas del lazo (operaciones **)

- El número de iteraciones se ha reducido en una unidad.

- La consecuencia más importante de este proceso es que antes del plegamiento se necesitan 4 estados para realizar una iteración, ahora sólo 2. El tiempo de ejecución se ha reducido a la mitad.

3.2.- Algoritmos de localización del data-path.

En PSAL2 han sido implementados 3 tipos de algoritmos de localización del data path:

a) algoritmos de localización de registros, mediante técnicas de partición de grafos.

b) Algoritmo de asignación de unidades operacionales basado en la utilización de conjuntos de operaciones autoexcluyentes.

c) Algoritmo basado en una evaluación previa del sistema.

Los algoritmos 'a' y 'b' ya fueron explicados en el capítulo anterior, ya que su filosofía es muy similar a la utilizada en PSAL1. Ambos forman un sistema completo de síntesis del "data path", ya que la síntesis de interconexión no es imprescindible para que el programa ESP realice la síntesis estructural. Nos centraremos por lo tanto en el último algoritmo.

3.2.1.Algoritmo basado en probabilidad

3.2.1.1. Cálculo de la probabilidad.

Las técnicas implementadas en PSAL1, las hasta ahora comentadas en PSAL2 y muchas de las descritas en la bibliografía utilizan una estrategia predefinida para realizar la síntesis del "data-path". Sin embargo se piensa que es necesario disponer de una técnica que se base únicamente en la reducción del coste como estrategia de búsqueda. En la evaluación del coste del diseño, deben de utilizarse medidas que aproximen el coste real de la

implementación del sistema, no simples estimaciones como el numero de registros o el de unidades operacionales. Además se piensa que dicho coste debería poder tomar cualquier valor, con objeto de poder modelar cualquier situación. Por otra parte parece necesario que la metodología permita evaluar el coste del sistema durante el proceso de síntesis, sin necesidad de concluir el proceso de asignación. Todas estas consideraciones nos llevaron a desarrollar los métodos basados en probabilidades. El algoritmo parte de un modelado en forma de grafo del problema a resolver. En general los nudos representan objetos (variables, operaciones, transferencias, ...) y los arcos la posibilidad de que dos nudos compartan los mismos recursos hardware.

La metodología trata de asignar probabilidad a esta posibilidad de unión. Dado que el coste del objeto o nudo, y de las relaciones o arcos es conocido, la función de evaluación puede determinar el coste final probable para cada nudo, y el total como suma de dichos costes individuales

Se han estudiado dos técnicas basadas en probabilidad : asignación con recurso conocidos y asignación libre. En la primera se especifican inicialmente el número de elementos que deberá tener el sistema al final de la síntesis, tratando la metodología de asignar todos los nudos a dichos elementos. En la segunda la elección del número óptimo de objetos es dejada al sistema. La filosofía de ambas técnicas es muy similar, por lo que procedemos a comentar aspectos relacionados con la más general (la segunda), para después comentar como afrontar la primera.

La asignación de probabilidad a un arco está basada en la relación existente entre las ventajas y desventajas de seleccionar dicho arco. Esta relación debe de ser evaluada teniendo en cuenta los posibles arcos seleccionables. Este es el principal punto de discrepancia entre las dos técnicas basadas en probabilidad: en la primera solo son seleccionables los arcos que conectan el nudo en estudio con uno de los elementos del conjunto mínimo de recursos especificado por el usuario; en la segunda se consideran todos los arcos que tienen al nudo en estudio como vértice.

El cálculo de la probabilidad se divide en dos partes:

- Cálculo del coste "relativo" del arco.
- Determinación de la probabilidad.

Todo arco posee un "coste real" que señala el coste de la unión de los dos nudos que une el cual no tiene porque ser la suma de los costes de los dos nudos, ya que se pueden tener en cuenta similitudes, uso común de recursos hardware, interconexiones, etc. Precisamente, se

define la ganancia de un arco, como la diferencia de la suma de los costes reales de los nudos que une y su propio coste. Por otro lado, el "coste relativo" de un arco, tiene en cuenta además de su ganancia las ventajas o desventajas de escoger dicho arco.

Antes de comentar como se calcula, debemos definir los conceptos de arco ventajoso y arco desventajoso. Utilizaremos en dicha definición el concepto de vertices: si un arco A une los nudos n1y n2, entonces el par (n1,n2) se define como vertices de A.

Dado un arco A con vertices (n1,n2) se dice que un par de arcos (B,C) es un conjunto ventajoso con respecto a A, si B tiene por vértices (n1, m) y C tiene por vértices (n2, m). Como consecuencia, dos arcos ventajosos con respecto al arco A, son aquellos que permanecen seleccionables al seleccionar A.

Dado un arco A, con vértices (n1, n2), se dice que un arco B con vértices (n1, m) es desventajoso si no existe un arco C con vértices (n2, m). Alternativamente, un arco B se dice desventajoso al arco A, si no se puede seleccionar, al seleccionar A.

Con estas definiciones, el coste relativo se puede calcular con la siguiente expresión :

coste_relativo = ganancia - suma (arcos ventajosos) + suma (arcos desventajosos)

La suma de los arcos ventajosos se define como la ganancia media de los arcos que forman cada par de elementos por la mínima probabilidad de dichos arcos. La idea es que seleccionar el arco A, significa poder seleccionar el par (B, C) ventajoso a A. Por lo tanto se gana un arco (el B o el C), ya que en el caso de no seleccionar A, no se pueden seleccionar los otros dos conjuntamente. Esta es la justificacion de la ganancia promedio. La razón de multiplicar por la probabilidad mímina radica en el hecho de que esa ganancia se producirá unicamente en el caso de seleccionar B y C. Esta decisión no depende directamente de la de seleccionar A, sino que viene dada por la probabilidad de los arcos B y C, en concreto por la mímina probabilidad (peor caso).

La aportación al coste relativo de los arcos desventajosos se obtiene multiplicando su ganancia por su probablidad y modela la perdida que significa seleccionar el arco que se esta analizando. A esta aportación de los arcos desventajosos hay que sumarle una cantidad que modela la posibilidad de no escoger los 2 elementos de un par de arcos ventajosos. Cuando en uno de estos conjuntos, no se selecciona un arco, el otro se comporta como desventajoso. La aportación al coste desventajoso de esta posibilidad, se modela como el producto entre la ganancia de dicho arco, su probabilidad y un factor, función de la probabilidad de su pareja. Este factor vale 0 si la probabilidad de dicha pareja es mayor de un cierto valor umbral. Para valores de dicho parámetro, comprendidos entre 0 y el umbral,

el factor presenta una dependencia lineal, valiendo 1 en 0 y 0 en el umbral. Esto significa que cuando la probabilidad de un arco en un conjunto ventajoso es nula, su aportación a la suma de arcos ventajosos en nula (porque la probabilidad mínima es 0), mientras que el otro arco se comporta como desventajoso (ya que el factor, que acabamos de definir, vale 1).

Una vez se ha determinado el coste relativo de todos los arcos, se determina la probabilidad asociada a cada uno. Esta "probabilidad" se calcula como el cociente entre el coste relativo del arco y la suma de los costes relativos del conjunto de arcos formado por el que se esta estudiando y los arcos que su elección excluye. Los costes relativos de los arcos son multiplicados por las probabilidades actuales de dichos elementos antes de realizar el calculo de la próxima probabilidad, lo que obliga a la utilización de un método iterativo para determinar la probabilidad del arco. Este producto modela el hecho de que además de una ganancia, todo arco tiene una posibilidad de ser escogido que debe de tenerse en cuenta a la hora del calculo de la nueva probabilidad.

El conjunto de elementos excluidos por la selección de un arco, esta formado, no solo por los arcos desventajosos al selecionado sino tambien por aquellos arcos que pertenecen a conjuntos ventajosos en los cuales uno de los elementos no ha sido selecionado. Esta aportación se calcula de la misma forma que se calculaba en el caso del coste relativo, sin mas que sustituir la gananacia por el coste relativo.

La probabilidad asi calculada es multiplicada por un factor, que modela el hecho de que la elección de un arco desventajoso impide la selección del arco bajo análisis. El efecto de dicha selección es, por lo tanto, mucho mas importante que un simple incremento del coste desventajoso siendo modelado por un factor cuyo valor es 1 cuando la probabilidad de los arcos desventajosos es menor que un cierto valor umbral, y 0 cuando algún arco desventajoso ha sido seleccionado (teniendo, por lo tanto, probabilidad 1). La relación entre el valor máximo de la probabilidad de un arco desventajoso y el factor que estamos definiendo es lineal, para valores de probabilidad entre el umbral y 1. Este valor umbral es calculado en función del número de arcos mediante una función empírica. En el cálcula de este factor se utiliza, además de la probabilidad de los arcos desventajosos el aporte de los pares de arcos ventajosos en el cuales uno de los elementos no ha sido eleccionado. El calculo de la aportación de cada arco es similar al comentado anteriormente en casos similares: el producto de la probabilidad del arco por la probabilidad de que su pareja no sea seleccionada (es decir, uno menos su probabilidad), por un factor que vale 0 cuando la probabilidad de la pareja es mayor que un cierto valor umbral y 1 cuando dicha probabilidad es nula.

Como ya se ha comentado, la probabilidad es calculada mediante un método iterativo, que tomando como base la última probabilidad calculada, obtiene un nuevo valor de dicho parámetro. Este proceso se repite hasta que la nueva probabilidad discrepa de la antigua menos que un cierto valor (definido por el usuario) denominado tolerancia.

En este esquema es importante señalar como se calcula la probabilidad inicial. El sistema inicialmente asigna a cada arco una probabilidad que es función del número de nudos (en concreto la inversa de dicho número). Acto seguido se inicia un proceso iterativo similar a los descritos anteriormente, lo que permite asignar inicialmente una probabilidad a cada arco. Como ejemplo de asignación de probabilidad, se muestran las probabilidades iniciales asignadas por al algoritmo al ejemplo propuesto en [TsSi86]. En este ejemplo se ha asignado coste 1 a los nudos y arcos. El algoritmo tratará, por lo tanto, de minimizar el número de elementos.

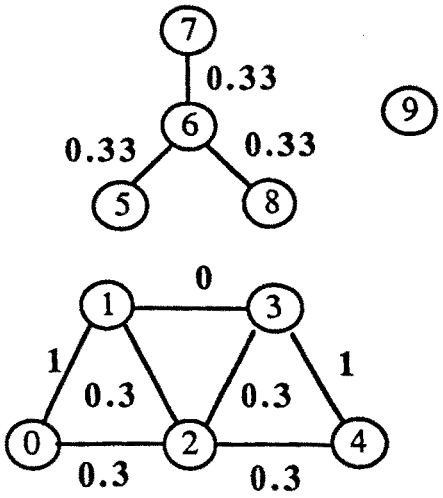


Figura 4.7

En la figura 4.7 se puede observar como ya desde la primera asignación, el algoritmo ha desechado el arco (1,3), al tiempo que ha seleccionado (asignado probabilidad 1) a los arcos (0,1) y (0,3). Estas elecciones son comunes a las 2 soluciones óptimas ({(0,1,2),(3,4)} y {(0,1),(2,3,4)}). Por otro lado, ha asignado probabilidad 0.33 a los arcos (5,6), (7,6) y (8,6), lo cual es lógico si se tiene encuenta que dichas decisiones son independientes y con el mismo coste. En cuanto al resto de los arcos, se les ha asignado una probabilidad de 0.3. La razón es que la selección de uno de dichos arcos {(0,2),(1,2),(3,2) y (4,2)}, significa reducir el coste del conjunto (0,1,2,3,4) en una unidad, pasando de coste 3 a coste 2, con una ganancia del 33%.

3.2.1.2 Estrategia de síntesis

El algoritmo presentado, utiliza una técnica inspirada en "simulated annealing" para realizar la síntesis del data-path. Esta metodologia modela el proceso de enfriamiento de un material. En él, existe una parámetro, la temperatura, que marca un límite al movimiento de las moléculas: por debajo de un determinado valor de dicho parámetro, ciertos movimientos no son posibles. Traducido a la técnica que se esta presentando, esto significa que algoritmo ira desechando de forma iterativa las peores elecciones, los arcos de menor probabilidad, hasta llegar a una solución del problema planteado.

Inicialmente toda decisión o arco posee una probabilidad de selección. El algoritmo selecciona aquella cuya probabilidad sea minima y le asigna probabilidad 0, es decir, la desecha definitivamente, ya que dicho valor no puede ser alterado por el algoritmo. A continuación se inicia un nuevo proceso de asignación de probabilidad y selección de un nuevo arco. El algoritmo de selección busca aquel arco que posea la probabilidad mínima, sin considerar los elementos que posean probabilidad 0 o 1. Acto seguido fuerza la probabilidad del arco a valor 0, continuando la ejecución con una nueva iteración. El proceso finaliza cuando el módulo de selección no es capaz de escoger un arco, porque todos tienen probabilidad 0 ó 1. Este algoritmo puede resumirse en los siguientes puntos:

1.- Calcular la probabilidad inicial.
2.- Mientras sea posible seleccionar un arco:
  2.1.- Asignar al arco seleccionado probabilidad 0.
  2.2.- Calcular la probabilidad
  2.3.- Evaluar el coste.
3.- Mostrar resultados.

Como puede observarse, cada vez que se realiza una selección se procede a evaluar el sistema resultante, utilizando una función que será mas tarde comentada. Dicha evaluación permite determinar la bondad de la decisión tomada.

Por otro lado, en la selección se utiliza la probabilidad en vez del coste del arco, porque aquel parámetro refeja mejor que este la relación entre las distintas elecciones, evitando que las diferencias de coste entre los distintos arcos afecten a la selección.

A continuación se puede observar este proceso, partiendo del ejemplo de la figura 4.7. En dicha figura los arcos de menor probabilidad son (0,2),(1,2),(3,2) y (4,2). El algoritmo selecciona uno de ellos, por ejemplo el arco (4,2), y le asigna probabilidad 0. A continuación recalcula la probabilidad. El resultado aparece en la figura 4.8.

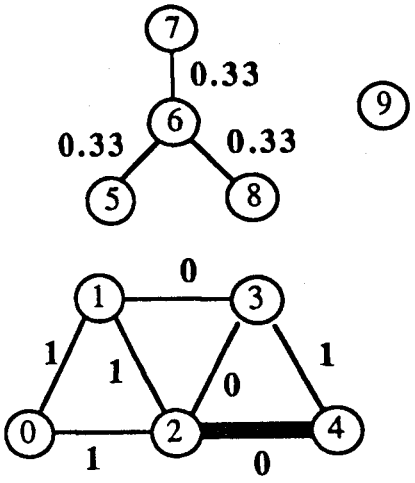


Figura 4.8

Al asignar probabilidad 0 al arco (2,4), el arco (2,3) toma probabilidad 0 y los arcos (1,2) y (0,2) probabilidad 1. Como se puede observar, con esta selección el algoritmo ha determinado el estado final del subgrafo (0,1,2,3,4). Esta es una de las caracteristicas mas importantes del nuevo algoritmo: realiza la síntesis tomando pocas decisiones, en este caso de 7 posibles arcos, ha necesitado desechar solo uno para conocer el estado final del subgrafo. Como aun quedan arcos seleccionables, en el subgrafo (5,6,7,8), el algoritmo selcciona un nuevo arco a desechar en dicho subgrafo, en concreto el arco (6,5). A continuación se reinicia el calculo de las probabilidades, obteniendose los valores mostrados en la figura 4.9.

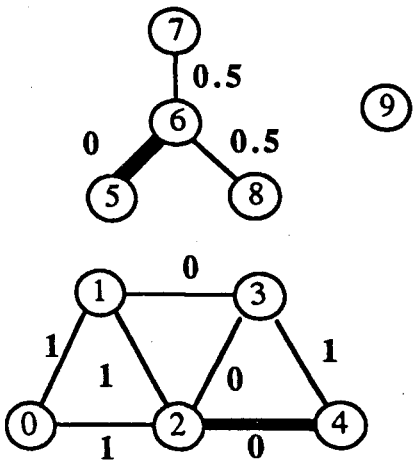


Figura 4.9

Por último se selecciona el arco (6,8), concluyendose el proceso de síntesis con el resultado

mostrado en la figura 4.10. La caracteristica antes mencionada (bajo número de decisiones), se mantiene para ejemplos de maor tamaño. Asi en el caso de la asignación de variables a registros, en ejemplo del PDP8, el algoritmo toma dos decisiones, estando el grafo formado por 20 nudos y 106 arcos.

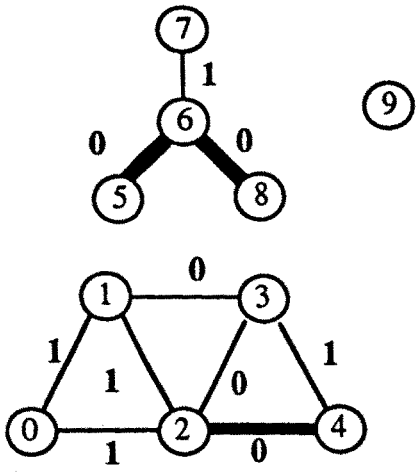


Figura 4.10

Otro aspecto a comentar es el número de iteraciones necesarias para obtener la probabilidad en cada iteración. Este número suele ser mas alto en la primera iteración, la que determina las probabilidades iniciales, que en el resto. A pesar de todo se mantiene en valores muy razonables durante todo el proceso. En el ejemplo presentado el número de iteraciones en cada paso fueron:

Paso	Iteraciones
1º (figura 4.7)	25
2º (figura 4.8)	3
3º (figura 4.9)	2
4º (figura 4.10)	2

3.2.1.3 Algoritmo de evaluación.

Una caracteristica importante que incorpora el algoritmo descrito es la posibilidad de proporcionar una estimación del coste del sistema resultante del proceso de síntesis en cualquier fase de la metodologia descrita.

Antes de pasar a describir el algoritmo, debe definirse la diferencia existente entre los conceptos de "coste de evaluación" y "coste de síntesis". Se denomina coste de síntesis al utilizado durante la asignación de probabilidades a los arcos del grafo. Se denominan costes

de evaluación a los utilizados durante dicha fase, que no tienen porque coincidir con los anteriores. Supongase, por ejemplo, que se desea realizar una síntesis de registros utilizando como criterio de coste el area ocupada por dicho elemento. Sin embargo durante la evaluación además del area total ocupada por los registros se desea conocer el número de registros que con dicho criterio el algoritmo produce. En este caso se utilizará como coste de síntesis el area de los registros, y como coste de evaluación aquel que como resultado produzca el número de registros. Por ejemplo, asignado coste 1 a los nudos (todo nudo modela un registro potencial) y el mismo valor a los arcos (cuando se unen dos nudos, por la selección del arco que los une, se produce un único elemento). Este coste de evaluación será utilizado con las probabilidades obtenidas con el coste de síntesis para determinar la medida deseada.

La función de evaluación actualmente implementada, calcula el coste total como la suma de los costes individales de cada nudo. De esta forma:

$$\text{Coste\_total} = \sum_{i=0}^{\text{número de nudos}} (\text{coste\_nudo}(i))$$

El coste de un nudo se calcula teniendo en cuenta dos aportaciones:

- El coste propio del nudo.
- Un valor que modela la relación de dicho nudo con el resto de los nudos del grafo.

El coste propio de un nudo se calcula a partir de la probabilidad de que dicho nudo no sea unido con otro, es decir, sea un nudo aislado. Esta probabilidad se calcula restando de 1 el valor máximo de la probabilidad de los arcos que tienen al nudo bajo análisis como vertice. El coste propio toma por lo tanto el valor:

Coste_propio = coste_nudo * prob_nudo_aislado

prob_nudo_aislado= 1 - max { probabilidad_arco_con_nudo }

El valor que modela la relación de un arco con el resto, es calculado a partir de los costes y probabilidades de los arcos que tienen a dicho nudo como vertice, siendo una especie de valor medio de dichas relaciones. En concreto, la función implementada calcula el valor asociado al arco como el producto de su coste por su probabilidad. Dichos valores son promediados con las probabilidades asociadas a cada arco. Por lo tanto, el valor asociado a un arco viene dado por la expresión:

valor_arco(i) = probabilidad_arco(i) * coste_arco (i)

Y el coste promedio de los arcos vendrá dado por la ecuación:

$$coste\_promedio = \left(\sum_{i} probabilidad\_arco(i) * valor\_arco(i) \right) / \left(\sum_{i} probabilidad\_arco(i) \right)$$

Este coste es compartido por el nudo que se esta analizando y aquellos nudos unidos con él con un arco con probabilidad apreciable. Por esta razón dicho coste es dividido entre el número de arcos con probabilidad mayor que 0 que tienen a dicho arco por vertice, mas uno. Por lo tanto el coste del nudo "i" toma el valor:

Coste_nudo(i) = Coste_propio(i) + (coste_promedio / (número_arcos + 1))

En el caso del ejemplo de la figura 4.8, esta función obtiene un coste de 6.0, teniendo en cuenta los coste y probabilidades de dicha figura. Como el coste final es 6, se puede observar la buena evaluación que el algoritmo realiza en dicho ejemplo. Los costes obtenidos en las distintas iteraciones fueron:

Inicial	6.09
1º iteración	6.25
2º iteración	6.16
3º iteración	6.00

Se puede observar un aumento del coste al pasar de la evaluación inicial, a la que se realiza tras la primera selección. Este aumento es debido a la simetria del problema, ya que en esa selección se ha desechado una posibilidad de igual coste y simétrica a la seleccionada, lo que conlleva un cierto efecto en el coste global del sistema. Esta tendencia no se mantiene en ejemplos de mayor tamaño como el ya comentado del PDP8. En él, el coste se va reduciendo según se realizan las iteraciones, aunque incluso la evaluación inicial del sistema proporciona una buena estimación de su coste, como se puede ver a continuación:

Inicial	7.96
1º iteración	7.86
2º iteración	7.00

CAPITULO 5

EJEMPLOS DE SINTESIS

I) Introducción

En el presente capítulo se pondrá a prueba la capacidad de síntesis de PSAL1 y PSAL2. Para ello se han escogido dos sistemas de pequeña/mediana complejidad (el computador PDP8 y el EDC, computador digital elemental propuesto en [Di79]) para los que se dispone de implementaciones que permiten, por lo tanto, el análisis de los resultados obtenidos.

El capítulo se encuentra dividido en tres apartados mas esta introducción. En los dos primeros se describe la sintesis de EDC y del PDP8. En el último se señalan algunas conclusiones extraidas de ambos ejemplos. Los apartados dedicados a la síntesis se inician con una breve descripción del ejemplo, para a continuarión mostrar los resultados obtenidos por PSAL1 y por último los obtenidos por PSAL2. En cada caso se muestra la descripción de entrada (en ISPS y VHDL respectivamente) y la arquitectura resultante (en DDL para PSAL1 y VHDL, CVS_BK y ESP para PSAL2). Además se señalan en un apartado las diferencias entre los resultados obtenidos por ambos sistemas.

II) Síntesis de un computador digital elemental

1) Introducción

Se ha escogido el EDC (computador digital elemental) descrito por Donald Dietmeyer en

su libro " Logic Design of Digital Systems"[Di79], en DDL. El autor utiliza la descripción para explicar diversos aspectos de la síntesis lógica a partir de una descripción a nivel de transferencias de registros, además de los principios básicos de funcionamiento de un computador. La razón principal por la cual ha sido elegida esta descripción es que dicho sistema se encuentra perfectamente descrito en el lenguaje de salida del sistema PSAL1 (DDL), por lo cual es muy fácil la comparación.

El EDC es una máquina de palabra única por instrucción y de dirección simple. La longitud de los datos y de las palabras es de 16 bits. En una instrucción se diferencian dos campos:

a) Los 6 primeros bits, contienen el código de la operación. A este campo se le denomina campo de código.

b) Los 10 restantes contienen la dirección . En la descripción DDL aparece como la variable ADR.

La capacidad de direccionamiento a memoria es, por lo tanto, de 2^{10} palabras (1024 palabras).
El primero de los campos, el campo de código, se encuentra a su vez dividido en dos subcampos:

a) OP, con 4 bits.
b) IX, con 2 bits.

Una variable (CAR, de 10 bits) almacena la dirección de la instrucción a ejecutar. Existen además 2 registros RUN y CLEAR que van a afectar a la secuencia y ejecución del computador: el estado del primer registro determina si el computador continua la ejecución o se detiene; el segundo indica cuando debe inicializarse la máquina.
Existen además una serie de buses utilizados para el intercambio de información entre el microprocesador y el exterior. Estos buses aparecen en la descripción como las variables INPUT y OUTPUT.

EDC posee además, un único acumulador , ACC, sobre el cual se realizan operaciones tales como sumas, restas, desplazamientos u operaciones lógicas. Existe además un registro OVF destinado a almacenar la ocurrencia de "overflow" en las operaciones. Este registro es peculiar, ya que no se carga por transferencias de datos de otros registros, sino que recoge una señal producida después de realizada una operación. Debe por tanto ser marcado como registro especial (nunca se carga, solo se utiliza). El conjunto de instrucciones del EDC aparece reflejado en la siguiente tabla:

Instrucción	Mnemónico	Código OP IX
Introduce un dato de INPUT en la memoria	INP	0000 00
Saca un dato de memoria a OUTPUT	OUT	0001 00
Introduce un bloque de datos de INPUT en memoria	BIN	0010 00
Pone el acumulador a " 0 "	ZRO	0011 00
Incrementa el acumulador	ADV	0011 01
Dobla el acumulador	DBL	0011 10
Cambia de signo el acumulador	CSN	0011 11
Carga el acumulador con un dato de memoria	LDA	0100 00
Suma el dato de memoria al acumulador	ADD	0101 11
Resta el dato de memoria del acumulador	SUB	0110 00
Carga el acumulador en una posición de memoria	STO	0111 00
Producto lógico entre el acumulador y un dato de memoria	AND	1000 00
Salto del programa a una posición de memoria	TRA	1001 00
Salto condicional si el acumulador es " 0 "	TRZ	1001 01
Salto condicional si el acumulador es negativo	TRN	1001 10
Salto condicional si se produce "overflow" en com-2	TRO	1001 11
Carga el acumulador con la dirección del dato	ENA	1010 00
Suma al acumulador la dirección del dato	INC	1010 01
Rotación aritmética a la derecha n* veces	SRA	1011 00
Rotación lógica a la derecha n* veces	SRL	1011 01
Rotación circular a la derecha n* veces	CIR	1011 10
Intercambia los dos bytes del acumulador entre sí	BEX	1011 11
No realiza ninguna operación	NOP	11xx xx

* n representa el valor decimal del campo de direcciones de la instrucción.

2) Síntesis del EDC utilizando el sistema PSAL1

2.1 Descripción de entrada en ISPS

Partiendo de las especificaciones antes reseñadas se ha realizado una descripción en ISPS del EDC. Esta descripción aparece a continuación:

```
!Descripcion en ISPS del computador digital elemental EDC de Dietmeyer
!
! Realizada en el Departamento de Electronica de la Universidad de Cantabria
!              Grupo de diseño y verificación de ASIC's
!
!                          febrero de 1991

EDC :=
      begin

**Processor.state**

      IR<1:16>,
            OP<0:3> := IR<1:4>,
            IX<1:2> := IR<5:6>,
            ADR<1:10> := IR<7:16>,
      CAR<1:10>,
      ACC<1:16>,
      OVF<>, RUN<>, CLEAR<>

**Memory**

      M[0:1023]<1:16>,

**Input.Output**

      INPUT<1:16>,
      OUTPUT<1:16>,
      IN<>, OUT<>

**Interpretation.Process**

      main interpret :=
            begin
            repeat
                  begin
                  decode CLEAR@RUN =>
                        begin
                        0 :=   restart interpret,
                        1 :=   begin
                               IR=M[CAR] next
                               CAR=CAR+1
                               end,
                        2:3 :=  begin
                               RUN=0; CLEAR=0; CAR=0; IR=0
                               end
                        end next
                  wait(RUN) next
                  execute()
                  end
            end,
```

```
execute :=
      begin
      decode OP =>
            begin
            0 :=   begin
                  wait(IN) next
                  M[ADR]=INPUT; IN=0
                  end,
            1 :=   begin
                  wait(not OUT) next
                  OUTPUT=M[ADR]; OUT=1
                  end,
            2 :=   begin
                  ACC=ADR next
                  ADR=1 next
                  repeat
                        begin
                        if ACC eql 0 => leave execute next
                        ACC=ACC-1 next
                        wait(IN) next
                        M[ADR]=INPUT; IN=0 next
                        ADR=ADR+1
                        end
                  end,
            3 :=   decode IX =>
                        begin
                        0 :=   ACC=0,
                        1 :=   ACC=ACC+1,
                        2 :=   ACC=ACC+ACC,
                        3 :=   ACC=-ACC
                        end,
             4 :=   ACC=M[ADR],
             5 :=   ACC=ACC+M[ADR],
             6 :=   ACC=ACC-M[ADR],
             7 :=   M[ADR]=ACC,
             8 :=   ACC=ACC and M[ADR],
             9 :=   decode IX =>
                        begin
                        0 :=   CAR=ADR,
                        1 :=   if ACC eql 0 => CAR=ADR,
                        2 :=   if ACC<1> => CAR=ADR,
                        3 :=   if OVF  => CAR=ADR
                        end,
            10 :=   decode IX =>
                        begin
                        0 :=   ACC<=ADR,
                        1 :=   ACC=ACC+ADR
                        end,
            11 :=   repeat
                        begin
                        if ADR eql 0 => leave execute next
                        ADR=ADR-1;
```

```
                    decode IX =>
                        begin
                        0 :=   ACC=ACC srd 1,
                        1 :=   ACC=ACC sr0 1,
                        2 :=   ACC=ACC srr 1,
                        3 :=   ACC=ACC srr 8
                        end
                    end
                end
            end
        end
```

Una vez tenemos el EDC descrito en ISPS, el siguiente paso es simularlo y verificar la corrección de la descripción. Hecho esto se puede proceder a la síntesis.

2.2) Generación de la base de datos

Lo primero que hace el sistema PSAL1 es convertir la descripción ISPS en una representación interna. En nuestro computador, el programa que genera dicha base de datos es creado mediante la macro gendb. Dicha macro tiene como argumento el nombre de la descripción a traducir. Al ejecutar dicha macro el sistema proporcionará la siguiente salida (se resaltan en negrita los comandos que escribe el usuario) :

$ gendb edc	Comando que permite crear el programa que
$ Set NoVerify	genera la base de datos
$ ISPS EDC	Parser de ISPS. Genera el fichero edc.gdb
ISPS V6(12)-MI	
$ gdbrtm edc	
GDB to RTM translator V10.5.5	Generador del código rtm para el simulador
GDB:I;ISPS V6(12)-MI;EDC.ISP;24-FEB-1989 11:40:42.94;	
GETGDB done, generating RTM code...	
Output will go to RTM.MAR	Fichero que contiene la descripción RTM
$ CONVERT EDC	Primer programa del sistema PSAL. Este programa transforma el fichero RTM.MAR, de forma que puede ser tratado por el Parser de PSAL
Nombre del fichero de salida:edc.mar	
Borro fichero RTM.MAR	

En EDC.MAR se encuentra la descripción del sistema en macroensamblador. Para poderla utilizar la descripción tiene que ser compilada y linkada con las librerías del sistema PSAL

```
$ MACRO/NOLIST EDC.

$ LINK/NOMAP/EXE:EDC EDC,usuarios:[espeso.grafo]gendb,
    usuarios:[espeso.grafo]gendb2/library

$ RUN edc
```

```
PSAL - GENERACION DE LA BASE DE DATOS

Desea guardar los mensajes en un fichero?:
```

El parser ofrece la posiblilidad de almacenar en un fichero información referente al propio proceso de traducción.

```
CREANDO EL NUEVO GRAFO

SUSTITUCION DE MARCAS POR VALORES

Estadistica : entidades 4
              nombres 17
              simbolos 25
              sentencia 94
```

Resultados de la traducción. Obsérvese el elevado número de bloques básicos detectados inicialmente por el Parser: 94.

```
PRIMER PASO DE LA OPTIMIZACION
Elimino entidad EXECUTE
Elimino entidad INTERPRET
Elimino entidad PRELUDE
```

Eliminación de entidades que solo son llamadas en una ocasión.

Esta entidad es introducida por el programa gdbrtm. No aparece en la descripción

```
Estadistica : entidades 1
              nombres 17
              simbolos 25
              sentencia 85
```

Observar la reducción del número de bloques básicos

SEGUNDO PASO DE LA OPTIMIZACION Compactación de bloques básicos

Estadistica : entidades 1
nombres 17
simbolos 25
sentencia 71

TERCER PASO DE LA OPTIMIZACION Se Asocian nudos

Han sido asociados 4 nudos

Estadistica : entidades 1
nombres 17
simbolos 25
sentencia 67

Eliminando sinonimos

Se denomina sinónimo a aquella variable que sirve para denotar ciertos bits de otra. Por ejemplo OP.

Han sido eliminados 3 sinonimos

SELECCION DE VARIABLES << RETURN >>

PSAL1 proporciona al usuario un mecanismo para indicar que variables deben de localizarse automáticamente en registros, o deben ser asociadas a terminales

OPCIONES
0 Salir del menu
1 Listar variables
2 Listar variables asignadas a conexiones externas.
3 Listar variables no agrupables
4 Seleccionar variable asignada a conexion
5 Seleccionar variable no agrupable

Opcion ? 4

Nombre de la variable a seleccionar
input Terminal

Seleccionada la variable INPUT

Opcion ? **4**
Nombre de la variable a seleccionar
output

Seleccionada la variable OUTPUT

Opcion ? **5**
Nombre de la variable a seleccionar
ovf Variable localizada automáticamente en un registro

Seleccionada la variable OVF

Opcion ? **2**
Lista de variables asignadas a conexiones externas
1 INPUT
2 OUTPUT

Opcion ? **3**
Lista de variables no agrupables
1 OVF OVF se declara como no agrupable porque en la declaración ISPS de entrada no se ha especificado su obtención, pero si su uso.
Opcion ? **0**

GUARDO GRAFO EN FICHERO:EDC.grf
El grafo de flujo es descrito en EDC.GRF

-- ESCRIBO LA DB EN EL FICHERO:EDC.DB

ESTADISTICAS

Numero de entidades	:	1
Numero de sentencias	:	67
Numero de simbolos	:	22
Numero de nombres	:	14
Numero de operaciones	:	60
Numero de st. otro	:	52
Numero de st. sucesor	:	41
Numero de letras	:	63
Numero de digitos es.	:	0

La generación de la base de datos ha concluido. Dos ficheros han sido generados:

- EDC.GRF. En el se almacena la base de datos en un formato que puede ser leido directamente. Es utilizado principalmente para verificar el funcionamiento de este módulo.

- EDC.DB. En este fichero se almacena la base de datos en formato binario. Gracias a su formato estos datos son rápidamente leidos por los demás módulos. Esta representación es utilizada como entrada por el resto del sistema.

2.4) Síntesis del EDC

A continuación se inicia la síntesis propiamente dicha. Hasta ahora la descripción había sufrido una serie de optimizaciones relacionadas con la estructura del grafo. El siguiente paso será la síntesis del data path y del control, así como la generación de la descripción en formato DDL. En nuestro sistema utilizamos un símbolo, "psal", para ejecutar el programa de síntesis. A continuación se muestran los resultados:

```
$ psal

Nombre de la  Base de Datos: edc

INICIO LA LECTURA DE LA BASE DE DATOS

INICIO LA RECOMPOSICION DEL GRAFO

    ESTADISTICAS
            Numero de entidades     :   1
            Numero de sentencias    :   67
            Numero de simbolos      :   22
            Numero de nombres       :   14
            Numero de operaciones   :   60
            Numero de st. otro      :   52
            Numero de st. sucesor   :   41
            Numero de letras        :   63
            Numero de digitos es.   :   0
            Numero de var. loc.     :   5
            Numero de var. glo.     :   9
```

Como se puede observar, el programa

no solo lee y recompone la base de datos, sino que también determina que variables son locales al bloque. En este caso 5.

INICIO EL ANALISIS DEL FLUJO DE DATOS

Comienza la ejecución de ANBLOCK

Analisis de bloques basicos

Numero de bytes necesarios en el analisis: 2

El análisis de variables se realiza con segmentos. El que para el análisis solo se necesiten 2 bytes implica que hay menos de 16 segmentos que analizar.

Eliminada operacion redundante en bloque 39

Eliminada una redundancia

Han sido eliminadas 1 operaciones

Inicio el analisis global

NUMERO DE ITERACIONES 18

Desea escribir los vectores de analisis de datos en un fichero ? (s/N) s

El sistema genera un fichero con el resultado del análisis. Más tarde veremos este fichero.

INICIO LA ESCRITURA DE LOS VECTORES DEL ANALISIS DEL FLUJO DE DATOS
ESCRITURA FINALIZADA

Defino operaciones encadenadas

Se definen operaciones encadenadas, asociadas con el control

La variable T35 no es localizable
La variable CHAIN2 no es localizable
La variable CHAIN3 no es localizable
La variable CHAIN4 no es localizable
La variable T50 no es localizable
Finalizada la busqueda de operaciones encadenadas

Se inicia el scheduling

INICIO LA COMPACTACION DE OPERACIONES

INICIO LA PRE-ASIGNACION OPERACION-ESTADO

0 Respetar el orden de las operaciones.
1 Asignacion sin limite de recursos hardware
2 Asignacion con recursos limitados

Opcion? **1** Asignación mas rápida

En la sentencia 6,se ha pasado de 2 a 1 estados
En la sentencia 16,se ha pasado de 2 a 1 estados
En la sentencia 22,se ha pasado de 2 a 1 estados

Numero de OUs: 1
Numero maximo :1

FIN DE LA COMPACTACION

A continuación se inicia la localización de variables

INICIO LA GENERACION DE LA TABLA DE EXCLUSIONES

Modulo GENTALIVE

FINALIZADA LA GENERCION DE LA TABLA DE EXCLUSIONES
Escribo en un fichero dicha tabla ?(s/N) **n**

INICIO LA SINTESIS DE VARIABLES
FIN DEL PROCESO DE SINTESIS
Resumen: 8 variables han sido agrupadas en 7 registros

Como minimo se necesitaban 7

El sistema necesita 11 registros

Inicio el preanalisis Identificación de operaciones no localizables en OU's
Fin del preanalisis

INICIO LA ASIGNACION OPERACION-ALU
OBTENIDA LA TABLA OPERACION-ALU Módulo ANALU
Se necesitan 1 alus

INICIO LA SINTESIS DE OPERACIONES

Describo la asignacion realizada en el fichero edc.ous
Informacion completada
FIN DE LA SINTESIS
INICIO LA COMPACTACION DE OPERACIONES
COMPACTO BLOQUES — Módulo compac
FIN DE LA COMPACTACION
INICIO LA ESCRITURA DE LA DESCRIPCION DDL — Módulo ESDDL
ESCRITURA FINALIZADA
INICIO LA ESCRITURA DE LA DESCRIPCION
FIN DE LA ESCRITURA DE LA DESCRIPCION

En este momento la síntesis ha finalizado. En el fichero edc.ddl se dispone de la descripción DDL resultante del proceso. Además del fichero edc.ddl, el sistema ha generado:

bit	bitmax	bitmin	nombre
1	0	0	RUN
2	0	0	CLEAR
3	9	0	IR
4	11	10	
5	15	12	
6	9	0	CAR
7	0	0	IN
8	15	0	INPUT
9	0	0	OUT
10	14	0	ACC
11	15	15	
12	0	0	OVF

TABLA DE DATOS

nud:1
in:1100011111110000
ou:1100011111110000
nud:2
in:1100011111110000
ou:1100011111110000
nud:3
in:1100011111110000
ou:1100011111110000
nud:4
in:1100011111110000
us:1100000000000000
de:0000000000000000
ou:1100011111110000

Figura 5.1

- EDC.DFA. Este fichero contiene los vectores del análisis del flujo de datos. En la figura 5.1 se puede observar las primeras líneas de este fichero. En primer lugar se señalan los segmentos asociados a cada variable, así como el bit asociado a cada segmento. Por ejemplo, el bit 4 esta asociado al segundo segmento de la variable IR. Este segmento esta formado por los bits 10 y 11 de IR. A continuación se encuentran los valores de los vectores del análisis de variables.

GRAFO DE EXCLUSIONES

Nudos:
0 RUN
1 CLEAR
2 IR
3 CAR
4 IN
5 OUT
6 ACC
7 T45

Arco			tipo	comun	excluidos
0	1	-	0	0	0
0	2	-	0	0	0
0	3	-	0	0	0
0	4	-	0	0	0
0	5	-	0	0	0
0	6	-	0	0	0
0	7	-	0	0	0
1	2	-	0	0	0
1	3	-	0	0	0
1	4	-	0	0	0
1	5	-	0	0	0
1	6	-	0	0	0
1	7	-	0	0	0
2	3	-	0	0	0
2	4	-	0	0	0
2	5	-	0	0	0
2	6	-	0	0	0
2	7	-	1	0	0
3	4	-	0	0	0
3	5	-	0	0	0
3	6	-	0	0	0
3	7	-	0	0	0
4	5	-	0	0	0
4	6	-	0	0	0
4	7	-	0	0	0
5	6	-	0	0	0
5	7	-	0	0	0
6	7	-	0	0	0

Figura 5.2

- EDC.TABLA. En esta tabla se recoge el grafo de dependencias calculado por PSAL1 antes de realizar la síntesis de variables. El grafo correspondiente al EDC aparece en la figura 5.2. En él primero se indica la asignación variable-nudo que ha realizado el programa, y a continuación se muestra el grafo. En la tabla, la columna tipo indica si existe un arco entre los nudos asociados a la linea. Como se observa, solo existe un arco , el 2-7, es decir, el arco {IR,T45}. Las 2 últimas columnas nos indican los vecinos comunes y excluidos.

- EDC.OUS. En este fichero se escribe el resultado de la asignación operación-OU. En la figura 5.3 se muestra el contenido de este fichero en el ejemplo que nos ocupa. Su formato ya fue comentado anteriormente.

```
                ASIGNACIONES OPERACION-ALU REALIZADAS POR PSAL

                Modulo        : edc
                Numero de ALUs :1

        ALU 1

        operacion     bits
         sr0          16
         srr          16
         srd          16
         -            16
         +            16
         -            16

         INTERCONEXIONES DEFINIDAS

        formato 123... 1 salida de la alu,2 input 1,3 input 2
        valores 0 .. conectado,1 .. desconectado,x .. optativo

         ALUs ----> 0 ... Maxalu
                    001 IR
                    111 ACC
                    001 CONSTANTES
```

Figura 5.3

```
EDC()
        ** variables locales a EDC **
            RUN<0:0>,
            CLEAR<0:0>,
            T35<1:0>,
            IR<15:0>,
            M[0:1023]<15:0>,
            CAR<9:0>,
            IN<0:0>,
            INPUT<15:0>,
            OUT<0:0>,
            T50<0:0>,
            OUTPUT<15:0>,
            ACC<15:0>,
            OVF<0:0>,
            CHAIN2<0:0>,
            CHAIN3<0:0>,
            CHAIN4<0:0>,
        ** entidades locales a EDC **
        ** Instrucciones de EDC **
    BEGIN
     bloque 2     :
        $0$ CONTINUE
     bloque 3     :
        $0$ CONTINUE
     bloque 4     :
        $0$ T35<1:0> = CLEAR<0:0> @ RUN<0:0> NEXT
        $0$  control T35<1:0> NEXT
        $0$  DECODE T35<1:0>
             0     GO TO 5
             1     GO TO 6
             2 : 3  GO TO 7
     bloque 5     :
        $0$  LOOP 2
     bloque 6     :
        $0$ CAR<9:0> = CAR<9:0>+{US} '1 ; $0$ IR<15:0>=M[CAR<9:0>]<15:0> NEXT
        $0$     GO TO 8
```

Figura 5.4

- EDC.DSC. Es éste un fichero, con un formato parecido a EDC.GRF, pero con la ventaja de que es más fácil de leer, además de no contener tanta información como aquel. En este fichero se muestra el grafo de flujo, indicando entre el signo de admiración (!) la OU a la cual esta asociada la operación a la que precede y entre dólares ($) el estado al cual esta asociada la operación. En la figura 5.4 se refleja el comienzo de dicho fichero.

2.4) Descripción DDL del sistema a ivel RT

El objetivo de PSAL1 es generar una descripción DDL, a partir de una descripción ISPS del sistema digital. En la figura 5.5 se muestra la descripción DDL que PSAL propone para el EDC.

```
<SY> edc:

<RE> RUN[1] ,
     CLEAR[1] ,
     IR[16] ,
     CAR[10] ,
     IN[1] ,
     OUT[1] ,
     ACC[16] ,
     OVF[1].

<TE> INPUT[16] ,
     OUTPUT[16] ,
     _M[10].

<ME> M[1024:16].

 <TI> clk_edc.

<AU> CPU_edc:clk_edc:

<ST>


   S0:
     ! CLEAR ' RUN
    # 0    ->S0
    # 1    CAR <- INC(CAR),
      _M=CAR,IR<-M[_M],->S1
    # 2:3    RUN <- 0D1,
     CLEAR <- 0D1,
     CAR <- 0D10,
     IR <- 0D16,->S1 ..
```

figura 5.5

```
S1:
  RUN :! IR[15 : 12]
 # 0   ->S2
 # 1   ->S3
 # 2   IR[9 : 0] <- 1D10 ,
  ACC <-IR[9 : 0],->S4
 # 3   ! IR[11 : 10]
 # 0   ACC <- 0D16
 # 1   ACC <- INC(ACC)
 # 2   ACC <- OU0(ACC, ACC, <<< + >>> )
 # 3   ACC <- OU0(ACC, 0d16, <<< - >>> ) .,->S0
 # 4   _M=IR[9 : 0],ACC<-M[_M],->S0
 # 5   _M=IR[9 : 0],IR<-M[_M],->S6
 # 6   _M=IR[9 : 0],IR<-M[_M],->S7
 # 7   _M=IR[9 : 0],M[_M] <-ACC,->S0
 # 8   _M=IR[9 : 0],IR<-M[_M],->S8
 # 9   ! IR[11 : 10]
 # 0   CAR <-IR[9 : 0]
 # 1   | +/(ACC @ 0D1 ) | CAR <-IR[9 : 0].
 # 2   | ACC[15] | CAR <-IR[9 : 0].
 # 3   | OVF | CAR <-IR[9 : 0]. .,->S0
 # 10  ! IR[11 : 10]
 # 0   ACC <-IR[9 : 0]
 # 1   ACC <- OU0(ACC, IR[9 : 0], <<< + >>> ) .,->S0
 # 11  ->S9 ..
S2:
  IN : _M=IR[9 : 0],M[_M] <-INPUT,
   IN <- 0D1,->S0.
S3:
   ~OUT : _M=IR[9 : 0],OUTPUT<-M[_M],
  OUT <- 1D1 ,->S0.
S4:
  | +/(ACC @ 0D1 ) | ->S0;ACC <- DEC(ACC),->S5 ..
S5:
  IN :IR[9 : 0] <- INC(IR[9 : 0]),
  IN <- 0D1,
   _M=IR[9 : 0],M[_M] <-INPUT,->S4.
S6:
   ACC <- OU0(ACC, IR, <<< + >>> ),->S0.
S7:
   ACC <- OU0(ACC, IR, <<< - >>> ),->S0.
S8:
   ACC <-ACC * IR,->S0.
S9:
  | +/(IR[9 : 0] @ 0D1 ) | ->S0;IR[9 : 0] <- DEC(IR[9 : 0]),! IR[11 : 10]
 # 0   ACC <- OU0(ACC, 1D16 , <<< srd >>> ),->S9
 # 1   ACC <- OU0(ACC, 1D16 , <<< sr0 >>> ),->S9
 # 2   ACC <- OU0(ACC, 1D16 , <<< srr >>> ),->S9
 # 3   ACC <- OU0(ACC, 8D16 , <<< srr >>> ),->S9 .....
```

Figura 5.5 (continuación)

En la descripción de la figura 5.5, se habrá observado que aparecen ciertos elementos que no estan definidos. Las unidades operacionales utilizadas para localizar las operaciones de la descripción, son introducidas por PSAL como elementos en DDL. El programa considera que la definición de dichas unidades operacionales es introducida por el usuario. En la descripción de salida las OU's presentan el siguiente formato:

OUnúmero(primer operando , segundo operando , código)

Siendo:

- número: valor asociado a la OU en el fichero "nombre.OUS".
- código: valor que identifica la operación a realizar. Dado que PSAL no implementa la OU, el programa no sabe que código tiene asociado la operación. Por ello escribe, en el campo asociado al código, la operación a realizar entre paréntesis angulares (<<<code>>>). Una vez la OU haya sido implementada, la expresión antes mencionada será sustituida por el código de la operación.

3) Síntesis del EDC utilizando el sistema PSAL2

3.1) Descripción VHDL del EDC

A partir de la descripción ISPS del EDC presentada en el apartado 2.1 de este mismo capítulo se propuso la siguiente representación para el EDC en VHDL:

```
-- Descripcion en VHDL del computador digital elemental EDC
-- propuesto por D. Dietmeyer.
--
-- Esta version esta basada en la descripcion en ISPS de dicho sistema
-- realizada en el Dpto. de Electronica de la Universidad de Cantabria
--      Grupo de Microelectronica
--

entity EDC is
        port ( IN_OK        : in   BIT;
               OUT_OK       : in   BIT;
               INPUT        : in   BIT_VECTOR(1 to 16);
               OUTPUT       : out  BIT_VECTOR(1 to 16);
               RUN          : in   BIT;
               CLEAR        : in   BIT;
               OVF          : in   BIT
             );
end;
```

```
-- Descripcion del comportamiento

architecture ALGORITHM of EDC is
begin
        process
                subtype word  is BIT_VECTOR(1 to 16);
                variable IR  : word;                                    -- registro de instrucciones
                variable ACC : word;                                    -- accumulador
                variable CAR : BIT_VECTOR( 1 to 10);                    -- Registro de direcciones
                type memory is array (POSITIVE range <>) of word;
                variable M       : memory(0 to 1023);                   -- Memory
                constant cero : BIT_VECTOR( 16 downto 1):= 0;
                constant uno  : BIT_VECTOR( 16 downto 1):= 1;           -- Campos del codigo de instrucciones
                alias OP    : BIT_VECTOR(0 to 3)  is IR(1 to 4 );
                alias IX    : BIT_VECTOR(1 to 2)  is IR(5 to 6 );
                alias ADR   : BIT_VECTOR(1 to 10) is IR(7 to 16);

        -- Decodificacion de la instruccion

                procedure execute is
                        begin
                                case OP is
                                        when X"0" =>
                                                wait until IN_OK;
                                                M(ADR):= INPUT;
                                        when X"1" =>
                                                wait until OUT_OK;
                                                OUTPUT<= M(ADR);
                                        when X"2" =>
                                                ACC:= ADR;
                                                ADR:= uno;
                                                while ACC /= cero  loop
                                                        wait until IN_OK;
                                                        M(ADR):=INPUT;
                                                        ACC :=ACC- uno;
                                                        ADR:=ADR+ uno(10 downto 1);
                                                end loop;
                                        when X"3" =>
                                                case IX is
                                                        when "00" =>
                                                                ACC:=cero;
                                                        when "01" =>
                                                                ACC:=ACC+uno;
                                                        when "10" =>
                                                                ACC:=ACC-uno;
                                                        when "11" =>
                                                                ACC:= -ACC;
                                                end case;
                                        when X"4" =>
                                                ACC:=M(ADR);
                                        when X"5" =>
                                                ACC:=ACC+M(ADR);
```

```
                when X"6" =>
                        ACC:=ACC-M(ADR);
                when X"7" =>
                        M(ADR):=ACC;
                when X"8" =>
                        ACC:=ACC and M(ADR);
                when X"9" =>
                        case IX is
                                when "00" =>
                                        CAR:= ADR;
                                when "01" =>
                                        if ACC = cero then
                                            CAR:=ADR;
                                        end if;
                                when "10" =>
                                        if ACC(1) then
                                            CAR:=ADR;
                                        end if;
                                when "11" =>
                                        if OVF then
                                            CAR:=ADR;
                                        end if;
                        end case;
                when X"A" =>
                        case IX is
                                when "00" =>
                                        ACC:= ADR;
                                when "01" =>
                                        ACC:=ACC+ADR;
                                when others =>
                                        null ;
                        end case;
                when X"B" =>
                        while ADR /= cero(10 downto 1) loop
                          ADR:=ADR-uno(10 downto 1);
                          case IX is
                                when "00" =>            -- shift right (signo)
                                        ACC:=ACC(1)&ACC(1 to 15);
                                when "01" =>            -- shift right ( 0)
                                        ACC:="0"&ACC(1 to 15);
                                when "10" =>            -- rotate
                                        ACC:=ACC(16)&ACC(1 to 15);
                                when "11" =>
                                        ACC:=ACC(9 to 16) & ACC(1 to 8);
                           end case;
                          end loop;
        end case;
end ;
```

```
        -  _____ Process body _____
        -  ..... MAIN DESCRIPTION .....

        begin
        case CLEAR & RUN is
                when "00" =>
                        null;
                when "01" =>
                        IR:=M(CAR);
                        CAR:=CAR + uno(10 downto 1);
                when "10" =>
                        CAR := cero(10 downto 1);
                        IR  := cero;
                when "11" =>
                        CAR := cero(10 downto 1);
                        IR  := cero;
        end case;
        wait until RUN;
        execute;
        end process;
end;
```

3.2) Ejecución de PSAL2

Una vez el comportamiento del sistema fue descrito en VHDL, se procedio a su síntesis con la herramienta PSAL2. En este ejemplo se ha utilizado minmo conjunto de instrucciones necesarias para realizar la síntesis. Debido a la flexibilidad del programa se podrían haber pedido as datos o utilizado otros algoritmos.

% psal2	LLamada al programa desde unix
Is your Terminal VT-100 compatible [Y/n] ?n	Determinacion del terminal
PSAL 2 High Level Synthesis Program 1991	Presentación
Psal>set file edc	Nombre de fichero = edc.XXX
Psal>set listing on	
Psal>set input vhdl	Tipo de entrada vhdl (entonces XXX = vhdl)
Psal>load	Carga del fichero

Input file : edc.vhdl
Listing file : edc.lis

Steps 1-2: Compiling vhdl -> psal2 — Módulo VHDL2PSAL
Step 3: Compiling psal2 -> Data base

Done !
Psal>**data** — Analisis del flujo de datos

Inicio el analisis del flujo de datos

INLINE : funcion execute — Expansión inline de entidades que solo son llamadas en una ocasión.

Inicio analisis global
Numero de iteraciones :4
Fin del analisis del flujo de datos

Scheduling

Psal>**scheduling**

Tipo de puertos

Tipo de modulo de los puertos de entrada del sistema **terminal**
Tipo de modulo de los puertos de salida del sistema **terminal**
Tipo de modulo de los puertos de entrada-salida del sistema **terminal**
Tipo de modulo de los puertos de datos de las memorias **terminal**
Tipo de modulo de los puertos de direcciones del sistema **terminal**

PSAL2 SCHEDULING SHELL

Opciones:
1 Force Directed Scheduling
2 List scheduling Force Directed Scheduling
3 List scheduling Force Directed Scheduling (restricted)
4 Modificado

scheduling/Psal>**2** — Se ha seleccionado List scheduling sin limitación en el número de OUs.

Running schedule algorithm

IDENTIFICADO TERMINAL. Variable: psal_vtmp_6
IDENTIFICADO TERMINAL. Variable: psal_vtmp_0
IDENTIFICADO TERMINAL. Variable: psal_vtmp_1
IDENTIFICADO TERMINAL. Variable: psal_vtmp_2
IDENTIFICADO TERMINAL. Variable: psal_vtmp_3
IDENTIFICADO TERMINAL. Variable: psal_vtmp_4
IDENTIFICADO TERMINAL. Variable: psal_vtmp_5

Inicio el analisis del flujo de datos

Repite el analisis del flujo de datos

Inicio analisis global
Numero de iteraciones :4
Fin del analisis del flujo de datos

Done !!

Allocation

Psal>**allocation**

PSAL2 HARDWARE ALLOCATION SHELL

Elige algoritmo:
0 global
1 probabilistico

Opcion ? **0**

Running variable synthesis algorithm

Escribo tabla de vida en fichero edc.tlive
Done !

INICIO LA SINTESIS DE VARIABLES

FIN DEL PROCESO DE SINTESIS
Resumen: 3 variables han sido agrupadas en 3 registros

Como minimo se necesitaban 3
Inicio el preanalisis
Fin del preanalisis

INICIO LA ASIGNACION OPERACION-ALU
OBTENIDA LA TABLA OPERACION-ALU

Tabla de exclusiones entre operaciones

Se necesitan 2 alus
INICIO LA SINTESIS DE OPERACIONES
FIN DE LA SINTESIS

Done !!

Una vez terminada la síntesis se escribe el resultado en lenguaje PSAL

Psal>**write**

La descripcion sera almacenada en el fichero edc_s.psal2

Done !
Psal>**set out cvs** Representación CVS_BK
Psal>**write**

La descripcion sera almacenada en el fichero edc.cvs

-- DESCRIPCION CVS_BK CONCLUIDA --

Done !
Psal>**set out vhdl** Representación VHDL
Psal>**write**

La descripcion sera almacenada en el fichero edc_s.vhdl

-- DESCRIPCION VHDL CONCLUIDA --

Done !
Psal>**set out esp** Representación ESP
Psal>**write**

-- DESCRIPCION ESP CONCLUIDA --

Done !

Psal>**set out dbm** Representación dbm
Psal>**write**

-- Almaceno la base de datos en el fichero edc.xxx

Done !

Psal>**exit** Fin de la ejecución

*** EJECUCION DEL PROGRAMA PSAL2 TERMINADA *****

ADIOS !

3.3) Descripción de la arquitectura resultante en VHDL

Una vez realizada la síntesis, la arquitectura resultante es descrita en diversos lenguajes. A continuación se puede observar la descripción VHDL. Las pecularíedades de esta descripción fueron comentadas en el capitulo 2.

```
entity edc is
        port
        (
        psal_vhdl_reset_terminal:in bit;
        ovf:in bit;
        in_ok:in bit;
        out_ok:in bit;
        input:in bit_vector(15 downto 0);
        run:in bit;
        clear:in bit;
        output:out bit_vector(15 downto 0);
        clock:in bit );
        end edc;
```

```
architecture RTL of edc

        type psram_0 is array (1023 downto 0) of bit_vector(15 downto 0);
        constant psal_const_1:bit:=0;
        constant psal_const_3:bit_vector(15 downto 0):=0;
        constant psal_const_2:bit_vector(15 downto 0):=1;
        signal psal_vhdl_reset:bit;
        signal ir:bit_vector(15 downto 0);
        signal car:bit_vector(9 downto 0);
        signal acc:bit_vector(15 downto 0);
        type statevalue is (
                        edc_state_3,
                        edc_state_4,
                        edc_state_5,
                        edc_state_6) ;
        signal reg_state : statevalue ;

        BEGIN

        PROCESS

        variable psal_vtmp_5:bit;
        variable psal_vtmp_4:bit;
        variable psal_vtmp_3:bit_vector(15 downto 0);
        variable psal_vtmp_2:bit_vector(15 downto 0);
        variable psal_vtmp_1:bit_vector(15 downto 0);
        variable psal_vtmp_0:bit;
        variable psal_vtmp_6:bit_vector(1 downto 0);
        variable psal_address_m:bit_vector(9 downto 0);
                variable m:psram_0;
        BEGIN

        IF  (psal_vhdl_reset OR psal_vhdl_reset_terminal )THEN
                IF(psal_vhdl_reset AND psal_vhdl_reset_terminal)THEN
                psal_vhdl_reset<='0';

        END IF;
        reg_state<= edc_state_3;
                ELSE
        CASE reg_state IS

WHEN edc_state_3 =>

        psal_vtmp_6 := clear &  run;

                CASE  psal_vtmp_6 IS
        when 0 =>
        NULL;
        when 1 =>
        psal_address_m := car;
         ir <= m( psal_address_m);
         car <= ALU_1(SUM, car,psal_const_2(9 downto 0));
```

```
when 2 =>
 car <= psal_const_3(9 downto 0);
 ir <= psal_const_3;
when 3 =>
 car <= psal_const_3(9 downto 0);
 ir <= psal_const_3;
END CASE;
        reg_state <= edc_state_4;
WHEN edc_state_4 =>

IF ( run = 1 ) THEN
        CASE ir(15 downto 12) IS
when 0 =>
IF ( in_ok = 1 ) THEN
psal_address_m := ir(9 downto 0);
 m( psal_address_m ) := input;

        reg_state <= edc_state_3;
 ELSE
 reg_state <= edc_state_4;
        END IF;
when 1 =>
IF ( out_ok = 1 ) THEN
psal_address_m := ir(9 downto 0);
output :=m( psal_address_m);
        reg_state <= edc_state_3;
 ELSE
 reg_state <= edc_state_4;
        END IF;
when 2 =>
 acc <= ir(9) & ir(9) & ir(9) & ir(9) & ir(9) & ir(9) & ir(9 downto 0);
 ir <= ir(15 downto 10) & (psal_const_2(9 downto 0));
        reg_state <= edc_state_5;
when 3 =>
        CASE ir(11 downto 10) IS
when 0 =>
 acc <= psal_const_3;
when 1 =>
 acc <= ALU_0(SUM, acc,psal_const_2);
when 2 =>
 acc <= ALU_0(RES, acc,psal_const_2);
when 3 =>
 acc <= ALU_0(NEG, acc);
END CASE;
         reg_state <= edc_state_3;

when 4 =>
psal_address_m := ir(9 downto 0);
 acc <= m( psal_address_m);
         reg_state <= edc_state_3;
```

```
when 5 =>
psal_address_m := ir(9 downto 0);
psal_vtmp_1 :=m( psal_address_m);
 acc <= ALU_0(SUM, acc, psal_vtmp_1);
          reg_state <= edc_state_3;

when 6 =>
psal_address_m := ir(9 downto 0);
psal_vtmp_2 :=m( psal_address_m);
 acc <= ALU_0(RES, acc, psal_vtmp_2);
          reg_state <= edc_state_3;

when 7 =>
psal_address_m := ir(9 downto 0);
 m( psal_address_m ) :=  acc;

          reg_state <= edc_state_3;

when 8 =>
psal_address_m := ir(9 downto 0);
psal_vtmp_3 :=m( psal_address_m);
 acc <= AND( acc, psal_vtmp_3);
          reg_state <= edc_state_3;

when 9 =>
          CASE  ir(11 downto 10) IS
when 0 =>
 car <=  ir(9 downto 0);
when 1 =>
psal_vtmp_4 :=EQL( acc,psal_const_3);
          IF( not ( psal_vtmp_4=BITVAL(0))) then
 car <=  ir(9 downto 0);
ELSE
NULL;
          END IF;
 when 2 =>
          IF( not ( acc(15)=BITVAL(0))) then
 car <=  ir(9 downto 0);
ELSE
NULL;
          END IF;
 when 3 =>
          IF( not ( ovf=BITVAL(0))) then
 car <=  ir(9 downto 0);
ELSE
NULL;
          END IF;
 END CASE;
           reg_state <= edc_state_3;

 when 10 =>
          CASE  ir(11 downto 10) IS
```

```
            when 0 =>
             acc <= ir(9) & ir(9) & ir(9) & ir(9) & ir(9) & ir(9) & ir(9 downto 0);
            when 1 =>
             acc <= ALU_0(SUM, acc, ir(9) & ir(9) & ir(9) & ir(9) & ir(9) & ir(9) & ir(9 downto 0));
                    when others =>
            NULL;
            END CASE;
                    reg_state <= edc_state_3;

            when 11 =>
                    reg_state <= edc_state_6;
                     when others =>
                     reg_state <= edc_state_3;

            END CASE;
             ELSE
             reg_state <= edc_state_4;
                    END IF;
WHEN edc_state_5 =>

            psal_vtmp_0 :=NEQ( acc,psal_const_3);
                    IF ( psal_vtmp_0 = BITVAL(0)) THEN

                    reg_state <= edc_state_3;
            ELSE

            IF ( in_ok = 1 ) THEN
            psal_address_m := ir(9 downto 0);
             m( psal_address_m ) := input;

             acc <= ALU_0(RES, acc,psal_const_2);
             ir <= ir(15 downto 10) & (ALU_1(SUM, ir(9 downto 0),psal_const_2(9 downto 0)));
                    reg_state <= edc_state_5;
             ELSE
             reg_state <= edc_state_5;
                    END IF;
                    END if;
WHEN edc_state_6 =>

            psal_vtmp_5 :=NEQ( ir(9 downto 0),psal_const_3(9 downto 0));
                    IF ( psal_vtmp_5 = BITVAT(0)) THEN

                    reg_state <= edc_state_3;
            ELSE

             ir <= ir(15 downto 10) & (ALU_1(RES, ir(9 downto 0),psal_const_2(9 downto 0)));
                    CASE ir(11 downto 10) IS
            when 0 =>
             acc <= acc(15) & acc(15 downto 1);
            when 1 =>
             acc <= psal_const_1 & acc(15 downto 1);
```

```
            when 2 =>
             acc <= acc(0) &  acc(15 downto 1);
            when 3 =>
             acc <= acc(7 downto 0) &  acc(15 downto 8);
            END CASE;
                    reg_state <= edc_state_6;
                    END if;
END case;
            END if;
            WAIT UNTIL (clock=0 and not clock'stable);
            END process;
END RTL;
```

En esta descripción las unidades operacionales aparecen como funciones cuyo nombre es ALU_"nº de la OU". El primer argumento es la operación que se realiza y el resto los operandos. La operación se representa por medio de simbolos (SUM,RES,...) que serán reemplazados por valores numéricos durante el proceso de síntesis a nivel RT. El sistema precisa 2 OUs. En la primera, ALU_0, se realizan las operaciones sobre el acumulador, mientras que en la segunda, ALU_1, se realizan los incrementos y decrementos de IR y CAR.

Se ha introducido una función BITVAL, para transformar enteros en vectores de bits.

3.4) Descripción CVS_BK del EDC.

La filosofía de esta descripción ya fue comentada en el capítulo 2. Unicamente conviene señalar la utilización e funciones externas (como "func $pop_wait_at_166") para modelar el test de la condición de la primitiva wait. Su introducción tiene como misión modelar condiciones dificiles de expresar en CVS_BK, como las que produce el comando "wait on" de VHDL.

```
module edc (
        in
                $psal_reset_terminal;
                ovf[0..0];
                in_ok[0..0];
                out_ok[0..0];
                input[15..0];
                run[0..0];
                clear[0..0]
        out
                output[15..0]
        );
```

```
func $pop_wait_at_166(data1_166[0..0]); external;
func $pop_wait_at_6(data1_6[0..0]); external;
func $pop_wait_at_11(data1_11[0..0]); external;
func $pop_wait_at_24(data1_24[0..0]); external;
        clock $psal_clock
                default;
        ram
                m[1023..0;15..0];
        node
                psal_vtmp_5[0..0];
                psal_vtmp_4[0..0];
                psal_vtmp_3[15..0];
                psal_vtmp_2[15..0];
                psal_vtmp_1[15..0];
                psal_vtmp_0[0..0];
                psal_vtmp_6[1..0];
                psal_address_m[9..0];
                output[15..0];
        register
                $psal_reset;
                ir[15..0];
                car[9..0];
                acc[15..0];
        constant
                $psal_const_1[0..0].=0;
                $psal_const_3[15..0].=0;
                $psal_const_2[15..0].=1;
        state
                edc_state_list = (

                        edc_state_3,
                        edc_state_4,
                        edc_state_5,
                        edc_state_6  );
        begin

         whenever ( $psal_reset or $psal_reset_terminal ) goto edc_state_3;
         sequence

        edc_state_3:
                 if ($psal_reset and $psal_reset_terminal ) then $psal_reset:='0 endif;
                psal_vtmp_6.= clear. run;
                case  psal_vtmp_6 of
        0:
        noop
        1:
                psal_address_m.= car;
                ir:=m[ psal_address_m] { ir <- psal_data_m } ;
                car:= car + $psal_const_2[9..0]
        2:
                car:=$psal_const_3[9..0];
                ir:=$psal_const_3
```

```
3:
        car:=$psal_const_3[9..0];
        ir:=$psal_const_3
endcase;
        goto edc_state_4
edc_state_4:
        if $pop_wait_at_166( run )  then
        case ir[15..12] of
0:
        if $pop_wait_at_6( in_ok )  then
        psal_address_m.= ir[9..0];
        m[ psal_address_m ] :=  input { psal_data_m<-input } ;
        goto edc_state_3
        else
         goto edc_state_4
        endif
1:
        if $pop_wait_at_11( out_ok )  then
        psal_address_m.= ir[9..0];
        output.=m[ psal_address_m] { output <- psal_data_m } ;
        goto edc_state_3
        else
         goto edc_state_4
        endif
2:
        acc:= ir[9]. ir[9]. ir[9]. ir[9]. ir[9]. ir[9]. ir[9..0];
        ir:=ir[15..10]. ($psal_const_2[9..0]);
        goto edc_state_5
3:
        case ir[11..10] of
0:
        acc:=$psal_const_3
1:
        acc:= acc + $psal_const_2
2:
        acc:= acc - $psal_const_2
3:
        acc:= -  acc
endcase;
         goto edc_state_3
4:
        psal_address_m.= ir[9..0];
        acc:=m[ psal_address_m] { acc <- psal_data_m } ;
         goto edc_state_3
5:
        psal_address_m.= ir[9..0];
        psal_vtmp_1.=m[ psal_address_m] { psal_vtmp_1 <- psal_data_m } ;
        acc:= acc +  psal_vtmp_1;
         goto edc_state_3
6:
        psal_address_m.= ir[9..0];
        psal_vtmp_2.=m[ psal_address_m] { psal_vtmp_2 <- psal_data_m } ;
        acc:= acc -  psal_vtmp_2;
```

```
        goto edc_state_3
7:
        psal_address_m.= ir[9..0];
        m[ psal_address_m ] := acc { psal_data_m<-acc } ;
        goto edc_state_3
8:
        psal_address_m.= ir[9..0];
        psal_vtmp_3.=m[ psal_address_m] { psal_vtmp_3 <- psal_data_m } ;
        acc:= acc and psal_vtmp_3;
        goto edc_state_3
9:
        case ir[11..10] of
0:
        car:= ir[9..0]
1:
        psal_vtmp_4.= acc - $psal_const_3;
        if(not(EQU( psal_vtmp_4))) then
        car:= ir[9..0]
else
noop
        endif
2:
        if(not(EQU( acc[15]))) then
        car:= ir[9..0]
else
noop
        endif
3:
        if(not(EQU( ovf))) then
        car:= ir[9..0]
else
noop
        endif
endcase;
        goto edc_state_3
10:
        case ir[11..10] of
0:
        acc:= ir[9]. ir[9]. ir[9]. ir[9]. ir[9]. ir[9]. ir[9..0]
1:
        acc:= acc + ir[9]. ir[9]. ir[9]. ir[9]. ir[9]. ir[9]. ir[9..0]
        else
noop
endcase;
        goto edc_state_3
11:
        goto edc_state_6
        else
        goto edc_state_3
endcase
        else
        goto edc_state_4
        endif
```

```
edc_state_5:
        psal_vtmp_0.= acc <> $psal_const_3;
        if  psal_vtmp_0   then
        goto edc_state_3
        else
        if $pop_wait_at_24( in_ok )  then
        psal_address_m.= ir[9..0];
        m[ psal_address_m ] :=  input { psal_data_m<-input } ;
        acc:= acc - $psal_const_2;
        ir:=ir[15..10]. ( ir[9..0] + $psal_const_2[9..0]);
        goto edc_state_5
        else
         goto edc_state_5
        endif
        endif
 edc_state_6:
        psal_vtmp_5.= ir[9..0] <> $psal_const_3[9..0];
        if  psal_vtmp_5   then
        goto edc_state_3
        else
        ir:=ir[15..10]. ( ir[9..0] - $psal_const_2[9..0]);
        case  ir[11..10] of
 0:
        acc:= acc[15]. acc[15..1]
 1:
        acc:=$psal_const_1. acc[15..1]
 2:
        acc:= acc[0]. acc[15..1]
 3:
        acc:= acc[7..0]. acc[15..8]
 endcase;
        goto edc_state_6
        endif
end.
```

3.5) Descripción ESP del EDC

Como ya se comentó en el capítulo 2, la descripción en ESP esta fomado por varios módulos. Esta es la descripción del módulo principal, en donde esta contenido el comportamiento del sistema. En los otros módulos se describen las unidades operacionales. Por esta razón aparecen como puertos de la descripción las conexiones con las OUS. PSAL2 define 3 puertos de salida y uno de entrada para modelar la conexión entre el módulo principal y una OUS. Por ejemplo, los puertos de salida:

```
aluinALU_0_1<15:0>
aluinALU_0_2<15:0>
alucont_ALU_0<1:0>
```

modelan las dos entradas de la OU número 0, asi como la señal de control que deermina la operación a realizar. El puerto de entrada :

```
alures_ALU_0<15:0>
```

modela la salida de la OU 0. La asignación simbólica de valores a los códigos de las OUs, se modela mediante constantes. Asi la contante:

```
contalu_ALU_1_0=0,  !65
```

Modela la operacion relacional igual en la OU 0. El simbolo '!' señala el inicio de una línea de comentario. Los caracteres existentes entre dicho simbolo y el final de la linea serán ignorados.

```
model edc
        output<15:0>,
        aluinALU_1_1<15:0>,
        aluinALU_1_2<15:0>,
        alucont_ALU_1,
        aluinALU_0_1<15:0>,
        aluinALU_0_2<15:0>,
        alucont_ALU_0<1:0>
        =
        psal_esp_reset_terminal,
        ovf,
        clear,
        run,
        input<15:0>,
        out_ok,
        in_ok,
        alures_ALU_0<15:0>,
        alures_ALU_1<15:0>,
        clock ;


        $RAMNAME
                m<1023:15>;

        $REGISTERNAME
        psal_esp_reset,
        ir<15:0>,
        acc<15:0>,
        car<9:0>;
```

```
CONSTANT
psal_const_1=0#2,
psal_const_2<15:0>=0000000000000001#2,
psal_const_3<15:0>=0000000000000000#2,
contalu_ALU_1_0=0,!65
contalu_ALU_1_1=1,!66
contalu_ALU_0_0<1:0>=00#2,!65
contalu_ALU_0_1<1:0>=01#2,!66
contalu_ALU_0_2<1:0>=10#2;!64

STATE
        psal_address_m<9:0>,
        psal_vtmp_6<1:0>,
        psal_vtmp_0,
        psal_vtmp_1<15:0>,
        psal_vtmp_2<15:0>,
        psal_vtmp_3<15:0>,
        psal_vtmp_4,
        psal_vtmp_5;
$STATENAME

                edc_state_3,
                edc_state_4,
                edc_state_5,
                edc_state_6;

ROUTINE edc_edc;
$BODY
BEGIN
 IF  ( $REG psal_esp_reset OR psal_esp_reset_terminal )THEN
        BEGIN
IF( $REG psal_esp_reset AND psal_esp_reset_terminal)THEN
BEGIN
$LOAD psal_esp_reset=0;
END;
$GOTO edc_state_3;
END
ELSE
BEGIN
 $TABLA

 [edc_state_3]:BEGIN
 psal_vtmp_6 = clear &  run;
        SELECTONE  psal_vtmp_6 FROM
 [0]: BEGIN
 BUF;
 END;
 [1]: BEGIN
 psal_address_m = $REG car;
 $LOAD ir =  $READ m< psal_address_m>;
 alucont_ALU_1 = contalu_ALU_1_0;
 aluinALU_1_1 =  SXT {WIDTH = 15 } $REG car;
```

```
aluinALU_1_2 = SXT {WIDTH = 15 }psal_const_2<9:0>;
$LOAD car = alures_ALU_1<9:0>;
END;
[2]: BEGIN
$LOAD car = psal_const_3<9:0>;
$LOAD ir = psal_const_3;
END;
[3]: BEGIN
$LOAD car = psal_const_3<9:0>;
$LOAD ir = psal_const_3;
END;
ENDSELECTONE;
        $GOTO edc_state_4;
END;
[edc_state_4]:BEGIN
        IF  run EQL 0    THEN
BEGIN
        SELECTONE  $REG ir<15:12> FROM
[0]: BEGIN
        IF  in_ok EQL 0    THEN
BEGIN
psal_address_m = $REG ir<9:0>;
$WRITE m< psal_address_m > =  input;

        $GOTO edc_state_3;
END
        ELSE
BEGIN
         $GOTO edc_state_4;
        END;
END;
[1]: BEGIN
        IF  out_ok EQL 0    THEN
BEGIN
psal_address_m = $REG ir<9:0>;
output = $READ m< psal_address_m>;
        $GOTO edc_state_3;
END
         ELSE
BEGIN
         $GOTO edc_state_4;
        END;
END;
[2]: BEGIN
$LOAD acc =  (SXT { WIDTH = 15} $REG ir<9:0> );
$LOAD ir = ir<15:10> & (psal_const_2<9:0>);
        $GOTO edc_state_5;
END;
[3]: BEGIN
        SELECTONE  $REG ir<11:10> FROM
[0]: BEGIN
$LOAD acc = psal_const_3;
```

```
END;
[1]: BEGIN
alucont_ALU_0 = contalu_ALU_0_0;
aluinALU_0_1 = $REG acc;
aluinALU_0_2 = psal_const_2
$LOAD acc = alures_ALU_0<15:0>;
END;
[2]: BEGIN
alucont_ALU_0 = contalu_ALU_0_1;
aluinALU_0_1 = $REG acc;
aluinALU_0_2 = psal_const_2
$LOAD acc = alures_ALU_0<15:0>;
END;
[3]: BEGIN
alucont_ALU_0 = contalu_ALU_0_2;
aluinALU_0_1 = $REG acc;
$LOAD acc = alures_ALU_0<15:0>;
END;
ENDSELECTONE;
        $GOTO edc_state_3;

END;
[4]: BEGIN
psal_address_m = $REG ir<9:0>;
$LOAD acc = $READ m< psal_address_m>;
        $GOTO edc_state_3;

END;
[5]: BEGIN
psal_address_m = $REG ir<9:0>;
psal_vtmp_1 = $READ m< psal_address_m>;
alucont_ALU_0 = contalu_ALU_0_0;
aluinALU_0_1 = $REG acc;
aluinALU_0_2 = psal_vtmp_1
$LOAD acc = alures_ALU_0<15:0>;
        $GOTO edc_state_3;

END;
[6]: BEGIN
psal_address_m = $REG ir<9:0>;
psal_vtmp_2 = $READ m< psal_address_m>;
alucont_ALU_0 = contalu_ALU_0_1;
aluinALU_0_1 = $REG acc;
aluinALU_0_2 = psal_vtmp_2
$LOAD acc = alures_ALU_0<15:0>;
        $GOTO edc_state_3;

END;
[7]: BEGIN
psal_address_m = $REG ir<9:0>;
$WRITE m< psal_address_m > = $REG acc;

        $GOTO edc_state_3;
```

```
END;
[8]: BEGIN
psal_address_m = $REG ir<9:0>;
psal_vtmp_3 = $READ m< psal_address_m>;
$LOAD acc = $REG acc AND psal_vtmp_3;
        $GOTO edc_state_3;

END;
[9]: BEGIN
        SELECTONE $REG ir<11:10> FROM
[0]: BEGIN
$LOAD car = $REG ir<9:0>;
END;
[1]: BEGIN
psal_vtmp_4 = $REG acc EQL psal_const_3;
        IF( not ( psal_vtmp_4 EQL 0 )) THEN
BEGIN
$LOAD car = $REG ir<9:0>;
        END;
END;
[2]: BEGIN
        IF( not ( $REG acc<15> EQL 0 )) THEN
BEGIN
$LOAD car = $REG ir<9:0>;
        END;
END;
[3]: BEGIN
        IF( not ( ovf EQL 0 )) THEN
BEGIN
$LOAD car = $REG ir<9:0>;
        END;
END;
ENDSELECTONE;
        $GOTO edc_state_3;

END;
[10]: BEGIN
        SELECTONE $REG ir<11:10> FROM
[0]: BEGIN
$LOAD acc = (SXT { WIDTH = 15} $REG ir<9:0> );
END;
[1]: BEGIN
alucont_ALU_0 = contalu_ALU_0_0;
aluinALU_0_1 = $REG acc;
aluinALU_0_2 = SXT {WIDTH = 15 } $REG ir<9:0>;
$LOAD acc = alures_ALU_0<15:0>;
END;
ENDSELECTONE;
        $GOTO edc_state_3;

END;
```

```
[11]: BEGIN
        $GOTO edc_state_6;
END;
        [OTHERWISE]:
BEGIN
        $GOTO edc_state_3;
END;
ENDSELECTONE;
END
        ELSE
BEGIN
         $GOTO edc_state_4;
         END;
END;
[edc_state_5]:BEGIN
psal_vtmp_0 = $REG acc NEQ psal_const_3;
        IF  psal_vtmp_0   THEN
BEGIN
        $GOTO edc_state_3;
END
ELSE
BEGIN
        IF  in_ok EQL 0    THEN
BEGIN
psal_address_m = $REG ir<9:0>;
$WRITE m< psal_address_m > =  input;

alucont_ALU_0 = contalu_ALU_0_1;
aluinALU_0_1 =  $REG acc;
aluinALU_0_2 = psal_const_2
$LOAD acc = alures_ALU_0<15:0>;
alucont_ALU_1 = contalu_ALU_1_0;
aluinALU_1_1 =  SXT {WIDTH = 15 } $REG ir<9:0>;
aluinALU_1_2 =  SXT {WIDTH = 15 }psal_const_2<9:0>;
$LOAD ir = ir<15:10> & (alures_ALU_1<9:0>);
        $GOTO edc_state_5;
END
         ELSE
BEGIN
          $GOTO edc_state_5;
         END;
         END;
END;
[edc_state_6]:BEGIN
psal_vtmp_5 = $REG ir<9:0> NEQ psal_const_3<9:0>;
         IF  psal_vtmp_5   THEN
BEGIN
         $GOTO edc_state_3;
END
ELSE
BEGIN
alucont_ALU_1 = contalu_ALU_1_1;
aluinALU_1_1 =  SXT {WIDTH = 15 } $REG ir<9:0>;
```

```
        aluinALU_1_2 = SXT {WIDTH = 15 }psal_const_2<9:0>;
        $LOAD ir = ir<15:10> & (alures_ALU_1<9:0>);
                SELECTONE $REG ir<11:10> FROM
        [0]: BEGIN
        $LOAD acc = $REG acc<15> & $REG acc<15:1>;
        END;
        [1]: BEGIN
        $LOAD acc = psal_const_1 & $REG acc<15:1>;
        END;
        [2]: BEGIN
        $LOAD acc = $REG acc<0> & $REG acc<15:1>;
        END;
        [3]: BEGIN
        $LOAD acc = $REG acc<7:0> & $REG acc<15:8>;
        END;
        ENDSELECTONE;
                $GOTO edc_state_6;
                END;
        END;

        $ENDTABLA

END;
END;

ENDROUTINE;

ENDMODEL;
```

4) Evaluación de los resultados

Se va a realizar a continuación un estudio de las implementaciónes de EDC obtenidas por PSAL1 y PSAL2, realizando además una comparación con la implementación propuesta por D. Dietmeyer en [Di79].

En [Di79], el EDC sufre una serie de depurados desde una descripción inicial de 16 estados, hasta la descripción revisada siguiente:

```
<SY>  EDC:
<TI>  P(1E-6).
<ME> M[1024 : 16].
<RE>  IR[16] = OP[0:3] ' IX[2] ' ADR[10], CAR[10], ACC[16], MAR[10],
 INPUT[16], OUTPUT[16], OVF, RUN, CLEAR, IN, OUT.
<TE>  EADR[6].<ID>  XADR = EADR ' ADR.                    extensión de ADR a partir
<BO>  EADR = ADR[1].                                      de su bit mas significativo
<AU> CPU: P:
 <ST> IF1 :  | CLEAR | RUN <- 0D1, CLEAR <- 0D1, CAR <- 0D10, IR <- 0D16, ->IF3;
       | RUN | MAR <- CAR, CAR <- INC(CAR), -> IF2; -> IF1...
       IF2 :  IR <- M[MAR], ->IF3.
       IF3 : RUN : MAR <- ADR,
          ! OP   #2  ADR <- 1D10, ACC <- 0D6 ' ADR, -> EX  #12:15 -> IF1; -> EX..
       EX :  ! OP   #0 | IN | M[MAR] <- INPUT, IN <- 0D1, -> IF1; -> EX.
             #1 | ~OUT | OUTPUT <- M[MAR], OUT <- 1D1, -> IF1; -> EX.
             #2 | +/ACC | MAR <- ADR, ACC <- DEC(ACC), -> EXBIX; -> IF1.
             #3 ACC <-                          ! IX  #0 0D16 #1 ACC + 1D16 #2 ACC + ACC
                    #1 0D16 - ACC., ->IF1.
             #4   ACC <- M[MAR], -> IF1
             #5   ACC <- ACC + M[MAR], -> IF1
             #6   ACC <- ACC - M[MAR], -> IF1
             #7   M[MAR] <-  ACC, -> IF1
             #8   ACC <- ACC * M[MAR], -> IF1
             #9 ! IX   #0 CAR = <- ADR #1 | ~+/ACC | CAR = <- ADR.
                  #2 | ACC[1]  | CAR = <-ADR.
                  #3 | OVF | CAR = <- ADR.., -> IF1
             #10 ACC <- ! IX                    #0 XADR    #1 ACC + XADR., -> IF1
             #11 | +/ADR | ADR <- DEC(ADR),
                 ACC <- ! IX                    #0 ACC[1] ' ACC[1:15] #1 ACC[0] ' ACC[1:15]
                      #2 ACC[16] ' ACC[1:15]
                      #3 ACC[9:16] ' ACC[1:8]., -> EX; -> IF1.
       EXBIX : IN : M[MAR] <- INPUT, ADR <- INC(ADR), IN <- 0D1, -> EX....
```

En ella, se utilizan operadores arítmeticos no definidos en el simulador, como la suma "+" y la substracción "-". En cualquier caso, esta descripción no recoge la implementación de las operaciones sobre el acumulador en una Unidad Aritmética y Lógica (ALU) tal y como luego comentaremos. Vamos a proceder al análisis comparativo las tres descripciones, tanto en cuanto al número de estados de la UC como a los recursos "hardware" utilizados en la UD.

1º) Registros y terminales

PSAL2 propone básicamente los mismos registros que PSAL1 y [Di79]. A pesar de ello, existen diferencias entre las descripciones debido a su diferente filosofia. Por ejemplo, como PSAL2 define explicitamente los puertos no les asocia directamente a registros ni terminales. En cambio existen diferencias entre PSAL1 y [Di79] en este punto. Asi PSAL1 propone que INPUT y OUTPUT sean implementadas como terminales de entrada y salida mientras que en [Di79] se utilizan registros. Esta modificación solo afecta al protocolo de comunicación de EDC con el exterior no descrito explícitamente en el programa ISPS. Con respecto a la variable de direccionamiento a memoria, como ya hemos comentado, PSAL1 supone el uso de un "bus" que denomina "_M" mientras que en [Di79] se utiliza un registo. Todos estos probremas estan resueltos en PSAL2, en donde el usuario puede elegir como quiere que se modelen en cada caso los puertos del sistema y de las memorias. En el ejemplo mostrado se han supuesto los mismos valores que en PSAL1, para poder compararlos.

2º) Estados

A pesar de que el direccionamiento a memoria se realiza a través del terminal "_M", lo que permite reducir el número de estados, el autómata de control de EDC propuesto por PSAL1 tiene 10 estados mientras que la descripción de EDC de [16] se ha realizado con solo 5. Este resultado se debe a varias causas:

a) En la implementación actual de PSAL1, la lectura de un dato de memoria requiere su almacenamiento en un registro como paso previo a su utilización en alguna operación aritmética. Esta limitación, que no existe en [Di79], obliga al uso de los estados S6, S7 y S8.

b) En la implementación actual de PSAL1, el estamento "wait" ISPS obliga a la utilización de un estado en DDL de espera de la condición. Este hecho genera los estados S2 y S3 cuyas acciones podrían ejecutarse en S1.

c) En la implementación actual de PSAL1, el estamento "repeat" ISPS obliga a la utilización de un estado en DDL de retorno del lazo independiente del estado en que dicho lazo es llamado. Este hecho genera el estado S9 cuya acción podría ejecutarse en S1.

PSAL2 supera estos obstáculos, por lo cual propone una descripción con 4 estados. Esta descripción es muy parecida a [Di79], con la diferencia que PSAL2 se ahora el estado IF2 por no necesitar cargar el bus de direcciones en el estado anterior. Frente a la descripción de PSAL1 el ahorro es de 6 estados (¡mas del 50% de reducción!), debido a que PSAL2 puede encadenar operaciones, modela las operaciones de espera de una forma mas eficiente y además la síntesis no se ve afectada por los problemas específicos del lenguaje de entrada como pasaba con PSAL1.

3º) Unidades operacionales

Aunque no descritas explicítamente en el programa DDL anterior, D. Dietmeyer propone la realización de todas las operaciones aritméticas, lógicas y de desplazamiento sobre el acumulador ACC en una única unidad operacional o ALU. Las operaciones sobre los registros CAR (incrementación) y ADR (incrementación y decrementación) se realizan en el propio registro.

PSAL1, en su implementación actual, implementa las operaciones de autoincrementación y autodecrementación directamente sobre el registro al que se refieren. Esto hace que PSAL proponga también una única unidad operacional para la realización de todas las operaciones aritméticas, lógicas y de desplazamiento sobre el acumulador ACC, salvo la incrementación y decrementación que se implementan en el mismo registro.

PSAL2 ofrece la misma solución que [Di79]. El sistema propone implementar el EDC con dos unidades operacionales. En la primera, ALU_0, se realizan todas las operaciones aritméticas con el acumulador. Las incrementaciones de CAR y ADR se realizan en ALU_1.

III) Síntesis del computador PDP8

1) Introducción

Las razones de elegir el PDP8 fueron su caracter de computador comercial y el tratarse de un sistema facil de analizar. Además, al ser un computador que ha estado 15 años en el mercado (como veremos en la descripción de su evolución histórica) se dispone de diseños e implementaciones, lo que permite comparar los resultados.

El ancestro de la familia PDP8 fué el computador instrumental de laboratorio (LINC) inicialmente desarrollado en el "MIT Lincoln Laboratory" en 1.962. La empresa DEC comenzó a fabricar LINCs en 1.965. Eventualmente un PDP8 y un LINC fueron combinados dando lugar a un procesador dual llamado LINC8.

Por otro lado, en 1.962, la necesidad de construir un sistema de control analógico para un complejo control de reactor obliga a desarrollar un computador de 12 bits de control en tiempo real: el PDP5. La naturaleza analógica de la aplicación inicial se refleja en la implementación de un convertidor analógico-digital en el acumulador. Pronto se hizo evidente la necesidad de contar con una nueva máquina con unas prestaciones mucho mayores. La nueva tecnología digital prometía importantes mejoras en cuanto a velocidad y una nueva tecnología de memorias con nucleos de fenita (core memory) comenzaba a estar disponible permitiendo reducir el ciclo de acceso de los 6 microsegundos de PDP5 a los 1,6 de la nueva máquina. Además, el coste de la lógica había reducido tanto que permitía que el contador de programas (program counter) pudiera ser movido de la mejoria a un registro separado, reduciendo substancialmente los tiempos de ejecución de la instrucción. La nueva máquina, con 12 bits y con la mitad de tamaño que sus antecesores, se llamó PDP8. Por todo ello se puede afirmar que el PDP8 es el primer minicomputador.

Como su predecesor el PDP5, esta máquina era un computador de 12 bits de dimensionamiento único, diseñado para tareas con aritmética mínima y requerimientos de memoria primaria pequeños. El primer equipo fué producido en abril de 1.965, y otras máquinas de la familia PDP8 que la siguieron se mantuvieron en producción hata los 80. De esta forma se produjeron alrededor de 50.000 máquinas de esta familia en 1.979 excluyendo los computadores basados en CMOS. Durante los 15 años que estuvo en producción sufrió diversas modificaciones, tendentes a introducir las tecnologías emergentes respetando el conjunto de instrucciones original. De esta forma surgen el PDP8/S, PDP8/I, PDP8/L, PDP8/G, PDP8/M, PD8-8/A (estos últimos también conocidos como computadores omnibus-8).

En 1.976, Intersil ofreció el primer procesador PDP8 en un chip, con tecnología CMOS (el CMOS-8). Este chip fué utilizado como parte del terminal de vídeo VT78, ofreciendo enormes ventajas tanto en coste como en prestaciones.

El PDP8 opera con palabras de 12 bits, enteras de 12 bits o vectores booleanos del mismo tamaño. Tiene unas pocas operaciones: '=', '+', '-' (unario), not, and, rotación a la izquierda y rotación a la derecha. Opcionalmente puede tener '*', '/' y normalización.

El sistema tiene 2^{12} palabras divididas en 32 páginas de 128 palabras cada una. El cálculo de la dirección está basado en referencias a la primera página (page-zero) o a la página actual del pc (program counter), señala el alias 'pa'. Existe una función 'eadd' que calcula la dirección efectiva en cada caso. Esta estructura permite expresar direcciones con solo 7 bits (dirección en la página local).

El procesador posee un acumulador de 12 bits (AC), un bit de extensión del acumulador, llamado (L) el contador de programa (PC), el registro de instrucciones, el registro que indica cuando el sistema debe estar funcionando y el bit de permiso de interrupción. Los puertos son las conexiones con la consola (o switches) y los requerimientos externos de interrupción. Existen 8 instrucciones básicas codificadas en 3 bits del registro de instrucciones. Cada una de las 6 primeras instrucciones de referencia a memoria tiene 4 modos de dimensionamiento:

- directo a página cero
- directo a página actual
- indirecto a página uno
- indirecto a página actual

Estas 6 primeras instruccions se agrupan en 4 categorías:

1.- Transmisión de datos: deposita y limpia el acumulador DCA
2.- Operaciones Aritméticas: Suma en complemento 2 con el acumulador (TAD).
3.- Operaciones booleanos: AND con el acumulador (AND)
4.- Control del programa, como por ejemplo salto a subrutina (JMS) o salto incondicional (JMP).

Las instrucciones de transferencia entrada-salida (código de operación 6) unen los restantes 9 bits de la instrucción para especificar instrucciones a los dispositivos de entrada/salida. Los 6 bits de la variable id.select, seleccionan el dispositivo, y los bits de io-control la operación del dispositivo de I/O. Se pueden presentar 3 situaciones.

1.- Comprobar una condición booleana en un dispositivo de I/O.
2.- Leer un dato del dispositivo e introducirlo en el aumulador.
3.- Escribir el dato del acumulador en un dispositivo.

Las intrucciones con código de operación 7, reciben el nombre de instruccions de operario. Se dividen en 3 grupos:

- Operaciones del grupo 1: Operar ocn los registros del procesador
- Operacioens del grupo 2: Comprueban el estado del procesador
- Grupo de elementos de extensión aritmética paa multiplicaciones, divisiones, etc. (no incluídos en las descripciones).

Por otro lado, dispositivos externos pueden solicitar que el procesador se detenga utilizando la señal INTERRUPT-REQUEST. El procesador sólo se detendrá si además INTERRUPT-ENABLE vale 1.

2) Síntesis del PDP8 utilizando el sistema PSAL1

2.1) Descripción ISPS

La descripción algorítmica ha sido tomada del conjunto de ejemplos que se distribuyen conjuntamente al simulador y parser de ISPS. Esta descripción tambien se puede encontar en [SiBe82].

```
PDP8 := begin
! The basic PDP-8 instruction set, not including the extended arithmetic
! element (EAE) option.  I/O instructions are limited to those dealing
! with the interrupt mechanism

! Original description (ISPL) and conversion to ISPS by Mario Barbacci
! Addition of extended accumulator and additional group 3 microoperations
! by Larry Lai (1980)


** Mp.state **

M\Memory[0:4095]<0:11>,

** Pc.state **

PC\Program.Counter<0:11>,
cpage\current.page<0:4>,
lac<0:12>,
     L\Link<>          := lac<0>,
     AC\Accumulator<0:11>   := lac<1:12>,
MQ\Multiplier.Quotient.Register<0:11>,
interrupt.state<>,
interrupt.request<>,
switches<0:11>,

** Instruction.Format **

i\instruction<0:11>,

        op\operation.code<0:2>    := i<0:2>,
        ib\indirect.bit<>  := i<3>,
        pb\page.0.bit<>           := i<4>,
        pa\page.address<0:6>      := i<5:11>,

        io.select<0:5>            := i<3:8>,        ! device select
```

```
        io.control<0:2>          := i<9:11>,      ! device operation
                io.pulse.p1<>            := io.control<0>,
                io.pulse.p2<>            := io.control<1>,
                io.pulse.p4<>            := io.control<2>,

        group<> := i<3>,                 ! microinstruction group
        CLA<> := i<4>,                   ! clear AC (all groups)
! group 1 operations (group bit = 0)
        CLL<> := i<5>,                   ! clear L
        CMA<> := i<6>,                   ! complement AC
        CML<> := i<7>,                   ! complement L
        ROT<0:2>:= i<8:10>,              ! rotate group
        IAC<> := i<11>,                  ! increment AC
! group 2 operations (group bit = 1 and bit 11 = 0)
        SMA<> := i<5>,                   ! skip on minus AC (is bit = 0)
        SZA<> := i<6>,                   ! skip on zero AC (is bit = 0)
        SNL<> := i<7>,                   ! skip on L not zero (is bit = 0)
        SPA<> := i<5>,                   ! skip on positive AC (is bit = 1)
        SNA<> := i<6>,                   ! skip on AC not zero (is bit = 1)
        SZL<> := i<7>,                   ! skip on L zero (is bit = 1)
        is<>    := i<8>,                 ! invert skip sense
        OSR<> := i<9>,                   ! logical or AC with switches
        HLT<> := i<10>,                  ! halt the processor



** Address.Calculation **

eadd\effective.address<0:11> :=
    begin
    DECODE pb =>
            begin
            0 := eadd ='00000 @ pa,
            1 := eadd = cpage @ pa
            end next
    IF ib =>
                begin
                IF eadd<0:8> eql #001 => M[eadd] = M[eadd] + 1 next
                eadd = M[eadd]
                end
    end,

** Interpretation.Process **

main interpret :=
        begin
        REPEAT          begin
                i = M[PC]; cpage = PC<0:4> next
                PC = PC + 1 next
                execute() next
                IF interrupt.state and interrupt.request =>
                        begin
```

```
                        M[0] = PC next
                        PC = 1
                        end
                end
        end,

** Instruction.Set **
execute :=
        begin
        DECODE op =>
                begin
        #0\AND :=       AC = AC and M[eadd()],
        #1\TAD :=       lac = lac + ('0 @ M[eadd()]),
        #2\ISZ :=       begin
                        M[eadd] = M[eadd()] + 1 next
                        IF M[eadd] eql 0 => PC = PC + 1
                        end,
        #3\DCA :=       begin
                        M[eadd()] = AC next
                        AC = 0
                        end,
        #4\JMS :=       begin
                        M[eadd()] = PC next
                        PC = eadd + 1
                        end,
        #5\JMP :=       PC = eadd(),
        #6\iot := input.output(),
        #7\opr :=       operate()
                end
        end,

input.output :=
        begin
        DECODE i<3:11> =>
                begin
                #001\ION :=
                        begin           ! turn Interrupt ON
                        interrupt.state = 1 next
                        RESTART interpret
                        end,
                #002\IOF :=
                        begin           ! turn Interrupt OFF
                        interrupt.state = 0
                        end,
                Otherwise := no.op()    ! not implemented
                end
        end,

skip<>,
```

```
skip.group :=
        begin
        DECODE is =>
                begin
                0 :=    begin
                        skip = 0 next
                        IF SNL and (L eql{US} '1) => skip = 1;
                        IF SZA and (AC eql 0) => skip = 1;
                        IF SMA and (AC lss 0) => skip = 1
                        end,
                1 :=    begin
                        skip = 1 next
                        IF SZL and not (L eql{US} '0) => skip = 0;
                        IF SNA and not (AC neq 0) => skip = 0;
                        IF SPA and not (AC Geq 0) => skip = 0
                        end
                end next
        IF skip => PC = PC + 1                  ! Skip
        end,

operate :=
   begin
        DECODE group =>
           begin
                0 := begin                      ! group 1
                        IF CLA => AC = 0 next
                        IF CLL => L = 0 next
                        IF CMA => AC = not AC next
                        IF CML => L = not L next
                        IF IAC => lac = lac + 1 next
                        DECODE ROT =>                   ! rotate group
                           begin
                                #0      := no.op(),
                                #1\BSW := AC<6:11>@AC<0:5> = AC,
                                #2\RAL := lac = lac slr 1,
                                #3\RTL := lac = lac slr 2,
                                #4\RAR := lac = lac srr 1,
                                #5\RTR := lac = lac srr 2,
                                #6      := undefined(),
                                #7      := undefined()
                           end
                     end,
                1 := begin              ! groups 2 and 3
                        DECODE i<11> =>
                           begin
                             0 := begin     ! group 2
                                        skip.group() next
                                        IF CLA => AC = 0 next
                                        IF OSR => AC = AC or switches next
                                        IF HLT => STOP()
                                  end,
```

```
                                    1 := begin      ! group 3
                                           IF CLA => AC = 0 next
                                           DECODE i<5:7> =>
                                                 begin
                                                 0       := no.op(),
                                                 1\MQL   := MQ = AC,
                                                 2\SCA   := no.op(),
                                                 3       := no.op(),
                                                 4\MQA   := AC = AC or MQ,
                                                 5       := MQ@AC = AC@MQ,
                                                 6       := no.op(),
                                                 7       := no.op()
                                                 end
                                           end
                                    end
                              end
                        end
                  end
            end
```

2.2 Generación de l base de datos

$gendb pdp8
$ ISPS PDP8
ISPS V6(12)-MI
$gdbrtm PDP8
GDB to RTM translator V10.5.5
GDB:I;ISPS V6(12)-MI;PDP8.ISP;27-FEB-1991 00:18:18.68;
GETGDB done, generating RTM code...
Output will go to RTM.MAR

$ CONVERT PDP8

Nombre del fichero de salida:pdp8.mar

Borro fichero RTM.MAR
$ MACRO/NOLIST PDP8
$ LINK/NOMAP/EXE:PDP8 PDP8,usuarios:[espeso.grafo]gendb,
usuarios:[espeso.grafo]gendb2/library
$ DEL/NOLOG PDP8.obj;,PDP8.mar;,PDP8.gdb;
$ Set NoVerify

La base de datos para el sistema PSAL, sera generada por

el fichero ejecutable PDP8.EXE

```
PSAL - GENERACION DE LA BASE DE DATOS

Desea guardar los mensajes en un fichero?n

CREANDO EL NUEVO GRAFO

SUSTITUCION DE MARCAS POR VALORES

Estadistica : entidades 8
              nombres 56
              simbolos 96
              sentencia 143


PRIMER PASO DE LA OPTIMIZACION
Elimino entidad SKIP.GROUP
Elimino entidad OPERATE
Elimino entidad INPUT.OUTPUT
Elimino entidad EXECUTE
Elimino entidad INTERPRET
Elimino entidad PRELUDE

Estadistica : entidades 2
              nombres 56
              simbolos 96
              sentencia 124


SEGUNDO PASO DE LA OPTIMIZACION

Estadistica : entidades 2
              nombres 56
              simbolos 96
              sentencia 121
```

TERCER PASO DE LA OPTIMIZACION

Han sido asociados 2 nudos

Estadistica : entidades 2
nombres 56
simbolos 96
sentencia 119

Eliminando sinonimos

Han sido eliminados 22 sinonimos
Han sido eliminados 9 simbolos repetidos

SELECCION DE VARIABLES << RETURN >>

OPCIONES
0 Salir del menu
1 Listar variables
2 Listar variables asignadas a conexiones externas.
3 Listar variables no agrupables
4 Seleccionar variable asignada a conexion
5 Seleccionar variable no agrupable

Opcion ? 4
Nombre de la variable a seleccionar
interrupt.state

Seleccionada la variable INTERRUPT.STATE

Opcion ? 4
Nombre de la variable a seleccionar
interrupt.request

Seleccionada la variable INTERRUPT.REQUEST

Opcion ? **4**
Nombre de la variable a seleccionar
switches

Seleccionada la variable SWITCHES

Opcion ? **4**
Nombre de la variable a seleccionar
skip

Seleccionada la variable SKIP

Opcion ? **2**
Lista de variables asignadas a conexiones externas
1 INTERRUPT.STATE
2 SKIP
3 SWITCHES
4 INTERRUPT.REQUEST

Opcion ? **0**
GUARDO GRAFO EN FICHERO:PDP8.grf

-- ESCRIBO LA DB EN EL FICHERO:PDP8 --

ESTADISTICAS

Numero de entidades	: 2
Numero de sentencias	: 119
Numero de simbolos	: 65
Numero de nombres	: 34
Numero de operaciones	: 108
Numero de st. otro	: 91
Numero de st. sucesor	: 81
Numero de letras	: 186
Numero de digitos es	.: 0

2.3 Síntesis del pdp8

Una vez obtenida la base de datos, se procedio a la síntesis del sistema.

$ psal

Nombre de la Base de Datos: **pdp8**
INICIO LA LECTURA DE LA BASE DE DATOS

INICIO LA RECOMPOSICION DEL GRAFO

ESTADISTICAS

Numero de entidades	: 2
Numero de sentencias	: 119
Numero de simbolos	: 65
Numero de nombres	: 34
Numero de operaciones	: 108
Numero de st. otro	: 91
Numero de st. sucesor	: 81
Numero de letras	: 186
Numero de digitos es	.: 0
Numero de var. loc.	: 18
Numero de var. glo.	: 15

INICIO EL ANALISIS DEL FLUJO DE DATOS

Analisis de bloques basicos

Numero de bytes necesarios en el analisis: 4
Operacion redundante.Rtm=55
Eliminada operacion redundante en bloque 10
Eliminada operacion redundante en bloque 19
Operacion redundante.Rtm=160
Eliminada operacion redundante en bloque 50
Eliminada operacion redundante en bloque 90
Eliminada operacion redundante en bloque 113

Han sido eliminadas 5 operaciones

Inicio el analisis global

NUMERO DE ITERACIONES 33

Desea escribir los vectores de analisis de datos en un fichero ? (s/N)**n**
Defino operaciones encadenadas
La variable T85 no es localizable
La variable CHAIN2 no es localizable
La variable CHAIN3 no es localizable
La variable CHAIN4 no es localizable
La variable CHAIN5 no es localizable
La variable T104 no es localizable
La variable T109 no es localizable
La variable T80 no es localizable
La variable T74 no es localizable
Finalizada la busqueda de operaciones encadenadas
INICIO LA COMPACTACION DE OPERACIONES

INICIO LA PRE-ASIGNACION OPERACION-ESTADO

0 Respetar el orden de las operaciones.
1 Asignacion sin limite de recursos hardware
2 Asignacion con recursos limitados

Opcion? **1**
En la sentencia 4,se ha pasado de 3 a 2 estados
En la sentencia 10,se ha pasado de 5 a 4 estados
En la sentencia 14,se ha pasado de 2 a 1 estados
En la sentencia 16,se ha pasado de 2 a 1 estados
En la sentencia 42,se ha pasado de 4 a 3 estados
En la sentencia 52,se ha pasado de 2 a 1 estados
En la sentencia 63,se ha pasado de 3 a 2 estados
En la sentencia 96,se ha pasado de 3 a 2 estados
En la sentencia 104,se ha pasado de 2 a 1 estados

Numero de OUs: 2
Numero maximo :2

FIN DE LA COMPACTACION
INICIO LA GENERACION DE LA TABLA DE EXCLUSIONES
La variable T72 no se utiliza
La variable T89 no se utiliza
La variable T126 no se utiliza
La variable T133 no se utiliza
FINALIZADA LA GENERCION DE LA TABLA DE EXCLUSIONES
Escribo en un fichero dicha tabla ?(s/N)

INICIO LA SINTESIS DE VARIABLES

FIN DEL PROCESO DE SINTESIS
Resumen: 20 variables han sido agrupadas en 7 registros

Como minimo se necesitaban 7

El sistema necesita 12 registros

Inicio el preanalisis
Fin del preanalisis

INICIO LA ASIGNACION OPERACION-ALU

OBTENIDA LA TABLA OPERACION-ALU

Se necesitan 2 alus
INICIO LA SINTESIS DE OPERACIONES

Describo la asignacion realizada en el fichero pdp8.ous

Informacion completada

FIN DE LA SINTESIS

INICIO LA COMPACTACION DE OPERACIONES
COMPACTO BLOQUES

```
FIN DE LA COMPACTACION
INICIO LA ESCRITURA DE LA DESCRIPCION DDL

ESCRITURA FINALIZADA
INICIO LA ESCRITURA DE LA DESCRIPCION

FIN DE LA ESCRITURA DE LA DESCRIPCION

GUARDO GRAFO EN FICHERO:pdp8.grf
```

2.4 Resultados de la síntesis

La descripción resultante del proceo de síntesis aparece a continuación. Es importante observar las transferencias a don't care (-> ?), producidas por las instrucciones "undefined" de ISPS.

```
<SY> pdp8:  _STOP:

<RE> I[12] ,
     PC[12] ,
     CPAGE[13] ,
     EADD[12] ,
     T75[24] ,
     LAC[13] ,
     MQ[12],
     _STOP[1],
     _EADD[3].

<TE> INTERRUPT.STATE[1] ,
     SKIP[1] ,
     SWITCHES[12] ,
     INTERRUPT.REQUEST[1] ,
     _M[12].

<ME> M[4096:12].

<TI> clk_pdp8.

<AU> CPU_pdp8:clk_pdp8:

<ST>

   S0:
     PC <- INC(PC),
     CPAGE[4 : 0] <-PC[11 : 7],
     _M=PC,I<-M[_M],->S1 .
```

```
S1:
  ! I[11 : 9]
 # 0   _EADD <- 0D3 ,->S32
 # 1   _EADD <- 1D3 ,->S32
 # 2   _EADD <- 2D3 ,->S32
 # 3   _EADD <- 3D3 ,->S32
 # 4   _EADD <- 4D3 ,->S32
 # 5   _EADD <- 5D3 ,->S32
 # 6   ! I[8 : 0]
 # 1   INTERRUPT.STATE = 1D1 ,->S0
 # 2   INTERRUPT.STATE = 0D1,->S30
   ;   ->S30 .
 # 7   ! I[8]
 # 0   | I[7] | LAC[11 : 0] <- 0D12.,
  | I[6] | LAC[12] <- 0D1.,->S14
 # 1   ! I[0]
 # 0   ! I[3]
 # 0   CPAGE[0] <-~+/(LAC[12] @ 1D1 ),
   EADD[0] <- OU1(LAC[11 : 0], 0D1 , <<< lss >>> ),
   T75[0] <- OU0(LAC[11 : 0], 0D1 , <<< eql >>> ),
   SKIP = 0D1,->S20
 # 1   CPAGE[0] <-~+/(LAC[12] @ 0D1 ),
   T75[0] <- OU1(LAC[11 : 0], 0D1 , <<< geq >>> ),
   EADD[0] <- OU0(LAC[11 : 0], 0D1 , <<< neq >>> ),
   SKIP = 1D1 ,->S21 .
 # 1   | I[7] | LAC[11 : 0] <- 0D12.,->S25 . . ..
S2:
   _M=EADD,T75[11 : 0]<-M[_M],->S3 .
S3:
   LAC[11 : 0] <-LAC[11 : 0] * T75[11 : 0],->S30.
S4:
   _M=EADD,T75[11 : 0]<-M[_M],->S5 .
S5:
   CPAGE <- 0D13 ' T75[11 : 0],->S6 .
S6:
   LAC <- OU1(LAC, CPAGE, <<< + >>> ),->S30.
S7:
   _M=EADD,T75[11 : 0]<-M[_M],->S8 .
S8:
   CPAGE <- OU1(T75[11 : 0], 1D13 , <<< + >>> ),->S9 .
S9:
   _M=EADD,M[_M] <-CPAGE,
   | +/(T75[11 : 0] @ 0D1 ) | PC <- INC(PC).,->S30.
S10:
   LAC[11 : 0] <- 0D12,
   _M=EADD,M[_M] <-LAC[11 : 0],->S30.
S11:
   PC <- OU1(EADD, 1D12 , <<< + >>> ),
   _M=EADD,M[_M] <-PC,->S30.
S12:
   PC <-EADD,->S30.
S14:
   | I[5] | LAC[11 : 0] <- ~LAC[11 : 0].,
   | I[4] | LAC[12] <- ~LAC[12].,->S15.
S15:
   | I[0] | LAC <- INC(LAC).,->S16.
```

```
S16:
  ! I[3 : 1]
 # 0   ->S30
 # 1   T75[11 : 0] <-LAC[11 : 0],->S17
 # 2   LAC <- OU1(LAC,  1D13 , <<<  slr  >>> ),->S30
 # 3   LAC <- OU1(LAC,  2D13 , <<<  slr  >>> ),->S30
 # 4   LAC <- OU1(LAC,  1D13 , <<<  srr  >>> ),->S30
 # 5   LAC <- OU1(LAC,  2D13 , <<<  srr  >>> ),->S30
 # 6   -> ?
 # 7   -> ? ..
S17:
  T75[11 : 0] <-T75[11 : 6],
  LAC[11 : 6] <-T75[11 : 0],->S18 .
S18:
  LAC[5 : 0] <-T75[11 : 0],->S30.

S20:
  | I[4] * CPAGE[0] | SKIP =  1D1 .,
  | I[5] * T75[0] | SKIP =  1D1 .,
  | I[6] * EADD[0] | SKIP =  1D1 .,->S23 .
S21:
  T75[0] <- ~T75[0],
  EADD[0] <- ~EADD[0],
  CPAGE[0] <- ~CPAGE[0],->S22 .
S22:
  | I[4] * CPAGE[0] | SKIP =  0D1.,
  | I[5] * EADD[0] | SKIP =  0D1.,
  | I[6] * T75[0] | SKIP =  0D1.,->S23 .
S23:
  | SKIP | PC <- INC(PC).,
  | I[7] | LAC[11 : 0] <- 0D12.,->S24.
S24:
  | I[2] | LAC[11 : 0] <-LAC[11 : 0] + SWITCHES.,
  | I[1] | _STOP <- 0D1, ->S0;->S30..
S25:
  ! I[6 : 4]
 # 0   ->S30
 # 1   MQ <-LAC[11 : 0],->S30
 # 2   ->S30
 # 3   ->S30
 # 4   LAC[11 : 0] <-LAC[11 : 0] + MQ,->S30
 # 5   T75 <-LAC[11 : 0] ' MQ,->S26
 # 6   ->S30
 # 7   ->S30 ..
S26:
  MQ <-T75[23 : 12],
  LAC[11 : 0] <-T75,->S30.
S30:
  | INTERRUPT.STATE * INTERRUPT.REQUEST | PC <- 1D12 ,
    _M= 0D12 ,M[_M] <-PC.,->S0.
S32:
  ! I[7]
 # 0   EADD <- 0D12  ' I[6 : 0],->S33
 # 1   EADD <-CPAGE[4 : 0] ' I[6 : 0],->S33 ..
S33:
  | I[8] | | +/(EADD[11 : 3] @  1D1 ) | _M=EADD,T75[11 : 0]<-M[_M],->S34. ..
```

```
S34:
  T75[12 : 0] <- INC(T75[11 : 0]),->S35 .
S35:
  _M=EADD,M[_M] <-T75[12 : 0],->S36.
S36:
  _M=EADD,EADD<-M[_M],! _EADD
   # 0  ->S2
   # 1  ->S4
   # 2  ->S7
   # 3  ->S10
   # 4  ->S11
   # 5  ->S12.....
```

Unidades operacionales y conexiones realizadas por el sistema PSAL1

ASIGNACIONES OPERACION-ALU REALIZADAS POR PSAL

Modulo : pdp8
Numero de ALUs :2

ALU 1

operacion	bits
eql	12
neq	12

ALU 2

operacion	bits
slr	13
srr	13
+	13
lss	12
geq	12

INTERCONEXIONES DEFINIDAS

formato 123... 1 salida de la alu,2 input 1,3 input 2
valores 0 .. conectado,1 .. desconectado,x .. optativo

ALUs ----> 0 ... Maxalu
000 100 PC
000 101 CPAGE
100 110 EADD
100 110 T75
010 110 LAC
001 001 CONSTANTES

3) Síntesis del PDP8 utilizando el sistema PSAL2

3.1) Descripción VHDL del PDP8

A partir de la descripción ISPS del PDP8 presentada en el apartado 2.1 de este mismo capítulo se propuso la siguiente representación para dicha descripción en VHDL:

```
entity PDP8 is

 port (  interrupt_state    :       in      BIT;
         interrupt_request  :       in      BIT;
         switches           :       in      BIT_VECTOR(0 to 11);
         psal_reset         :       out     BIT     -- Interno en PSAL2
      );
end;


--      PDP8 VHDL description based on the original ISPS description by
--      Mario Barbacci and Larry Lai

architecture ALGORITHM of PDP8 is
begin
        process
                subtype word    is BIT_VECTOR(0 to 11);
                type memory is array (POSITIVE range <>) of word;
                variable M      : memory(0 to 4095);            -- Memory
                variable PC     : word;                         -- Program.Counter
                variable cpage  : BIT_VECTOR(0 to 4);           -- current_page
                variable lac    : BIT_VECTOR(0 to 12);
```

```
        alias L             : BIT is lac(0);    -- Link
        alias AC: word is lac(1 to 12);         -- Accumulator
variable MQ      : word;                        -- Multiplier.Quotient_Register
variable i       : word;                        -- instruction
        alias op : BIT_VECTOR(0 to 2) is i(0 to 2);-- operation.code
        alias ib : BIT is i(3);                 -- indirect_bit
        alias pb : BIT is i(4);                 -- page.0.bit
        alias pa : BIT_VECTOR(0 to 6) is i(5 to 11);-- page.address
        alias io_select    : BIT_VECTOR(0 to 5) is i(3 to 8);-- device select
        alias io_control   : BIT_VECTOR(0 to 2) is i(9 to 11);-- device operation
                alias io_pulse_p1 : BIT is i(9);
                alias io_pulse_p2 : BIT is i(10);
                alias io_pulse_p4 : BIT is i(11);
        alias group      : BIT is i(3);             -- microinstruction group
        alias    CLA     : BIT is i(4);             -- clear AC (all groups)
-- group 1 operations (group bit = 0)
        alias    CLL     : BIT is i(5);                     -- clear L
        alias    CMA     : BIT is i(6);                     -- complement AC
        alias    CML     : BIT is i(7);                     -- complement L
        alias    ROT     : BIT_VECTOR(0 to 2) is i(8 to 10);-- rotate group
        alias    IAC     : BIT is i(11);                    -- increment AC
-- group 2 operations (group bit = 1 and bit 11 = 0)
        alias    SMA     : BIT is i(5);     -- skip on minus AC (is bit = 0)
        alias    SZA     : BIT is i(6);     -- skip on zero AC (is bit = 0)
        alias    SNL     : BIT is i(7);     -- skip on L not zero (is bit = 0)
        alias    SPA     : BIT is i(5);     -- skip on positive AC (is bit = 1)
        alias    SNA     : BIT is i(6);     -- skip on AC not zero (is bit = 1)
        alias    SZL     : BIT is i(7);     -- skip on L zero (is bit = 1)
        alias    ISS     : BIT is i(8);     -- invert skip sense
        alias    OSR     : BIT is i(9);     -- logical or AC with switches
        alias    HLT     : BIT is i(10);    -- halt the processor
        constant cero    : word := 0;
        constant uno     : word := 1;

function eadd return word is
        variable eadd    : word;
        variable e_tmp   : word;
begin
        if pb then
                eadd := B"00000" & pa;
          else
                eadd := cpage & pa;
        end if;
        if ib then
                if eadd(0 to 8) = O"001" then
                        e_tmp := M(eadd) + uno;
                        M(eadd):= e_tmp;
                        eadd:=e_tmp;
                end if;
        end if;
        return eadd;
end;
```

```
procedure input_output is
        begin
                case i(3 to 11) is
                        when O"001" =>          -- ION
                        interrupt_state := 1;
                        when O"002" =>          -- IOF
                        interrupt_state := 0;
                        when others =>          -- not implemented
                        null;
                end case;
        end;


procedure skip_group is
        variable skip       : BIT;
        begin
                if ISS then
                        if SZL and not (L = 0) then skip := 0;
                        elsif
                         SNA and not (AC /= cero) then skip := 0;
                        elsif
                         SPA and not (AC >= cero) then skip := 0;
                        else
                         skip:=1;
                        end if;
                else
                        if SNL and (L = 1) then skip := 1;
                        elsif
                         SZA and (AC = cero) then skip := 1;
                        elsif
                         SMA and (AC < cero) then skip := 1;
                        else
                          skip:=0;
                        end if;

                end if;
                if skip then PC := PC + uno;        -- Skip
                end if;
          end;


 procedure operate is
          variable masc : BIT_VECTOR(0 to 12);
          begin
                  if group then                          -- groups 2 and 3
                          if i(11) then                  -- group 3
                                  if CLA then lac := L & cero;
                                  end if;
                                  case i(5 to 7) is
                                          when 0 => null;
                                          when 1 => MQ := AC;-- MQL
```

```
                                        when 2 => null;  -- SCA
                                        when 3 => null;
                                        when 4 => AC := AC or MQ; -- MQA
                                        when 5 => MQ := AC;

                                        when 6 => null;
                                        when 7 => null;
                                end case;
                        else                    -- group 2
                                skip_group;
                                if CLA then AC := cero;
                                end if;
                                if OSR then AC := AC or switches;
                                end if;
                                if HLT then psal_reset <= 1;
                                end if;
                        end if;
                else                            -- group 1
                        if CLA then  masc:= (not CLL)& cero;
                                else masc:= (not CLL)& uno;
                        end if;
                        lac:=lac and masc;
                        if CML  then masc:= CMA & uno;
                                else masc := CMA & cero;
                        end if;
                        lac:= masc xor lac;
                        if IAC then lac := lac + uno;
                        end if;
                        case ROT is       -- rotate group
                                when 0 => null;
                                when 1 => lac := L & AC(6 to 11) & AC(0 to 5);
                                when 2 => lac := lac(1 to 12) & lac(0);-- RAL
                                when 3 => lac := lac(2 to 12) & lac(0 to 1);-- RTL
                                when 4 => lac := lac(12) & lac(0 to 11);-- RAR
                                when 5 => lac := lac(11 to 12) & lac(0 to 10);
                                                                        -- RTR
                        --      when 6 => unpredictable;
                        --      when 7 => unpredictable;
                        end case;
                end if;
        end;

-- Instruction.Set
procedure execute is
        variable eadd_temp1     : word;
        variable eadd_temp2     : word;
        begin
                case op is
                        when 0 =>                       -- AND
                                AC := AC and M(eadd( ));
                        when 1 =>                       -- TAD
                                lac := lac + M(eadd());
```

```
                                        when 2 =>                       -- ISZ
                                                eadd_temp1 := eadd();
                                                eadd_temp2 := M(eadd_temp1) + 1;
                                                M(eadd_temp1) := eadd_temp2;
                                                if eadd_temp2 = cero then
                                                        PC := PC + uno;
                                                end if;
                                        when 3 =>                       -- DCA
                                                M(eadd()) := AC;
                                                AC := cero;
                                        when 4 =>                       -- JMS
                                                eadd_temp1:=eadd();
                                                M(eadd_temp1) := PC;
                                                PC := eadd_temp1 + uno;
                                        when 5 =>                       -- JMP
                                                PC := eadd();
                                        when 6 =>                       -- iot
                                                input_output;
                                        when 7 =>                       -- opr
                                                operate;
                                end case;
                        end;

                begin
                        i := M(PC);
                        cpage := PC(0 to 4);
                        PC := PC + uno;
                        execute;
                        if interrupt_state and interrupt_request then
                                M(cero) := PC;
                                PC := uno;
                        end if;
                end process;
end;
```

3.2) Ejecución de PSAL2

% psal2

Is your Terminal VT-100 compatible [Y/n] ?**n**

PSAL 2
High Level Synthesis Program
1989

Psal>set file pdp8

```
Psal>set listing on
Psal>set input vhdl
Psal>load

 Input file  : pdp8.vhdl
 Listing file : pdp8.lis

 Steps 1-2: Compiling vhdl -> psal2
 Step 3: Compiling psal2 -> Data base

 Done !
Psal>data

 Inicio el analisis del flujo de datos

 INLINE : funcion input_output
 INLINE : funcion skip_group
 INLINE : funcion operate
 INLINE : funcion execute

 Inicio analisis global

 Numero de iteraciones :3

 Fin del analisis del flujo de datos
Psal>scheduling
Tipo de modulo de los puertos de entrada del sistema          terminal
Tipo de modulo de los puertos de salida del sistema           terminal
Tipo de modulo de los puertos de entrada-salida del sistema   terminal
Tipo de modulo de los puertos de datos de las memorias        terminal
Tipo de modulo de los puertos de direcciones del sistema      terminal

 PSAL2 SCHEDULING SHELL

        Opciones:
            1 Force Directed Scheduling
            2 List scheduling Force Directed Scheduling
            3 List scheduling Force Directed Scheduling (restricted)
            4 Modificado
```

scheduling/Psal>2

Running schedule algorithm

IDENTIFICADO ENCADENAMIENTO: variable lac, state=9,d=1,u=3

IDENTIFICADO TERMINAL. Variable: TER_9_2_lac
IDENTIFICADO ENCADENAMIENTO: variable lac, state=9,d=1,u=1

IDENTIFICADO TERMINAL. Variable: TER_9_6_lac
IDENTIFICADO ENCADENAMIENTO: variable lac, state=9,d=1,u=1

IDENTIFICADO TERMINAL. Variable: TER_9_8_lac
IDENTIFICADO ENCADENAMIENTO: variable i, state=9,d=1,u=24

IDENTIFICADO TERMINAL. Variable: TER_9_1_i
IDENTIFICADO TERMINAL. Variable: psal_vtmp_24
IDENTIFICADO ENCADENAMIENTO: variable eadd_temp1, state=12,d=1,u=1

IDENTIFICADO TERMINAL. Variable: TER_12_1_eadd_temp1
IDENTIFICADO ENCADENAMIENTO: variable eadd_temp1, state=15,d=1,u=2

IDENTIFICADO TERMINAL. Variable: TER_15_1_eadd_temp1
IDENTIFICADO TERMINAL. Variable: psal_vtmp_20
IDENTIFICADO TERMINAL. Variable: psal_vtmp_21
IDENTIFICADO TERMINAL. Variable: psal_vtmp_22
IDENTIFICADO TERMINAL. Variable: psal_vtmp_23
IDENTIFICADO TERMINAL. Variable: masc
IDENTIFICADO TERMINAL. Variable: psal_vtmp_17
IDENTIFICADO TERMINAL. Variable: psal_vtmp_18
IDENTIFICADO TERMINAL. Variable: psal_vtmp_19
IDENTIFICADO TERMINAL. Variable: skip
IDENTIFICADO TERMINAL. Variable: psal_vtmp_2
IDENTIFICADO TERMINAL. Variable: psal_vtmp_3
IDENTIFICADO TERMINAL. Variable: psal_vtmp_4
IDENTIFICADO TERMINAL. Variable: psal_vtmp_5
IDENTIFICADO TERMINAL. Variable: psal_vtmp_6
IDENTIFICADO TERMINAL. Variable: psal_vtmp_7
IDENTIFICADO TERMINAL. Variable: psal_vtmp_8

IDENTIFICADO TERMINAL. Variable: psal_vtmp_9
IDENTIFICADO TERMINAL. Variable: psal_vtmp_10
IDENTIFICADO TERMINAL. Variable: psal_vtmp_11
IDENTIFICADO TERMINAL. Variable: psal_vtmp_12
IDENTIFICADO TERMINAL. Variable: psal_vtmp_13
IDENTIFICADO TERMINAL. Variable: psal_vtmp_14
IDENTIFICADO TERMINAL. Variable: psal_vtmp_15
IDENTIFICADO TERMINAL. Variable: psal_vtmp_16
IDENTIFICADO ENCADENAMIENTO: variable eadd, state=3,d=2,u=2

IDENTIFICADO TERMINAL. Variable: TER_3_4_eadd
IDENTIFICADO TERMINAL. Variable: psal_vtmp_0
IDENTIFICADO TERMINAL. Variable: psal_vtmp_1

Inicio el analisis del flujo de datos

Inicio analisis global

Numero de iteraciones :3

Fin del analisis del flujo de datos

Done !!

Psal>**allocation**

PSAL2 HARDWARE ALLOCATION SHELL

Elige algoritmo:
    0 global
    1 probabilistico
Opcion? 0

Running variable synthesis algorithm

Escribo tabla de vida en fichero pdp8.tlive
Done !

INICIO LA SINTESIS DE VARIABLES

FIN DEL PROCESO DE SINTESIS
Resumen: 17 variables han sido agrupadas en 9 registros

Como minimo se necesitaban 9

Inicio el preanalisis
Fin del preanalisis

INICIO LA ASIGNACION OPERACION-ALU

OBTENIDA LA TABLA OPERACION-ALU

Tabla de exclusiones entre operaciones

Se necesitan 2 alus
INICIO LA SINTESIS DE OPERACIONES

FIN DE LA SINTESIS

Done !!

Psal>**write**

La descripcion sera almacenada en el fichero pdp8_s.psal2

Done !
Psal>**set out vhdl**
Psal>**write**

La descripcion sera almacenada en el fichero pdp8_s.vhdl

-- DESCRIPCION VHDL CONCLUIDA --

Done !
Psal>**set out cvs**
Psal>**write**

La descripcion sera almacenada en el fichero pdp8.cvs

-- DESCRIPCION CVS_BK CONCLUIDA --

Done !
Psal>**set out esp**
Psal>**write**

-- DESCRIPCION ESP CONCLUIDA --

Done !
Psal>**set out dbm**
Psal>**write**

-- Almaceno la base de datos en el fichero pdp8_ver3.XXX

Done !

Psal>**exit**

*** EJECUCION DEL PROGRAMA PSAL2 TERMINADA *****

ADIOS !

3.4 Descripción del sistema resultante en VHDL

```
entity pdp8 is
        port
        (
        psal_vhdl_reset_terminal   :in bit;
        switches                   :in bit_vector(11 downto 0);
        interrupt_state            :in bit;
        interrupt_request          :in bit;
        psal_reset                 :out bit;
        clock                      :in bit );
        end pdp8;
```

```
architecture RTL of pdp8

        type psram_0 is array (4095 downto 0) of bit_vector(11 downto 0);
        constant psal_const_1:bit_vector(8 downto 0):=1;
        constant psal_const_18:bit_vector(11 downto 0):=0;
        constant psal_const_17:bit_vector(11 downto 0):=1;
        constant psal_const_16:bit:=1;
        constant psal_const_15:bit:=1;
        constant psal_const_14:bit:=0;
        constant psal_const_13:bit:=0;
        constant psal_const_12:bit:=0;
        constant psal_const_11:bit:=0;
        constant psal_const_10:bit:=1;
        constant psal_const_9:bit:=1;
        constant psal_const_8:bit:=1;
        constant psal_const_7:bit:=1;
        constant psal_const_6:bit:=1;
        constant psal_const_5:bit:=0;
        constant psal_const_4:bit:=1;
        constant psal_const_3:bit:=0;
        constant psal_const_2:bit_vector(4 downto 0):=0;
        signal psal_vhdl_reset:bit;
        signal eadd:bit_vector(11 downto 0);
        signal mq:bit_vector(11 downto 0);
        signal lac:bit_vector(12 downto 0);
        signal pc:bit_vector(11 downto 0);
        signal e_tmp:bit_vector(11 downto 0);
        signal re_eadd:bit_vector(2 downto 0);

        type statevalue is (
                        pdp8_state_9,
                        pdp8_state_10,
                        pdp8_state_11,
                        pdp8_state_12,
                        pdp8_state_13,
                        pdp8_state_14,
                        pdp8_state_15,
                        pdp8_state_16,
                        pdp8_state_17,
                        pdp8_state_18,
                        pdp8_state_20,
                        pdp8_state_27,
                        eadd_state_3,
                        eadd_state_4,
                        eadd_state_6) ;
        signal reg_state : statevalue ;


        BEGIN
```

```
PROCESS

variable psal_vtmp_1:bit_vector(11 downto 0);
variable psal_vtmp_0:bit;
variable TER_3_4_eadd:bit_vector(11 downto 0);
variable psal_vtmp_16:bit;
variable psal_vtmp_15:bit;
variable psal_vtmp_14:bit;
variable psal_vtmp_13:bit;
variable psal_vtmp_12:bit;
variable psal_vtmp_11:bit;
variable psal_vtmp_10:bit;
variable psal_vtmp_9:bit;
variable psal_vtmp_8:bit;
variable psal_vtmp_7:bit;
variable psal_vtmp_6:bit;
variable psal_vtmp_5:bit;
variable psal_vtmp_4:bit;
variable psal_vtmp_3:bit;
variable psal_vtmp_2:bit;
variable skip:bit;
variable psal_vtmp_19:bit_vector(6 downto 0);
variable psal_vtmp_18:bit;
variable psal_vtmp_17:bit;
variable masc:bit_vector(12 downto 0);
variable psal_vtmp_23:bit;
variable psal_vtmp_22:bit_vector(11 downto 0);
variable psal_vtmp_21:bit_vector(11 downto 0);
variable psal_vtmp_20:bit_vector(11 downto 0);
variable TER_15_1_eadd_temp1:bit_vector(11 downto 0);
variable TER_12_1_eadd_temp1:bit_vector(11 downto 0);
variable psal_vtmp_24:bit;
variable TER_9_1_i:bit_vector(11 downto 0);
variable TER_9_8_lac:bit_vector(12 downto 0);
variable TER_9_6_lac:bit_vector(12 downto 0);
variable TER_9_2_lac:bit_vector(12 downto 0);
variable psal_address_m:bit_vector(11 downto 0);
        variable m:psram_0;
BEGIN

IF  (psal_vhdl_reset OR psal_vhdl_reset_terminal )THEN
        IF(psal_vhdl_reset AND psal_vhdl_reset_terminal)THEN
        psal_vhdl_reset<='0';

END IF;
reg_state<= pdp8_state_9;
        ELSE
CASE reg_state IS
```

```
WHEN eadd_state_3 =>

            IF( not ( e_tmp(7)=BITVAL(0))) then
        TER_3_4_eadd :=psal_const_2 &  e_tmp(6 downto 0);
        ELSE
        TER_3_4_eadd := eadd(4 downto 0) &  e_tmp(6 downto 0);
            END IF;
         eadd <=  TER_3_4_eadd;
            IF( not ( e_tmp(8)=BITVAL(0))) then
        psal_vtmp_0 :=EQL( TER_3_4_eadd(11 downto 3),psal_const_1);
            IF( not ( psal_vtmp_0=BITVAL(0))) then
        psal_address_m := TER_3_4_eadd;
        psal_vtmp_1 :=m( psal_address_m);
         e_tmp <= ALU_1(SUM, psal_vtmp_1,psal_const_17);
             reg_state <= eadd_state_4;
        ELSE
        NULL;
             reg_state <= eadd_state_6;

            END IF;
        ELSE
        NULL;
             reg_state <= eadd_state_6;

            END IF;
WHEN eadd_state_4 =>

        psal_address_m := eadd;
         m( psal_address_m ) :=  e_tmp;

         eadd <=  e_tmp;
            reg_state <= eadd_state_6;
WHEN eadd_state_6 =>

         e_tmp <=  eadd;
            CASE re_eadd is
        when 0 =>
        reg_state<= pdp8_state_10;
        when 1 =>
        reg_state<= pdp8_state_11;
        when 2 =>
        reg_state<= pdp8_state_12;
        when 3 =>
        reg_state<= pdp8_state_14;
        when 4 =>
        reg_state<= pdp8_state_15;
        when 5 =>
        reg_state<= pdp8_state_16;
WHEN pdp8_state_9 =>

         psal_address_m := pc;
         TER_9_1_i :=m( psal_address_m);
          eadd <= eadd(11 downto 5) & ( pc(11 downto 7));
```

```
pc <= ALU_0(SUM, pc,psal_const_17);
e_tmp <= TER_9_1_i;
        CASE TER_9_1_i(11 downto 9) IS
when 0 =>
         re_eadd (2 downto 0)<=bit_val(0);
         reg_state <= eadd_state_3;
when 1 =>
         re_eadd (2 downto 0)<=bit_val(1);
         reg_state <= eadd_state_3;
when 2 =>
         re_eadd (2 downto 0)<=bit_val(2);
         reg_state <= eadd_state_3;
when 3 =>
         re_eadd (2 downto 0)<=bit_val(3);
         reg_state <= eadd_state_3;
when 4 =>
         re_eadd (2 downto 0)<=bit_val(4);
         reg_state <= eadd_state_3;
when 5 =>
         re_eadd (2 downto 0)<=bit_val(5);
         reg_state <= eadd_state_3;
when 6 =>
         CASE TER_9_1_i(8 downto 0) IS
when 1 =>
interrupt_state :=psal_const_4;
when 2 =>
interrupt_state :=psal_const_3;
          when others =>
NULL;
END CASE;
          reg_state <= pdp8_state_27;

when 7 =>
          IF( not ( TER_9_1_i(8)=BITVAL(0))) then
          IF( not ( TER_9_1_i(0)=BITVAL(0))) then
          IF( not ( TER_9_1_i(7)=BITVAL(0))) then
TER_9_2_lac := lac(12) & psal_const_18;
ELSE
TER_9_2_lac := lac;
          END IF;
          CASE TER_9_1_i(6 downto 4) IS
when 0 =>
 lac <= TER_9_2_lac;
when 1 =>
 mq <= TER_9_2_lac(11 downto 0);
 lac <= TER_9_2_lac;
when 2 =>
 lac <= TER_9_2_lac;
when 3 =>
 lac <= TER_9_2_lac;
when 4 =>
 lac <= lac(12) & (AND( TER_9_2_lac(11 downto 0), mq));
when 5 =>
```

```
mq <= TER_9_2_lac(11 downto 0);
lac <= TER_9_2_lac;
when 6 =>
lac <= TER_9_2_lac;
when 7 =>
lac <= TER_9_2_lac;
END CASE;
        reg_state <= pdp8_state_27;

ELSE
        IF( not ( TER_9_1_i(3)=BITVAL(0))) then
psal_vtmp_2 :=EQL( lac(12),psal_const_14);
psal_vtmp_3 :=NOT( psal_vtmp_2;
psal_vtmp_4 :=AND( TER_9_1_i(4), psal_vtmp_3);
        IF( not ( psal_vtmp_4=BITVAL(0))) then
skip :=psal_const_13;
ELSE
psal_vtmp_5 :=NEQ( lac(11 downto 0),psal_const_18);
psal_vtmp_6 :=NOT( psal_vtmp_5;
psal_vtmp_7 :=AND( TER_9_1_i(5), psal_vtmp_6);
        IF( not ( psal_vtmp_7=BITVAL(0))) then
skip :=psal_const_12;
ELSE
psal_vtmp_8 :=GEQ( lac(11 downto 0),psal_const_18);
psal_vtmp_9 :=NOT( psal_vtmp_8;
psal_vtmp_10 :=AND( TER_9_1_i(6), psal_vtmp_9);
        IF( not ( psal_vtmp_10=BITVAL(0))) then
skip :=psal_const_11;
ELSE
skip :=psal_const_10;
        END IF;
        END IF;
        END IF;
ELSE
psal_vtmp_11 :=EQL( lac(12),psal_const_9);
psal_vtmp_12 :=AND( TER_9_1_i(4), psal_vtmp_11);
        IF( not ( psal_vtmp_12=BITVAL(0))) then
skip :=psal_const_8;
ELSE
psal_vtmp_13 :=EQL( lac(11 downto 0),psal_const_18);
psal_vtmp_14 :=AND( TER_9_1_i(5), psal_vtmp_13);
        IF( not ( psal_vtmp_14=BITVAL(0))) then
skip :=psal_const_7;
ELSE
psal_vtmp_15 :=LSS( lac(11 downto 0),psal_const_18);
psal_vtmp_16 :=AND( TER_9_1_i(6), psal_vtmp_15);
        IF( not ( psal_vtmp_16=BITVAL(0))) then
skip :=psal_const_6;
ELSE
skip :=psal_const_5;
        END IF;
        END IF;
```

```
                END IF;
                END IF;
                IF( not ( skip=BITVAL(0))) then
                reg_state <= pdp8_state_17;
        ELSE
        NULL;
                 reg_state <= pdp8_state_18;

                END IF;
                END IF;
        ELSE
                IF( not ( TER_9_1_i(7)=BITVAL(0))) then
        psal_vtmp_17 :=NOT( TER_9_1_i(6);
        masc := psal_vtmp_17 & psal_const_18;
        ELSE
        psal_vtmp_18 :=NOT( TER_9_1_i(6);
        masc := psal_vtmp_18 & psal_const_17;
                END IF;
        TER_9_6_lac :=AND( lac, masc);
                IF( not ( TER_9_1_i(4)=BITVAL(0))) then
        masc := TER_9_1_i(5) & psal_const_17;
        ELSE
        masc := TER_9_1_i(5) & psal_const_18;
                END IF;
        TER_9_8_lac :=AND( masc, TER_9_6_lac);
                IF( not ( TER_9_1_i(0)=BITVAL(0))) then
         lac <= ALU_1(SUM, TER_9_8_lac,psal_const_17(11) & psal_const_17);
        ELSE
         lac <=  TER_9_8_lac;
                END IF;
                reg_state <= pdp8_state_20;
                END IF;
                 when others =>
                 reg_state <= pdp8_state_27;

        END CASE;
WHEN pdp8_state_10 =>

        psal_address_m := e_tmp;
        psal_vtmp_20 :=m( psal_address_m);
         lac <= lac(12) & (AND( lac(11 downto 0), psal_vtmp_20));
                reg_state <= pdp8_state_27;
WHEN pdp8_state_11 =>

        psal_address_m := e_tmp;
        psal_vtmp_21 :=m( psal_address_m);
         lac <= ALU_1(SUM, lac, psal_vtmp_21(11) &  psal_vtmp_21);
                reg_state <= pdp8_state_27;
WHEN pdp8_state_12 =>

        TER_12_1_eadd_temp1 := e_tmp;
        psal_address_m := TER_12_1_eadd_temp1;
        psal_vtmp_22 :=m( psal_address_m);
```

```
        e_tmp <= ALU_1(SUM, psal_vtmp_22,psal_const_16 & psal_const_16 & psal_const_16 &
psal_const_16 & psal_const_16 & psal_const_16 & psal_const_16 & psal_const_16 &
psal_const_16 & psal_const_16 & psal_const_16 & psal_const_16);
        eadd <= TER_12_1_eadd_temp1;
                reg_state <= pdp8_state_13;
WHEN pdp8_state_13 =>

        psal_address_m := eadd;
         m( psal_address_m ) := e_tmp;

        psal_vtmp_23 :=EQL( e_tmp,psal_const_18);
                IF( not ( psal_vtmp_23=BITVAL(0))) then
         pc <= ALU_0(SUM, pc,psal_const_17);
        ELSE
        NULL;
                END IF;
                reg_state <= pdp8_state_27;
WHEN pdp8_state_14 =>

        psal_address_m := e_tmp;
         m( psal_address_m ) := lac(11 downto 0);

         lac <= lac(12) & (psal_const_18);
                reg_state <= pdp8_state_27;
WHEN pdp8_state_15 =>

        TER_15_1_eadd_temp1 := e_tmp;
        psal_address_m := TER_15_1_eadd_temp1;
         m( psal_address_m ) := pc;

         pc <= ALU_0(SUM, TER_15_1_eadd_temp1,psal_const_17);
                reg_state <= pdp8_state_27;
WHEN pdp8_state_16 =>

         pc <= e_tmp;
                reg_state <= pdp8_state_27;
WHEN pdp8_state_17 =>

         pc <= ALU_0(SUM pc,psal_const_17);
                reg_state <= pdp8_state_18;
WHEN pdp8_state_18 =>

                IF( not ( TER_9_1_i(7)=BITVAL(0))) then
         lac <= lac(12) & (psal_const_18);
        ELSE
        NULL;
                END IF;
                IF( not ( TER_9_1_i(2)=BITVAL(0))) then
         lac <= lac(12) & (AND( lac(11 downto 0), switches));
        ELSE
        NULL;
                END IF;
```

```
                IF( not ( TER_9_1_i(1)=BITVAL(0))) then
        psal_reset :=psal_const_15;
        ELSE
        NULL;
                END IF;
                reg_state <= pdp8_state_27;
WHEN pdp8_state_20 =>

                CASE  TER_9_1_i(3 downto 1) IS
        when 0 =>
        NULL;
                 reg_state <= pdp8_state_27;

        when 1 =>
        psal_vtmp_19 := lac(12) &  lac(5 downto 0);
         lac <=  psal_vtmp_19 &  lac(11 downto 6);
                 reg_state <= pdp8_state_27;

        when 2 =>
         lac <=  lac(11 downto 0) &  lac(12);
                 reg_state <= pdp8_state_27;
        when 3 =>
         lac <=  lac(10 downto 0) &  lac(12 downto 11);
                 reg_state <= pdp8_state_27;
        when 4 =>
         lac <=  lac(0) &  lac(12 downto 1);
                 reg_state <= pdp8_state_27;
        when 5 =>
         lac <=  lac(1 downto 0) &  lac(12 downto 2);
                 reg_state <= pdp8_state_27;
                 when others =>
                 reg_state <= pdp8_state_27;

        END CASE;
WHEN pdp8_state_27 =>

        psal_vtmp_24 :=AND( interrupt_state, interrupt_request);
                IF( not ( psal_vtmp_24=BITVAL(0))) then
        psal_address_m :=psal_const_18;
         m( psal_address_m ) :=  pc;

         pc <= psal_const_17;
        ELSE
        NULL;
                END IF;
                reg_state <= pdp8_state_9;
END case;
        END if;
        WAIT UNTIL (clock=0 and not clock'stable);
        END process;
END RTL;
```

4) Evaluación de los resultados

Vamos a realizar a continuación un estudio de las implementaciónes de PDP8 obtenidas por PSAL1 y PSAL2.

1º) Registros.

PSAL2 propone 2 registros menos que PSAL1. Ello es debido a que en la descripción que propone PSAL2, algunas variables han pasado a ser localizadas en terminales en vez de registros, como consecuencia del encadenamiento de operaciones.

2º) Estados

PSAL1 propone una descripción con 31 estados, mientras la solución propuesta por PSAL2 tiene solo 15. Esta importante reducción del número de estados es consecuencia de la mayor flexibilidad de la base de datos de PSAL2 asi como de los nuevos algoritmos de "scheduling" implementados en este sistema.

3º) Unidades operacionales

PSAL1 y PSAL2 proponen el mismo número de unidades operacionales. En un primer análisis se puede concluir que segundo sistema realiza una mejor asignación, ya que implementa en una de las unidades operacionales todas las operaciones que se realizan sobre el "program counter" (pc), mientras el resto (operaciones con el acumulador y memoria) son localizadas en la otra unidad operacional. PSAL1 sin embargo realiza algunas operaciones en los registros (incrementos y decrementos), mientras localiza las operaciones aritméticas en una OU, (la segunda) y algunas operaciones relacionales en la otra.

El resultado anterior debe de ser matizado ya que PSAL1 y PSAL2 no han utilizado los mismos criterios de asignación de OU's. Si se fuerza a PSAL2 a utilizar el mismo criterio que el otro sistema, el sistema propone una arquitectura con 3 OU's. La nueva asignación de operaciones es muy similar a la comentada, localizandose en la nueva OU únicamente operaciones relacionales.

IV) Conclusiones

Como se ha podido ver en los apartados anteriores, el sistema PSAL2 obtiene unos resultados mucho mejores que PSAL1, comparables a las soluciones propuestas en la

bibliografia para el mismo ejemplo. Especialmente importante es la diferencia entre la asignación de estados que realiza PSAL1 y la que realiza PSAL2. Esta diferencia es consecuencia en gran medida de la mayor flexibilidad del formato intermedio de PSAL2, que permite encadenar operaciones o definir multiciclo o pipelining.

En cuanto a la asignación de registos, ambos sistemas han asignados todos elementos en el menor conjunto de elementos que permitian las dependencias entre datos. Este mismo hecho se produce en la asignación de operaciones a unidades operacionales. Por ello se puede afirmar que la asignación de recursos hardware es la optima, ya que se obtiene una asignación en el menor número de elementos que permiten las dependencias de datos y el "scheduling" realizado anteriormente.

CONCLUSIONES

1º) Se ha estudiado, en primer lugar, la utilización de lenguajes de descripción de hardware en síntesis de alto nivel lo que ha llevado a la implementación de compiladores y generadores de código que permiten iniciar la síntesis a partir de descripciones a nivel algorítmico y describir la arquitectura resultante a nivel de transferencia de registros.

2º) Se ha procedido a la implementación del sistema PSAL1, que partiendo de una descripción a nivel algorítmico en ISPS, propone una arquitectura que describe utilizando el lenguaje DDL.

3º) Del análisis de los resultados de PSAL1 se ha concluido la necesidad de contar con lenguajes más homogeneos para describir el sistema a nivel algorítmico y RT, asi como, que las construcciones de control del lenguaje de entrada no deben de ser tan rígidas como las de ISPS. Además, se ha evaluado la posibilidad de contar con un lenguaje de salida que fuese utilizado por herramientas de síntesis a nivel RT, con el fin de implementar la arquitectura resultante. La elección de KARL IV como lenguaje de entrada y CVS_BK como salida estuvo motivada por el hecho de que estos lenguajes del entorno KARL, cumplian los requisitos antes mencionados. El sistema PSAL2 utiliza CVS_BK para representar la arquitectura resultante del proceso de síntesis y no pudo utilizar KARL IV debido a que este lenguaje no estaba lo suficientemente desarrollado.

4º) Se ha procedido al desarrollo de los interfaces de entrada y salida VHDL con objeto de permitir el uso de PSAL2 en entornos estandar multinivel.

5º) El uso de VHDL en PSAL2 ha precisado de un estudio de la utilización de este lenguaje en la descripción del sistema a nivel algorítmico y a nivel de transferencia de registros.

6º) Se ha conectado el sistema PSAL2 con las herramientas de síntesis a nivel de transferencias de registros desarrolladas en nuestro grupo, mediante la utilización del sistema ESP.

7º) Se han estudiado e implementado representaciones intermedias para soportar el proceso de síntesis de alto nivel. El sistema PSAL1 utiliza un formato muy ligado al lenguaje ISPS, lo

que constituyó un inconveniente a la hora de mejorar los algoritmos de síntesis. Por ello se desarrolló un nuevo formato intermedio, mucho mas flexible y no ligado a un lenguaje de entrada particular. Este formato constituye la base del sistema PSAL2. Al mismo tiempo se definió un lenguaje que permite conocer y definir la información contenida en la base de datos del sistema PSAL2 en cualquier fase del proceso de síntesis. La base de datos propuesta almacena información sobre el comportamiento del sistema, la arquitectura propuesta y la relación entre ambas. A diferencia del formato de PSAL1, es muy flexible permitiendo modelar arquitecturas de muy variados estilos, introducir nuevos algoritmos de síntesis con facilidad e interactuar con el proceso de síntesis.

8º) Se han propuesto y desarrollado algoritmos de asignación de estados y síntesis del data-path. De esta forma, en el sistema PSAL1, se ha desarrollado un algoritmo de asignación de estados basado en la técnica ASAP que trata de optimizar el número de estados en las sentencias condicionales y que permite poner límite al número de unidades operacionales, lo cual reduce el coste de la implementación resultante. Además se ha propuesto e implementado un algoritmo iterativo/constructivo que realiza la localización de operaciones en OUs teniendo en cuenta además del coste y similitud de las operaciones, el coste de las interconexiones entre las OUs y los registros. La localización de variables en registros se lleva a cabo en PSAL1 mediante una metodologia global basada en técnicas de partición de grafos.

9º) En el sistema PSAL2 se han implementado versiones mejoradas del algoritmo de asignación de operaciones dirigido por fuerza y del "list scheduling" para realizar la asignación de operaciones a estados. Además, se ha desarrollado un nuevo algoritmo capaz de realizar la asignación conjunta de operaciones de control y datos. Por otro lado, se ha propuesto e implementado una técnica que puede ser utilizada tanto para localización de variables en registros como para asignación de operaciones a unidades operacionales basada en la utilización de probabilidades. Esta técnica permite evaluar el coste de la implementación sin necesidad de haber concluido el proceso de síntesis.

10º) Los algoritmos desarrollados han sido verificados sobre distintos ejemplos, mostrando su eficacia.

11º) Todo este trabajo se ha plasmado en los sistemas de síntesis de alto nivel PSAL1 y PSAL2. El primero esta constituido por alrededor de 15000 lineas en lenguaje C y funciona en un μVAX II con sistema operativo VMS. El segundo, esta formado por alrededor de 30.000 lineas en lenguaje C y funciona en una SUN con sistema operativo UNIX.

APENDICE A

EJEMPLO DE GRAFO DE FLUJO

Grafo de flujo generado por PSAL1 a partir de la descripción ISPS del computador MARK1.

TABLA DE ENTIDADES

Numero= 1

numero	defini	contex	coste	nombre
1	1	1	0	MARK1

TABLA DE NOMBRES

Numero= 8

numero	flag	pali	pald	biti	bitd	padre	contex	incre	nombre
1	2	0	0	31	0	0	1	0	T38
2	20	0	8191	31	0	0	1	1	M
3	0	0	0	12	0	0	1	0	CR
4	0	0	0	15	0	0	1	0	PI
5	1	0	0	0	2	9	1	0	F
6	1	0	0	0	12	10	1	0	S
7	0	0	0	31	0	0	1	0	ACC
8	2	0	0	0	0	0	1	0	T41

TABLA DE SIMBOLOS

Numero= 14

numero	rtm	flag	nombre	pali	pald	biti	bitd
1	40	22	0	0	0	1	0
2	43	22	1	0	0	1	0
3	38	24	1	0	0	31	0
4	34	24	1	0	0	15	0
5	42	60	2	7	0	31	0
6	37	60	2	12	0	31	0
7	42	21	3	0	0	12	0
8	33	21	4	0	0	15	0
9	3	221	4	0	0	15	13
10	4	221	4	0	0	12	0
11	35	21	5	0	0	0	2
12	37	21	6	0	0	0	12
13	39	21	7	0	0	31	0
14	41	24	8	0	0	0	0

TABLA DE SENTENCIAS

Numero= 16

numero	nante	noper	ante	oper	rtm	control fin	nsuc	suc
				oper	flag	result	dato1	dato2
1	0	0	1					
				342	0	1	16	1	2
2	2	0						
			1					
			14					
				333	8	0	0	1	3
3	1	2	2					
				23	21	3	5	0	
				25	41	8	4	0	
				301	11	11	13	7	
							1	4	0	0
							1	5	1	0

1	6	2	0
1	7	3	0
2	8	4	5
1	9	6	0
1	12	7	0

---

4 1 1 3

23 61 7 6 0

206 13 0 0 1 13

---

5 1 2 3

23 21 3 6 0
101 43 7 7 3

206 16 0 0 1 13

---

6 1 2 3

23 21 3 6 0
100 41 13 3 0

206 19 0 0 1 13

---

7 1 1 3

24 61 6 13 0

206 21 0 0 1 13

---

8 1 2 3

23 21 3 6 0
102 43 13 13 3

206 24 0 0 1 13

---

9 1 1 3

113 63 14 13 1

300 26 14 11 2

1 11 0 0

0 10 0 0

---

10 1 1 9

101 63 7 7 2

206 28 0 0 1 11

---

11 2 0

9

10

333 28 0 9 1 13

---

12 1 0 3

343 30 0 0 1 13

---

13 7 0

4

5

6

7

8

11

12

333 31 0 3 1 14

---

14 1 1 13

101 63 7 7 2

206 33 0 0 1 2

15	0	0							
			335	33	0	0	1	16	
16	1	0	15						
			205	0	0	1	0		

APENDICE B

CODIGOS DE OPERACION DEL SISTEMA PSAL1

Los códigos de operación se clasifican, en función del valor de los dos primeros bits (de los ocho que lo forman), en cuatro grupos:

Bits 0:1	Tipo
00	Varios
01	Aritméticos y relacionales
10	Especiales
11	Control

Los códigos del primer grupo, se clasifican a su vez en función de los bits 2º y 3º. De esta forma:

Bits 2:3	Tipo
00	Clase 1
01	Clase 2
10	Shift y rotación
11	Lógica

Los códigos aritméticos y relacionales se dividen en dos clases (según el valor del bit 4º):

Bit 4	Tipo
0	Aritmético
1	Relacional

Los códigos aritméticos (4º bit = 0) utilizan los bits 2º y 3º, para señalar en que representación se realizan las operaciones. Así:

Bits 2:3	Tipo
00	Complemento 1
01	Complemento 2
10	Magnitudes con bit de signo
11	Magnitudes sin signo

Una vez descritos los tipos de códigos, veamos las operaciones que pueden formar parte del grafo de flujo:

a) Códigos clase 1

Expresaremos el valor de los bits 2:7 en octal. Cuando el código no se corresponda con una operación en el lenguaje ISPS, pondremos un ejemplo detrás de dicho código. Entonces:

Bits 2:7	Código	Ejemplo
1	clear	a=0
2	incrementa	a=a+1
3	decrementa	a=a-1
4	concatenación	
5	mueve	a=b

b) Códigos clase 2

Utilizaremos el mismo formato que en el caso anterior. Los códigos que forman esta clase son:

Bits 2:7	Código	Ejemplo
20	noop	
21	conexión	a:=b
23	lee de memoria	a=m[2]
24	escribe en memoria	
25	lee bits	a=b<2:5>
26	escribe bits	b<2:5>=a

c) Códigos de desplazamiento y rotación

Bits 2:7	Código
40	sl0
41	sl1
42	slr
43	sld
44	sr0
45	sr1
46	srr
47	srd
50	sli
51	sri

d) Códigos de operaciones lógicas

Bits 2:7	Código
60	not
61	and
62	or
63	nand
64	nor
65	xor
66	eqv

e) Códigos de operaciones aritméticas

Estos códigos se asignan en función del valor de los bits 5:7.

Bits 5:7	Código	Ejemplo
0	neg	
1	add	
2	sub	
3	mult	
4	div	
5	mod	
6	amove	a<=b

f) Códigos de operaciones relacionales

Bits 5:7	Código
0	test
1	eql
2	neq
3	lss
4	leq
5	geq
6	gtr

g) Códigos de las operaciones especiales

Bits 2:7	Código	Ejemplo
00	inicio de accion marcada	
01	begin (PROCESS)	
02	begin (CRITICAL)	
03	begin (PROCESS+ CRITICAL)	
05	fin de grafo	
06	bloque básico	
20	lee bit genérico	a=b<c>
21	declaración de argumento	
22	escribe bit genérico	b<c>=a

h) Códigos de acciones de control

Bits 2:7	Código	Ejemplo
00	if	
01	select	DECODE ...
02	bselect	DECODE ...=> OTHERWISE..
03	branch	
10	leave	
11	restart	
12	resume	
13	terminate	
20	begin	
21	end	
22	call	
30	diverge	

31	pmerge
32	pjoin
33	smerge
34	sjoin
40	undefine
41	unpredictable
42	inicio de grafo
43	stop
51	wait

APENDICE C

EJEMPLO DE BASE DE DATOS DE PSAL2

A continuación se muestra la base de datos utilizada por PSAL2 en la síntesis del EDC.

```
System edc:=
 begin
 defarq:=
  begin
  defmod:=
   begin
   creamod 1,called CONSTANT_MOD_25,10,4,1 : 1 ,1 : 2 ,1,#r 1 : 0 ,#d 0,#c 0 ;
   creamod 2,called CONSTANT_MOD_15,10,4,1 : 3 ,1 : 4 ,16,#r 1 : 1 ,#d 0,#c 0 ;
   creamod 3,called CONSTANT_MOD_14,10,4,1 : 5 ,1 : 6 ,16,#r 1 : 0 ,#d 0,#c 0 ;
   creamod 4,called ovf,2,4096,0 : ,1 : 7 ,1,#d 0,#c 0 ;
   creamod 5,called clear,2,4096,0 : ,1 : 8 ,1,#d 0,#c 0 ;
   creamod 6,called run,2,4096,0 : ,1 : 9 ,1,#d 0,#c 0 ;
   creamod 7,called output,2,8192,1 : 10 ,0 : ,16,#d 0,#c 0 ;
   creamod 8,called input,2,4096,0 : ,1 : 11 ,16,#d 0,#c 0 ;
   creamod 9,called out_ok,2,4096,0 : ,1 : 12 ,1,#d 0,#c 0 ;
   creamod 10,called in_ok,2,4096,0 : ,1 : 13 ,1,#d 0,#c 0 ;
   creamod 11,called m,3,98304,#p 1 : 13 ,#p 1 : 12 ,16,1024,#d 0,#c 0 ;
   creamod 12,called psal_data_m,2,1032,1 : 14 ,1 : 15 ,16,#m 1 : 11 ,#d 0,#c 0 ;
   creamod 13,called psal_address_m,2,2056,1 : 16 ,0 : ,10,#m 1 : 11 ,#d 0,#c 0 ;
   creamod 14,called psal_vtmp_6,2,0,1 : 17 ,1 : 18 ,2,#d 0,#c 0 ;
   creamod 15,called psal_vtmp_0,2,0,1 : 19 ,1 : 20 ,1,#d 0,#c 0 ;
   creamod 16,called psal_vtmp_1,2,0,1 : 21 ,1 : 22 ,16,#d 0,#c 0 ;
   creamod 17,called psal_vtmp_2,2,0,1 : 23 ,1 : 24 ,16,#d 0,#c 0 ;
   creamod 18,called psal_vtmp_3,2,0,1 : 25 ,1 : 26 ,16,#d 0,#c 0 ;
   creamod 19,called psal_vtmp_4,2,0,1 : 27 ,1 : 28 ,1,#d 0,#c 0 ;
   creamod 20,called psal_vtmp_5,2,0,1 : 29 ,1 : 30 ,1,#d 0,#c 0 ;
   creamod 21,called ir,1,0,1 : 31 ,1 : 32 ,16,#d 0,#c 0 ;
   creamod 22,called acc,1,0,1 : 33 ,1 : 34 ,16,#d 0,#c 0 ;
   creamod 23,called car,1,0,1 : 35 ,1 : 36 ,10,#d 0,#c 0 ;
   creamod 24,called ALU_0,4,0,2 : 37 38 ,1 : 39 ,16,#o 2: 65 0 16 0  0 0| 66 0 16 0  0 0|,
     #d 0,#c 0 ;
   creamod 25,called ALU_1,4,0,2 : 40 41 ,1 : 42 ,16,#o 3: 64 0 16 0  0 0| 65 0 10 0  0 0|
        66 0 10 0  0 0|,#d 0,#c 0 ;
   creamod 26,called CONTROL_UNIT_26,12,0,2 : 43 44 ,1 : 45 ,0,#d 0,#c 0 ;
   end,
```

```
defpor:=
 begin
 creapor 1, 1, 0, 0, 1, 1, 0 ,#d 0 ;
 creapor 2, 2, 0, 0, 1, 0, 1 ,#d 0 ;
 creapor 3, 1, 0, 15, 2, 1, 0 ,#d 0 ;
 creapor 4, 2, 0, 15, 2, 0, 1 ,#d 0 ;
 creapor 5, 1, 0, 15, 3, 1, 0 ,#d 0 ;
 creapor 6, 2, 0, 15, 3, 0, 1 ,#d 0 ;
 creapor 7, 2, 0, 0, 4, 0, 1 ,#d 0 ;
 creapor 8, 2, 0, 0, 5, 0, 1 ,#d 0 ;
 creapor 9, 2, 0, 0, 6, 0, 1 ,#d 0 ;
 creapor 10, 1, 15, 0, 7, 1, 0 ,#d 0 ;
 creapor 11, 2, 0, 15, 8, 0, 1 ,#d 0 ;
 creapor 12, 2, 0, 0, 9, 0, 1 ,#d 0 ;
 creapor 13, 2, 0, 0, 10, 0, 1 ,#d 0 ;
 creapor 14, 9, 15, 0, 12, 1, 0 ,#d 0 ;
 creapor 15, 10, 0, 15, 12, 0, 1 ,#d 0 ;
 creapor 16, 5, 0, 9, 13, 1, 0 ,#w 1023 0 ,#d 0 ;
 creapor 17, 1, 1, 0, 14, 1, 0 ,#d 0 ;
 creapor 18, 2, 0, 1, 14, 0, 1 ,#d 0 ;
 creapor 19, 1, 0, 0, 15, 1, 0 ,#d 0 ;
 creapor 20, 2, 0, 0, 15, 0, 1 ,#d 0 ;
 creapor 21, 1, 15, 0, 16, 1, 0 ,#d 0 ;
 creapor 22, 2, 0, 15, 16, 0, 1 ,#d 0 ;
 creapor 23, 1, 15, 0, 17, 1, 0 ,#d 0 ;
 creapor 24, 2, 0, 15, 17, 0, 1 ,#d 0 ;
 creapor 25, 1, 15, 0, 18, 1, 0 ,#d 0 ;
 creapor 26, 2, 0, 15, 18, 0, 1 ,#d 0 ;
 creapor 27, 1, 0, 0, 19, 1, 0 ,#d 0 ;
 creapor 28, 2, 0, 0, 19, 0, 1 ,#d 0 ;
 creapor 29, 1, 0, 0, 20, 1, 0 ,#d 0 ;
 creapor 30, 2, 0, 0, 20, 0, 1 ,#d 0 ;
 creapor 31, 1, 15, 0, 21, 1, 0 ,#d 0 ;
 creapor 32, 2, 0, 15, 21, 0, 1 ,#d 0 ;
 creapor 33, 1, 15, 0, 22, 1, 0 ,#d 0 ;
 creapor 34, 2, 0, 15, 22, 0, 1 ,#d 0 ;
 creapor 35, 1, 9, 0, 23, 1, 0 ,#d 0 ;
 creapor 36, 2, 0, 9, 23, 0, 1 ,#d 0 ;
 creapor 37, 1, 16, 0, 24, 1, 0 ,#d 0 ;
 creapor 38, 1, 16, 0, 24, 2, 0 ,#d 0 ;
 creapor 39, 2, 16, 0, 24, 0, 1 ,#d 0 ;
 creapor 40, 1, 16, 0, 25, 1, 0 ,#d 0 ;
 creapor 41, 1, 10, 0, 25, 2, 0 ,#d 0 ;
 creapor 42, 2, 10, 0, 25, 0, 1 ,#d 0 ;
 creapor 43, 1, 15, 0, 26, 1, 0 ,#d 0 ;
 creapor 44, 1, 15, 0, 26, 2, 0 ,#d 0 ;
 creapor 45, 2, 15, 0, 26, 0, 1 ,#d 0 ;
 end,

defwir:=
 begin
 end
end,
```

```
defbeh:=
 begin
 defent:=
  begin
  creaent 1, called edc, 8,  1, #i 146 ;
  end,
 defvar:=
  begin
  creavar 25, 1, 32, value 0 , 0, 0, 1 :  0 0 |,#c 1, 1, 0, 0, 0;
  creavar 24, 0, 48, called psal_vtmp_5, 0, 0, 1 :  0 0 |,#c 1, 20, 0, 0, 0,#s 1:  0 0 0 |,1,1;
  creavar 23, 0, 48, called psal_vtmp_4, 0, 0, 1 :  0 0 |,#c 1, 19, 0, 0, 0,#s 1:  0 0 0 |,1,1;
  creavar 22, 0, 48, called psal_vtmp_3, 15, 0, 1 :  15 0 |,#c 1, 18, 0, 0, 0,#s 1:  15 0 0 |,1,1;
  creavar 21, 0, 48, called psal_vtmp_2, 15, 0, 1 :  15 0 |,#c 1, 17, 0, 0, 0,#s 1:  15 0 0 |,1,1;
  creavar 20, 0, 48, called psal_vtmp_1, 15, 0, 1 :  15 0 |,#c 1, 16, 0, 0, 0,#s 1:  15 0 0 |,1,1;
  creavar 19, 0, 48, called psal_vtmp_0, 0, 0, 1 :  0 0 |,#c 1, 15, 0, 0, 0,#s 1:  0 0 0 |,1,1;
  creavar 26, 0, 48, called psal_vtmp_6, 1, 0, 1 :  1 0 |,#c 1, 14, 0, 0, 0,#s 2:  1 1 0 | 0 0 0 |,1,1;
  creavar 15, 1, 32, value 1 , 15, 0, 1 :  15 0 |,#c 1, 2, 0, 0, 0;
  creavar 14, 1, 32, value 0 , 15, 0, 1 :  15 0 |,#c 1, 3, 0, 0, 0;
  creavar 13, 6, 56, called psal_data_m, 15, 0, 1 :  1 16 |,#c 1, 12, 0, 0, 0,#v 1: 11 ,
      #s 1:  15 0 0 |,6,3;
  creavar 12, 7, 56, called psal_address_m, 9, 0, 1 :  9 0 |,#c 1, 13, 0, 0, 0,#v 1: 11 ,
      #s 1:  9 0 0 |,9,9;
  creavar 11, 9, 48, called m, 15, 0, 1 :  1 16 |,#w  1023 0 |,#c 1, 11, 0, 0, 0,#v 2: 12 13 ;
  creavar 10, 0, 0, called car, 9, 0, 1 :  9 0 |,#c 1, 23, 0, 0, 0,#s 1:  9 0 1 |,2,7;
  creavar 9, 0, 0, called acc, 15, 0, 1 :  15 0 |,#c 1, 22, 0, 0, 0,#s 4:  15 15 2 | 14 8 3 | 7 1 4 |
       0 0 5 |,19,16;
  creavar 8, 0, 0, called ir, 15, 0, 1 :  15 0 |,#c 1, 21, 0, 0, 0,#s 3:  15 12 6 | 11 10 7 | 9 0 8 |,23,6;
  creavar 7, 5, 312, called ovf, 0, 0, 1 :  0 0 |,#c 1, 4, 0, 0, 0,#s 1:  0 0 0 |,1,0;
  creavar 6, 5, 312, called clear, 0, 0, 1 :  0 0 |,#c 1, 5, 0, 0, 0,#s 1:  0 0 0 |,1,0;
  creavar 5, 5, 312, called run, 0, 0, 1 :  0 0 |,#c 1, 6, 0, 0, 0,#s 1:  0 0 0 |,2,0;
  creavar 4, 5, 568, called output, 15, 0, 1 :  1 16 |,#c 1, 7, 0, 0, 0,#s 1:  15 0 0 |,0,1;
  creavar 3, 5, 312, called input, 15, 0, 1 :  1 16 |,#c 1, 8, 0, 0, 0,#s 1:  15 0 0 |,2,0;
  creavar 2, 5, 312, called out_ok, 0, 0, 1 :  0 0 |,#c 1, 9, 0, 0, 0,#s 1:  0 0 0 |,1,0;
  creavar 1, 5, 312, called in_ok, 0, 0, 1 :  0 0 |,#c 1, 10, 0, 0, 0,#s 1:  0 0 0 |,2,0;
  end,
 deflin:=
  begin
  crealin 1, 2, 8, 1, 15, 12, 26, 1, 3, 0, 0 ;
  crealin 2, 2, 1, 1, 0, 0, 26, 1, 0, 0, 0 ;
  crealin 3, 0, 8, 1, 9, 0, 12, 1, 9, 0, 0 ;
  crealin 4, 0, 3, 1, 15, 0, 13, 1, 15, 0, 0 ;
  crealin 5, 2, 2, 1, 0, 0, 26, 1, 0, 0, 0 ;
  crealin 6, 0, 8, 1, 9, 0, 12, 1, 9, 0, 0 ;
  crealin 7, 0, 13, 1, 15, 0, 4, 1, 15, 0, 0 ;
  crealin 8, 0, 8, 1, 9, 0, 9, 1, 15, 0, 0 ;
  crealin 9, 0, 15, 1, 15, 0, 8, 1, 9, 0, 0 ;
  crealin 10, 2, 9, 1, 15, 0, 0, 0, 0, 0, 0 ;
  crealin 11, 2, 14, 1, 15, 0, 0, 0, 0, 0, 0 ;
  crealin 12, 1, 0, 0, 0, 0, 19, 1, 0, 0, 0 ;
  crealin 13, 2, 19, 1, 0, 0, 26, 1, 0, 0, 0 ;
  crealin 14, 2, 1, 1, 0, 0, 26, 1, 0, 0, 0 ;
  crealin 15, 0, 8, 1, 9, 0, 12, 1, 9, 0, 0 ;
  crealin 16, 0, 3, 1, 15, 0, 13, 1, 15, 0, 0 ;
  crealin 17, 2, 9, 1, 15, 0, 24, 1, 16, 0, 0 ;
  crealin 18, 2, 15, 1, 15, 0, 24, 2, 16, 0, 0 ;
```

```
crealin 19, 1, 24, 1, 16, 0, 9, 1, 15, 0, 0 ;
crealin 20, 2, 8, 1, 9, 0, 25, 1, 10, 0, 0 ;
crealin 21, 2, 15, 1, 9, 0, 25, 2, 10, 0, 0 ;
crealin 22, 1, 25, 1, 10, 0, 8, 1, 9, 0, 0 ;
crealin 23, 2, 8, 1, 11, 10, 26, 1, 1, 0, 0 ;
crealin 24, 0, 14, 1, 15, 0, 9, 1, 15, 0, 0 ;
crealin 25, 2, 9, 1, 15, 0, 24, 1, 16, 0, 0 ;
crealin 26, 2, 15, 1, 15, 0, 24, 2, 16, 0, 0 ;
crealin 27, 1, 24, 1, 16, 0, 9, 1, 15, 0, 0 ;
crealin 28, 2, 9, 1, 15, 0, 24, 1, 16, 0, 0 ;
crealin 29, 2, 15, 1, 15, 0, 24, 2, 16, 0, 0 ;
crealin 30, 1, 24, 1, 16, 0, 9, 1, 15, 0, 0 ;
crealin 31, 2, 9, 1, 15, 0, 25, 1, 16, 0, 0 ;
crealin 32, 1, 25, 1, 16, 0, 9, 1, 15, 0, 0 ;
crealin 33, 0, 8, 1, 9, 0, 12, 1, 9, 0, 0 ;
crealin 34, 0, 13, 1, 15, 0, 9, 1, 15, 0, 0 ;
crealin 35, 0, 8, 1, 9, 0, 12, 1, 9, 0, 0 ;
crealin 36, 0, 13, 1, 15, 0, 20, 1, 15, 0, 0 ;
crealin 37, 2, 9, 1, 15, 0, 24, 1, 16, 0, 0 ;
crealin 38, 2, 20, 1, 15, 0, 24, 2, 16, 0, 0 ;
crealin 39, 1, 24, 1, 16, 0, 9, 1, 15, 0, 0 ;
crealin 40, 0, 8, 1, 9, 0, 12, 1, 9, 0, 0 ;
crealin 41, 0, 13, 1, 15, 0, 21, 1, 15, 0, 0 ;
crealin 42, 2, 9, 1, 15, 0, 24, 1, 16, 0, 0 ;
crealin 43, 2, 21, 1, 15, 0, 24, 2, 16, 0, 0 ;
crealin 44, 1, 24, 1, 16, 0, 9, 1, 15, 0, 0 ;
crealin 45, 0, 8, 1, 9, 0, 12, 1, 9, 0, 0 ;
crealin 46, 0, 9, 1, 15, 0, 13, 1, 15, 0, 0 ;
crealin 47, 0, 8, 1, 9, 0, 12, 1, 9, 0, 0 ;
crealin 48, 0, 13, 1, 15, 0, 22, 1, 15, 0, 0 ;
crealin 49, 2, 9, 1, 15, 0, 26, 1, 15, 0, 0 ;
crealin 50, 2, 22, 1, 15, 0, 26, 2, 15, 0, 0 ;
crealin 51, 1, 26, 1, 15, 0, 9, 1, 15, 0, 0 ;
crealin 52, 2, 8, 1, 11, 10, 26, 1, 1, 0, 0 ;
crealin 53, 0, 8, 1, 9, 0, 10, 1, 9, 0, 0 ;
crealin 54, 2, 9, 1, 15, 0, 0, 0, 0, 0, 0 ;
crealin 55, 2, 14, 1, 15, 0, 0, 0, 0, 0, 0 ;
crealin 56, 1, 0, 0, 0, 0, 23, 1, 0, 0, 0 ;
crealin 57, 2, 23, 1, 0, 0, 26, 1, 0, 0, 0 ;
crealin 58, 0, 8, 1, 9, 0, 10, 1, 9, 0, 0 ;
crealin 59, 2, 9, 1, 15, 15, 26, 1, 0, 0, 0 ;
crealin 60, 0, 8, 1, 9, 0, 10, 1, 9, 0, 0 ;
crealin 61, 2, 7, 1, 0, 0, 26, 1, 0, 0, 0 ;
crealin 62, 0, 8, 1, 9, 0, 10, 1, 9, 0, 0 ;
crealin 63, 2, 8, 1, 11, 10, 26, 1, 1, 0, 0 ;
crealin 64, 0, 8, 1, 9, 0, 9, 1, 15, 0, 0 ;
crealin 65, 2, 9, 1, 15, 0, 24, 1, 16, 0, 0 ;
crealin 66, 2, 8, 1, 9, 0, 24, 2, 10, 0, 0 ;
crealin 67, 1, 24, 1, 16, 0, 9, 1, 15, 0, 0 ;
crealin 68, 2, 8, 1, 9, 0, 0, 0, 0, 0, 0 ;
crealin 69, 2, 14, 1, 9, 0, 0, 0, 0, 0, 0 ;
crealin 70, 1, 0, 0, 0, 0, 24, 1, 0, 0, 0 ;
crealin 71, 2, 24, 1, 0, 0, 26, 1, 0, 0, 0 ;
crealin 72, 2, 8, 1, 9, 0, 25, 1, 10, 0, 0 ;
crealin 73, 2, 15, 1, 9, 0, 25, 2, 10, 0, 0 ;
```

```
crealin 74, 1, 25, 1, 10, 0, 8, 1, 9, 0, 0 ;
crealin 75, 2, 8, 1, 11, 10, 26, 1, 1, 0, 0 ;
crealin 76, 0, 9, 1, 15, 15, 9, 1, 15, 15, 0 ;
crealin 77, 0, 9, 1, 15, 1, 9, 1, 14, 0, 0 ;
crealin 78, 0, 25, 1, 0, 0, 9, 1, 15, 15, 0 ;
crealin 79, 0, 9, 1, 15, 1, 9, 1, 14, 0, 0 ;
crealin 80, 0, 9, 1, 0, 0, 9, 1, 15, 15, 0 ;
crealin 81, 0, 9, 1, 15, 1, 9, 1, 14, 0, 0 ;
crealin 82, 0, 9, 1, 7, 0, 9, 1, 15, 8, 0 ;
crealin 83, 0, 9, 1, 15, 8, 9, 1, 7, 0, 0 ;
crealin 84, 0, 6, 1, 0, 0, 26, 1, 1, 1, 0 ;
crealin 85, 0, 5, 1, 0, 0, 26, 1, 0, 0, 0 ;
crealin 86, 2, 26, 1, 1, 0, 26, 1, 1, 0, 0 ;
crealin 87, 0, 10, 1, 9, 0, 12, 1, 9, 0, 0 ;
crealin 88, 0, 13, 1, 15, 0, 8, 1, 15, 0, 0 ;
crealin 89, 2, 10, 1, 9, 0, 25, 1, 10, 0, 0 ;
crealin 90, 2, 15, 1, 9, 0, 25, 2, 10, 0, 0 ;
crealin 91, 1, 25, 1, 10, 0, 10, 1, 9, 0, 0 ;
crealin 92, 0, 14, 1, 9, 0, 10, 1, 9, 0, 0 ;
crealin 93, 0, 14, 1, 15, 0, 8, 1, 15, 0, 0 ;
crealin 94, 0, 14, 1, 9, 0, 10, 1, 9, 0, 0 ;
crealin 95, 0, 14, 1, 15, 0, 8, 1, 15, 0, 0 ;
crealin 96, 2, 5, 1, 0, 0, 26, 1, 0, 0, 0 ;
end,
defsen:=
begin

creasen 1, 192, 3, ante 0 : , suc 1 : 0 0 0 146 |, 2, 0, 0, 0,#f 174,#c 1 ;
creasen 146, 194, 3, ante 2 : 170 1 , suc 1 : 0 0 0 147 |, 3, 0, 0, 0,#f 173,#c 1 ;
creasen 147, 144, 1, ante 1 : 146 , 0 :, 3, 0, 0, 0,#f 172 ;
creasen 148, 4, 0, 2 : 6 0 0 84 | 5 0 0 85 | , 1 : 26 1 0 |, 3, 0, 0, 0 ;
creasen 149, 165, 50, 1 : 26 1 0 86 | , suc 4 : 1 0 0 150 | 1 1 0 152 | 1 2 0 157 | 1 3 0 161 |,
    3, 0, 0, 0,#f 165 ;
creasen 150, 144, 1, ante 1 : 149 , 0 :, 3, 0, 0, 0,#f 151 ;
creasen 151, 160, 2, 0 : , suc 1 : 0 0 0 165 |, 3, 0, 0, 0,#f 150 ;
creasen 152, 144, 1, ante 1 : 149 , 0 :, 3, 0, 0, 0,#f 156 ;
creasen 153, 70, 0, 1 : 10 9 0 87 | , 1 : 12 9 0 |, 3, 0, 0, 0 ;
creasen 154, 19, 0, 2 : 13 15 0 88 | 12 9 0 | , 1 : 8 15 0 |, 3, 0, 0, 0 ;
creasen 155, 65, 0, 2 : 10 9 0 89 | 15 9 0 90 | , 1 : 10 9 0 91 |,#m 25,2, 3, 0, 0, 0 ;
creasen 156, 160, 2, 0 : , suc 1 : 0 0 0 165 |, 3, 0, 0, 0,#f 152 ;
creasen 157, 144, 1, ante 1 : 149 , 0 :, 3, 0, 0, 0,#f 160 ;
creasen 158, 70, 0, 1 : 14 9 0 92 | , 1 : 10 9 0 |, 3, 0, 0, 0 ;
creasen 159, 70, 0, 1 : 14 15 0 93 | , 1 : 8 15 0 |, 3, 0, 0, 0 ;
creasen 160, 160, 2, 0 : , suc 1 : 0 0 0 165 |, 3, 0, 0, 0,#f 157 ;
creasen 161, 144, 1, ante 1 : 149 , 0 :, 3, 0, 0, 0,#f 164 ;
creasen 162, 70, 0, 1 : 14 9 0 94 | , 1 : 10 9 0 |, 3, 0, 0, 0 ;
creasen 163, 70, 0, 1 : 14 15 0 95 | , 1 : 8 15 0 |, 3, 0, 0, 0 ;
creasen 164, 160, 2, 0 : , suc 1 : 0 0 0 165 |, 3, 0, 0, 0,#f 161 ;
creasen 165, 149, 49, ante 4 : 164 160 156 151 , 0 :, 3, 0, 0, 0,#f 149 ;
creasen 166, 25, 48, 1 : 5 0 0 96 | , 0 :, 4, 1, 0, 0 ;
creasen 4, 165, 50, 1 : 8 15 12 1 | , suc 12 : 1 0 0 5 | 1 1 0 10 | 1 2 0 15 | 1 3 0 32 | 1 4 0 48 |
    1 5 0 52 | 1 6 0 57 | 1 7 0 62 | 1 8 0 66 | 1 9 0 71 | 1 10 0 106 | 1 11 0 118 |, 4, 0, 0, 0,
    #f 143 ;
creasen 5, 144, 1, ante 1 : 4 , 0 :, 4, 0, 0, 0,#f 9 ;
creasen 6, 25, 48, 1 : 1 0 0 2 | , 0 :, 4, 0, 0, 0 ;
```

```
creasen 7, 70, 0, 1 : 8 9 0 3 | , 1 : 12 9 0 |, 4, 0, 0, 0 ;
creasen 8, 20, 0, 2 : 3 15 0 4 | 12 9 0 | , 1 : 13 15 0 |, 4, 0, 0, 0 ;
creasen 9, 160, 2, 0 : , suc 1 : 0 0 0 143 |, 4, 0, 0, 0,#f 5 ;
creasen 10, 144, 1, ante 1 : 4 , 0 :, 4, 0, 0, 0,#f 14 ;
creasen 11, 25, 48, 1 : 2 0 0 5 | , 0 :, 4, 0, 0, 0 ;
creasen 12, 70, 0, 1 : 8 9 0 6 | , 1 : 12 9 0 |, 4, 0, 0, 0 ;
creasen 13, 19, 0, 2 : 13 15 0 7 | 12 9 0 | , 1 : 4 15 0 |, 4, 0, 0, 0 ;
creasen 14, 160, 2, 0 : , suc 1 : 0 0 0 143 |, 4, 0, 0, 0,#f 10 ;
creasen 15, 144, 1, ante 1 : 4 , 0 :, 4, 0, 0, 0,#f 31 ;
creasen 16, 70, 0, 1 : 8 9 0 8 | , 1 : 9 15 0 |, 4, 0, 0, 0 ;
creasen 17, 70, 0, 1 : 15 15 0 9 | , 1 : 8 9 0 |, 4, 0, 0, 0 ;
creasen 18, 169, 50, 0 : , suc 1 : 0 0 0 19 |, 4, 0, 0, 0,#f 30 ;
creasen 19, 188, 3, ante 2 : 29 18 , suc 1 : 0 0 0 20 |, 5, 0, 0, 0 ;
creasen 20, 146, 1, ante 1 : 19 , 0 :, 5, 0, 0, 0,#f 22 ;
creasen 21, 74, 0, 2 : 9 15 0 10 | 14 15 0 11 | , 1 : 19 0 0 12 |, 5, 0, 0, 0 ;
creasen 22, 162, 50, 1 : 19 0 0 13 | , suc 2 : 3 0 0 23 | 1 0 0 30 |, 5, 0, 0, 0,#f 20 ;
creasen 23, 144, 1, ante 1 : 22 , 0 :, 5, 0, 0, 0,#f 29 ;
creasen 24, 25, 48, 1 : 1 0 0 14 | , 0 :, 5, 0, 0, 0 ;
creasen 25, 70, 0, 1 : 8 9 0 15 | , 1 : 12 9 0 |, 5, 0, 0, 0 ;
creasen 26, 20, 0, 2 : 3 15 0 16 | 12 9 0 | , 1 : 13 15 0 |, 5, 0, 0, 0 ;
creasen 27, 66, 0, 2 : 9 15 0 17 | 15 15 0 18 | , 1 : 9 15 0 19 |,#m 24,2, 5, 0, 0, 0 ;
creasen 28, 65, 0, 2 : 8 9 0 20 | 15 9 0 21 | , 1 : 8 9 0 22 |,#m 25,2, 5, 0, 0, 0 ;
creasen 29, 160, 2, 0 : , suc 1 : 0 0 0 19 |, 5, 0, 0, 0,#f 23 ;
creasen 30, 153, 1, ante 1 : 22 , 0 :, 5, 0, 0, 0,#f 18 ;
creasen 31, 160, 2, 0 : , suc 1 : 0 0 0 143 |, 5, 0, 0, 0,#f 15 ;
creasen 32, 144, 1, ante 1 : 4 , 0 :, 4, 0, 0, 0,#f 47 ;

creasen 33, 165, 50, 1 : 8 11 10 23 | , suc 4 : 1 0 0 34 | 1 1 0 37 | 1 2 0 40 | 1 3 0 43 |,
    4, 0, 0, 0,#f 46 ;
creasen 34, 144, 1, ante 1 : 33 , 0 :, 4, 0, 0, 0,#f 36 ;
creasen 35, 70, 0, 1 : 14 15 0 24 | , 1 : 9 15 0 |, 4, 0, 0, 0 ;
creasen 36, 160, 2, 0 : , suc 1 : 0 0 0 46 |, 4, 0, 0, 0,#f 34 ;
creasen 37, 144, 1, ante 1 : 33 , 0 :, 4, 0, 0, 0,#f 39 ;
creasen 38, 65, 0, 2 : 9 15 0 25 | 15 15 0 26 | , 1 : 9 15 0 27 |,#m 24,1, 4, 0, 0, 0 ;
creasen 39, 160, 2, 0 : , suc 1 : 0 0 0 46 |, 4, 0, 0, 0,#f 37 ;
creasen 40, 144, 1, ante 1 : 33 , 0 :, 4, 0, 0, 0,#f 42 ;
creasen 41, 66, 0, 2 : 9 15 0 28 | 15 15 0 29 | , 1 : 9 15 0 30 |,#m 24,2, 4, 0, 0, 0 ;
creasen 42, 160, 2, 0 : , suc 1 : 0 0 0 46 |, 4, 0, 0, 0,#f 40 ;
creasen 43, 144, 1, ante 1 : 33 , 0 :, 4, 0, 0, 0,#f 45 ;
creasen 44, 64, 0, 1 : 9 15 0 31 | , 1 : 9 15 0 32 |,#m 25,1, 4, 0, 0, 0 ;
creasen 45, 160, 2, 0 : , suc 1 : 0 0 0 46 |, 4, 0, 0, 0,#f 43 ;
creasen 46, 149, 49, ante 4 : 45 42 39 36 , 0 :, 4, 0, 0, 0,#f 33 ;
creasen 47, 160, 2, 0 : , suc 1 : 0 0 0 143 |, 4, 0, 0, 0,#f 32 ;
creasen 48, 144, 1, ante 1 : 4 , 0 :, 4, 0, 0, 0,#f 51 ;
creasen 49, 70, 0, 1 : 8 9 0 33 | , 1 : 12 9 0 |, 4, 0, 0, 0 ;
creasen 50, 19, 0, 2 : 13 15 0 34 | 12 9 0 | , 1 : 9 15 0 |, 4, 0, 0, 0 ;
creasen 51, 160, 2, 0 : , suc 1 : 0 0 0 143 |, 4, 0, 0, 0,#f 48 ;
creasen 52, 144, 1, ante 1 : 4 , 0 :, 4, 0, 0, 0,#f 56 ;
creasen 53, 70, 0, 1 : 8 9 0 35 | , 1 : 12 9 0 |, 4, 0, 0, 0 ;
creasen 54, 19, 0, 2 : 13 15 0 36 | 12 9 0 | , 1 : 20 15 0 |, 4, 0, 0, 0 ;
creasen 55, 65, 0, 2 : 9 15 0 37 | 20 15 0 38 | , 1 : 9 15 0 39 |,#m 24,1, 4, 0, 0, 0 ;
creasen 56, 160, 2, 0 : , suc 1 : 0 0 0 143 |, 4, 0, 0, 0,#f 52 ;
creasen 57, 144, 1, ante 1 : 4 , 0 :, 4, 0, 0, 0,#f 61 ;
creasen 58, 70, 0, 1 : 8 9 0 40 | , 1 : 12 9 0 |, 4, 0, 0, 0 ;
creasen 59, 19, 0, 2 : 13 15 0 41 | 12 9 0 | , 1 : 21 15 0 |, 4, 0, 0, 0 ;
creasen 60, 66, 0, 2 : 9 15 0 42 | 21 15 0 43 | , 1 : 9 15 0 44 |,#m 24,2, 4, 0, 0, 0 ;
```

```
creasen 61, 160, 2, 0 : , suc 1 : 0 0 0 143 |, 4, 0, 0, 0,#f 57 ;
creasen 62, 144, 1, ante 1 : 4 , 0 :, 4, 0, 0, 0,#f 65 ;
creasen 63, 70, 0, 1 : 8 9 0 45 | , 1 : 12 9 0 |, 4, 0, 0, 0 ;
creasen 64, 20, 0, 2 : 9 15 0 46 | 12 9 0 | , 1 : 13 15 0 |, 4, 0, 0, 0 ;
creasen 65, 160, 2, 0 : , suc 1 : 0 0 0 143 |, 4, 0, 0, 0,#f 62 ;
creasen 66, 144, 1, ante 1 : 4 , 0 :, 4, 0, 0, 0,#f 70 ;
creasen 67, 70, 0, 1 : 8 9 0 47 | , 1 : 12 9 0 |, 4, 0, 0, 0 ;
creasen 68, 19, 0, 2 : 13 15 0 48 | 12 9 0 | , 1 : 22 15 0 |, 4, 0, 0, 0 ;
creasen 69, 49, 16, 2 : 9 15 0 49 | 22 15 0 50 | , 1 : 9 15 0 51 |, 4, 0, 0, 0 ;
creasen 70, 160, 2, 0 : , suc 1 : 0 0 0 143 |, 4, 0, 0, 0,#f 66 ;
creasen 71, 144, 1, ante 1 : 4 , 0 :, 4, 0, 0, 0,#f 105 ;
creasen 72, 165, 50, 1 : 8 11 10 52 | , suc 4 : 1 0 0 73 | 1 1 0 76 | 1 2 0 86 | 1 3 0 95 |,
    4, 0, 0, 0,#f 104 ;
creasen 73, 144, 1, ante 1 : 72 , 0 :, 4, 0, 0, 0,#f 75 ;
creasen 74, 70, 0, 1 : 8 9 0 53 | , 1 : 10 9 0 |, 4, 0, 0, 0 ;
creasen 75, 160, 2, 0 : , suc 1 : 0 0 0 104 |, 4, 0, 0, 0,#f 73 ;
creasen 76, 144, 1, ante 1 : 72 , 0 :, 4, 0, 0, 0,#f 85 ;
creasen 77, 73, 0, 2 : 9 15 0 54 | 14 15 0 55 | , 1 : 23 0 0 56 |, 4, 0, 0, 0 ;
creasen 78, 165, 50, 1 : 23 0 0 57 | , suc 2 : 3 0 0 79 | 1 0 0 82 |, 4, 0, 0, 0,#f 84 ;
creasen 79, 144, 1, ante 1 : 78 , 0 :, 4, 0, 0, 0,#f 81 ;
creasen 80, 70, 0, 1 : 8 9 0 58 | , 1 : 10 9 0 |, 4, 0, 0, 0 ;
creasen 81, 160, 2, 0 : , suc 1 : 0 0 0 84 |, 4, 0, 0, 0,#f 79 ;
creasen 82, 144, 1, ante 1 : 78 , 0 :, 4, 0, 0, 0,#f 83 ;
creasen 83, 160, 2, 0 : , suc 1 : 0 0 0 84 |, 4, 0, 0, 0,#f 82 ;
creasen 84, 149, 49, ante 2 : 83 81 , 0 :, 4, 0, 0, 0,#f 78 ;
creasen 85, 160, 2, 0 : , suc 1 : 0 0 0 104 |, 4, 0, 0, 0,#f 76 ;
creasen 86, 144, 1, ante 1 : 72 , 0 :, 4, 0, 0, 0,#f 94 ;
creasen 87, 165, 50, 1 : 9 15 15 59 | , suc 2 : 3 0 0 88 | 1 0 0 91 |, 4, 0, 0, 0,#f 93 ;
creasen 88, 144, 1, ante 1 : 87 , 0 :, 4, 0, 0, 0,#f 90 ;
creasen 89, 70, 0, 1 : 8 9 0 60 | , 1 : 10 9 0 |, 4, 0, 0, 0 ;
creasen 90, 160, 2, 0 : , suc 1 : 0 0 0 93 |, 4, 0, 0, 0,#f 88 ;
creasen 91, 144, 1, ante 1 : 87 , 0 :, 4, 0, 0, 0,#f 92 ;
creasen 92, 160, 2, 0 : , suc 1 : 0 0 0 93 |, 4, 0, 0, 0,#f 91 ;
creasen 93, 149, 49, ante 2 : 92 90 , 0 :, 4, 0, 0, 0,#f 87 ;
creasen 94, 160, 2, 0 : , suc 1 : 0 0 0 104 |, 4, 0, 0, 0,#f 86 ;
creasen 95, 144, 1, ante 1 : 72 , 0 :, 4, 0, 0, 0,#f 103 ;
creasen 96, 165, 50, 1 : 7 0 0 61 | , suc 2 : 3 0 0 97 | 1 0 0 100 |, 4, 0, 0, 0,#f 102 ;
creasen 97, 144, 1, ante 1 : 96 , 0 :, 4, 0, 0, 0,#f 99 ;
creasen 98, 70, 0, 1 : 8 9 0 62 | , 1 : 10 9 0 |, 4, 0, 0, 0 ;
creasen 99, 160, 2, 0 : , suc 1 : 0 0 0 102 |, 4, 0, 0, 0,#f 97 ;
creasen 100, 144, 1, ante 1 : 96 , 0 :, 4, 0, 0, 0,#f 101 ;
creasen 101, 160, 2, 0 : , suc 1 : 0 0 0 102 |, 4, 0, 0, 0,#f 100 ;
creasen 102, 149, 49, ante 2 : 101 99 , 0 :, 4, 0, 0, 0,#f 96 ;
creasen 103, 160, 2, 0 : , suc 1 : 0 0 0 104 |, 4, 0, 0, 0,#f 95 ;
creasen 104, 149, 49, ante 4 : 103 94 85 75 , 0 :, 4, 0, 0, 0,#f 72 ;
creasen 105, 160, 2, 0 : , suc 1 : 0 0 0 143 |, 4, 0, 0, 0,#f 71 ;
creasen 106, 144, 1, ante 1 : 4 , 0 :, 4, 0, 0, 0,#f 117 ;
creasen 107, 165, 50, 1 : 8 11 10 63 | , suc 3 : 1 0 0 108 | 1 1 0 111 | 3 0 0 114 |, 4, 0, 0, 0,
    #f 116 ;
creasen 108, 144, 1, ante 1 : 107 , 0 :, 4, 0, 0, 0,#f 110 ;
creasen 109, 70, 0, 1 : 8 9 0 64 | , 1 : 9 15 0 |, 4, 0, 0, 0 ;
creasen 110, 160, 2, 0 : , suc 1 : 0 0 0 116 |, 4, 0, 0, 0,#f 108 ;
creasen 111, 144, 1, ante 1 : 107 , 0 :, 4, 0, 0, 0,#f 113 ;
creasen 112, 65, 0, 2 : 9 15 0 65 | 8 9 0 66 | , 1 : 9 15 0 67 |,#m 24,1, 4, 0, 0, 0 ;
creasen 113, 160, 2, 0 : , suc 1 : 0 0 0 116 |, 4, 0, 0, 0,#f 111 ;
```

```
  creasen 114, 144, 1, ante 1 : 107 , 0 :, 4, 0, 0, 0,#f 115 ;
  creasen 115, 160, 2, 0 : , suc 1 : 0 0 0 116 |, 4, 0, 0, 0,#f 114 ;
  creasen 116, 149, 49, ante 3 : 115 113 110 , 0 :, 4, 0, 0, 0,#f 107 ;
  creasen 117, 160, 2, 0 : , suc 1 : 0 0 0 143 |, 4, 0, 0, 0,#f 106 ;
  creasen 118, 144, 1, ante 1 : 4 , 0 :, 4, 0, 0, 0,#f 142 ;
  creasen 119, 169, 50, 0 : , suc 1 : 0 0 0 120 |, 4, 0, 0, 0,#f 141 ;
  creasen 120, 188, 3, ante 2 : 140 119 , suc 1 : 0 0 0 121 |, 6, 0, 0, 0 ;
  creasen 121, 146, 1, ante 1 : 120 , 0 :, 6, 0, 0, 0,#f 123 ;
  creasen 122, 74, 0, 2 : 8 9 0 68 | 14 9 0 69 | , 1 : 24 0 0 70 |, 6, 0, 0, 0 ;
  creasen 123, 162, 50, 1 : 24 0 0 71 | , suc 2 : 3 0 0 124 | 1 0 0 141 |, 6, 0, 0, 0,#f 121 ;
  creasen 124, 144, 1, ante 1 : 123 , 0 :, 6, 0, 0, 0,#f 140 ;
  creasen 125, 66, 0, 2 : 8 9 0 72 | 15 9 0 73 | , 1 : 8 9 0 74 |,#m 25,3, 6, 0, 0, 0 ;
  creasen 126, 165, 50, 1 : 8 11 10 75 | , suc 4 : 1 0 0 127 | 1 1 0 130 | 1 2 0 133 | 1 3 0 136 |,
      6, 0, 0, 0,#f 139 ;
  creasen 127, 144, 1, ante 1 : 126 , 0 :, 6, 0, 0, 0,#f 129 ;
  creasen 128, 4, 0, 2 : 9 15 15 76 | 9 15 1 77 | , 1 : 9 15 0 |, 6, 0, 0, 0 ;
  creasen 129, 160, 2, 0 : , suc 1 : 0 0 0 139 |, 6, 0, 0, 0,#f 127 ;
  creasen 130, 144, 1, ante 1 : 126 , 0 :, 6, 0, 0, 0,#f 132 ;
  creasen 131, 4, 0, 2 : 25 0 0 78 | 9 15 1 79 | , 1 : 9 15 0 |, 6, 0, 0, 0 ;
  creasen 132, 160, 2, 0 : , suc 1 : 0 0 0 139 |, 6, 0, 0, 0,#f 130 ;
  creasen 133, 144, 1, ante 1 : 126 , 0 :, 6, 0, 0, 0,#f 135 ;
  creasen 134, 4, 0, 2 : 9 0 0 80 | 9 15 1 81 | , 1 : 9 15 0 |, 6, 0, 0, 0 ;
  creasen 135, 160, 2, 0 : , suc 1 : 0 0 0 139 |, 6, 0, 0, 0,#f 133 ;
  creasen 136, 144, 1, ante 1 : 126 , 0 :, 6, 0, 0, 0,#f 138 ;
  creasen 137, 4, 0, 2 : 9 7 0 82 | 9 15 8 83 | , 1 : 9 15 0 |, 6, 0, 0, 0 ;
  creasen 138, 160, 2, 0 : , suc 1 : 0 0 0 139 |, 6, 0, 0, 0,#f 136 ;
  creasen 139, 149, 49, ante 4 : 138 135 132 129 , 0 :, 6, 0, 0, 0,#f 126 ;
  creasen 140, 160, 2, 0 : , suc 1 : 0 0 0 120 |, 6, 0, 0, 0,#f 124 ;
  creasen 141, 153, 1, ante 1 : 123 , 0 :, 6, 0, 0, 0,#f 119 ;
  creasen 142, 160, 2, 0 : , suc 1 : 0 0 0 143 |, 6, 0, 0, 0,#f 118 ;
  creasen 143, 149, 49, ante 12 : 142 117 105 70 65 61 56 51 47 31 14 9 , 0 :, 7, 0, 0, 0,#f 4 ;
  creasen 170, 168, 2, 0 : , suc 1 : 0 0 0 146 |, 7, 0, 0, 0,#f 171,#c 1 ;
  creasen 171, 136, 1, ante 0 : , 0 :, 8, 0, 0, 0,#f 170 ;
  creasen 172, 160, 2, 0 : , suc 1 : 0 0 0 173 |, 8, 0, 0, 0,#f 147 ;
  creasen 173, 195, 3, ante 1 : 172 , suc 0 :, 8, 0, 0, 0,#f 146,#c 1 ;
  creasen 174, 193, 3, ante 0 : , suc 0 :, 9, 0, 0, 0,#f 1,#c 1 ;
  end
 end
end
```

APENDICE D

CODIGOS DE LAS SENTENCIAS DE LA BASE DE DATOS DEL SISTEMA PSAL2

Los códigos de operación se clasifican, en función del valor del primer bits (de los ocho que lo forman), en dos grupos:

Bit 0	Tipo
0	Operaciones de datos
1	Operaciones de control

Las operaciones con datos se clasifican a su vez dependindo del valor del bit 1 en dos grupos:

Bit 1	Tipo
0	Operaciones varias
1	Operaciones aritméticas y relacionales.

Los códigos del primer grupo, se clasifican a su vez en función de los bits 2º y 3º. De esta forma:

Bits 2:3	Tipo
00	Tipo 1
01	Tipo 2
10	Shift y rotación
11	Lógica

Los códigos aritméticos y relacionales se dividen en dos clases (según el valor del bit 4º):

Bit 4	Tipo
0	Aritmético
1	Relacional

Los códigos aritméticos (4º bit = 0) utilizan los bits 2º y 3º, para señalar en que representación se realizan las operaciones. Así:

Bits 2:3	Tipo
0	Complemento 1
1	Complemento 2
2	Magnitudes con bit de signo
3	Magnitudes sin signo

Una vez descritos los tipos de códigos, veamos las sentencias que pueden formar parte de la base de datos:

a) Códigos tipo 1:

Se expresa el valor de los bits 4:7 en octal.:

Bits 2:7	Código	Ejemplo
01	clear	
02	incrementa	
03	decrementa	
04	concatenación	
05	carga dato sin signo	
10	retraso.	
11	inicia ejecución de caja negra.	
12	carga resultado de caja negra.	
13	lee dato de caja negra.	
14	escribe dato de caja negra.	

b) Códigos tipo 2

Utilizaremos el mismo formato que en el caso anterior. Los códigos que forman

esta clase son:

Bits 4:7	Código	Ejemplo
00	noop	
01	escribe bit (decodificador).	
02	lee bit (multiplexor).	
03	lee de memoria	
04	escribe en memoria	
05	lee variable con multiples indices.	
06	escribe en variable con multiples indices	
10	wait on	
11	wait until	

c) Códigos de desplazamiento y rotación. Se utilizan los nombres que ISPS utilizaba para estas operaciones:

Bits 4:7	Código
00	sl0
01	sl1
02	slr
03	sld
04	sr0
05	sr1
06	srr
07	srd
10	sli
11	sri

d) Códigos de operaciones lógicas

Bits 4:7	Código
00	not
01	and
02	or
03	nand
04	nor
05	xor
06	xnor

e) Códigos de operaciones aritméticas
Estos códigos se asignan en función del valor de los bits 5:7.

Bits 5:7	Código
0	neg
1	+
2	-
3	*
4	/
5	resto
6	carga con signo

f) Códigos de operaciones relacionales

Bits 5:7	Código
0	test (en el sentido ya comentado en PSAL1)
1	=
2	|=
3	<
4	<=
5	>
6	>=

Antes de pasar a describir dichos códigos, combiene señalar que siempre que aparecen "operandos" de control, aparecen otros nudos que tienen igualmente operandos de control y que "cierran" al anterior, es decir, si en primero tenia como resultado control (ejemplo: una operación if), el segundo tiene por datos control (en el ejemplo un nudo equivalente al smerge en el formato intermedio de PSAL1). Las operaciones de control se dividen en dos grupos en función del valor del bit 1:

Bit 1	Tipo
0	Operaciones estructuradas
1	Operaciones no estructuradas y especiales

Las operaciones del primer tipo se dividen a su vez en 4 subtipos:

Bits 2:3	Tipo
0	Operaciones con control de entrada y datos de salida. Se caracterizan por no ejecutar nunca. Un ejemplo puede ser la operacion que hay despues de un return.
1	Operaciones con entrada de control y salida de datos. Por ejemplo el nudo en donde se unen las ramas de un if.
2	Operaciones con entrada de datos y salida de control. Ejemplo: una operación if.
3	Operaciones con control de entrada y salida. Ejemplo: un lazo.

Las operaciones estrucutras se dividen ademas en función de los bits 4:7 en:

Bit 4:7	Código
0	Bloque
1	Bloque de inicialización en for
2	Bloque condicionale en lazo.
4	fork
5	decode
6	call
7	return
8	recall
9	while
10	do-while
11	for
12	loop
13	break
14	next.

Las operaciones especiales se dividen en :

Bits 3:7	Código
0	Inicio de descripción
1	Fin de descripción
2	Inicio de entidad
3	Fin de entidad

Las operaciones especiales se clasifican en 4 tipos:

Bit 3:6	Tipo
0	Stop
1	Salto no estructurado hacia adelante
2	Salto no estructurado hacia atras
3	Entrada no estructurada.

Todas ellas se componen de dos operaciones una de inicio y otra de final de la acción dependenciendo del bit 7:

Bit 7	Código
0	begin
1	end.

BIBLIOGRAFIA

[Ba73] M.R. Barbacci; "Automated Exploration of the Design Space for Register Transfer (RT) systems"; Ph.D. Preliminar Proposal. Univ. Carnegie Mellon. Noviembre 1973.

[COM77] Computer. Junio 1977.

[SiTh77] D.P. Siewiorek, D.E. Thomas; "Measuring Designer Performance to verify Design Automation Systems"; Procs. Design Automation Conference. Vol 14. 1977.

[PaSi78] A. Parker, D.P. Siewiorek, M.R. Barbacci,L. Hafer, G. Leive, J. Kim; "The CMU Design Automation Systems"; Proc 16th Design Automation Conference. Junio 1978.

[SnSi78] E. Snow, D.P. Siewiorek, D.E. Thomas; "A Technology-Relative Computer-Aided Design Systems: Abstract Representation, Transforms and Design Tradeoffs"; Procs. of the15th. Design Automation Conference. Junio .1978.

[BaSh79] U. Banerjee, C. Shyh-Ching,D.j. Kuck,R.A. Towle; "Time and Parallel Processor Bounds for Fortran-like Loops"; IEEE Trans. on Computers, VOL. C-26,No 9. Septiembre 1979.

[Di79] D. Dietmeyer; "Logi Design of Digital Systems"; Allyn and Bacon,Inc. 1979.

[AlOl80] S. Allan,A. Oldehoeft; "A Flow Analysis Procedure for the Translaton of High-Level Languages to a Data Flow Language". IEEE Trans. on Computers. VOL.C-29.No9. Septiembre 1980.

[BaNo80] M. R. Barbacci, J. D. Northcutt; "Applications of ISPS, an Architecture Description Languaje". Journal of Digital System. Volumen IV.1980.

[PaKu80] D.A. Padua, D.J. Kuck,D.H. Lawrie; "High-speed multiprocessors and compilations techniques"; IEEE Trans. on Computers; VOL.C-29,No.9. Septiembre 1980.

[Zi80] G. Zimmermam; "MDS- The MIMOLA Design Method"; Journal of Digital Systems. Volumen IV.No. 3. 1980.

[Ba81] M.R. Barbacci; "Instruction Set Processor Specification (ISPS): The notation and its applications"; IEEE Trans. on Computers, VOL.C-30,No 1, Enero 1981.

[Ch81] G.J. Chaitin at al.; " Register Allocation via Coloring"; Computer Languajes, Vol 6, No 1; 1981.

[DiPa81] S.W. Director, A. Parker, D.P. Siewiorek, D.E. Thomas; " A Design Methodology and Computers Aids for Digital VLSI Sysytems"; IEEE Trans. on Circuits and Systems. VOL. CAS-28.NO 7, Julio 1981.

[SiBe82] D.P. Siewiorek, C.G. Bell,A. Newell; "Computer Structures: Principles and Examples"; McGraw Hill. 1982.

[HaPa83] L.J. Hafer, A.C. Parker; "A Formal Method for the Specification, Analysis and Design of Register-Transfer Level Digital Logic"; IEEE Trans. on Computer-Aided Design. Vol CAD-2, No 1, Enero 1983.

[MF83] M.C. McFarland; "Computer-Aided Partitioning of Behavioral Hardware Description"; Procs. 20th Design Automation Conference. Junio 1983.

[Ch84] F. Chow,J. Hennessy; "Register Allocation by Priority-based Coloring"; Proc. ACM SIGPLAN'84 Symp. on Compiler Construction. Vol 19. Number 6; Junio 1984.

[Fo84] R. Forsyth; " Expert System : Principles and case studies"; Chapman and Hall, Ltd; 1984.

[HwBr84] K. Hwang, F. Briggs; "Compter Architecture and Parallel Processing"; McGraw-Hill. 1984. Capítulo 10.

[KoTh84] T.J. Kowalski, D.E. Thomas; "The VLSI Design Automation Assistant: An IBM System/370 Design"; IEEE Design & Test. Febrero 1984.

[TsSi84] C.J. Tseng,D. Siewiorek; "Emeral: A Bus Style Desiner"; 21st Design Automation Conference. 1984.

[Ba85] T. Backman et al.; "The Silc compiler: Language and features"; Proc. 22nd Design Automation Conference. 1985.

[JaJe85] R. Jamier, A.A. Jerraya; "APOLLON, A Data-path Silicon Compiler"; IEEE Circuits and Devices Magazine; Mayo 1985.

[Ko85] T.J. Kowalski; "An Artificial Intelligence Approach to VLSI Design"; Kluwer Academic Publishers. 1985.

[KoTh85a] T.J. Kowalski, D.E. Thomas; "The VLSI Design Automation Assistant: What's in a Knowledge Base"; IEEE Proc. 22nd Design Automation Conference. 1985.

[KoGe85] T.J. Kowalski, D. Geiger,W. Wolf,W. Fichtner; "The VLSI Design Automation Assistant: From Algorithms to Silicon"; IEEE Design & Test. Agosto. 1985.

[LaMo85] M. Lam, J. Mostow; "A Transformational Model of VLSI Systolic Design"; IEEE Computer. Febrero 1985.

[Sh85] M. Shahdad et al.;"VHSIC Hardware Description Language"; Computer; Febrero

1985.

[Ue85] T. Uehara; "A Knowledge-based Logic Design System"; IEEE Design & Test; Octubre 1985.

[AhSe86] A.V. Aho,R. Sethi,J.D. Ullman; "Compilers: Principles, Techniques and Tools"; Addison Wesley. 1986.

[AlBr86] M. A. Allende, S. Bracho, E. Villar; "Descripción y simulación de sistemas digitales en las etapas iniciales de diseño mediante las herramientas implemantadas sobre el lenguaje DDL: Estudio de un filtro digital"; Actas de las II Jornadas de Electrónica Militar; Febrero , 1986.

[Ba86] A. Baker; "Selecting a Silicon Compiler"; VLSI System Design; Mayo 1986.

[CVS86] "Software Catalogue of the CVS Proyect"; University of Kaiserslautern. Mayo 1986.

[GeMo86] E. Gelenbe, E. Montagne; "A Performance Model of Block Structured Parallel Programs", en "Parallel Algorithms & Arquitectures"; North-Holland. 1986.

[DM86] H. De Man et al.; "Cathedral II: A silicon compiler for digital signal processing"; IEEE Design & Test. Diciembre 1986.

[LiMa86] R. Lipsett, E. Marschner,M. Shahdad; "VHDL- The Language"; IEEE Design & Test; Abril 1986.

[MoFo86] D. I. Moldovan, A.B. Fortes; "Partitioning and Mapping Algorithms into Fixed Size Systolic Arrays"; IEEE Trans. on Computer. VOL. C-35,No 1. Enero 1986.

[NaSa86] J. Nash, L. Saunders; "VHDL Critique"; IEEE Design and Test. Abril.1986.

[NeSV86] A.R. Newton,A.L. Sangiovanni-Vincentelli; "Computer-Aided Design for VLSI Circuits"; IEEE Computer; Abril 1986.

[Pa86] N. Park et al.; "MAHA: A program for data path synthesis"; 23rd Design Automation Conference. Julio 1986.

[Su86] D.A. Subrahmanyam; "Synapse: An Expert System for VLSI Design"; Computer; Julio 1986.

[Th86] D.E. Thomas; "Automated Synthesis of Data Path in Digital System"; en "Design Methodologies". Advances in CAD for VLSI; Vol 6, North-Holland. 1986.

[TsSi86] C. Tseng, D.P. Siewiorek; "Automated synthesis of data paths in digital systems"; IEEE Trans. Computer-Aided Design; VOL CAD-5, Julio 1986.

[AlBr87] M.A. Allende, S. Bracho,E. Villar; "Modificación del sintetizador lógico DDLSYN para el diseño sindrome-testable"; Actas de las IV Jornadas de Diseño Lógico; Abril. 1987.

[BrRu87] R.L Brayton, R. Rudell, A. Sangiovanni-Vicentelli, A.R. Wang; "MIS: A

Multiple-Level Logic Optimization System"; IEEE Transactions on Computer-Aided Design; Vol. CAD-6; Nº 6; November, 1987.

[GeEl87] C.H. Gebotys,M.I. Elmasy; "A VLSI methodology with testability constraints"; Procs. 1987 Canadian Conf. VLSI. Octubre 1987.

[Gi87] E.F. Girczyc; "Loop winding- A data flow approach to functional pipeling"; Procs. Int. Symp. Circuits Systems. Mayo 1987.

[Go87] R. Goering; "Silicon Compilers bridge gap between concepts and silicon"; Computer Design; Noviembre 1, 1987.

[Mo87] D. Moldovan; "ADVIS: A Software Package for the Design of Systolic Arrays"; IEEE Trans. on Computer-Aided Design. Vol CAD-6,No 1. Enero 1987.

[NeDe87] A. Newton.S. Devadas; "Algorithms for Hardware Allocation in Data Path Synthesis". IEEE.1987.

[PaGa87a] B.M. Pangrle, D.D. Gajski; "Slicer: A state synthesizer for intelligent silicon compilation"; Procs. IEEE Int. Conf. Computer Design. Octubre 1987.

[PaGa87b] B.M. Pangrle, D.D. Gajski; "Design Tools for Intelligent Silicon Compilation"; IEEE Trans. on Computer-Aided Design. Vol CAD-6. No. 6; Noviembre 1987.

[PaMa87] A.C. Parker, S. Mayati; "Automating the VLSI Design Process Using Expert Systems and Silicon Compilation"; Proc. IEEE, Vol 75,No 6. Junio 1987.

[Sc87] M. Schindler; "Designers Build Smarts into their CAE tools with expert system shell"; Electronic Design; Julio 1987.

[Tr87] H. Trickey; "Flamel: A high-level hardware compiler"; IEEE Trans. Computer-Aided Design. VOL. CAD-6. Marzo 1987.

[Co88] D.R. Coelho; "VHDL: A call for standards"; Procs. 25th ACM/IEEE Design Automation Conference. Junio.1988.

[Ba88] M. Balakrishnan et al.; "Allocation of Multiport Memories in Data Path Synthesis"; IEEE Trans. on Computer-Aided Design. Vol 7. No. 4; Abril 1988.

[Bh88] J. Bhasker; "Process-Graph analyser: A front-end tool for VHDL behavioural synthesis"; Software-practice and Experience. Vol.18(5). Mayo.1988.

[Bo88a] A. Bonomo et al.; "Easily testable data part synthesis in the BACH Silicon Compiler"; 2nd ABAKUS Workshop. Septiembre 1988.

[Bo88b] A. Bonomo et al.; "Control part synthesis in the BACH Silicon Compiler"; 2nd ABAKUS Workshop. Septiembre 1988.

[BoDe88] G. Borriello,E. Detjens;" High-Level Synthesis: Current status and future directions"; Procs. 25th ACM/IEEE Design Automation Conference; 1988.

[BrCa88] R.K. Brayton, R. Camposano, G. De Micheli, R.H. Otten,J. VanEijndhovem;

"The Yorktown Silicon Compiler"; en "Silicon Compilation", D.D. Gajski (ed.); Addison-Wesley. 1988.

[BrMa88] S. Bracho,M. Martinez; "Test funcional de arquitecturas RISC a nivel de su conjunto de instrucciones"; Actas del 3º Simposio de Electrónica de las Comunicaciones; Oporto. 1988.

[Co88] D.R. Coelho; "VHDL: A Call for Standards"; Procs. 25th ACM/IEEE Design Automation Conference. 1988.

[CVS88] CVS_BK. Language Reference Manual. Kaiserslautern University. Junio 1988.

[NeHi88] S. Nevadas, M. Hi-Keung,R. Newton, A. Sangiovanni-Vicentelli; "MUSTANG: State Assignment of Finite State Machines Targeting MultiLevel Logic Implementations"; IEEE Transactions on Computer-Aided Design, Vol. 7, Nº 12, December, 1988.

[DM88] H. De Man et al.; "Cathedral-II: A Synthesis System for Multiprocessor DSP Systems"; Capitulo 8 de "Silicon Compilation"; D.D. Gajski (ed.); Addison-Wesley. 1988.

[DMKu88] G. De Micheli,D.C. Ku; "Hercules- A system for High-Level Synthesis"; Procs 25th ACM/IEEE DEsign Automation Conference. 1988.

[HaLe88] R.W Hartenstein,K. Lemmert; "A Systolic Design System Using KARL"; 2 ABAKUS Workshop. Innsbruck-Igls (Austria).Septiember 1988.

[Ku88] S. Y. Kung;"VLSI Array Processors"; Prentice Hall; 1988.

[Li88] J. Lis, D.D. Gajski; "Synthesis from VHDL"; Proc. of the IEEE Int. Conf. on Computer Design, VLSI in Computers and Processors, ICCD,1988.

[MFPa88] M.C. McFarland,A.C. Parker,R. Camposano; "Tutorial on High-Level Synthesis"; Procs 25th ACM/IEEE Design Automation Conference. Junio 1988.

[ML88] J. McLeod; "Logic Synthesizers step up to hierarchical design"; Electronics, Junio 1988.

[PaPa88a] N. Park, A. Parker; "Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications"; IEEE Trans. on Computer-Aided Design. Vol 7. No 3. Marzo 1988.

[PaPa88b] N. Park, A.C. Parker; "Theory of clocking for maximum execution overlap of High-speed Digital Systems"; IEEE Trans. on Computers; VOL.37,No 6, Junio 1988.

[Po88] C.D. Polychronopoulos; "Compiler optimizations for enhancing parallelism and their impact on architecture design"; IEEE Trans. on Computers; VOL.37,No.8. Agosto 1988.

[Ro88] R. Rohrer; "Evolution of the Electronic Design Automation Industry"; IEEE Design & Test; Diciembre 1988.

[SeSo88] J.Setien,A.Sotelo,D.Mozos,F.Tirado,M.Fernández; "Behavioral Specification and Internal Representation in FIDIAS System"; IASTED International Symposium APPLIED Informatics; Febrero 1988.

[ShTa88] K. Shirai,T. Takezawa; "Expert system for designing digital signal processor architectures"; Microprocessors and Microsystems. Vol 12. No 2. Marzo 1988.

[Th88] D.E. Thomas et al.; "The System Architect's Workbench"; Procs. 25th ACM/IEEE Design Automation Conference. 1988.

[BaMa89] M. Balakrishnan, P. Marwedel; "Integrated scheduling and binding. A synthesis approach for design space exploration"; Proc. 26th ACM/IEEE Design Automation Conference. 1989.

[Ben89] "Benchmarks for the Fourth International Workshop on High-Level Synthesis",1989. Disponible atraves de Email en HLSW@cs.wisc.edu (InterNet).

[Bi89] W.P. Birmingham; "The MICON System for Computer Design"; Proc. 26th ACM/IEEE Design Automation Conference. 1989.

[BoIt89] A. Bonomo,M. Italiano,M. Maggiulli,M. Melgara,M. Paolini,I. Stamelos; "BACH: Behavioural-level Automated Compilation Hardware"; CSELT Technical Reports. Vol XVII. No 1. Febrero 1989.

[Ca89] R. Camposano; "Behavior-Preserving Transformations for High-Level Synthesis"; Procs. Workshop on Hardware Specification and Synthesis: Mathematical Aspects, Springler Verlag, 1989.

[CaBe89] R. Camposano,R.A. Bergamaschi; "Synthesis using Path-based Scheduling: Algorithms and Exercices"; High-Level Synthesis Workshop, Maine. Octubre 1989.

[CaRo89] R. Camposano,W. Rosenstiel; "Synthesizing Circuits from Behavioral Description"; IEEE Trans. on Computers-Aided Design. VOL.8.No 2. Febrero 1989.

[CaTa89] R. Camposano, R. M. Tabet; "Design Representation for the Synthesis of Behavioral VHDL Models"; CHDL'89.

[Co89] B. C. Cole; "Making the right moves in ASICs"; Electronics; Noviembre 1989.

[DA89] M D'Abreu; "Putting Synthesis to Work"; IEEE Design & Test; Abril 1989.

[DeNe89] S. Devadas,R. Newton; "Algorithms for Hardware Allocation in Data Path Synthesis"; IEEE Trans. on Computer-Aided Design, Vol 8, No. 7; Julio 1989.

[DM89] H. De Man et al.; "Loop optimization in register-transfer scheduling for DSP-systems"; Procs. 26th ACM/IEEE Design Automation Conference. 1989.

[HaEl89] B. Haroum,I. Elmasry; "Architectural Synthesis for DSP Silicon Compilers"; IEEE Trans. on Computer-Aided Design. Vol. 8,No. 4. Abril 1989.

[HaPa89] S. Hayati,A. Parker; "Automatic Production of Controller Specifications from

Control and Timing Behavioral Descriptions"; Procs. 26th ACM/IEEE Design Automation Conference.1989.

[IDT89a]"CAD for System Design: Is it practical?"; IEEE Design & Test; Abril 1989.

[IDT89b] "Expert system in CAD"; A D&T Roundtable; IEEE Design & Test; Agosto 1989.

[KuKu89] A. Kumar,S. Kumar,P. Kulshreshtha,S Ghose: "Automatic Synthesis of Microprogrammed Control Units from Behavioral Descriptions"; Procs. 26th ACM/IEEE Design Automation Conference.1989.

[Ma89] M.C. Markowitz; "Logic synthesis prepares for VHDL"; EDN; 30 de Marzo. 1989.

[Me89a] E. Meyer; "VHDL opens the road to top-down design"; Computer Design; 1 de Febrero. 1989.

[Me89b] E. Meyer; "VHDL strives to cover both synthesis and modeling"; Computer Design. 1 de octubre. 1989.

[ML89] J.McLeod: "A gigant leap for simulation"; Electronics. Febrero. 1989.

[NuMe89] G. M. Nurie,P.J. Menchini; "VHDL Model Portability"; High Performance Systems; Julio 1989.

[Le89] S.H. Leibson; "VHDL"; EDN.16 de Marzo.1989.

[Pa89a] A. Parker et al.;" Experience with the ADAM Synthesis System"; Proc. 26th ACM/IEEE Design Automation Conference. Las Vegas. 1989.

[Pa89b] P. Paulin; "Scheduling and Binding Algorithms for High-level Synthesis"; Procs. 26th ACM/IEEE Design Automation Conference. Las Vegas. 1989.

[PaKn89a] P. Paulin,J. Knight; "Forced-directed Scheduling for the Behavioral Synthesis of ASICs"; IEEE Trans. on Computer-Aided Design. Vol 8.No 6. Junio 1989.

[PaKn89b] P. Paulin,J. Knight; "Algorithms for High-level Synthesis"; IEEE Design & Test; Diciembre 1989.

[RuTh89] R. Rutenbar,K. Tham; "Design Automation: Even when it's old it's new"; IEEE Design & Test, Diciembre 1989.

[Sa89] P. Sánchez; "Análisis y síntesis de sistemas digitales descritos a nivel algorítmico"; Tesina de Licenciatura. Febrero 1989.

[SaAl89] P. Sanchez,M.A. Allende,M. Martinez,F. Llacer y E. Villar; "Estudio comparativo de dos implementaciones de la Transformada Rápida de Haar"; Actas de las V Jornadas de Diseño de Circuitos Integrados. Pag 11-17, Huelva, Diciembre 1989

[Se89] R.BSegal., "BDSYN Users' Manual Version 1.1", UCB, April, 1989.

[SoSe89] A. Sotelo,J. Setien,R. Hermida,M. Fernández; "An Approach to minimal-time scheduling of micro-operations"; Procs. 15th Symposium on Microprocessing and Microprogramming EUROMICRO'89. 1989.

[Sp89] R.L. Spickelmier, "Octtools Distribution 3.0", UCB, April, 1989.

[St89] J. Straus; "Synthesis from register-transfer level VHDL"; Digest of papers COMPCON SPRING'89, 34th IEEE Int. Conference.1989.

[ThDr89] D.E. Thomas, E. Drikes; "Architectural Partitioning for System Level Design"; Procs. 26th ACM/IEEE Design Automation Conference. 1989.

[Ts89] S. Tsubota; "Satisfying designers' voracious appetite for tools"; High Performance Systems; Mayo 1989.

[VHDL89] "IEEE Standard 1076 VHDL Tutorial"; CAD Language System,Inc. Marzo.1989.

[WaTh89] R. Walker, D. E. Thomas; "Behavioral Transformation for Algorithmic Level IC Design"; IEEE Trans. on Computer-Aided Design; VOL.8, No 10, Octubre 1989.

[Al90] J. Allen; "Performance-Directed Synthesis of VLSI System". Proc. of the IEEE. VOL 78,NO 2, Febrero 1990.

[BePa90] N. Berry,B. Pangrle; "SCHALLOC: An Algorithm for Simultaneous Scheduling & Connectivity Binding in a Datapath Syntheis System"; Procs. of the EDAC. 1990.

[BhLe90] J. Bhasker,H. Lee; "An Optimizer for Hardware Synthesis". IEEE Design & Test. Octubre 1990.

[BrGa90] F. Brewer,D. Gajski;"Chippe: A System for Constraint Driven Behavioral Synthesis". IEEE Trans. on Computer-Aided Design Vol-9,No 7. Julio 1990.

[ClTh90] R. Cloutier,D. Thomas;"The Combination of Scheduling,Allocation and Mapping in a Single Algorithm"; 27th ACM/IEEE Design Automation Conference. 1990.

[DeMA90] A. Delaruelle,O. McArdle,J. van Meerbergen, C. Niessen; "Synthesis of delay functions in DSP compilers"; Proc. of the EDAC. 1990.

[DM90] H. De Man et al.;"Architecture-Driven Synthesis Techniques for VLSI Implementation of DSP Algorithms". Proc. of the IEEE. Vol 78. NO 2. Febrero 1990.

[Ca90] R. Camposano; "From Behavior to Structure: High-Level Synthesis". IEEE Design & Test. Octubre 1990.

[CaBe90] R. Camposano, R. Bergamaschi; "Redesign Using State Splitting"; Procs. of the EDAC. 1990.

[Go90] G. Goossens,J. Rabaey,J. Vandewalle,H. de Man;"An efficient Microcode Compiler for Application Specific DSP Processors". IEEE Trans on Computer-Aided Design. VOL 9. NO 9. Septiembre 1990.

[Ga90] W. Grass; "A Branch-and-Bound Method for Optimal Transformation of Data Flow Graphs for Observing Hardware Constraints"; Procs. of de EDAC. 1990.

[KrRo90] H. Kramer,W. Rosenstiel; "System Synthesis using Behavioural Descriptions";

Procs. of the EDAC. 1990.

[Ma90] P. Marwedel; " Matching System and Component Behaviour in MIMOLA Synthesis Tools"; Procs. of the EDAC. 1990.

[MaDe90] D. Mallon,P. B. Denyer; "A New Approach To Pipeline Optimization"; Procs. of the EDAC. 1990.

[HwHs90] C. Hwang, Y. Hsu,Y. Lin; "Optimum and Heuristic Data Path Scheduling Under Resource Constraints"; 27th ACM/IEEE Design Automation Conference. 1990.

[MFKo90] M. McFarland, T. Kowalski; "Incorporating Bottom-Up Design into Hardware Synthesis"; IEEE Trans. on Computer-Aided Design. LO 9. NO 9. Septiembre 1990.

[MFPa90] M. McFarland,A. Parker, R. Camposano; "The Hihg-Level Synthesis of Digital Systems"; Proc. of the IEEE, Vol 78,NO 2, Febrero 1990.

[Mi90] G. De Micheli; "High-Level Synthesis of Digital Circuits". IEEE Design & Test of Computers.Octubre 1990.

[ML90] McLeod; "Top-down design is the watchword at DAC"; Electronics; Junio 1990.

[PaKo90] C. A. Papachristou, H Konuk; "A Linear Program Driven Scheduling and Allocation Method Followed by an Interconnect Optimization Algorithm"; 27th ACM/IEEE Design Automation Conference. 1990.

[SaDo90] R. Sarma, M. Dooley,N. Craig newman and G. Hetherington; "High-Lvel Synthesis: Technology Transfer To Industry"; 27th ACM/IEEE Design Automation Conference. 1990.

[SaRa90] P. Sánchez,E. Randon,E. Villar; " Some experiences in the use of VHDL in High-Level Synthesis"; Proc. of the First European Conference on VHDL. Marsella. Septiembre 1990.

[SaVi90a] P. Sánchez, E. Villar; "PSAL2: A High-Level Synthesis Program"; IASTED Inter. Symp. Applied Informatics; Febrero 1990.

[SaVi90b] P. Sánchez, E. Villar; "An Iterative Constructive Technique for Operation Allocation in High Level Synthesis"; IASTED Inter. Symp. Applied Informatics; Febrero 1990.

[SaZa90] A. Safir, B. Zavidovique; "Towards a Global Solution to Hig Level Synthesis Poblems"; Procs. of the EDAC. 1990.

[ScGr90] J. Scheichenzuber, Werner Grass,U. Lauther, S. Marz; "Global Hardware Synthesis from Behavioral Dataflow Description"; 27th ACM/IEEE Design Automation Conference. 1990.

[St90] L. Stock; "Interconnect Optimisation During Data Path Allocation"; Procs. of the EDAC 1990.

[Ti90] F. Tirado et al.; "Hardware Allocation in FIDIAS System"; EUROCOMP'90.1990.

[We90] S. Weber; "Here's a Synthesizer that support VHDL"; Electronics. Abril 1990.

[WhNe90] G. Whitcomb, R newton; "Abstract Data Types and High-Level Synhesis"; 27th ACM/IEEE Design Automation Conference. 1990.

[SaVi91] P. Sánchez,E. Villar; "Análisis y síntesis de sistemas digitales descritos a nivel algorítmico"; Revista de Informática y Automática. Aceptado.

[RaSa91] E. Randon,P. Sánchez,E. Villar; "ESP: An Structural Synthesis Program"; EUROMICRO'91. Remitido.