2 PROCESADOR LEON

El LEON es un modelo de microprocesador en VHDL que implementa una arquitectura de 32 bits de acuerdo a la especificación SPARC V8.

Presenta las siguientes características: caché de instrucciones y datos separados, multiplicador y divisor hardware, controlador de interrupciones, unidad de depuración DSU con buffer para modo traza, dos *timers* de 24–bit, dos UARTs, función de ahorro de energía, función *watch-dog*, puerto I/O de 16-bit y un flexible controlador de memoria. El modelo en VHDL ha sido diseñado pensando en la síntesis y es completamente sintetizable con la mayoría de herramientas de síntesis y puede ser implementado en FPGAs y ASICs.

La versión de libre distribución GNU se diferencia de la completa en la ausencia del periférico PCI, y la ausencia del código VHDL que implementa una estructura tolerante a fallos, especialmente diseñada para aplicaciones muy críticas.

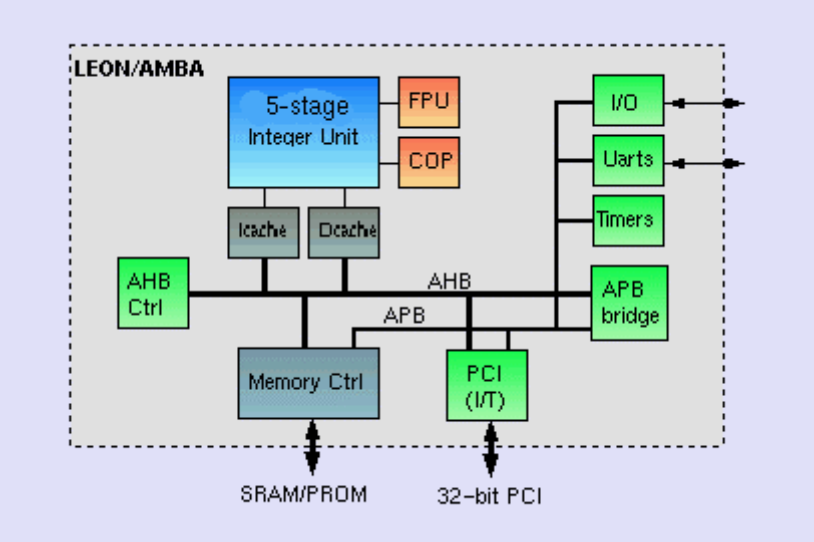

Figura 2-1:Esquema general del microprocesador LEON

El procesador tiene una completa implementación de los buses AMBA, APB y AHB (*on-chip*). Un fácil esquema de configuración hace simple añadir nuevos módulos, o comunicarse con procesadores que implementen el mismo bus

2.1 CARACTERÍSTICAS

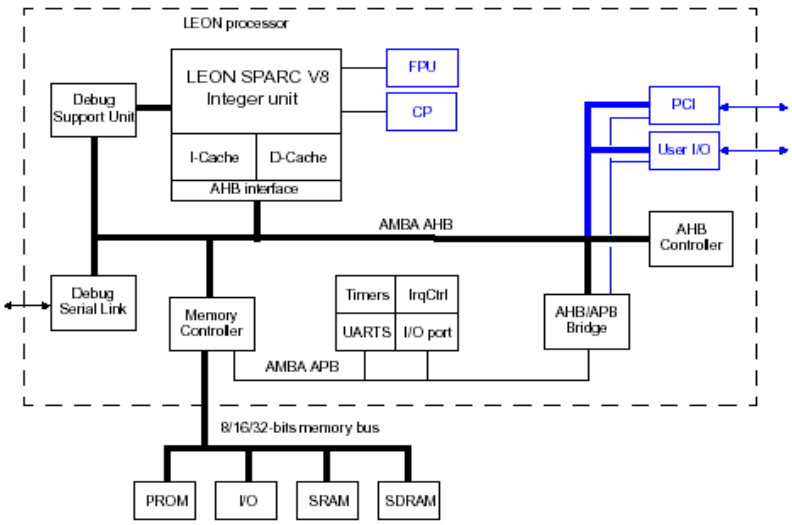

Figura 2-2: Estructura detallada del microprocesador LEON

2.1.1 IU

LEON implementa por completo el estándar SPARC V8, incluyendo instrucciones de multiplicación y división. El número de ventanas de registros se puede fijar dentro de los límites del estándar SPARC (2 – 32), por defecto 8 (compatibilidad con los compiladores existentes)

La implementación se centra en la portabilidad y baja complejidad. Las características más destacadas son:

- Etapa *pipeline* de 5 instrucciones
- Interfaz de caché de datos e instrucciones separados
- Soporte para ventanas de registros (2 – 32)
- Multiplicador configurable (16x16, 32x1, 32x8, 32x16 & 32x32)
- Opcional MAC 16 x 16 bit con acumulador de 40 – bit.
- Divisor Base 2

La figura 2-3 muestra el diagrama de bloques de la IU del procesador LEON.

El pipeline se desarrolla en 5 etapas, cada una de ellas ejecuta una instrucción: FE (*Instruction Fetch*), DE (*Decode*), EX (*Execute*), ME (*Memory*), WR (*Write*).

Asumiendo que las instrucciones se encuentran en la caché y la no existencia de bloqueos en lectura, una instrucción *store, load* se realiza en 2 ciclos máquina. Ejecutar la instrucción *sdiv/udiv* toma 35 ciclos máquina.

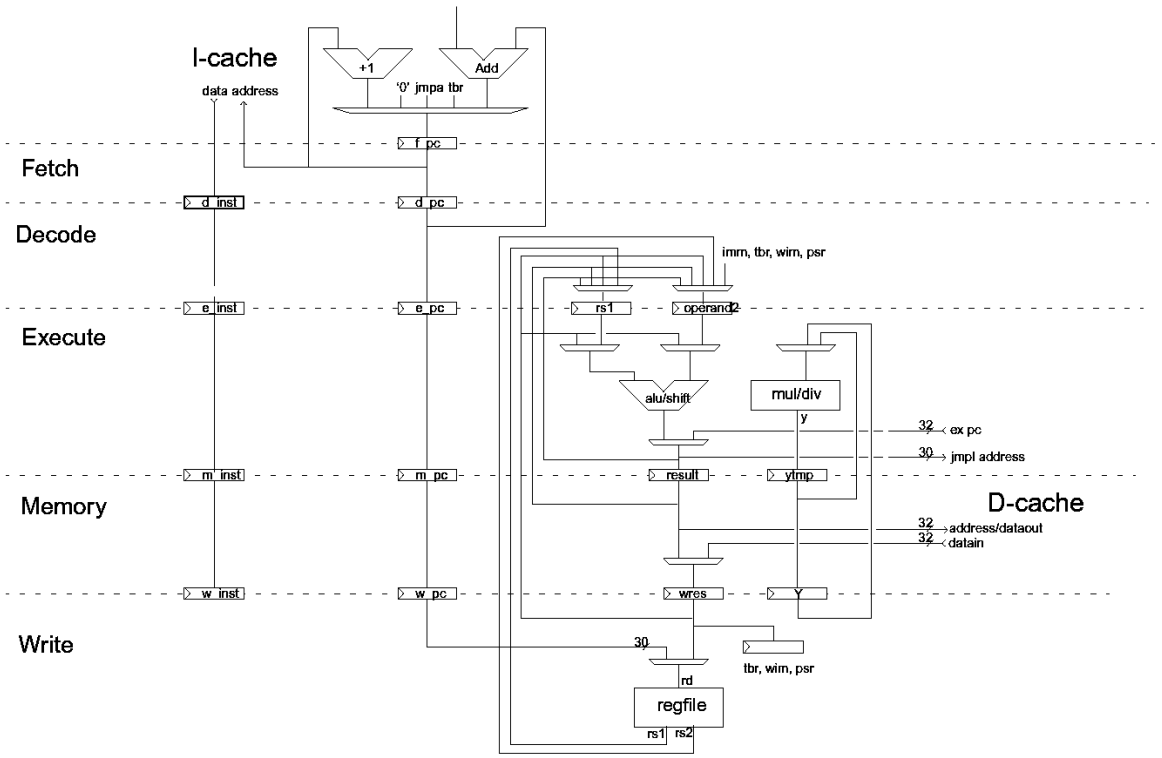

Figura 2-3: LEON IU

2.1.1.1 Operación de RESET

Para realizar el *reset* es necesario mantener al menos 1 ciclo de reloj a nivel bajo la entrada. La ejecución comienza en la dirección 0. Todos los registros tienen valores impredecibles excepto los siguientes que sí se ven afectados por el *reset*

Registro	**Valor tras *Reset***
PC	0x0
nPC	0x4
PSR	ET=0, S=1
CCR	0x0

Tabla 2-1: Valores tras *Reset*

LEON sigue el modelo de excepciones propuesto en la norma SPARC. Soporta un total de 256 excepciones, incluidas hardware y software junto con las interrupciones. Las excepciones se agrupan en una tabla que por cada entrada tiene las primeras cuatro instrucciones del manejador, lo que supone un total de 4 Kbytes necesarios para almacenarla. La dirección base de la tabla se fija en el registro `&TBR`.

2.1.1.2 Unidad de punto flotante (FPU) y coprocesador

El modelo del procesador LEON no incluye una FPU pero provee una interfaz directa par conectarlo a la FPU Meiko, y una interfaz general para la conexión con otro tipo de FPUs. También existe una interfaz para un coprocesador que pueda ser incluido en el mismo chip.

2.1.2 CACHÉ

El procesador LEON implementa una arquitectura Harvard con buses de instrucciones y datos separados. El tamaño es configurable entre 1 – 64 byte, con 8 – 32 bytes por línea. A cada línea le corresponde una etiqueta (*tag*) que permite identificar el contenido de la correspondiente caché. Se usa una técnica de relleno automático de la línea para mejorar la latencia del bus.

La caché de datos implementa un buffer doble de escritura y una política de escritura coherente (*write-through*). También puede caracterizar un protocolo de coherencia de datos en sistemas multi-maestro (bus-snooping)

Las zonas que pueden ser utilizadas por la caché suelen ser la RAM y la PROM, aunque en la configuración se pueden delimitar otras.

Los procesadores SPARC generan además de la dirección, una dirección alternativa de 8 bits conocida como ASI (Address Space Identifier), lo que provee el acceso a 256 espacios separados de direcciones de 32 bits. Durante la ejecución, por defecto se usa los ASIs 0x8-0xA (definido por SPARC). El resto según la norma, se dejan libres para distintas implementaciones. Usando las instrucciones `lda/sta`, se puede acceder a espacios alternativos.

El procesador LEON usa el resto de espacios direccionables para permitir realizar instrucciones sobre la caché como, `flush` (borrado), forzar lectura directa desde memoria (*forced cache miss*), o leer/modificar las etiquetas (*tags*) de la caché de datos e instrucciones.

2.1.3 PERIFÉRICOS

Un gran número de funciones son implementadas en el propio chip que incluye el procesador. Los periféricos son controlados a través de registros que son mapeados a direcciones del bus APB.

2.1.3.1 Controlador de Interrupciones

El controlador de interrupciones gestiona un total de 15 interrupciones, originadas por fuentes externas e internas. Cada interrupción puede ser programada a uno de dos niveles de prioridad. La siguiente figura muestra es diagrama de bloques del controlador.

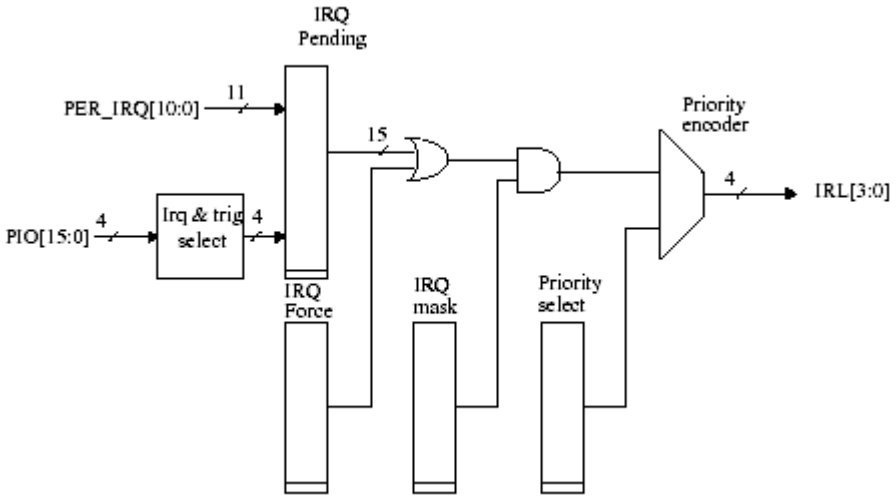

Figura 2-4: Controlador de interrupciones

Mediante los correspondientes registros, se puede forzar una interrupción, enmascararla o fijar su prioridad. Dentro de cada nivel de prioridad de los dos que existen, la interrupción 15 es la que tiene una prioridad más alta y la interrupción 1 la más baja.

Después de un *reset*, el registro de máscara esta puesto a cero y el resto de registros tienen valores indefinidos

En la tabla 2-2 se muestran las asignaciones de las interrupciones a los distintos periféricos, quedando cuatro libre para que puedan ser utilizadas por nuevos periféricos.

El controlador de interrupciones secundario tiene una interrupción asignada en el controlador primario, la cuál se activa cuando lo hacen alguna de las interrupciones asignadas al controlador secundario.

La interrupción número 15 no es enmascarable por la IU, y por tanto, debe ser manejada con precaución.

Interrupción	Fuente
15	Definido por usuario
14	Definido por usuario
13	Definido por usuario
12	Definido por usuario
11	DSU trace buffer
10	Control. Int. Secundario
9	Timer 2
8	Timer 1
7	Paralelo I/O[4]
6	Paralelo I/O[3]
5	Paralelo I/O[2]
4	Paralelo I/O[1]
3	UART1
2	UART2
1	AHB Error

Tabla 2-2: Asignación de interrupciones

2.1.3.2 Controlador de Interrupciones Secundario

Se dispone de un segundo controlador de interrupciones que puede añadir hasta 32 nuevas interrupciones, para ser usadas por nuevos módulos implementados en el chip. La figura 2-5 muestra el diagrama de bloques de controlador de interrupciones secundario.

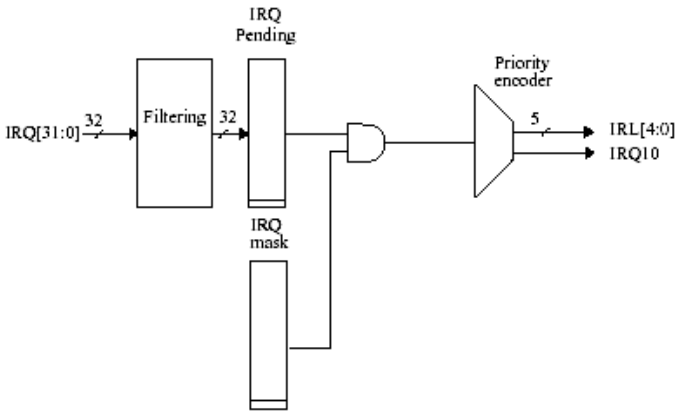

Figura 2-5: Controlador de interrupciones secundario

Las interrupciones asignadas a este segundo controlador pueden ser filtradas previamente, para activarse sólo en determinados eventos. Sólo se implementan las interrupciones que se configuren. La configuración de éstas se realiza en el módulo `mcore.vhd`.

2.1.3.3 Timers

Se implementan dos *timers* de 24-bit, un 24-bit *watchdog* y un 10-bit *prescaler*.(Figura 2-6)

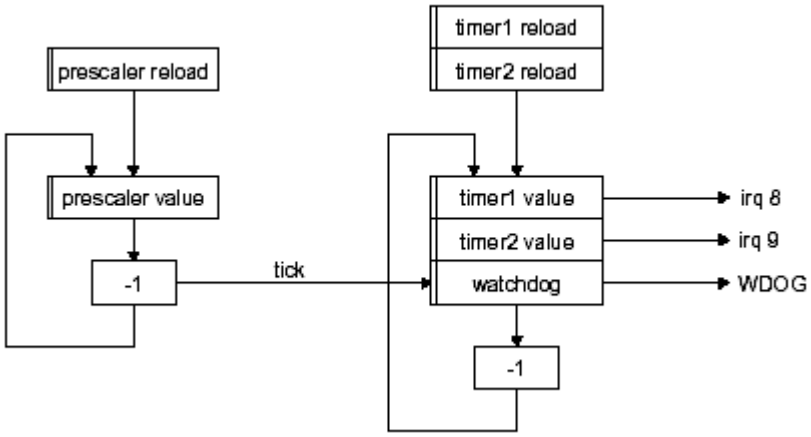

Figura 2-6: Unidad *timer*

El *prescaler* se rige por el reloj del sistema, y es decrementado en cada ciclo del mismo. Cuando termina la cuenta vuelve a cargarse y genera para los *timers* un *tick*.

Mediante el registro de control de cada *timer*, se puede comenzar la cuenta, reiniciar o situarlo en el estado de reinicio automático. Se genera una interrupción cuando se termina la cuenta, pero sólo será tenida en cuenta por la IU si no está enmascarada.

El *watchdog* opera de forma similar a los *timers*, con las diferencias de que siempre está habilitado y cuando termina la cuanta afirma la señal WDOG. Esta señal puede ser utilizada para generar un *reset.*

2.1.3.4 UARTs

Se incorporan dos UARTs idénticas para comunicaciones serie.

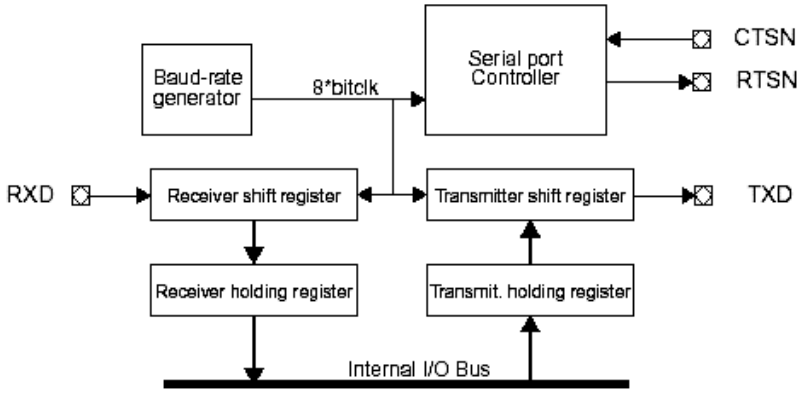

Figura 2-7: UART

Soportan tramas de datos de 8 bits, un bit opcional de paridad y un bit de stop. Para genera la velocidad de comunicación, cada UART dispone de un divisor de reloj de 12 bits. El control de flujo hardware se realiza mediante las señales RTSN/CTSN.

Admite un modo de realimentación con el objeto de testar el correcto funcionamiento de la UART.

Cada UART tiene asignada una línea de interrupción que puede generar en diferentes situaciones, todas ellas programables mediante el registro de control, como bit recibido, bit transmitido, error en paridad de trama...

2.1.3.5 Puerto paralelo I/O

Se dispone de un puerto de entrada/salida programable. El puerto está dividido en dos grupos, los 16 bits más bajos se encuentra en los pins PIO[15:0] mientras que los 16 bits más altos usan D[15:0] y sólo pueden ser usados cuando todas loas zonas de memoria estén programadas en modo 8 o 16 bit. Si el controlador SDRAM está habilitado tampoco puede ser usado.

Los 16 bits más bajos pueden ser programados individualmente como entrada o salida en el registro de dirección. La dirección de los 16 bits superiores sólo puede ser programada en bytes.

Algunos bits tienen usos alternativos, como entradas/salidas de UARTs o entradas externas de interrupciones.

2.1.3.6 Otros periféricos

Desde el bus APB se pueden acceder a otros tres registros que constituyen por sí mismos cada uno un periférico.

- Registro de configuración del procesador LEON. Es un registro de 32 bits que muestran las opciones de configuración que fueron elegidas antes de la síntesis y que se han implementado: tamaño de caché. Es un registro cableado en hardware.
- Registro *Power-down*. Escribiendo una cantidad arbitraria en este registro el procesador entra en un modo de espera o ahorro de energía. Se reanuda la ejecución cuando llega una interrupción no enmascarada. El resto de periféricos funcionan normalmente.
- Registro de estado AHB. Cualquier respuesta errónea en el bus AHB es almacenada en este registro, junto con la dirección que ha propiciado el fallo.

2.1.4 ACCESOS A MEMORIA EXTERNA

El bus de memoria provee una interfaz de acceso directa a PROM, dispositivos de memoria de E/S mapeados, SRAM y SDRAM. Soporta 2 bancos de PROM, un banco E/S, 5 bancos de memoria SRAM y dos bancos de memoria SDRAM.

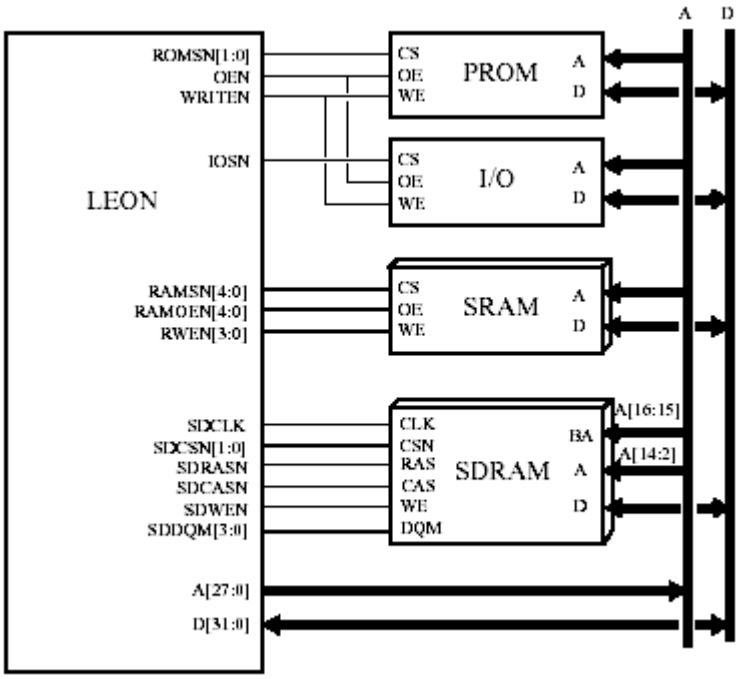

Figura 2-8:Interfaz de memoria

El controlador de memoria actúa como esclavo del bus AHB. Aunque su configuración es programada a través del bus APB. Dispone de tres registros de configuración: MCR1 (configura ROM e I/O), MCR2 (configura SRAM y SDRAM), y MCR3 (contador de refresco de la memoria SRAM).

El bus de memoria se puede programar para realizar accesos a dispositivos de 8 o 16 bit (para esto se necesitan que los bancos de memoria esté conectados de forma que admitan esta posibilidad), y la memoria PROM se puede deshabilitar si se incluye en el mismo chip que el procesador.

El controlador decodifica un espacio de direcciones de 2 Gbyte.

Tamaño	Mapeado
512 M	Prom
512 M	I/O
1 G	SRAM/SDRAM

Tabla 2-3: Mapa de memoria

Dispone de una señal para alargar los ciclos de lectura `brdyn` y otra para indicar errores en un acceso por parte del periférico (`bexcn`).

2.1.5 DSU (DEBUG SUPPORT UNIT)

El procesador LEON incluye una unidad de depuración hardware (DSU) que posibilita ejecución de programas paso a paso, trazado de la ejecución, puntos de ruptura hardware, etc.

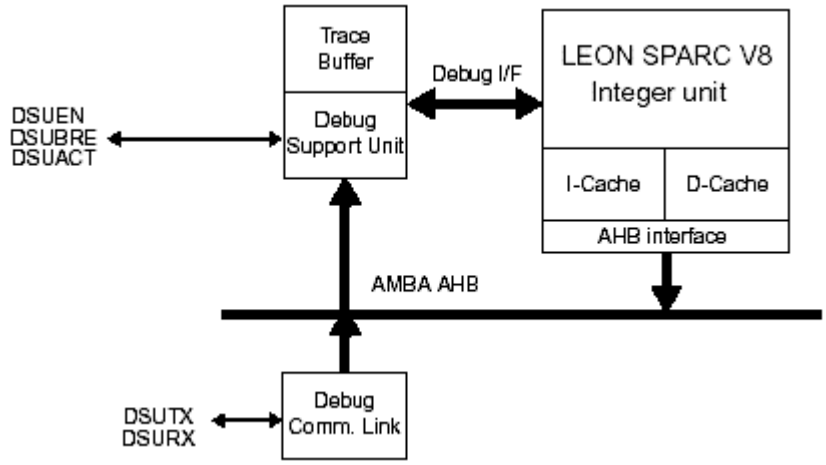

Figura 2-9: DSU

Incluye un dispositivo de comunicación exterior propio (UART), que implementa el estándar asíncrono de comunicación RS232.

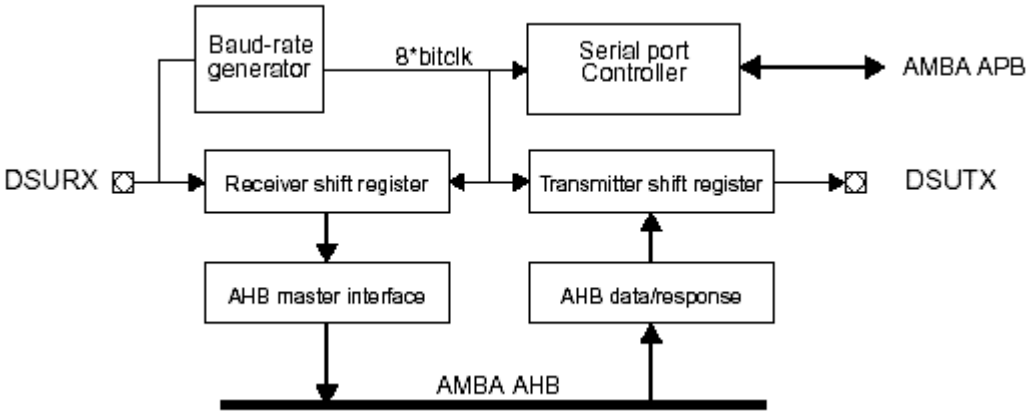

Figura 2-10: Unidad de comunicaciones DSU

La DSU se encuentra enclavado en el bus AHB como un esclavo, y le corresponde un espacio de direcciones de 2 Mbyte. Mediante estas direcciones un maestro del bus AHB puede acceder a los registros del procesador y a los contenidos de *buffer* de trazado (2 Kbytes).

2.1.6 ARQUITECTURA VHDL

El diseño del procesador LEON está constituido por un conjunto de módulos VHDL que siguen una determinada jerarquía. Además de un conjunto de *packages*, que definen distintas tecnologías y configuraciones del dispositivo.

2.1.6.1 Clocking scheme

El modelo implementa dos esquemas a la hora de usar la señal CLK: CLK continua y el uso de multiplexores para habilitar los registros *pipeline* de la IU, o *gated CLK* que es parado cuando sea necesario en las etapas de *pipeline*. El esquema de reloj continuo produce un análisis temporal más sencillo, pero *gated CLK* ocupa menos área y consume menos. Puede ser seleccionado mediante el elemento GATEDCLK.

2.2 BUS AMBA

Las especificaciones AMBA definen un conjunto de buses de altas características especialmente diseñados para sistemas integrados en un chip. Entre ellos se encuentra el bus AHB, o bus de alta velocidad para la comunicación del procesador con el gestor de memoria, DMA u otros procesadores. Por el contrario, el bus APB permite la comunicación con dispositivos de E/S de baja velocidad u otros periféricos. Para conectar los dos buses el controlador del bus APB es un esclavo del bus AHB y actúa como puente.

2.2.1 TERMINOLOGÍA

- Ciclo del Bus: Es la unidad básica y corresponde con un ciclo del reloj del bus definido entre dos transiciones consecutivas con pendiente positiva.
- Transferencia del bus: Una transferencia AHB es una operación de lectura o de escritura de un objeto de datos, que puede tomar uno o más ciclos del bus. La transferencia termina cuando existe una respuesta de finalización por parte del esclavo seleccionado. El bus AHB soporta transferencias de 8, 16, 32, 64, 128 bits. Una transferencia del bus APB siempre requiere dos ciclos de bus.
- Operación ráfaga (*burst*): Es definida como una o más transferencias de datos por parte de un maestro del bus AHB todas seguidas. El bus APB no soporta transferencias ráfaga.

2.2.2 BUS AHB

Es un bus de sistema de altas características y elevada tasa de transferencia que soporta múltiples maestros.

Entre sus prestaciones destacan:

- Transferencias en modo ráfaga
- División de transferencias (*Split*)
- Operaciones en un único ciclo de bus
- Implementación no tri-estado
- Configuraciones de bus de datos ancho (54/128 bits)

Un sistema suele incluir como maestro al procesador, aunque también puede tener una DMA, o un DSP incluidos como maestros.

Como esclavos suelen estar el puente APB/AHB, el controlador de memoria y cualquier memoria interna que exista.

Un sistema AMBA AHB incluye los siguientes componentes:

- AHB maestro. Inicia las transferencias de lectura y escritura, dando la dirección y la información de control. Sólo un maestro puede usar el bus a la vez.
- AHB esclavo. Un esclavo responde ante una operación de escritura o lectura en un rango de direcciones.
- AHB arbitro. Asegura que un solo maestro está usando el bus al mismo tiempo. AMBA fija el protocolo de arbitraje, no el algoritmo que escoge a un maestro de entre los que requieren el bus. Si hay sólo un maestro es trivial.
- AHB decodificador. Se encarga de generar las señales de selección de cada esclavo en función de la dirección de la transferencia.

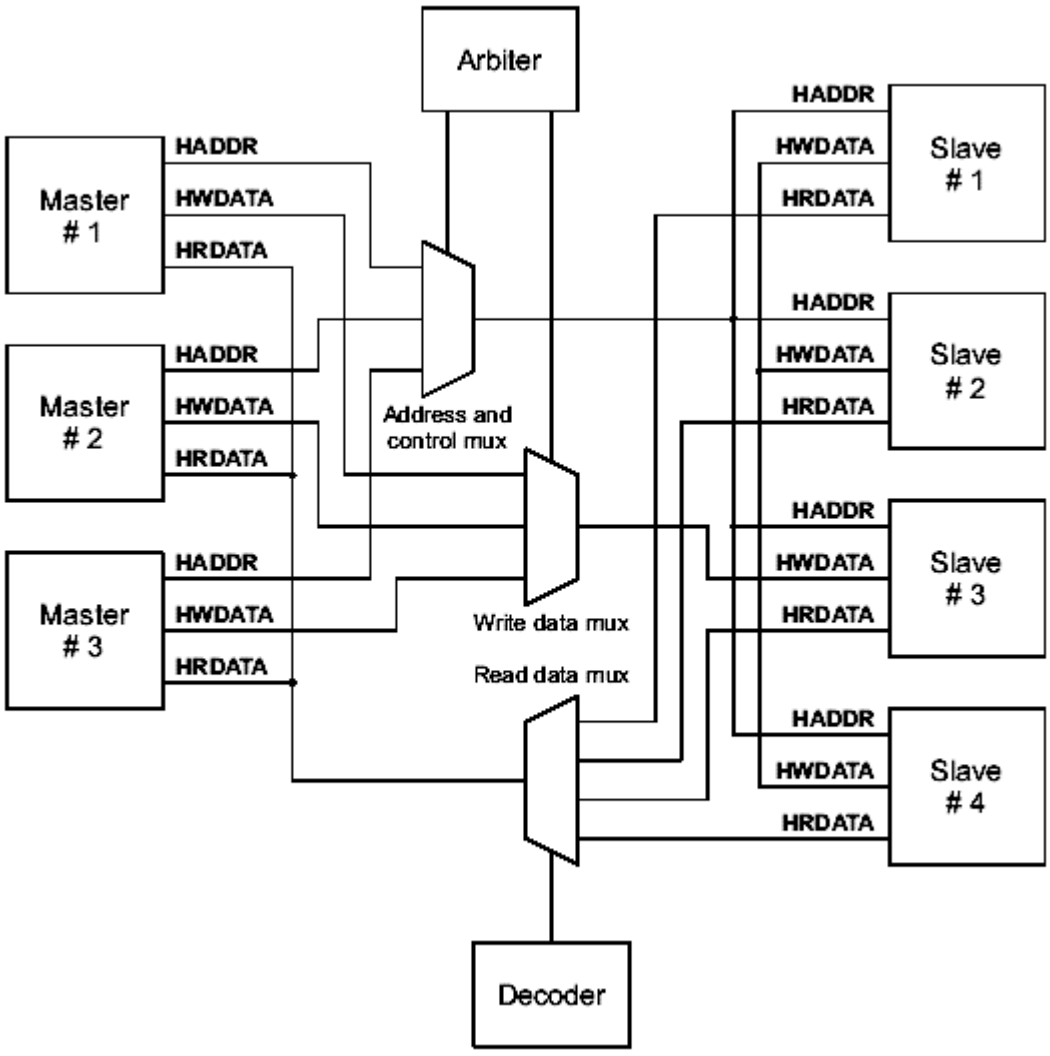

Figura 2-11: Conexión típica en un sistema multi-maestro

2.2.2.1 Señales AHB

Las señales que se refieren al bus AHB suelen incluir la letra H como prefijo, para diferenciarlas del resto.

- **HCLK** (reloj del bus). Sincroniza todas las transferencias
- **HRESETn.** Esta señal es activa a nivel bajo, e inicia el sistema y el bus
- HADDR[31:0]. Dirección de 32 bits generada por el maestro
- **HTRANS [1:0].** Tipo de transferencia actual (NONSEQUENTIAL, SEQUENTIAL, IDLE, BUSY) generada por el maestro.
- **HWRITE.** A nivel alto indica transferencia de escritura y a nivel bajo transferencia de lectura. Generada por el maestro.
- **HSIZE[2:0]**. Tamaño de la transferencia (8, 16, 32, 64 bits). El protocolo permite transferencias de hasta 24 bits. Generada por el maestro.
- **HBURST[2:0].** Indica si la transferencia es en modo ráfaga. Generada por el maestro.
- **HPROT[3:0].** Provee información adicional acerca de la transferencia. Generada por el maestro.
- **HWDATA[31**:0]. Bus de datos 32 bits en una operación de escritura. Generado por el maestro.
- **HSELx**. Señal de selección para el esclavo x. Generada por el decodificador.
- **HRDATA[31:0].** Bus de datos 32 bits en una operación de lectura. Generada por el esclavo seleccionado.
- **HREADY**. Si está a nivel alto indica que la transferencia ha finalizado, si está a nivel bajo extiende la transferencia otro ciclo de bus más. Generada por el esclavo
- **HRESP[1:0].** Provee información acerca de la transferencia. (OK, ERROR, RETRY, SPLIT). Generada por el esclavo seleccionado.

Además para las situaciones multi-maestro existen otro grupo de señales:

- **HBUSREQx.** Señal generada por el maestro x para pedir el bus al arbitro.
- **HLOCKx.** Bloqueo del bus por parte del maestro x.
- **HGRANTx.** Generada por el arbitro, indica al maestro que se le ha concedido el bus.
- **HMASTER[3:0].** Estas señales del arbitro indican que maestro tiene en cada instante asignado el bus.
- **HMASTLOCK**. Generada por el arbitro indica que el maestro actual está realizando una transferencia bloqueada.

- **HPSPLITx[15:0].** El esclavo x indica a que maestro se le debe de conceder el bus para terminar una transferencia dividida.

Una transferencia normal en el bus AHB consiste se realiza en dos ciclos de bus. Pero si se realizan varias transferencias, sin estados de espera, se consiguen realizar en un promedio de un ciclo de bus por transferencia.

La transferencia consta de dos fases:

- Fase de dirección. Dura un ciclo de bus, y en ella el maestro coloca la dirección y las señales de control.
- Fase de datos. Dura un ciclo como mínimo. Se responde con el dato correspondiente a la dirección del ciclo anterior.

2.2.2.2 AHB en el procesador LEON

El bus AHB implementado en el procesador LEON permite hasta 16 maestros. El procesador LEON suele ser el maestro 0. Cada maestro se conecta a través de dos *records*:

```
-- AHB master inputs (HCLK and HRESETn routed separately)
   type AHB_Mst_In_Type is  record
    HGRANT: Std_ULogic;           -- bus grant
    HREADY: Std_ULogic;           -- transfer done
    HRESP:  Std_Logic_Vector(1downto 0); -- response type
    HRDATA: Std_Logic_Vector(HDMAX-1 downto 0); -- read data bus
   end record;

   -- AHB master outputs
   type AHB_Mst_Out_Type is   record
    HBUSREQ:    Std_ULogic;            -- bus request
    HLOCK:      Std_ULogic;       -- lock request
    HTRANS:     Std_Logic_Vector(1 downto 0); -- transfer type
    HADDR: Std_Logic_Vector(HAMAX-1 downto 0); -- address bus
    HWRITE:     Std_ULogic;                  -- read/write
    HSIZE: Std_Logic_Vector(2 downto 0); -- transfer size
    HBURST:Std_Logic_Vector(2 downto 0); -- burst type
    HPROT: Std_Logic_Vector(3 downto 0); -- protection control
    HWDATA:Std_Logic_Vector(HDMAX-1 downto 0); -- write data bus
   end record;

   -- AHB slave inputs (HCLK and HRESETn routed separately)
   type AHB_Slv_In_Type is record
    HSEL:    Std_ULogic;                 -- slave select
    HADDR: Std_Logic_Vector(HAMAX-1 downto 0); -- address bus
    HWRITE:  Std_ULogic;                 -- read/write
    HTRANS:  std_Logic_Vector(1 downto 0); -- transfer type
    HSIZE:  Std_Logic_Vector(2       downto 0); -- transfer size
    HBURST: Std_Logic_Vector(2       downto 0); -- burst type
    HWDATA: Std_Logic_Vector(HDMAX-1 downto 0); -- write data
    HPROT:  Std_Logic_Vector(3  downto 0); -- protection control
    HREADY: Std_ULogic;                 -- transfer done
    HMASTER:Std_Logic_Vector(3 downto 0); -- current master
    HMASTLOCK:  Std_ULogic;              -- locked access
   end record;
```

```
-- AHB slave outputs
type AHB_Slv_Out_Type is    record
 HREADY:     Std_ULogic;                  -- transfer done
 HRESP:      Std_Logic_Vector(1 downto 0); -- response type
 HRDATA: Std_Logic_Vector(HDMAX-1 downto 0); -- read data bus
 HSPLIT: Std_Logic_Vector(15 downto 0); -- split completion
end record;
```

2.2.3 BUS APB

El bus APB está optimizado para una complejidad de interfaz reducida y bajo consumo. Es un bus secundario dentro de un sistema con bus AHB.

El puente AHB/APB es el encargado de realizar las transferencias como maestro en el bus APB, siendo un esclavo en el bus AHB. Es usado como interfaz a periféricos que requieren una baja tasa de transferencia.

2.2.3.1 Señales APB

Suelen referirse con la letra P como prefijo. Son las siguientes:

- **PADDR[31:0]**. Bus de direcciones APB
- **PELx.** Señal de selección del esclavo x APB, generada por el puente AHB/APB.
- **PENABLE.** Señal de habilitación. Se usa para sincronizar la transferencia.
- **PWRITE.** A nivel alto indica acceso de escritura, a nivel bajo acceso de lectura.
- **PRDATA.** Bus de datos en transferencias de lectura. Anchura hasta 32 bits.
- **PWDATA.** Bus de datos en transferencias de escritura. Anchura hasta 32 bits.

2.2.3.2 APB en el procesador LEON

Los periféricos se conectan mediante los siguientes *records*:

```
-- APB slave inputs (PCLK and PRESETn routed separately)
   type APB_Slv_In_Type is     record
    PSEL:     Std_ULogic;             -- slave select
    PENABLE:  Std_ULogic;             -- strobe
    PADDR:Std_Logic_Vector(PAMAX-1 downto 0); -- address bus
    PWRITE:   Std_ULogic;                      -- write
    PWDATA: Std_Logic_Vector(PDMAX-1 downto 0); -- write data
   end record;

 -- APB slave outputs
   type APB_Slv_Out_Type is      record
    PRDATA: Std_Logic_Vector(PDMAX-1 downto 0); -- read data bus
   end record;
```

2.3 SPARC V8

SPARC es un conjunto de especificaciones que definen un entorno a un determinado software (compilador). El hardware no tiene que ser el mismo en todas las implementaciones, pero si es visto igual por parte del software que se ejecuta.

SPARC es una CPU ISA (*Instruction Set Achitecture*), derivada de la arquitectura RISC. SPARC fue diseñado como un objetivo para optimizar compiladores e implementaciones hardware *pipelin*e sencillas.

Fue formulado por SUN Microsystem en 1985, basándose en la arquitectura RISC I & II de 1982 propuesta por Berkeley.

Para lenguajes como C++, donde predomina la programación orientada a objetos, la arquitectura de ventanas de registros que SPARC proponía resultaba en un menor número de instrucciones ejecutadas.

2.3.1 CARACTERÍSTICAS

- Espacio linear de direcciones de 32 bits
- Pocas y simples instrucciones. Formato homogéneo, todas las instrucciones tienen un ancho de 32 bits, y deben de estar alineadas en memoria en los múltiplos de 4 (32 bits). Sólo las instrucciones *load/store* permiten acceder a memoria y I/O.
- Ficheros de registros “en ventanas”. En todo momento un programa ve 8 registros globales y una ventana de 24 registros, que pueden ser utilizados para almacenar direcciones de retorno, argumentos o variables locales.
- Un fichero de registros exclusivo para la unidad de punto flotante (FPU), programables en tamaño mediante software
- Transferencia de control retrasada. El procesador recoge la siguiente instrucción que hay después de ejecutar una instrucción de control de transferencia.
- Rápidos manejadores de excepciones. Las excepciones son vectorizadas en una tabla, donde se encuentran las cuatro primeras instrucciones de la rutina. La dirección base de la tabla se fija mediante el registro %TBR.
- Instrucciones para sincronización con otros procesadores. Garantizando una lectura y almacenamiento atómicos con ciertas instrucciones.
- Conjunto de instrucciones para coprocesador definidas.

La arquitectura SPARC admite dos modos de funcionamiento, modo usuario y modo supervisor. Este último puede ejecutar cualquier instrucción, incluyendo las privilegiadas. Si en modo usuario se ejecuta alguna de éstas se produce una excepción.

La especificación V8 añade al conjunto de instrucciones la operación de multiplicación y división, además de la instrucción *smac* que realiza un módulo MAC.

SPARC es una máquina que sigue el convenio de direcciones *big-endian*, acceder a una dirección de memoria es acceder a su byte más significativo (MSB), si se siguen realizando accesos de un byte, se acceden a los siguientes hasta el byte menos significativo (LSB).

Los procesadores SPARC generan junto a la dirección, una dirección alternativa de 8 bits conocida como ASI (Address Space Identifier), lo que provee el acceso a 256 espacios separados de direcciones de 32 bits. SPARC define como espacios de direcciones normales los que usan ASIs entre 0x8-0xA. Usando instrucciones *lda/sta*, se puede acceder a espacios alternativos.

Las instrucciones de SPARC se agrupan en seis categorías:

- Lectura/Escritura
- Aritméticas/Lógicas/Desplazamientos
- Control de transferencia
- Escritura/Lectura de registros
- Unidad de Punto Flotante
- Coprocesador

2.3.1.1 Registros

Como ya se ha menciona, SPARC define una arquitectura que usa ventanas de registros. Una implementación de una IU puede contener desde 40 hasta 520 registros de propósito general de 32 bits. Corresponden a 8 registros globales (g0 –g7), más una cola circular de 2 a 32 conjuntos de 16 registros cada uno. El número de ventanas de registros depende de la implementación y varia entre 2 y 32 (NWINDOWS), y por tanto, el número total de registros es dependiente de la implementación.

En un determinado instante, una instrucción puede acceder a los 8 registros globales y a una ventana. La ventana incluye 24 registros, 16 propios que constituyen los registros locales (8 *local registers*) y registros de entrada (8 *in registers*), junto con 8 registros denominados de salida (*out registers*), que son los registros de entrada de la ventana adyacente.

Existe un registro que apunta a la ventana actual, CWP (*Current Window Register*), que es controlado por el software supervisor. El número de ventanas real en una implementación es invisible para el software en modo usuario.

Cuando ocurre una excepción, automáticamente se cambia a la ventana adyacente. También se puede cambiar de ventana mediante las instrucciones *save* y *restore,* permitiendo almacenar los argumentos en los registros *out* de la ventana que son los registros *in* de la siguiente.

La cola circula que implementa el conjunto de registros puede verse en la figura 2-12.

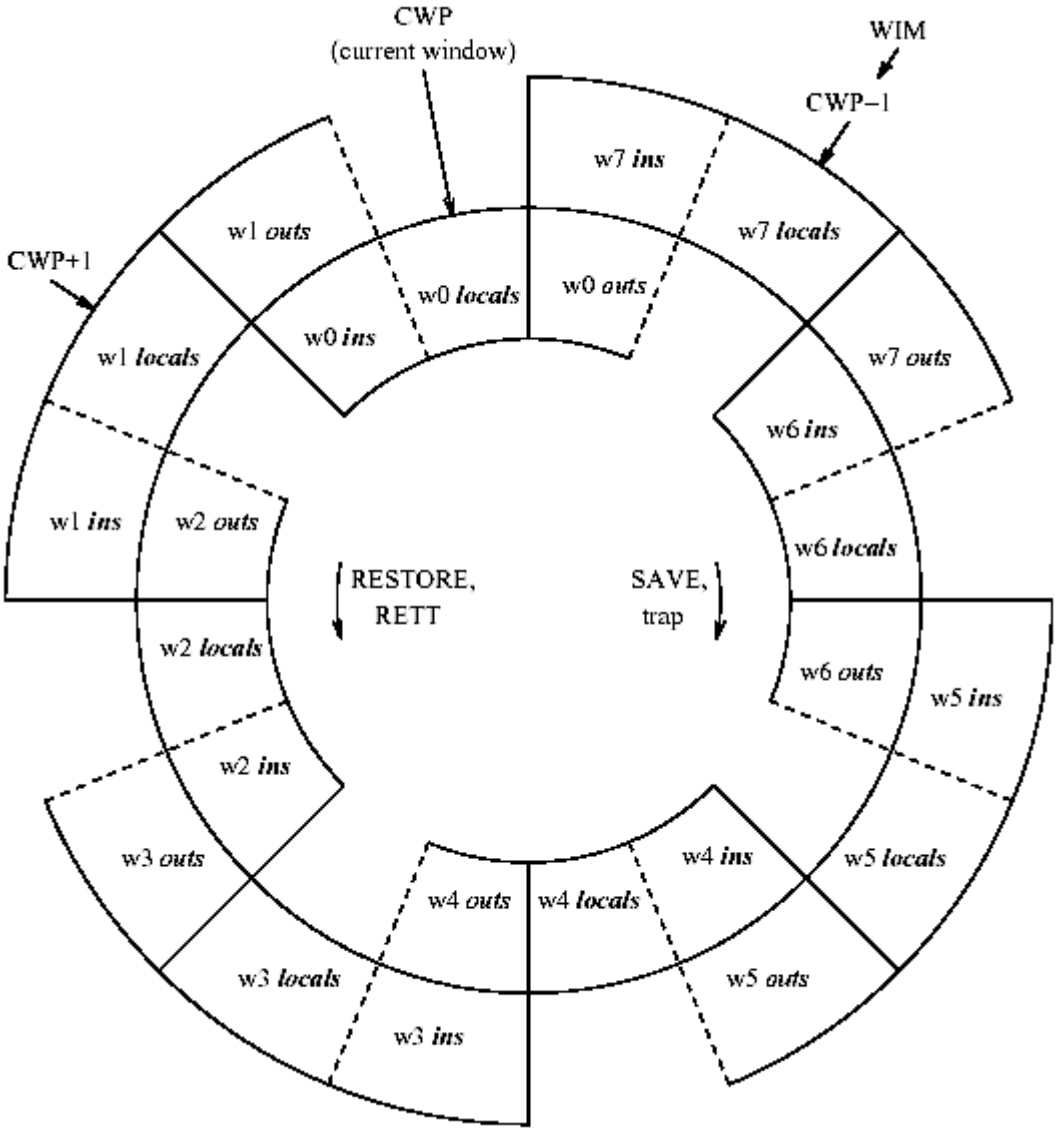

Figura 2-12: Cola de registros de propósito general