5 IMPLEMENTACIÓN DEL PROCESADOR LEON

Para demostrar el correcto funcionamiento del procesador LEON sobre una FPGA Virtex en una placa de desarrollo XSV, se realiza una implementación totalmente operativa del procesador, adecuada a la placa XSV, incluyendo el mayor número de periféricos disponibles que se pueden ubicar en la FPGA Virtex.

Los recursos físicos disponibles en la FPGA limitan el número de periféricos que se pueden habilitar durante la configuración del LEON. Además, las restricciones que imponen las conexiones existentes en la placa XSV fijan también la configuración y la forma de lanzamiento del procesador (*boot*).

Al no disponer la placa XSV de medios similares, se incorpora una memoria PROM interna en la FPGA para realizar el lanzamiento del procesador LEON tras el *reset*.

La prueba de la implementación operando sobre la placa XSV se realiza mediante el desarrollo de un software en código ensamblador SPARC V8 que inicia el procesador, y hace uso de los puertos de E/S para operar un display.

La implementación se obtiene siguiendo las fases que conforman el flujo de diseño Xilinx, junto con:

- Elección de restricciones de E/S para el fichero UCF, teniendo en cuenta las conexiones en la placa XSV con la memoria RAM y resto de dispositivos.

- Configuración del modelo de procesador LEON, mediante la edición del fichero *device.vhd*.

- Realización de un programa en código ensamblador SPARC V8 y generación de los ficheros necesarios para la incorporación en la memoria PROM interna durante el flujo de diseño.

5.1 RESTRICCIONES (UCF)

Se imponen las restricciones físicas que relacionan los pins de entrada y salida del dispositivo. A continuación se muestran las asignaciones realizadas a las diferentes señales que son necesarias para el funcionamiento.

```
resetn      174   SW1
clk         89
errorn      171   pull-up

address<27>
address<26>
address<25>
address<24>
address<23>
address<22>
address<21>
address<20>
address<19> 229
address<18> 230
address<17> 231
address<16> 232
address<15> 234
address<14> 235
address<13> 236
address<12> 237
address<11> 238
address<10> 187
address<9>  188
address<8>  189
address<7>  191
data<15>
data<14>
data<13>
data<12>
data<11>
data<10>
data<9>
data<8>
data<7>
data<6>
data<5>
data<4>
data<3>
ramoen<1>
ramoen<0>   228
rwen<3>
rwen<2>
rwen<1>
rwen<0>     201
brdyn
bexcn       161   DIPSW1
pio<15>     142   DIPSW7
pio<14>     185   SW4
pio<13>     147   SR6
```

```
address<6>  192
address<5>  193
address<4>  194
address<3>  195
address<2>  199
address<1>  200
address<0>

data<31>    224
data<30>    223
data<29>    222
data<28>    221
data<27>    220
data<26>    218
data<25>    217
data<24>    216
data<23>    215
data<22>    209
data<21>    208
data<20>    207
data<19>    206
data<18>    205
data<17>    203

data<16>    202
data<2>
data<1>
data<0>

ramsn<4>
ramsn<3>
ramsn<2>
ramsn<1>
ramsn<0>    186
ramoen<4>
ramoen<3>
ramoen<2>

romsn<1>
romsn<0>
iosn
oen
read
writen
pio<12>     144   SR5
pio<11>     141   SR4
pio<10>     139   SR3
pio<12>     144   SR5
pio<11>     141   SR4
```

```
pio<9>      133  SR2          pio<3>      156  SL3
pio<8>      132  SR1          pio<2>      163  SL2
pio<7>      124  SR0          pio<1>      167  SL1
pio<6>      134  SL6          pio<0>      177  SL0
pio<5>      138  SL5          wdogn       152  pull-up
pio<4>      145  SL4
```

Hay un gran número de señales que no son asignadas debido a que no son necesarias. Para este tipo las herramientas de *PAR* realizan sus asignaciones a diferentes pins de forma automática.

Las señales que forman partes del controlador de memoria SDRAM y de la unidad de control de depuración DSU no aparecen en la implementación concreta que se realiza pues han sido deshabilitadas.

Todas las asignaciones y restricciones que se imponen quedan fijadas en el fichero UCF que será usado por las herramientas de P&R durante el proceso de generación del diseño.

Debido a la configuración física que tiene la placa de desarrollo XSV no es posible dejar accesible para un uso posterior las señales necesarias para la configuración parcial mediante el método de programación *SelectMap* a través de sus puertos de expansión. Éstos sólo ponen a disposición las mismas líneas que conectan la FPGA con cada banco de memoria.

5.1.1 RESET

El *reset* utilizado por el microprocesador LEON es a nivel bajo, y se ha elegido el pin 174 para que sea la entrada del mismo.

Según el diseño de la placa de desarrollo XSV, dicho pin está conectado a un pulsador (SW1). Cuando se pulsa SW1 dicho pin es forzado a tierra, mientras que en su estado normal se encuentra a la tensión de alimentación a través de una resistencia *pull-up*.

5.1.2 CLK

El pin asignado al reloj es el 89. Dicho pin está conectado a un *buffer* dedicado especialmente a la línea de reloj que lo distribuye a lo largo de toda la FPGA mediante líneas especiales y haciendo uso de las DLLs que posee la arquitectura Virtex..

En los parámetros que se fijan antes de realizar la síntesis, se fija el objetivo de área mínima antes que velocidad. Se intenta dejar por tanto el mayor número posible de recursos para futuras incorporaciones de nuevos módulos incluidos junto el procesador LEON en la propia FPGA, a costa de reducir la frecuencia máxima de reloj del diseño.

La placa XSV utilizada dispone de una Virtex HQ240-4, información necesaria para el flujo de diseño para poder evaluar los retrasos y simulaciones temporales que proporcionan la siguiente información acerca de la máxima frecuencia de funcionamiento teórica.

En la síntesis que realiza el *FPGA Express* se obtiene como frecuencia máxima de reloj **20 Mhz**. Existen líneas que a esa frecuencia tienen un retraso máximo que es igual al periodo del reloj.

Se debe de tener en cuenta, que dicha frecuencia corresponde a un análisis realizado antes de realizar la fase de *placement & routing*, y por tanto sólo tiene en cuenta los retrasos característicos de los bloques que constituyen el diseño, sin incluir los retrasos debidos al rutado de las conexiones.

En el análisis temporal que se realiza tras las etapas de *placement & routing* y mapeado del diseño, se obtienen los siguientes resultados que también permiten determinar la frecuencia máxima de reloj:

```
Design statistics:
   Minimum period:  61.599ns (Maximum frequency:  16.234MHz)
   Maximum net delay:  15.998ns

The Average Connection Delay for this design is:        2.925 ns
The Maximum Pin Delay is:                              15.998 ns
The Average Connection Delay on the 10 Worst Nets is:  11.609 ns

Listing Pin Delays by value: (ns)

d < 3.00  < d < 6.00  < d < 9.00  < d < 12.00  < d < 16.00  d >= 16.00
---------   ---------   ---------   ---------   ---------   ---------
   17678        9750        1999         296          73           0
```

Teniendo en cuenta los retrasos de las conexiones y el mapeado final del diseño, la nueva frecuencia máxima de reloj es inferior, **16.23 Mhz**. Para obtener unas mejores características, se realiza de nuevo la fase r*outing* teniendo en cuenta los resultados ya obtenidos en la anterior. Este nuevo ciclo, intenta minimizar las líneas con mayor retardo, la nueva frecuencia máxima de reloj que se obtiene es de **17.51 Mhz**.

Se realiza la elección de **12,5 Mhz**, y se debe fijar un factor de división de 8 en el oscilador programable incorporado en la placa de desarrollo. De esta manera divide los 100 Mhz base en 12,5 Mhz. Esta tarea se realiza previa a la configuración de la FPGA por primera vez.

Aunque el programa implementado se ha probado fijando también la velocidad de la placa XSV en 20 Mhz, siendo completamente operativo.

5.1.3 SEÑALES TRI-ESTADO

Existen dos señales de salida que son tri-estado, la salida del *circuito "watch-dog"* (`wdogn`) y la salida que indica error en el sistema (`errorn`), cuando están activas se encuentran a nivel bajo. De esta forma si se desea se podrían utilizar para generar el *reset* del dispositivo.

La salida `errorn` se activa cuando ocurre un error en el procesador y éste se detiene. Puede ocurrir si las excepciones e interrupciones están deshabilitadas y ocurre una de ellas no enmascarable.

Para poder indicarlas mediante LEDs en la placa de desarrollo XSV, es necesario incorporar en el proceso de configuración de las restricciones resistencias de *pull-up* para ambas salidas. Se debe a que los LEDs disponibles se iluminan si se les aplica tensión, y no tienen ningún tipo de resistencia conectada a alimentación. Mediante esta configuración, están encendidos normalmente y en caso de que se activen alguna de las señales se apagará el LED correspondiente.

Se han asignado el pin 152 para `wdogn` y 171 para `errorn`. Corresponden a LEDs que forman partes de la barra gráfica, concretamente a los dos extremos, el pin 152 al segmento B0 y el pin 171 al segmento B9.

5.1.4 PIN ENTRADA/SALIDA

Los 16 puertos de entrada/salida configurables de los que dispone el microprocesador LEON se asignan a los 7 segmentos y a la barra gráfica, para poder realizar señalización durante el flujo del programa. Y las dos últimas, `pio[15]` y `pio[14]`, a un interruptor (*DIPSW7*) y pulsador (*SW4*) respectivamente. Se permite por tanto señales de entrada para controlar el flujo de decisión del programa.

Al usarlos todos como pins de E/S, quedan deshabilitados para el resto de funciones, como señales de UART1, UART2. Aunque mediante software puede cambiarse el uso, pero no el pin físico de salida que está conectado a los LEDs.

Además los pins utilizados para controlar el display son los mismos que utiliza el método de programación *Select-Map*, al utilizarlos durante el funcionamiento del procesador LEON no es posible utilizar las posibilidades de lectura de la configuración (*readback*) y reconfiguración parcial que ofrece la arquitectura Virtex.

En la tabla 5-1 se pueden ver las asignaciones que se realizan para controlar cada segmento del conjunto de LEDs. Además las siguientes figuras muestran la disposición de los elementos de la placa de desarrollo XSV que se utilizan como Entrada/Salida.

LED	FPGA pin	Señal LEON
SL0	177	PIO[0]
SL1	167	PIO[1]
SL2	163	PIO[2]
SL3	156	PIO[3]
SL4	145	PIO[4]
SL5	138	PIO[5]
SL6	134	PIO[6]
SR0	124	PIO[7]
SR1	132	PIO[8]
SR2	133	PIO[9]
SR3	139	PIO[10]
SR4	141	PIO[11]
SR5	144	PIO[12]
SR6	147	PIO[13]

Tabla 5-1: Asignación de LEDs a señales de Entrada/Salida

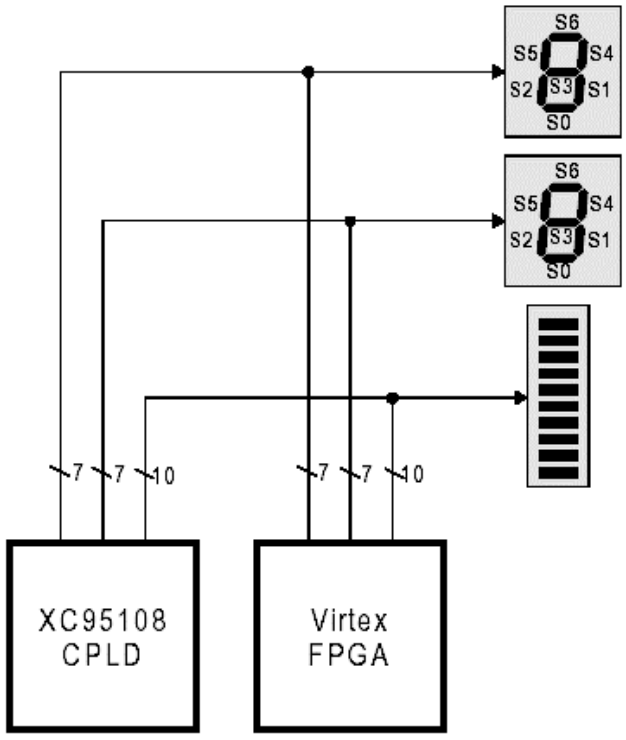


Figura 5-1: Disposición de conexiones hacia LEDs

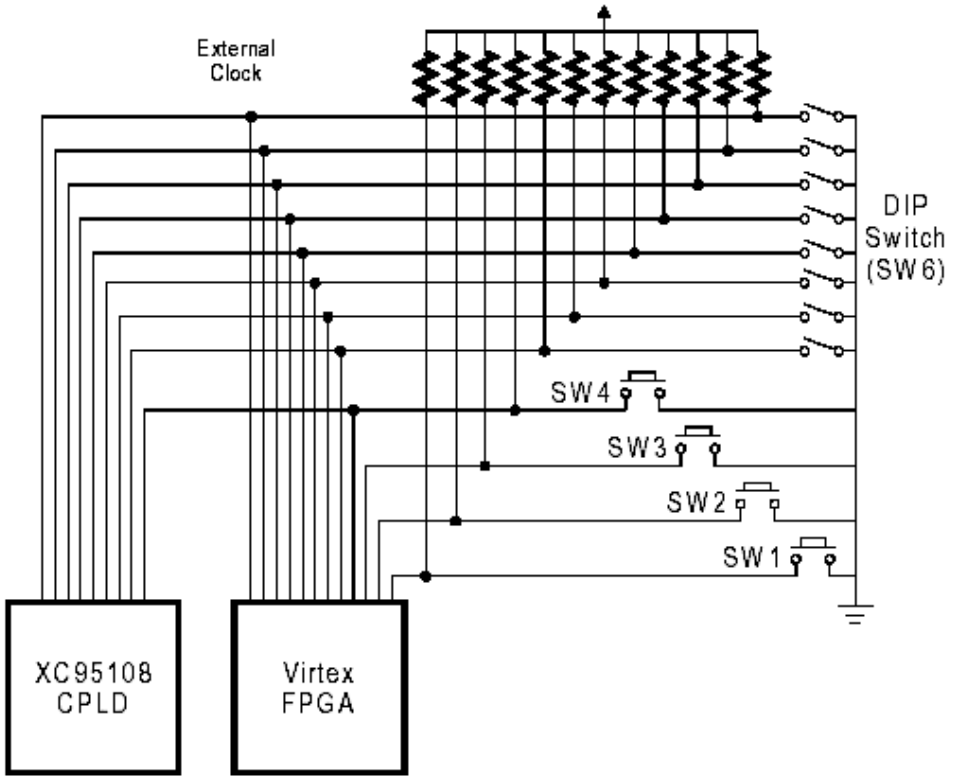


Figura 5-2: Disposición de conexiones hacia los interruptores y pulsadores

5.1.5 CONTROLADOR DE MEMORIA SRAM

El controlador de memoria dispone de señales para el control de la memoria PROM, dispositivos de I/O, memoria SRAM y SDRAM, pero sólo se asignan los de la memoria SRAM, ya que el resto de señales no son necesarias para esta implementación.

5.1.5.1 Placa XSV: Memoria

Para poder realizar las asignaciones, es necesario conocer la memoria SRAM existente en la placa de desarrollo XSV y el conexionado físico que necesita.

Dispone de dos bancos de memoria SRAM independientes, cada uno con un tamaño de 512K x 16 bits, situados a ambos lados de la FPGA. Se usará sólo banco izquierdo, dejando el otro libre para su posible uso por otros dispositivos integrados en el chip que se incorporen en un futuro. Por tanto existen 1024 Kbytes disponibles de memoria SRAM en la implementación.

Existen 16 líneas de datos, D[15:0], las correspondientes señales de control `ce`, `oe`, `we` activas todas ellas a nivel bajo y 19 líneas de datos (A[18:0]) que permiten direccionar 512K x 16 bits (Figura 5-3). La señal `we` habilita la escritura de los 16 bits completos, no existe habilitación de cada byte.

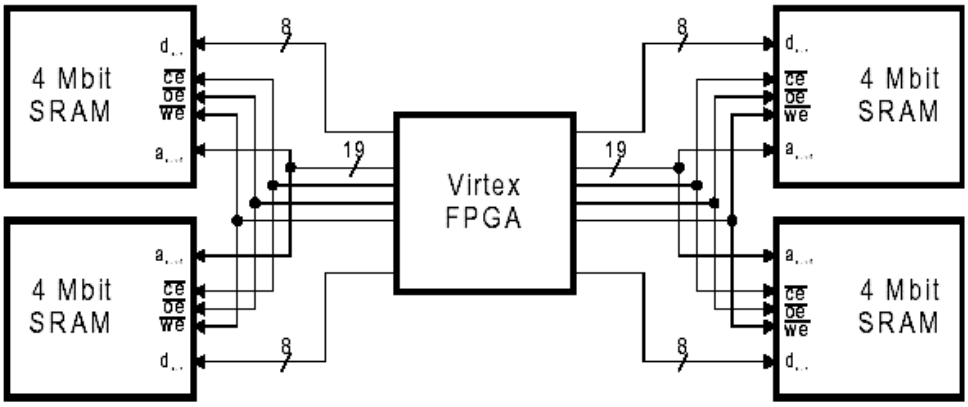


Figura 5-3: Disposición de conexiones hacia memoria SRAM

5.1.5.2 Accesos SRAM 16 bits

Para poder realizar accesos a memoria SRAM de 16 bits, es necesario modificar el conexionado que se realizaría en una situación normal.

Además cuando se inicie el microprocesador se debe de configurar la memoria SRAM para tener en cuenta el número de bancos (uno), el tamaño de cada banco (1024 KB), accesos en 16 bits y la señal de habilitación de escritura (`we`).

La interfaz que se propone para realizar accesos de 16 bits se muestra en la figura 5-4.

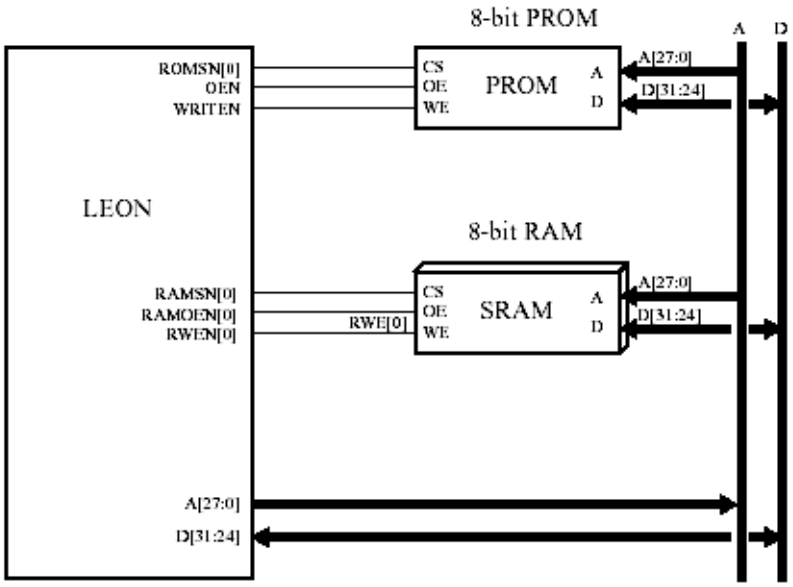


Figura 5-4: Disposición de conexionado para accesos de 16 bits

Las líneas de datos se conectan sólo a los 16 bits más altos del bus de datos (data<31:16>). La tabla 5-2 muestra la correspondencia entre las señales del controlador de memoria del microprocesador LEON y de la memoria SRAM integrada en la placa XSV, junto con las asignaciones de pins que se realizan para las líneas de datos.

SEÑAL MCRTL	SRAM	PIN
Data<31>	D15	224
Data<30>	D14	223
Data<29>	D13	222
Data<28>	D12	221
Data<27>	D11	220
Data<26>	D10	218
Data<25>	D9	217
Data<24>	D8	216
Data<23>	D7	215
Data<22>	D6	209
Data<21>	D5	208
Data<20>	D4	207
Data<19>	D3	206
Data<18>	D2	205
Data<17>	D1	203
Data<16>	D0	202

Tabla 5-2: Asignación: líneas del bus de datos

Las 19 líneas de direcciones se asignan al bus de direcciones pero a partir de address<1>, ya que sólo se pueden realizar accesos de 16 bits y por tanto address<0> siempre tiene valor 0.

Las señales de selección y habilitación de lectura, ce y we, se asignan a las correspondientes al primer banco de memoria SRAM, ramsn<0> y ramoen<0>.

La memoria SRAM disponible posee una señal de habilitación de escritura para los 16 bits que forman el dato. El controlador de memoria dispone de señales de habilitación para los 4 bytes que forman el bus de datos completo. Sólo se realiza la asignación a una de las señales, rwen<0>, y posteriormente en el inicio del procesador es necesario configurar el registro de control de memoria SRAM para habilitar el uso de una sola señal *strobe* para los ciclos de escritura.

La señal de entrada bexcn, error en bus, se asigna a uno de los interruptores DIP (DIPSW1), permitiendo su utilización de forma manual. Aunque mediante software se

realizará la programación del registro de control de memoria (MCFG1) para deshabilitarla, siempre se tiene la posibilidad de su activación.

Si está habilitada tiene validez para todas las áreas de memoria direccionadas (SRAM, SDRAM, ROM, I/O), al contrario de la señal `brdyn`, que sólo es válida para la zona I/O y la zona de memoria SRAM decodificada por la señal `ramsn[4]` (5º banco de memoria SRAM). En esta implementación no se usa y por tanto no se realiza asignación de pin a la señal `brdyn`.

La correspondencia entre las líneas de direcciones y control se muestran en la tabla 5-3.

SEÑAL MCRTL	SRAM	PIN
address<19>	A18	229
address<18>	A17	230
address<17>	A16	231
address<16>	A15	232
address<15>	A14	234
address<14>	A13	235
address<13>	A12	236
address<12>	A11	237
address<11>	A10	238
address<10>	A9	187
address<9>	A8	188
address<8>	A7	189
address<7>	A6	191
address<6>	A5	192
address<5>	A4	193
address<4>	A3	194
address<3>	A2	195
address<2>	A1	199
address<1>	A0	200
ramsn<0>	CE	186
ramoen<0>	OE	228
rwen<0>	WE	201
bexcn	DIPSW1	161

Tabla 5-3: Asignación líneas del bus de direcciones y bus de control

5.1.6 FICHERO UCF

A continuación se lista el fichero utilizado en el que quedan fijas las restricciones que se han impuesto en las entradas/salidas del diseño.

```
NET "resetn" LOC = "p174";
NET "clk" LOC = "p89";
NET "address<1>" LOC = "p200";
NET "address<2>" LOC = "p199";
NET "address<3>" LOC = "p195";
NET "address<4>" LOC = "p194";
NET "address<5>" LOC = "p193";
NET "address<6>" LOC = "p192";
NET "address<7>" LOC = "p191";
NET "address<8>" LOC = "p189";
NET "address<9>" LOC = "p188";
NET "bexcn" LOC = "p161";
NET "data<16>" LOC = "p202";
NET "data<17>" LOC = "p203";
NET "data<18>" LOC = "p205";
NET "data<19>" LOC = "p206";
NET "data<20>" LOC = "p207";
NET "data<21>" LOC = "p208";
NET "data<22>" LOC = "p209";
NET "data<23>" LOC = "p215";
NET "data<24>" LOC = "p216";
NET "data<25>" LOC = "p217";
NET "data<26>" LOC = "p218";
NET "data<27>" LOC = "p220";
NET "data<28>" LOC = "p221";
NET "data<29>" LOC = "p222";
NET "data<30>" LOC = "p223";
NET "data<31>" LOC = "p224";
NET "errorn" LOC = "p171";
NET "pio<0>" LOC = "p177";
NET "pio<1>" LOC = "p167";
NET "pio<2>" LOC = "p163";
NET "pio<3>" LOC = "p156";
NET "pio<4>" LOC = "p145";
NET "pio<5>" LOC = "p138";
NET "pio<6>" LOC = "p134";
NET "pio<7>" LOC = "p124";
NET "pio<8>" LOC = "p132";
NET "pio<9>" LOC = "p133";
NET "pio<10>" LOC = "p139";
NET "pio<11>" LOC = "p141";
NET "pio<12>" LOC = "p144";
NET "pio<13>" LOC = "p147";
NET "pio<14>" LOC = "p185";
NET "pio<15>" LOC = "p142";
NET "ramoen<0>" LOC = "p228";
NET "ramsn<0>" LOC = "p186";
NET "wdogn" LOC = "p152";
NET "rwen<0>" LOC = "p201";
NET "wdogn" PULLUP;
NET "errorn" PULLUP;
NET "address<10>" LOC = "p187";
NET "address<11>" LOC = "p238";
NET "address<12>" LOC = "p237";
NET "address<13>" LOC = "p236";
NET "address<14>" LOC = "p235";
NET "address<15>" LOC = "p234";
NET "address<16>" LOC = "p232";
NET "address<17>" LOC = "p231";
NET "address<18>" LOC = "p230";
NET "address<19>" LOC = "p229";
```

5.2 CONFIGURACIÓN

Se realiza la configuración del modelo VHDL del procesador LEON para obtener una implementación adecuada a las características del dispositivo objetivo. En este caso, la Virtex XCV800 y la placa de desarrollo XSV *Board*.

Se intenta obtener un modelo completo pero sin incluir los posibles elementos que no sean de aplicación en la implementación final ya sea por innecesarios o por imposibilidad de utilización debido a las características de la placa de desarrollo.

Para la síntesis es necesario que el fichero *device.vhd* contenga la configuración que se desea. Existe un conjunto de configuraciones predeterminadas para distintos tipos de dispositivos y que se pueden seleccionar en dicho fichero.

Para obtener un mayor control en la implementación obtenida, se realiza una configuración particular, sin usar ninguna de las estructuras ya predeterminadas en cuanto a configuración.

La configuración se realiza a través de la estructura de VHDL *record*. Existe una estructura maestra que contiene un numero de estructuras de menor nivel que configuran cada una un módulo o función específica.

```
type config_type is record
     synthesis: syn_config_type;
     iu : iu_config_type;
     fpu : fpu_config_type;
     cp : cp_config_type;
     cache: cache_config_type;
     ahb : ahb_config_type;
     apb : apb_config_type;
     mctrl: mctrl_config_type;
     boot : boot_config_type;
     debug: debug_config_type;
     pci : pci_config_type;
     peri : peri_config_type;
end record;
```

A continuación se configuran los distintos *records* para obtener las características adecuadas a la implementación deseada.

```
constant virtex_xcv800_xsv : config_type := (
     synthesis => syn_xsvconfig, iu => iu_xsvconfig, fpu =>
     fpu_xsvconfig, cp => cp_xsvconfig, cache =>
     cache_xsvconfig, ahb => ahb_xsvconfig,  apb =>
     apb_xsvconfig, mctrl => mctrl_xsvconfig, boot =>
     boot_xsvconfig,  debug => debug_xsvconfig, pci =>
     pci_xsvconfig, peri => peri_xsvconfig);
```

5.2.1 CONFIGURACIÓN : SÍNTESIS

En esta estructura se adapta el modelo a distintas herramientas de síntesis y a distintos dispositivos.

```
type targettechs is (gen, virtex, atc35, atc25);
-- synthesis configuration

type syn_config_type is record
  targettech: targettechs;
  infer_ram : boolean;-- infer cache ram automatically
  infer_regf : boolean;-- infer regfile automatically
  infer_rom: boolean;-- infer boot prom automatically
  infer_pads: boolean;-- infer pads automatically
  infer_mult: boolean;-- infer multiplier automatically
  rftype : integer;-- register file implementation option
end record;
```

La elección de las distintas posibilidades se hace teniendo en cuenta el objetivo a conseguir y el hardware disponible. La estructura configuración que se obtiene es:

```
constant syn_xsvconfig : syn_config_type := (
    targettech => virtex, infer_pads => true,
    infer_ram => false, infer_regf => false, infer_rom => false,
    infer_mult => true, rftype => 2);

------ infer_rom => true, generated from bprom.vhd; infer_rom =>
     false, VIRTEX prom is instantiated
```

En el proceso de síntesis se puede obligar a que las mega-celdas sean instanciadas o inferidas por las herramientas de síntesis. La tecnología objetivo Virtex permite la posibilidad de que se instancie directamente las celdas que hagan uso de dispositivos de memoria estática SRAM, en este caso la cache, ROM y registros.

Si se infiere la ROM, se toma como fuente en la síntesis de los contenidos de la misma, el fichero *bprom.vhd*. Si por el contrario se realiza la instancia, como posibilidad que permite Virtex, se toma como contenido de la ROM un fichero particular de Virtex que ya instancia el propio componente. Se detalla en el punto Configuración: *Boot-Prom*.

La elección de inferir el multiplicador se detalla en el siguiente punto.

5.2.2 CONFIGURACIÓN: IU

Con esta estructura se controla la implementación de la unidad de ejecución o IU (*Integer Unit*).

```
-- processor configuration
type multypes is (none, iterative, m32x8, m16x16, m32x16,
     m32x32);
type divtypes is (none, radix2);

type iu_config_type is record
   nwindows: integer;-- # register windows (2 - 32)
   multiplier: multypes;-- multiplier type
   divider : divtypes;-- divider type
   mac : boolean; -- multiply/accumulate
   fpuen: integer range 0 to 1;-- FPU enable
   cpen: boolean; -- co-processor enable
   fastjump : boolean;-- enable fast jump address generation
   icchold : boolean;-- enable fast branch logic
   lddelay: integer range 1 to 2; -- # load delay cycles (1-2)
   fastdecode : boolean;-- optimise instruction decoding (FPGA
    only)
   rflowpow : boolean;-- disable regfile when not accessed
   watchpoints: integer range 0 to 4; -- # hardware watchpoints
    (0-4)
   impl : integer range 0 to 15; -- IU implementation ID
   version: integer range 0 to 15; -- IU version ID
end record;
```

La configuración queda realizada mediante las siguientes asignaciones en la estructura:

```
constant iu_xsvconfig : iu_config_type := (
    nwindows => 8, multiplier => m16x16, divider => radix2, mac
    => true,fpuen => 0, cpen => false, fastjump => true,
    icchold => true, lddelay => 1, fastdecode => true,
    watchpoints => 0, impl => 0,
    version => 0, rflowpow => false);
```

`Nwindows` fija el número de ventanas de registros de la especificación SPARC V. Cada ventana está formada por 16 registros y dicho parámetro repercute de forma directa en el tamaño y ocupación de recursos de la implemetación. El estándar SPARC permite entre 2 – 32 ventanas. Para permitirla compatibilidad con los manejadores de error de *overflow/underflow* que poseen los compiladores de libre distribución LECCS, se fija en 8.

Para la síntesis en FPGAs se recomienda que se infiera un multiplicador de m16x16, ya que de esta forma es como se obtiene el mayor rendimiento. Además, la elección del multiplicador m16x16 permite habilitar el modulo MAC, ya que es el único tipo con el que puede funcionar.

La tabla 5-4 muestra las distintas opciones de multiplicadores existentes y el área que ocupan aproximadamente (Kgates).

Configuración	Latencia(ciclos CLK)	Aprox. área (Kgates)
Iterativa	35	1000
M16x16	4	6000
M32x8	4	5000
M32x16	2	9000
M32x32	1	15000

Tabla 5-4: Configuraciones del multiplicador

Se habilitan el divisor en base 2 y el módulo MAC, que permite la utilización de las instrucciones *smac* y *umac*. Para la implementación que se pretende conseguir no se habilitarán las interfaces de la unidad de punto flotante FPU y del coprocesador CP.

Las opciones `fastjump, fastdecode` son habilitadas para permitir la generación de una lógica adicional que permite generar más rápidamente las direcciones de salto y de acceso a los registros respectivamente.

Mediante `lddelay` se fija a 1 ciclo de retraso de lectura en la estructura *pipeline*, e `icchold` se habilita para permitir un ciclo de retraso si una instrucción de salto va precedida de una instrucción que modifique los códigos de condición (icc).

Los registros de comparación (`watchpoint`) se anulan, debido a que no se va a utilizar la depuración (debug).

`Rflowpow` se deshabilita, ya que puede llegar a crear un camino crítico importante al crear una señal de habilitación de lectura de los registros.

5.2.3 CONFIGURACIÓN: FPU

Como ya se ha mencionado, la Unidad de Punto Flotante (FPU) no se utiliza para esta implementación práctica.

El *record* que lo configura y los valores que se han asignado son los siguientes:

```
type fpucoretype is (meiko, lth); -- FPU core type
type fpuiftype is (none, serial, parallel); -- FPU interface
     type

type fpu_config_type is record
   core: fpucoretype;-- FPU core type
   interface: fpuiftype;-- FPU inteface type
   fregs: integer; -- 32 for serial interface, 0 for parallel
   version: integer range 0 to 7; -- FPU version ID
end record;
```

```
constant fpu_xsvconfig : fpu_config_type :=
    (core => lth, interface => none, fregs => 0, version => 0);
```

El tipo de FPU que se escoge si se habilitara es el lth, pues Meiko no viene distribuida con el microprocesador LEON en su versión de licencia libre GNU.

5.2.4 CONFIGURACIÓN: COPROCESADOR

La configuración se realiza a través del siguiente *record*:

```
-- co-processor configuration
type cptype is (none, cpc); -- CP type

type cp_config_type is record
  cp  : cptype;   -- Co-processor type
  version   : integer range 0 to 7; -- CP version ID
-- add your CP-specific configuration options here!!
end record;

constant cp_xsvconfig  : cp_config_type :=( cp => none, version
    => 0);
```

No se habilita la interfaz del coprocesador, pues no se utiliza para realizar la implementación práctica que se quiere obtener.

5.2.5 CONFIGURACIÓN: CACHÉ

Las siguientes declaraciones marcan la configuración de la caché:

```
-- cache configuration

type dsnoop_type is (none, slow, fast); -- snoop implementation
    type
constant PROC_CACHE_MAX : integer := 4;   -- maximum cacheability
    ranges
constant PROC_CACHE_ADDR_MSB : integer := 3; -- MSB address bits
    to decode cacheability
subtype proc_cache_addr_type is
    std_logic_vector(PROC_CACHE_ADDR_MSB-1 downto 0);

type proc_cache_config_type is record
  firstaddr : proc_cache_addr_type;
  lastaddr  : proc_cache_addr_type;
end record;

type proc_cache_config_vector is array (Natural Range <> ) of
    proc_cache_config_type;

constant proc_cache_config_void : proc_cache_config_type :=
```

```
  ((others => '0'), (others => '0'));

type cache_config_type is record
  icachesize      : integer;  -- size of I-cache in Kbytes
  ilinesize : integer;  -- # words per I-cache line
  dcachesize      : integer;  -- size of D-cache in Kbytes
  dlinesize : integer;  -- # words per D-cache line
  dsnoop          : dsnoop_type;    -- data-cache snooping
  cachetable      : proc_cache_config_vector(0 to PROC_CACHE_MAX-
    1);
end record;
```

La configuración realizada para la implementación en la VIRTEX XCV800 es la siguiente:

```
constant cache_xsvconfig : cache_config_type := ( icachesize =>
     2, ilinesize => 4, dcachesize => 2, dlinesize => 4, dsnoop
     => none,   cachetable => cachetbl_std);
```

El parámetro `dsnoop` implementa un protocolo para mantener la coherencia de los datos en la cache en caso de que exista un sistema multiprocesador que pueda llegar a modificarlos. En este caso, sólo se tiene un procesador que accede a los contenidos de la memoria y por tanto no es de aplicación.

Las zonas de memoria que se permiten sean accedidas a través de caché son dos, RAM y PROM. Dichas zonas y sus límites vienen determinados por `cachetable`. En la configuración del modelo no se modifican.

```
-- standard cacheability config
constant cachetbl_std : proc_cache_config_vector(0 to
     PROC_CACHE_MAX-1) := (
-- first    last        function   address[31:29]
  ("000", "000"),   -- PROM area    0x0- 0x0
  ("010", "011"),   -- RAM area     0x2- 0x3
   others => proc_cache_config_void);
```

Se fija 4 *words* por línea, lo que permite en la caché de instrucciones 4 instrucciones por línea y en la caché de datos cuatro enteros de tamaño 32 bits o dos de tamaño 16 bits.

Durante el proceso de síntesis se realiza la instancia de los bloques de memoria RAM necesarios para implementar la caché de datos e instrucciones, además de los bloques necesarios para los registros de etiquetas (*dtags* y *itags*) que corresponden a cada línea de caché.

Los tamaños de caché para instrucciones y datos se fijan en 2KB y 2KB respectivamente. La elección de tamaño corresponde a un compromiso que existe entre

recursos de RAM ocupados en la FPGA Virtex, tamaño de caché y tamaño de prom instanciada dentro de la FPGA. En **Configuración: Utilización de Recursos** se exponen las distintas posibilidades además de argumentar la elección realizada.

5.2.6 CONFIGURACIÓN: CONTROLADOR DE MEMORIA

El controlador de memoria es configurado a través del siguiente *record*:

```
-- memory controller configuration
type mctrl_config_type is record
  bus8en          : boolean;  -- enable 8-bit bus operation
  bus16en         : boolean;  -- enable 16-bit bus operation
  wendfb    : boolean;  -- enable wen feed-back to data bus
     drivers
  ramsel5         : boolean;  -- enable 5th ram select
  sdramen         : boolean;  -- enable sdram controller
  sdinvclk        : boolean;  -- invert sdram clock
end record;
```

Los parámetros se fijan mediante la siguiente declaración:

```
constant mctrl_xsvconfig : mctrl_config_type := (
    bus8en => true, bus16en => true, wendfb => true, ramsel5 =>
     false, sdramen => false, sdinvclk => false);
```

Se habilitan la posibilidad de accesos de memoria en 8 y 16 bits, debido a que la memoria física SRAM disponible en la placa de desarrollo XSV tiene un ancho de 16 bits. El controlador de memoria SDRAM se deshabilita como la señal de selección del 5º banco de RAM, ya que sólo existen en la placa dos bancos de memoria RAM.

Mediante `wendfb` se crea una realimentación hacia el interior de la señal de *write enable* para saber cuando deja de estar activa y asegurar que el cambio de dirección no afecta una vez terminado el ciclo de escritura.

5.2.6.1 Mapa de memoria del procesador LEON

El procesador LEON posee un bus de direcciones de 32 bits, aunque como los accesos se realizan en tamaño de *word* (32 bits), los últimos 2 bits del bus de direcciones no se implementan.

A continuación se detallan las distintas particiones que se realizan sobre el espacio de direcciones y funciones que se le asignan, permitiendo así comprobar las zonas que sean accedidas mediante la caché y las que no. Aunque siempre se puede modificar la configuración elegida para una implementación que necesite de características especiales.

Con 32 bits direcciona un espacio de 4 Gbytes, y para separarlo en distintas zonas sólo se utiliza los 3 primeros bits, A[31:29]. La tabla 5-5 muestra los valores de los bits que delimitan las distintas partes, el tamaño de las mismas así como la función a la que están destinadas.

A[31:29] inicial	A[31:29] final	Tamaño	Mapeado	Módulo
000	001	512 MB	Prom	Controlador Mem.
001	010	512 MB	Bus Memoria I/O	Controlador. Mem
010	100	1GB	SRAM y/o SDRAM	Controlador Mem.
100	100	512 MB	Registros y DSU	APB bridge y DSU
100	111	1.5 GB	Sin asignar	-

Tabla 5-5: Localización de direcciones en el bus AHB

En la figura que se muestra a continuación, aparece un esquema gráfico sobre la partición del espacio de direcciones a través de los 3 primeros bits para las distintas funciones.

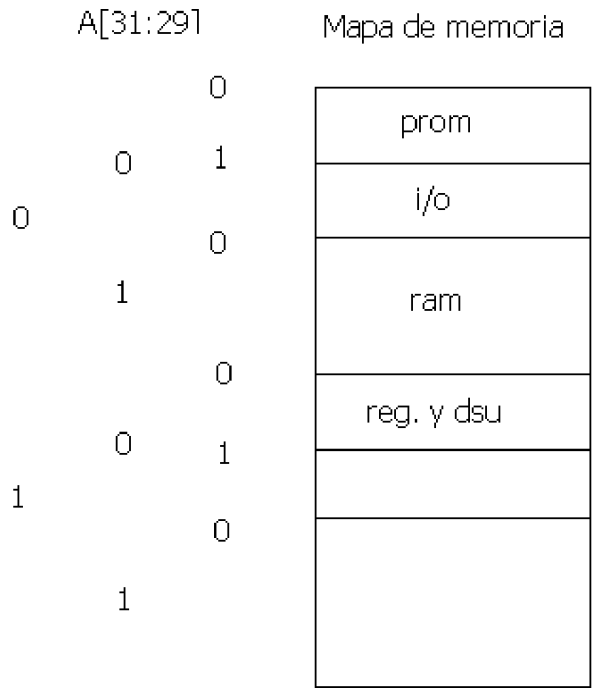


Figura 5-5: Mapa de memoria

Como ya se ha mencionado antes, sólo la zona de prom y la zona de ram se escogen para que puedan ser accedidas a través de caché.

5.2.7 CONFIGURACIÓN: DEBUG

Las características sobre depuración se configuran a través del siguiente *record*:

```
-- debug configuration
type debug_config_type is record
  enable          : boolean;  -- enable debug port
  uart      : boolean;  -- enable fast uart data to console
  iureg     : boolean;  -- enable tracing of iu register writes
  fpureg          : boolean;  -- enable tracing of fpu register
     writes
  nohalt          : boolean;  -- dont halt on error
  pclow           : integer;  -- set to 2 for synthesis, 0 for
     debug
  dsuenable       : boolean;  -- enable DSU
  dsutrace        : boolean;  -- enable trace buffer
  dsumixed        : boolean;  -- enable mixed-mode trace buffer
  dsudpram        : boolean;  -- use dual-port ram for trace
     buffer
  tracelines      : integer range 64 to 1024; -- # trace lines
     (needs 16 bytes/line)
end record;
```

Para esta implementación práctica no se usa el sistema de depuración que incorpora el LEON, y por tanto a través de las siguientes declaraciones se deshabilita el mismo.

```
constant debug_xsvconfig : debug_config_type := ( enable =>
     false, uart => false, iureg => false, fpureg => false,
     nohalt => false, pclow => 2, dsuenable => false, dsutrace
     => false, dsumixed => false, dsudpram => true, tracelines
     => 64);
```

El parámetro `enable` fija la incorporación de un desensamblador, pero sólo para la simulación, dicho valor no afecta para nada a la síntesis. Con `dsuenable` se deshabilita la unidad de depuración DSU. El resto de parámetros fijan las siguientes características:

`Uart`: fija el bit de preparado para transmitir permanentemente en '1' para simulación, no afecta para nada a la síntesis.

`Iureg,fpureg`: habilitan el trazado de los registros de la UI y FPU en caso de depuración.

`Nohalt`: para no violar la norma SPARC V se debe fijar en falso, pues permite que procesador no para la ejecución cuando se produzca un *error mode* (error en una excepción).

`Pclow`: se fija en 2 para síntesis, ya que no implementa las dos últimas líneas de direcciones en el bus AMBA pues se accede en tamaño de 32 bits.

`Dsutrace`: habilita en caso de depuración el *buffe*r de trazado.

`Dsumixed`: fija trazado mixto de accesos al bus AMBA e instrucciones de la IU.

`Dsupram`: fija el uso de *Dual Port-Ram* para el *buffer*.

`Tracelines`: número de líneas para el *buffer*, cada una de 128 bits.

5.2.8 CONFIGURACIÓN: PERIFÉRICOS

El *record* de configuración de periféricos determina varias características y funciones de los mismos.

```
type irq_filter_type is (lvl0, lvl1, edge0, edge1);
type irq_filter_vec is array (0 to 31) of irq_filter_type;

type irq2type is record
  enable    : boolean;  -- enable chained interrupt controller
  channels  : integer;  -- number of additional interrupts (1 -
     32)
  filter    : irq_filter_vec; -- irq filter definitions
end record;
```

```
type peri_config_type is record
  cfgreg    : boolean;  -- enable LEON configuration register
  ahbstat   : boolean;  -- enable AHB status register
  wprot     : boolean;  -- enable RAM write-protection unit
  wdog      : boolean;  -- enable watchdog
  irq2cfg   :irq2type;  -- chained interrupt controller config
end record;

constant peri_xsvconfig : peri_config_type := (
    cfgreg => true, ahbstat => true, wprot => false, wdog =>
     true, irq2cfg => irq2none);
```

Los distintos parámetros permiten incluir o no diversos periféricos, seleccionando por tanto un diseño más completo o más reducido.

Para esta implementación práctica se incluye el registro de configuración del procesador LEON, el registro de estado del bus AMBA, y el '*watchdog*' o perro guardián. No se utiliza el mecanismo de protección de RAM, ya que no se carga el programa en ella, sino en la PROM instanciada.

El controlador de interrupciones adicionales (hasta 32) tampoco se incluye, pues no es necesario para el ejemplo a desarrollar.

5.2.9 CONFIGURACIÓN: *BOOT*

En la configuración del *record boot* se puede seleccionar diversas características sobre la forma de lanzar el procesador una vez recibe la señal de reset, si se lanza de la memoria externa, de una memoria ROM interna, o desde la unidad DSU.

```
type boottype is (memory, prom, dual);

type boot_config_type is record
  boot           : boottype; -- select boot source
  ramrws    : integer range 0 to 3; -- ram read waitstates
  ramwws    : integer range 0 to 3; -- ram write waitstates
  sysclk    : integer;  -- cpu clock
  baud      : positive; -- UART baud rate
  extbaud   : boolean;  -- use external baud rate setting
  pabits    : positive; -- internal boot-prom address bits
end record;

constant boot_xsvconfig : boot_config_type := (boot => prom,
     ramrws => 0,    ramwws => 0, sysclk => 12500000, baud =>
     9600, extbaud => false, pabits => 10);
```

La opción `boot` fija la forma de lanzamiento del procesador: mediante `memory` se lanza a partir de una memoria externa, si es `prom` se instancia o infiere (dependiendo del parámetro `infer_prom` del *record* configuración de síntesis), y por último si es `dual` se escoge otra opción dependiendo del valor del pin PIO[4].

Para lanzarlo desde la unidad de depuración se deben afirmar las señales de los pin DSUEN y DSUBRE en el momento de realizar el *reset*.

Para la implementación en una placa de desarrollo XSV, y al no disponer de memoria *eeprom* externa (la memoria FLASH incluida en la placa se utiliza para guardar la configuración inicial de la FPGA Virtex) se elige el lanzamiento desde una memoria prom interna en la propia Virtex.

5.2.9.1 Lanzamiento desde PROM interna

Si el lanzamiento se realiza desde la prom interna, se fijan algunos parámetros de configuración del controlador de memoria y de la UART, como los ciclos de espera en lectura y escritura de memoria ram, y la velocidad de comunicación de la UART1.

Una vez que se realiza el *reset*, el contador que preescala los *timers* se fija para generar un pulso de 1 Mhz, a partir de la frecuencia de reloj fijada en el parámetro `sysclk`.

Además se ajusta el valor de los contadores existentes en la UART para obtener la velocidad indicada por `baud` o por PIO[7:0] si la opción `extbaud` está habilitada.

Si la prom es inferida, los contenidos de la misma vienen definidos por el fichero *bprom.vhd*, que por defecto contiene un monitor de lanzamiento denominado PMON. El tamaño de la misma es de 1Kbyte.

Si por el contrario se instancia, como permite la tecnología Virtex, `pabits` fija el tamaño de la prom instanciada eligiendo el número de bits que componen el bus de direcciones de la prom.

5.2.9.1.1 Instancia de memoria prom en Virtex

Para realizar la instancia de la memoria prom en Virtex es necesario seguir una serie de pasos en el flujo de diseño.

Los contenidos de la prom deben estar en un formato interno de Virtex denominado '*.mif*'. Se proporciona un programa en C (bin2mif) que convierte el formato binario '*.bin'* en el nuevo formato.

Una vez obtenido, se usa el programa *CoreGen™* proporcionado por Virtex para instanciar un bloque RAM síncrono sin escritura cuyo contenido viene determinado por el fichero '*.mif*'. El programa crea la correspondiente netlist '*.edn*' que se utilizará por parte de las herramientas de *Place & Route*.

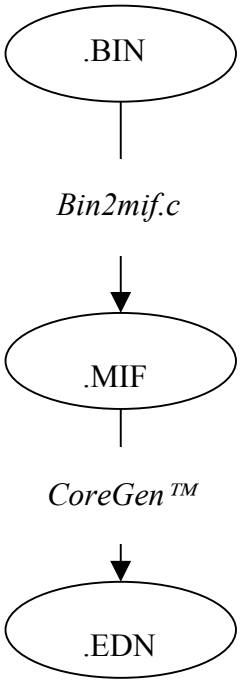


Figura 5-6: Flujo de diseño para instanciar memoria PROM en Virtex

5.2.9.2 Monitor PMON

PMON es un monitor proporcionado por el propio paquete del procesador LEON, para colocarlo en una memoria prom y que presenta una serie de características interesantes.

Tras realizar el *reset*, el monitor escanea todos los bancos de ram, detecta si tiene conectado memoria de 8,16 o 32 bits, el tamaño de cada banco y fija el puntero de pila en parte más alta de la RAM.

Para comunicarse con el monitor es necesario utilizar la UART1, usando una serie de paquetes de datos especiales denominados S-record.

En la implementación práctica que se realiza se usan parte de las características del monitor al comienzo de la ejecución.

5.2.10 CONFIGURACIÓN: AMBA

La incorporación de un bus AMBA en el propio sistema añade una gran funcionalidad y flexibilidad a la hora de añadir nuevos periféricos.

La configuración de los dos buses AHB y APB se realiza mediante los siguientes *record*:

```
type ahb_config_type is record
  masters   : integer range 1 to AHB_MST_MAX;
  defmst    : integer range 0 to AHB_MST_MAX-1;
  split     : boolean;  -- add support for SPLIT reponse
  slvtable  : ahb_slv_config_vector(0 to AHB_SLV_MAX-1);
  testmod   : boolean;  -- add AHB test module (not for
     synthesis!)
end record;

type apb_config_type is record
  table     : apb_slv_config_vector(0 to APB_SLV_MAX-1);
end record;

constant AHB_MST_MAX    : integer := 4;   -- maximum AHB masters
constant AHB_SLV_MAX    : integer := 7;   -- maximum AHB slaves
constant AHB_SLV_ADDR_MSB : integer := 4; -- MSB address bits to
     decode slaves
constant APB_SLV_MAX      : integer := 16;  -- maximum APB
     slaves
constant APB_SLV_ADDR_BITS : integer := 10;  -- address bits to
     decode APB slaves
```

Mediante las anteriores declaraciones se fijan el número máximo de esclavos en el bus APB y AHB y el número máximo de maestros en el bus AHB. Permiten configurar posteriormente los tamaños de las tablas que parten el bus de direcciones entre los distintos módulos conectados a ambos buses.

5.2.10.1 Configuración: AHB

La configuración de los esclavos que existen en el bus AHB se realiza en el siguiente conjunto de declaraciones:

```
subtype ahb_range_addr_type is
     std_logic_vector(AHB_SLV_ADDR_MSB-1 downto 0);

type ahb_slv_config_type is record
  firstaddr : ahb_range_addr_type;
  lastaddr  : ahb_range_addr_type;
  index     : integer range 0 to AHB_SLV_MAX-1;
  split           : boolean;
  enable    : boolean;
end record;

type ahb_slv_config_vector is array (Natural Range <> ) of
     ahb_slv_config_type;


constant ahb_slv_config_void : ahb_slv_config_type :=
  ((others => '0'), (others => '0'), 0, false, false);



constant ahbslvcfg_xsvconfig : ahb_slv_config_vector(0 to
     AHB_SLV_MAX-1) := (
  -- first    last  index  split  enable  function
     HADDR[31:28]
    ("0000", "0111",  0,   false, true), -- memory controller,
     0x0- 0x7
    ("1000", "1000",  1,   false, true), -- APB bridge, 128 MB
     0x8- 0x8
    ("1001", "1001",  2,   false, false), -- DSU         128 MB
     0x9- 0x9
    ("1010", "1111",  3,   false, false), -- PCI initiator
     0xA- 0xF
      others => ahb_slv_config_void);

  constant ahb_xsvconfig : ahb_config_type := ( masters => 1,
     defmst => 0,
    split => false, slvtable => ahbslvcfg_xsvconfig, testmod =>
     false);
```

Se configura el rango de direcciones de cada esclavo, si está habilitado o no y si tiene capacidad de realizar una respuesta *split* (especificación BUS AMBA).

Sólo existe un maestro, la UI y se configuran dos esclavos, el controlador de memoria y el puente al bus APB.

5.2.10.2 Configuración: APB

Las declaraciones que configuran el bus APB son las siguientes:

```
subtype apb_range_addr_type is
     std_logic_vector(APB_SLV_ADDR_BITS-1 downto 0);

type apb_slv_config_type is record
  firstaddr : apb_range_addr_type;
  lastaddr  : apb_range_addr_type;
  index     : integer;
  enable    : boolean;
end record;

type apb_slv_config_vector is array (Natural Range <> ) of
     apb_slv_config_type;
constant apb_slv_config_void : apb_slv_config_type :=
  ((others => '0'), (others => '0'), 0, false);

type apb_config_type is record
  table     : apb_slv_config_vector(0 to APB_SLV_MAX-1);
end record;


constant apbslvcfg_xsvconfig : apb_slv_config_vector(0 to
     APB_SLV_MAX-1) := (
  --   first         last      index  enable     function
     PADDR[9:0]
  ( "0000000000", "0000001000",  0,   true), -- memory
     controller, 0x00 - 0x08
  ( "0000001100", "0000010000",  1,   true), -- AHB status reg.,
     0x0C - 0x10
  ( "0000010100", "0000011000",  2,   true), -- cache
     controller,  0x14 - 0x18
  ( "0000011100", "0000100000",  3,   false), -- write
     protection,  0x1C - 0x20
  ( "0000100100", "0000100100",  4,   true), -- config register,
     0x24 - 0x24
  ( "0001000000", "0001101100",  5,   true), -- timers,
     0x40 - 0x6C
  ( "0001110000", "0001111100",  6,   true), -- uart1,
     0x70 - 0x7C
  ( "0010000000", "0010001100",  7,   true), -- uart2,
     0x80 - 0x8C
  ( "0010010000", "0010011100",  8,   true), -- interrupt ctrl
     0x90 - 0x9C
  ( "0010100000", "0010101100",  9,   true), -- I/O port
     0xA0 - 0xAC
  ( "0010110000", "0010111100", 10,   false),-- 2nd interrupt
     ctrl 0xB0 - 0xBC
  ( "0011000000", "0011001100", 11,   false), -- DSU uart
     0xC0 - 0xCC
  ( "0100000000", "0111111100", 12,   false), -- PCI
     configuration  0x100- 0x1FC
  ( "1000000000", "1011111100", 13,   false), -- PCI arbiter
     0x200- 0x2FC
     others => apb_slv_config_void);

  constant apb_xsvconfig  : apb_config_type := (table =>
     apbslvcfg_xsvconfig);
```

El número de esclavos y su rango de direcciones se configuran mediante la `APB slave table`, además de precisar si están habilitados o no. La tabla configura automáticamente el puente APB/AHB.

Si se desean añadir nuevos esclavos al bus APB es necesario indicarlos en la tabla, definir un rango de direcciones y posteriormente conectar los módulos en *mcore.vhd* a los correspondientes apbi/apbo.

En la configuración que se realiza se habilitan todos los posibles periféricos a excepción de la protección contra escritura en RAM, el segundo controlador de interrupciones y la unidad de depuración DSU (PCI no incluido).

En la tabla, los rangos de direcciones `padrr` indican el valor inicial y final de las últimas diez líneas del bus de direcciones.

5.2.11 CONFIGURACIÓN: UTILIZACIÓN DE RECURSOS

En el proceso de síntesis se puede realizar una extracción del número de recurso ocupados por parte del diseño. Los más escasos y en los cuáles repercute más la configuración son los recursos RAM, que se detallan en el punto siguiente.

El resto, formado por LUTS, IOBs, SLICES, etc, se fijan por el propio diseño y poco cambian mediante la configuración, a excepción de la inclusión o no de un multiplicador y divisor.

El informe que genera la herramienta de P&R muestra el porcentaje y cantidad recursos ocupados en la configuración que se ha detallado anteriormente.

```
Design Summary
--------------
   Number of errors:      0
   Number of warnings:  111
   Number of Slices:              4,217 out of  9,408   44%
   Number of Slices containing
      unrelated logic:                0 out of  4,217    0%
   Number of Slice Flip Flops:    1,916 out of 18,816   10%
   Total Number 4 input LUTs:     7,840 out of 18,816   41%
      Number used as LUTs:                     7,738
      Number used as a route-thru:               102
   Number of bonded IOBs:           101 out of    166   60%
   Number of Block RAMs:             22 out of     28   78%
   Number of GCLKs:                   2 out of      4   50%
   Number of GCLKIOBs:                1 out of      4   25%
Total equivalent gate count for design:  427,234
Additional JTAG gate count for IOBs:  4,896
```

5.2.11.1 Recursos RAM

Durante el proceso de configuración se pueden delimitar el tamaño de diversos elementos que repercuten directamente en los recursos RAM empleados en la FPGA Virtex. Son la caché de instrucciones y datos, la prom instanciada o inferida, los registros de la UI y la utilización del *buffer* de trazado de la DSU.

Los recursos RAM que posee la Virtex XCV800, son limitados y por tanto la elección del tamaño de los anteriores elementos es crucial para determinar los recursos utilizados y recursos que quedan libres para su utilización en futuros módulos conectados al procesador e incluidos en la propia FPGA.

5.2.11.1.1 Virtex XCV 800: RAM

La Virtex XCV 800 posee 28 bloques *Dual Port RAM* de un tamaño de 512 bytes. Por tanto los recursos disponibles de RAM son 14 Kbytes, sin tener en cuenta la posible memoria que se obtiene utilizando las tablas '*look-up table'* (LUT) de 4 entradas (16 bits) que existen en cada CLB.

Virtex tiene una matriz formada por 56x84 CLBs, cada una con 64 bits que provienen de las 4 LUTs, lo que hacen un total de 301.056 bits, o sea, 37632 bytes.

Dispositivo	**Bits Bloques RAM**	***SelectRAM Bits***
XCV 800	114.688	301.056

Tabla 5-6: Recursos de RAM disponible en Virtex XCV 800

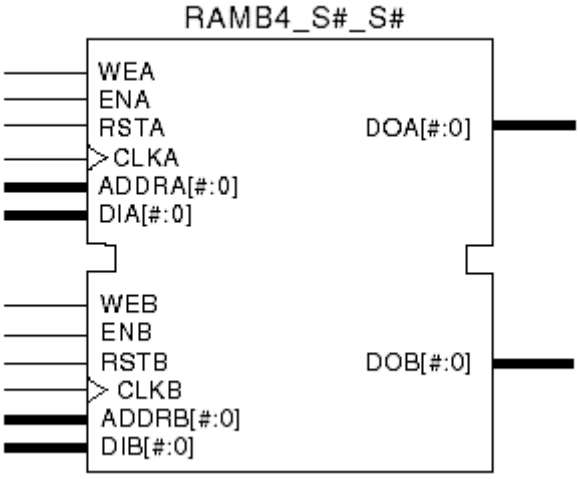


Figura 5-7: Bloque RAM Virtex

Cada bloque de RAM, ilustrado en la figura 5-7, es un módulo *dual-port* completamente síncrono y con señales de control independientes para cada puerto. Dichas señales de control pueden ser parcialmente utilizadas, permitiendo implementar memorias ROM al omitir los puertos de escritura. El ancho del bus de datos de cada puerto puede ser configurado de forma independiente.

La siguiente tabla muestra los ratios de profundidad y anchura que se pueden seleccionar:

Ancho	Profundidad	ADDR	DATA
1	4096	Addr<11:0>	Data<0>
2	2048	Addr<10:0>	Data<1:0>
4	1024	Addr<9:0>	Data<3:0>
8	512	Addr<8:0>	Data<7:0>
16	256	Addr<7:0>	Data<16:0>

Tabla 5-7: Ratio profundidad/ancho en bloque RAM

5.2.11.1.2 Recursos RAM ocupados

Existe un claro compromiso entre los recursos ocupados y los recursos que quedan libres para su uso futuro. Una configuración que implementa un gran tamaño de PROM y caché limita la incorporación de nuevos módulos en el proyecto.

En la asignación del tamaño de los distintos elementos se ha fijado como objetivo el no sobrepasar en ningún caso la capacidad de RAM ofrecida por los bloques *Dual-Port* RAM, dejando los bits ofrecidos por las LUTs para su completo uso por parte de la herramienta de síntesis y P & R para la construcción de las funciones lógicas necesarias.

Esta condición deja el número total de recursos disponibles en 14 Kbytes, que se asignan entre tamaño de caché, prom inferida o instanciada y recursos libres para uso futuro.

.

Elemento	Recursos asignados (KBytes)
Registros	2 KB
Caché Instrucciones	2 KB
Caché Datos	2 KB
Ins. Tags	0,5 KB
Data tags	0,5 KB
PROM instanciada	4 KB
Recursos libres	3 KB

Tabla 5-8: Asignación de recursos RAM

En la tabla 5-8 se reflejan las asignaciones de recursos realizadas, teniendo en cuenta que se reservan para registros de IU, posibles registros de FPU y coprocesador, además del resto de registros de los periféricos, un tamaño de aproximadamente 3 Kbyte.

Fijado el parámetro `nwindows` a 8 para compatibilidad con los compiladores existentes con la especificación SPARC, resulta un número de registros de la IU igual a:

Reg. IU=16*`nwindows`+8=136 registros (32 bits) [3-1]

A éstos hay que añadir los registros que se pueden direccionar a través del bus APB, y que incluyen los registros de configuración de los distintos periféricos, registros de estado (PSR), contador de programa (PC), contador de programa próximo (nPC), etc.

Todos registros tienen dos puertos de lectura y uno de escritura. Significaría ocupar dos bloques de RAM para la implementación de los registros. Ésto no es así ya que el fichero de registro se implementa de una forma especial en el microprocesador LEON. Para todas las tecnologías en la que se puede implementar el procesador, el fichero de registro es actualmente implementado mediante dos *dual-port* en paralelo, cada una con un puerto de escritura y uno de lectura, no utilizando el otro puerto de lectura que posee cada bloque de memoria.

Implica por tanto pasar de usar para conseguir registros de 32 bits, 2 bloques de RAM por cada 256 word (registros) a usar 4 bloques de RAM. Lo que hace un total aproximado de 3 Kbytes para el fichero de registros.

5.2.11.1.2.1 Caché memoria RAM

Con el tamaño de caché fijado se pueden almacenar 512 *words*, es decir, 512 instrucciones. Mientras que para datos se han fijado otras 512 *words* que permiten 512 enteros o 256 flotantes simples.

En la asignación de recursos se fija prioritario no limitar los recursos libres que podrían limitar futuras implementaciones con más periféricos. Esto se traduce en no elegir un elevado tamaño de caché de instrucciones y de datos.

Para la realización de la caché se usan bloques RAM *single-port* síncronos, cuyo ancho y profundidad dependen de la elección que se haya realizado en el *record* de configuración. La siguiente tabla muestra el tamaño de RAM ocupada para distintos tipos de configuraciones:

Tamaño Cache	***Words*/línea**	**RAM *tag***	**RAM datos**
1 kbyte	8	32x30	256x32
1 kbyte	4	64x26	256x32
2 kbyte	8	64x29	512x32
2 kbyte	4	128x25	512x32
4 kbyte	8	128x28	1024x32
4 kbyte	4	256x24	1024x32
8 kbyte	8	256x27	2048x32
8 kbyte	4	512x23	2048x32
16 kbyte	8	512x26	4096x32
16 kbyte	4	1024x22	4096x32

Tabla 5-9: Tamaños RAM caché

La caché de datos e instrucciones que se asigna ocupan un total de 8 bloques RAM. Aparte se encuentran los bloques necesarios para implementar las etiquetas (*tags*) de memoria de ambos tipos de caché, que para 2 + 2 Kbyte son un bloque (512 bytes) para cada uno.

El número total de bloques RAM ocupados por la caché de 2 + 2 Kbytes son 10.

5.2.11.1.2.2 PROM y recursos libres de asignación

Se ha elegido un mayor tamaño de la prom instanciada (4 kbyte) respecto al mínimo necesario, para permitir una mayor flexibilidad en el momento de incorporar nuevos programas. Hay que tener en cuenta que con la placa de desarrollo XSV no se dispone de memoria EEPROM para el almacenamiento de programa externo (la memoria Flash es utilizada para guardar la configuración de la FPGA).

El número de bloques RAM que son ocupados por la PROM instanciada es de 8 (4 Kbyte).

Quedan 3 Kbytes (6 bloques dual-port RAM) de recursos libres de RAM que pueden ser utilizador por futuros periféricos o módulos que se incluyan en la FPGA, aunque si se habilita la unidad de depuración DSU se necesitan al menos 1 Kbyte.

5.3 PROGRAMACIÓN

El programa que se realiza hace una demostración del uso de los puertos de Entrada/Salida y el *timer* del procesador LEON, además de verificar que la implementación del procesador funciona correctamente sobre una FPGA Virtex XCV800 en la placa de desarrollo XSV

Los puertos de E/S están conectados, según la configuración, a los dos displays (LEDs) y como entrada se utilizan dos pins, uno conectado a un interruptor (DIPSW7) y otro a un pulsador (SW4) que permiten controlar el flujo del programa.

Está realizado en lenguaje ensamblador compatible con las especificaciones SPARC V8.

El programa realiza la visualización en uno de los 7 segmentos de la cuenta desde 0 hasta 15 en hexadecimal (0 : F). Mediante el interruptor (DIPSW7) se fija la dirección de cuenta, hacia arriba o hacia abajo, y con el pulsador (SW4) se puede reanudar o parar la cuenta.

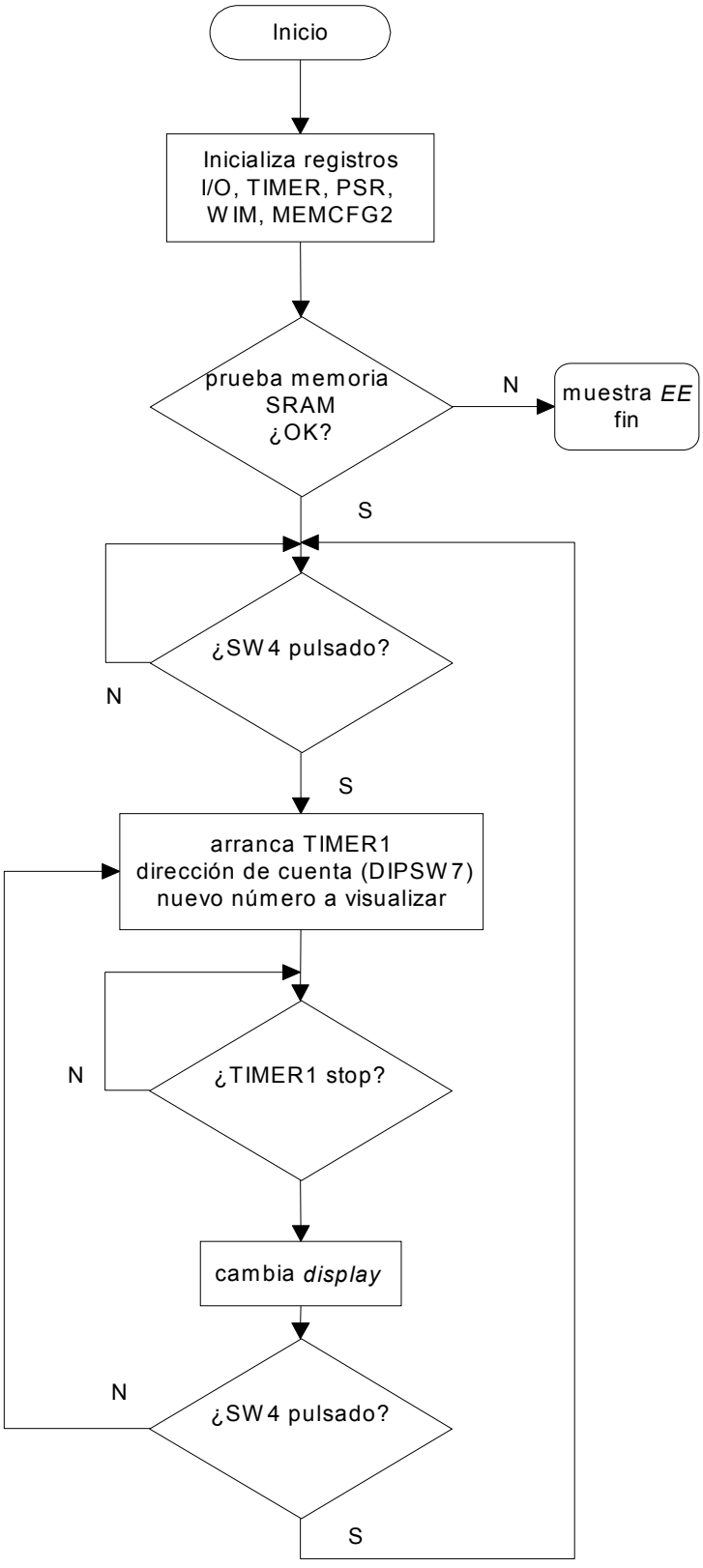
El programa además realiza la comprobación del funcionamiento de la memoria SRAM, y en caso de error detiene la ejecución.

El error de ejecución se indica mediante el apagado del segmento B9 de la barra gráfica. Además el segmento B0 tiene conectado la salida del periférico *watch-dog*.

Las interrupciones se deshabilitan al comienzo de la ejecución, ya que no se dispone de la tabla de excepciones almacenada en memoria.

El método utilizado para el seguimiento del contador *timer 1* es espera activa. Se detecta que se ha producido el final de la cuenta en el contador mediante la verificación continua del bit `EN` en el registro de control *timer 1*. El contador se programa para contar aproximadamente cada 1 s, teniendo en cuenta que el contador *scaler* ya ha sido fijado mediante hardware durante el lanzamiento (*boot*) para generar *ticks* a 1Mhz.

5.3.1 DIAGRAMA DE FLUJO



5.3.2 CODIGO FUENTE EN ENSAMBLADOR

```
! preescaler is set 1Mhz by boot
     ! %o0 register base
     ! %01 RAM base
     ! %g4 top of RAM
     ! %g0 0
     ! %g2 number display
     ! ***** I/O port configuration
     ! pio[14]  SW4
     ! pio[15]  DIPSW7
     ! reset SW1
     ! bexcn DIPSW1
     ! pio[13:7] SR[6:0] right display
     ! pio[6:0]  SL[6:0] left display
     !   *** display segment *****
     !          ---   S6
     !     S5  |   |  S4
     !          ---   S3
     !     S2  |   |  S1
     !          ---   S0


     .equ MASC, 0xffffc000
     .equ DIPSW, 0x8000
     .equ SW, 0x4000
     .equ CERO, 0x3bf7
     .equ UNO, 0x0977
     .equ DOS, 0x2ef7
     .equ TRES, 0x2df7
     .equ CUATRO, 0x1d77
     .equ CINCO, 0x35f7
     .equ SEIS, 0x37f7
     .equ SIETE, 0x2977
     .equ OCHO, 0x3ff7
     .equ NUEVE, 0x3d77
     .equ A, 0x3f77
     .equ B, 0x17f7
     .equ C, 0x06f7
     .equ D, 0x0ff7
     .equ E, 0x36f7
     .equ F, 0x3677
     .equ EE, 0x36ed
     .equ TIMERCOUNT, 0xfffff

     .equ BASERAM, 0x40000000
     .equ REGBASE, 0x80000000
     .equ TOPRAM,  0x40100000
     .equ IODIR, 0x3fff            ! input/output pio

    ! register address
    .equ IOPORT, 0xa0
    .equ IOREGDIR, 0xa4
    .equ TIMER1REG, 0x44
    .equ TIMER1CONTROL, 0x48
    .equ CACHECTRLREG, 0x14
    .equ MEMCFGREG2, 0x04

     .equ IOPORTINT, 0xa8

     .global _start
```

```
_start:
          wr      %g0, 0xc0, %psr        ! disable traps
          wr      %g0, 2, %wim
          flush
          set REGBASE , %o0
             set 0x100f, %o2
          st %o2, [%o0 + CACHECTRLREG]   ! enable cache

          set BASERAM, %o1
          ld [%o0 + MEMCFGREG2], %o2     ! read memcfg2
          and %o2, 0x0f, %o2
          set 0x0e50, %o3                ! set 1 Mbyte
          or %o3, %o2, %o2
             st %o2, [%o0 + MEMCFGREG2]  ! new memcfg2

          set TOPRAM, %g4
          sub %g4, 16, %g2
          or %g0, %g2, %sp               ! stack pointer
          or %g0, %g0, %g2
          st %g2, [%o0 + IOPORTINT]      ! disable interrupts

          set 0xffff0000, %g2
          ld [%o0 + IOREGDIR], %o2
          and %g2, %o2, %o2
          set IODIR, %g2
          or %g2, %o2, %o2
          st %o2, [%o0 + IOREGDIR] ! set I/O input and output

          ld [%o0 + IOPORT], %o2
          set 0xffff0000, %g2
          and %g2, %o2, %o2
          st %o2, [%o0 + IOPORT]         ! set outputs 0

          set TIMERCOUNT, %g2
          st %g2, [%o0 + TIMER1REG]      ! timer1 count
          set 0b100, %g2                 ! LD RL EN  timer reg bits
          st %g2, [%o0 + TIMER1CONTROL]  ! load counter

          ld [%o0 + IOPORT], %o2
          set UNO, %o3
          or %o3, %o2, %o2
          st %o2, [%o0 + IOPORT]         ! display 01

          st %o3, [%o1]                  ! check memory
          st %g0, [%o1 + 4]
          lda [%o1] 0, %o4
          subcc %o3, %o4, %g0
          be 1f
          st %g0, [%o1]
          set EE, %o4                    !  memory error
          ld [%o0 + IOPORT], %o2
          set MASC, %o3
          and %o3, %o2, %o2
          or %o2, %o4, %o2
          st %o2, [%o0 + IOPORT]         ! display EE (error)
          ba fin
          nop

 1:       set CERO, %o2                  ! store constants in memory
          st %o2, [%o1]
```

```
            set UNO, %o3
            st %o3, [%o1+4]
            set DOS, %o2
            st %o2, [%o1 + 8]
            set TRES, %o3
            st %o3, [%o1+ 12]
            set CUATRO, %o2
            st %o2, [%o1 + 16]
            set CINCO, %o3
            st %o3, [%o1+ 20]
            set SEIS, %o2
            st %o2, [%o1 + 24]
            set SIETE, %o3
            st %o3, [%o1+ 28]
            set OCHO, %o2
            st %o2, [%o1 + 32]
            set NUEVE, %o3
            st %o3, [%o1+ 36]
            set A, %o2
            st %o2, [%o1 + 40]
            set B, %o3
            st %o3, [%o1+ 44]
            set C, %o2
            st %o2, [%o1 + 48]
            set D, %o3
            st %o3, [%o1+ 52]
            set E, %o2
            st %o2, [%o1 + 56]
            set F, %o3
            st %o3, [%o1+ 60]

            or %g0, %g0, %g2
            set 4, %g2              ! %g2 -> number displayed (1)

1:          ld [ %o0 + IOPORT], %o2
            set SW, %o3
            and %o3, %o2, %o2
            srl %o2, 14, %o2        ! read SW value
            subcc %o2, 1, %g0
            be 1b
            nop

tim:        or %g0, %g0, %o3
            set 0b101, %o3          ! start timer1
            st %o3, [%o0 + TIMER1CONTROL]

            ld [ %o0 + IOPORT], %o4
            set DIPSW, %o5
            and %o5, %o4, %o4
            srl %o4, 15, %o4
            subcc %o4, 1, %g0       ! verify DIPSW value
            be up
            nop

            subcc %g2, %g0, %g0     ! down count
            be l_lim
            nop
            sub %g2, 4, %g2         ! new value to display

 lp_t:      ld [%o0 + TIMER1CONTROL], %o3
            set 0b001, %o5
```

```
          and %o5, %o3, %o3        ! loop until timer1 stop
          subcc %o3, %g0, %g0
          bne lp_t
          nop


          ld [%o1 + %g2], %o5
          ld [ %o0 + IOPORT], %o2
          set MASC, %o3
          and %o3, %o2, %o2
          or %o2, %o5, %o2
          st %o2, [ %o0 + IOPORT]  ! new number displayed


          set IOPORT, %o2
          lda [ %o0 + %o2] 0, %o5
          set SW, %o3
          and %o3, %o5, %o5        ! verify SW
          srl %o5, 14, %o5
          subcc %o5, 1, %g0
          be tim                   ! next count
          nop


          or %g0, %g0, %o3         ! SW push, stop count
          set 0b101, %o3           ! start timer1
          st %o3, [%o0 + TIMER1CONTROL]
lp_t2:    ld [%o0 + TIMER1CONTROL], %o3
          set 0b001, %o5
          and %o5, %o3, %o3        ! loop until timer1 stop
          subcc %o3, %g0, %g0
          bne lp_t2                ! delay 1s
          nop

          ba 1b                    ! wait new SW push


up:       subcc %g2, 60, %g0       ! up count
          be h_lim
          nop
          add %g2, 4, %g2
          ba lp_t                  ! go loop timer stop
          nop


h_lim:    or %g0, %g0, %g2         ! high limit (F)
          ba lp_t
          nop


l_lim:    or %g0, %g0, %g2         ! low limit (0)
          set 60, %g2
          ba lp_t
          nop


fin:      ta 0                     ! error mode: halt
          .end
```

5.3.3 ENSAMBLADO

El programa escrito es ensamblado para obtener el código ejecutable con el compilador de libre distribución GNU LECCS 1.1.5.

Las siguientes sentencias obtienen el código ejecutable:

```
sparc-rtems-as -o boot.o boot.s
sparc-rtems-ld -Ttext 0x00 boot.o
sparc-rtems-objcopy -O binary a.out boot.bin
bin2mif > virtex_prom1024.mif
```

El programa *as* ensambla el código, además de ejecutar el preprocesador. La utilidad *ld* realiza el linkado para obtener el código ejecutable, indicándole la dirección de memoria de comienzo del código de programa. Por último, *objcopy* crea el fichero ejecutable en el formato binario necesario.

Para poder realizar la instancia de la PROM en la FPGA Virtex es necesario generar mediante la utilidad *CoreGen* de Xilinx, una *netlist* que contenga los valores de la PROM. Estos valores deben ser leídos por *CoreGen* en un fichero *.mif*, que es el que contiene el código ejecutable del programa.

La utilidad bin2mif creada en C permite transformar el fichero "boot.bin" en un fichero *.mif* necesario para *CoreGen*.

5.4 PLACEMENT & ROUTING

En la siguiente figura se muestra la implementación física del procesador LEON que resulta tras la ejecución de las herramientas automáticas proporcionadas por Xilinx en la FPGA Virtex XCV 800.

Posteriormente se puede comparar con el resultado de realizar la misma implementación usando la técnica de control del rutado automático desarrollada, con la interfaz del bus AMBA rutada en unos recursos físicos fijados, y la lógica agrupada en zonas dentro de la FPGA.

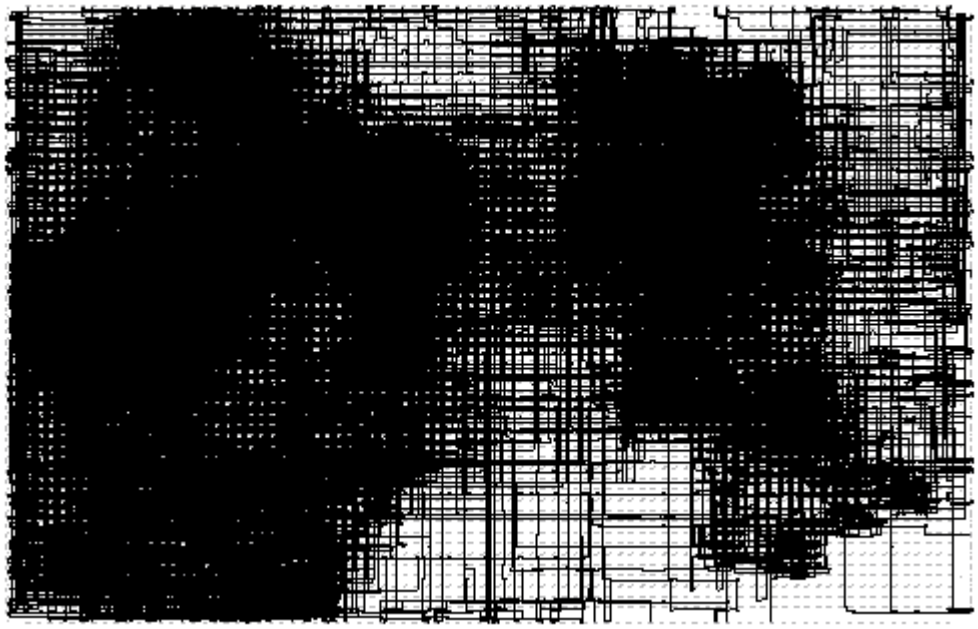


Figura 5-8: Implementación física del LEON en XCV800