Core PCITWBM - Manual de Usuario

Tabla de contenido

Características del core PCITWBM

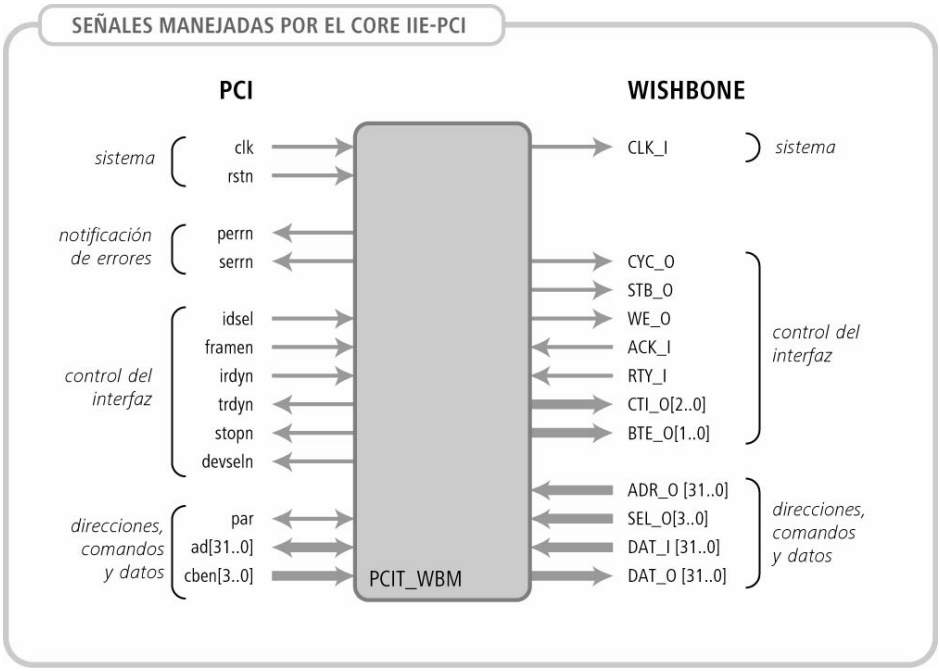
Principales características del core PCI:

- funcionamiento PCI target con las siguientes características:
  - 32 bit de ancho de palabra
  - detección de errores de paridad
  - hasta 6 registros de dirección de base (BARs) con tamaño y tipo ajustable en el momento de síntesis.
- soporte de la mayoría de los comandos PCI, incluyendo:
  - lectura y escritura de configuración
  - lectura y escritura de memoria
  - lectura y escritura de I/O
- soporte de transferencias en modo burst
- funcionamiento comprobado utilizando FPGA ACEX EP1K100 de Altera en buses PCI de 33MHz
- desarrollado en lenguaje VHDL
- interfaz de aplicación Wishbone compatible
- la aplicación y el bus PCI pueden utilizar diferentes relojes.
- no implementa el manejo de interrupciones

Descripción general

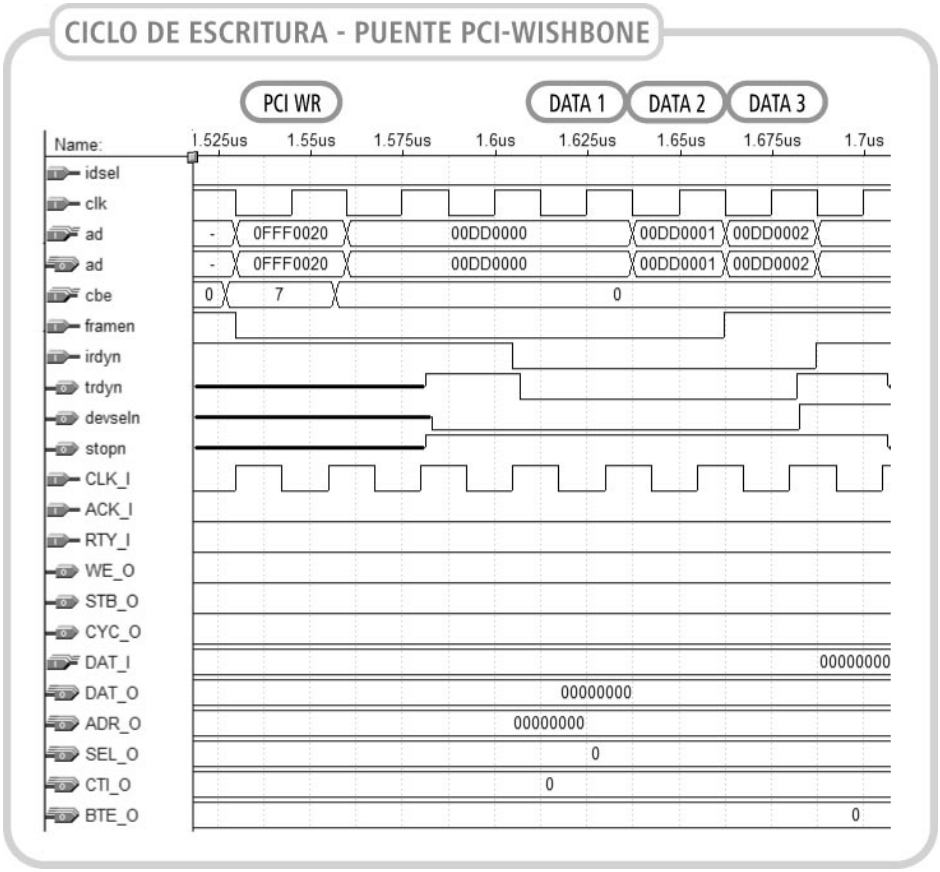
El core PCI es una implementación del interfaz PCI para ser utilizada por diseños que hagan uso del estándar Wishbone.
Su funcionamiento se puede interpretar como el de un puente entre los dos buses, el PCI y el Wishbone, es decir envía y recibe datos de un bus a otro.

La ventaja de pasar de un tipo de bus a otro radica en que la especificación Wishbone esta pensada para interconectar diseños dentro de un mismo integrado. Las interfaces y los ciclos de transferencia de datos son iguales para todos los cores IP sin importar su función (controlador de memoria, interfaz PCI, registros, etc.) y no es necesario conocer el funcionamiento del bus PCI para hacer un diseño. Esto además de facilitar la tarea, permite la reutilización. Si ya existe un controlador de memorias con interfaz Wishbone, basta con diseñar la interconexión entre dicho controlador y el core IIE-PCI.

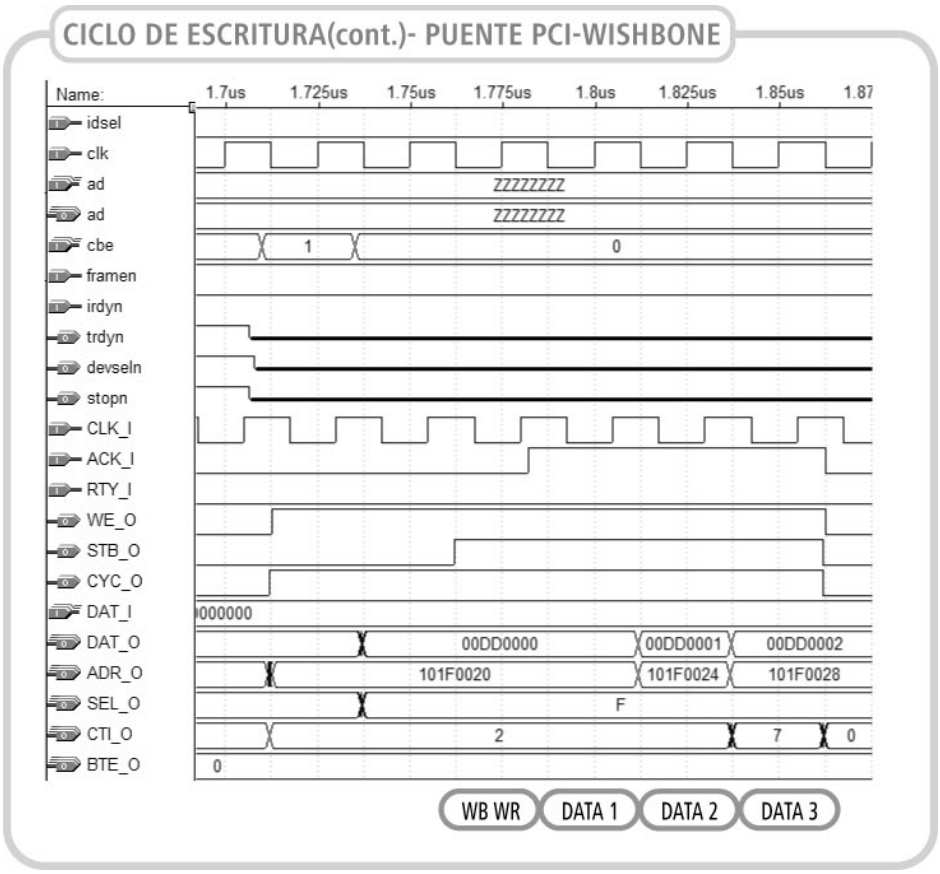


Funcionamiento

Cuando un master PCI realiza una escritura dentro de los rangos de direcciones asignados al core PCI, este acepta los datos y los almacena en un FIFO.



Luego al detectar que el FIFO de escritura no esta vacío, la interfaz Wishbone realiza tantos ciclos de escritura como sean necesarios hasta vaciar el FIFO.



Cuando un master PCI comienza una lectura dentro de los rangos de direcciones asignados al core PCI, este registra la dirección de comienzo y responde pidiendo un *retry*, esto hace que el master PCI aborte el ciclo y vuelva a intentar nuevamente la lectura cierto tiempo después.
Mientras esto sucede la interfaz Wishbone realiza ciclos de lectura, almacenando los datos leídos en el FIFO de lectura hasta llenarlo.
Cuando el master PCI reintenta el ciclo de lectura, la interfaz PCI contesta enviando los datos almacenados en el FIFO.

Detalles de la escritura al core PCI.

En caso de querer comenzar un ciclo de escritura PCI a una dirección asignada al core PCI, si la interfaz Wishbone no ha vaciado el FIFO de escritura, el core solicita un *retry* al master PCI.
Si ya se estaba realizando un ciclo de escritura y el FIFO de escritura se llena, se insertan tiempos de wait en el bus PCI (*trdyn='1'*) para dar tiempo al master Wishbone a sacar datos del FIFO. Si transcurrido 8 períodos de reloj el FIFO aún no se ha vaciado se pide un *disconect without data* lo que provoca que el ciclo se corte. Si aún restaban datos por escribir el master PCI automáticamente comenzará otro ciclo de

escritura.

Si la aplicación del lado Wishbone que recibe los datos escritos por el core esta mal implementada o no responde a las escrituras, el bus PCI podría quedar trancado pues se estarían intentando escrituras PCI continuamente y el FIFO nunca se encontraría vacio.

Detalles de la lectura al core PCI.

Cuando se realiza una lectura a una dirección asignada al core PCI, este chequea si hay datos en el FIFO de lectura y si corresponden a la dirección solicitada. En caso de coincidir las direcciones, se transfieren los datos desde el FIFO hasta que se vacíe o el ciclo sea finalizado por el master PCI.
En caso de que el FIFO de lectura se vacíe, se insertan tiempos de espera en el bus PCI (*trdyn='1'*) para dar tiempo a que el master Wishbone coloque más datos en el FIFO de lectura. Si transcurridos 8 períodos de reloj no hay nuevos datos en el FIFO de lectura se pide un *disconect without data* lo que provoca que el ciclo se corte. Si aún restaban datos por leer el master PCI automáticamente comenzará otro ciclo de lectura.
Si lo datos almacenados en la boca del FIFO de lectura no corresponden con la dirección solicitada, se registra la nueva dirección, se vacía el fifo y se vuelve al comienzo del proceso, solicitando un *retry* al master PCI.

Registro de traslación de direcciones

La dirección asignada a cada BAR del core PCI es asignada automáticamente en el momento de inicio del PC, y puede variar, dependiendo del PC que se esté utilizando y de cuantos dispositivos PCI estén colocados en el bus.

Para facilitar la tarea de diseño y decodificación de direcciones dentro de la aplicación en el bus Wishbone se implementaron registros de traslación de direcciones.
La función de los mismos es lograr que un acceso PCI a una dirección dentro del rango correspondiente a un BAR determinado se refleje como un acceso siempre a la misma dirección del bus Wishbone.

Estos registros pueden leerse y escribirse a través del BAR0 del core PCI en las direcciones indicadas por la siguiente tabla:

Registro de traslación	Ubicación en BAR0	Valor luego de Reset
para Bar 1	0x10	0x10000000
para Bar 2	0x14	0x20000000

Registro de traslación	Ubicación en BAR0	Valor luego de Reset
para Bar 3	0x18	0x30000000
para Bar 4	0x1C	0x40000000
para Bar 5	0x20	0x50000000
para Bar 6	0x24	0x60000000

Supongamos que al momento del booteo se le asigna a BAR0 el valor 0x80000000 y a BAR1 el 0x8F000000.
Si se quiere que los accesos al rango de direcciones correspondiente a BAR1 se mapeen a direcciones Wishbone que comiencen a partir de la dirección 0xE0000000, se debe escribir ese valor en la dirección 0x80000010.
Si se realiza una escritura a la dirección PCI 0x8F001000, se estará escribiendo en la dirección Wishbone 0xE0001000.

Independencia de relojes PCI y Wishbone

Los FIFOs entre la interfaz PCI y la interfaz Wishbone, no solo sirven como almacenamiento intermedio de los datos, sino que al contar los FIFOs con entradas de reloj independientes para la lectura y la escritura, permiten que las lógicas de las interfaces funcionen con diferentes velocidades de reloj.

Comandos del bus PCI

Durante la fase de direcciones de un ciclo, el bus *cben[3..0]* es utilizado para indicar el tipo de transacción a realizar. La siguiente tabla muestra un resumen de los tipos de transacciones, y cuales ciclos son soportados. En caso de que se le solicite al core un tipo de transacción no soportado, este la ignorará.

cben[3..0]	Tipo de ciclo	Acción
0000	Atención de interrupción	Ignorado
0001	Ciclo especial	Ignorado
0010	Lectura E/S	Aceptado
0011	Escritura E/S	Aceptado
0100	Reserved	Ignorado
0101	Reserved	Ignorado
0110	Lectura de memoria	Aceptado
0111	Escritura de memoria	Aceptado
1000	Reserved	Ignorado
1001	Reserved	Ignorado
1010	Lectura de configuración	Aceptado

cben[3..0]	Tipo de ciclo	Acción
1011	Escritura de configuración	Aceptado
1100	Lectura múltiple de memoria	Ignorado
1101	Ciclo de dirección doble	Ignorado
1110	Lectura de linea de memoria	Ignorado
1111	Lectura e invalidación	Ignorado

Uso del core PCITWBM

Interfaz y parámetros del core PCITWBM

El core PCI PCITWBM está escrito en VHDL. Su diseño esta divido en varios bloques, el de más alta jerarquía es *pcitwbm_top*.

Declaración del componente PCITWBM

```
COMPONENT pcitwbm_top
    GENERIC (
      vendor_id      : unsigned:= X"1172";
      device_id      : unsigned := X"ABBA";
      subsystem_id   : unsigned := X"10E9";
      subsystem_vid  : unsigned := X"10E9";
      NUMBER_OF_BARS : integer := 3;
      BAR_0_SIZE : integer := 8192;
      BAR_0_LOW_NIBBLE: integer := 0;
      BAR_1_SIZE : integer := 8192;
      BAR_1_LOW_NIBBLE: integer := 0;
      BAR_2_SIZE : integer := 8192;
      BAR_2_LOW_NIBBLE: integer := 0;
      BAR_3_SIZE : integer := 65536;
      BAR_3_LOW_NIBBLE: integer := 0;
      BAR_4_SIZE : integer := 65536;
      BAR_4_LOW_NIBBLE: integer := 0;
      BAR_5_SIZE : integer := 65536;
      BAR_5_LOW_NIBBLE: integer := 0;
      FIFO_NUMWORDS  : integer:= 14;
      LAT_TIMER_INITIAL_VALUE : integer := 7);
    PORT(
      -- PCI
      rstn     : IN  STD_LOGIC;
      clk      : IN  STD_LOGIC;
      irdyn    : IN  STD_LOGIC;
      idsel    : IN  STD_LOGIC;
      framen   : IN  STD_LOGIC;
      cbe      : IN  STD_LOGIC_VECTOR(3 downto 0);
      devseln  : OUT STD_LOGIC;
      stopn    : OUT STD_LOGIC;
      trdyn    : OUT STD_LOGIC;
      serrn    : OUT STD_LOGIC;
      perrn    : OUT STD_LOGIC;
      ad       : INOUT  STD_LOGIC_VECTOR(31 downto 0);
      par      : INOUT  STD_LOGIC;
      -- WB
      CLK_I    : IN  STD_LOGIC;
```

```
      DAT_I    : IN  STD_LOGIC_VECTOR(31 downto 0);
      DAT_O    : OUT STD_LOGIC_VECTOR(31 downto 0);
      ACK_I    : IN  STD_LOGIC;
      ADR_O    : OUT STD_LOGIC_VECTOR(31 downto 0);
      CYC_O    : OUT STD_LOGIC;
      RTY_I    : IN  STD_LOGIC;
      SEL_O    : OUT STD_LOGIC_VECTOR(3 downto 0);
      STB_O    : OUT STD_LOGIC;
      WE_O     : OUT STD_LOGIC;
      CTI_O    : OUT STD_LOGIC_VECTOR(2 downto 0);
      BTE_O    : OUT STD_LOGIC_VECTOR(1 downto 0)
      );
END COMPONENT;
```

Parámetros

VENDOR_ID
Tipo: unsigned
Valor por defecto: X"1172"
Descripción: Indica fabricante del dispositivo

DEVICE_ID
Tipo: unsigned
Valor: X"ABBA"
Descripción: Identifica tipo o modelo del dispositivo

SUBSYSTEM_ID y SUBSYSTEM_VID
Tipo:unsigned
Valor por defecto: X"10E9"
Descripción: Identifica la aplicación implementada

NUMBER_OF_BARS
Tipo: integer
Valor por defecto : 1
Descripción: número de BARs utilizados. Valores entre 1 y 6

BAR_i_SIZE
Tipo: integer
Valor por defecto : 8095
Descripción: tamaño del rango del i-ésimo BAR. El valor debe ser una potencia de 2.

BAR_i_LOW_NIBBLE
Tipo: integer

Valor por defecto : 0
Descripción: identifica el tipo de memoria asignado al i-ésimo BAR

FIFO_NUMWORDS
Tipo: integer
Valor por defecto : 14
Descripción: profundidad de los FIFO entre la interfaz Wishbone y PCI

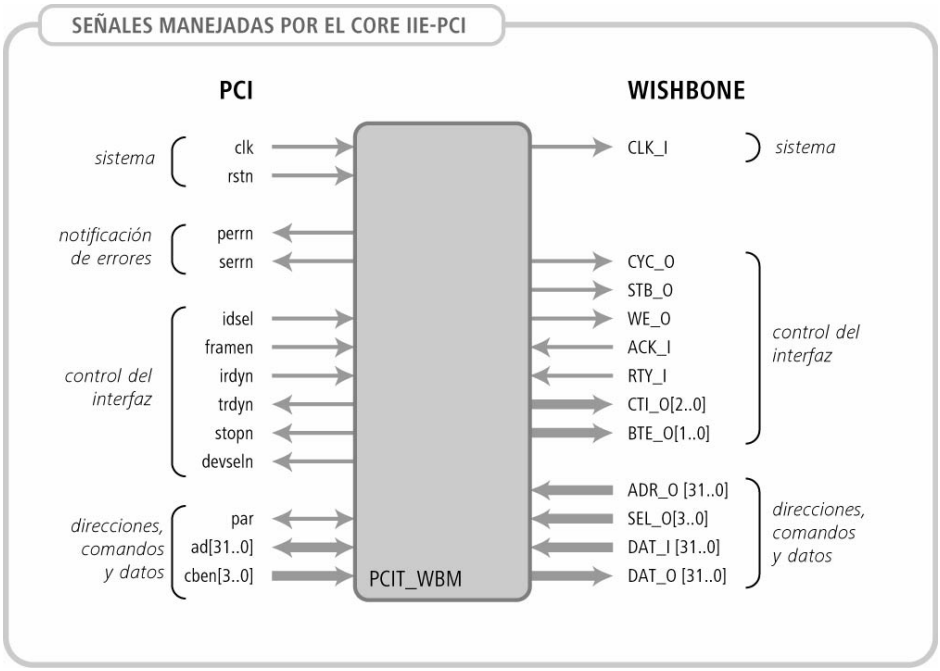
LAT_TIMER_INITIAL_VALUE
Tipo: integer
Valor por defecto : 7
Descripción: en caso de que el fifo de lectura se llene o el de lectura este vacio, el core espere LAT_TIMER_INITIAL_VALUE períodos de reloj e inicia un ciclo de desconexión.

Señales del bus PCI

Los señales del bus PCI pueden ser clasificadas según su tipo:

- **Entrada**
- **Salida**
- **Bidireccional**
- **STS** (tri-state sostenido): señales manejadas por un dispositivo a la vez. El dispositivo, antes de soltar la señal, debe mantenerla alta por un período de reloj. Otro dispositivo que desee manejar la señal debe esperar al menos un ciclo de reloj luego de que haya sido liberada por el dispositivo anterior.
- **Colector abierto**: señales que funcionan como un OR cableado entre los dispositivos que la manejan. Poseen un pull-up débil que las mantiene altas cuando no son manejadas por los dispositivos.

El estándar PCI define algunas señales como obligatorias y otras opcionales, a continuación se enumeran las señales del bus PCI son utilizadas por el core PCI.
Las señales activas en nivel bajo están terminadas con la letra *n*.



clk
Tipo: Entrada
Descripción: La entrada clk es el reloj del interfaz PCI. Excepto rstn (reset), todas las señales son síncronas, sus niveles son válidos solo durante el flanco de subida de reloj.

rstn
Tipo: Entrada
Nivel Activo: Bajo
Descripción: Reset. Es la señal de reset para el interfaz PCI y es asíncrona respecto al reloj. Cuando está activa, las señales de salida del bus PCI deben estar en tercer estado y las señales de colector abierto deben estar flotantes.

ad[31..0]
Tipo: Tri-estado
Descripción: Bus de direcciones/datos multiplexado en tiempo. Cada transacción consiste de una fase de dirección seguida de una o más fases de datos. Una transferencia de datos es llevada a cabo cuando *irdn* y *trdyn* están ambas activas.

cben[3..0]
Tipo: Tri-estado
Nivel Activo: Bajo

Descripción: Command/byte enable. Este bus está multiplexado en tiempo. Durante la fase de direcciones este bus indica el comando PCI deseado definiendo el tipo de transacción a realizar; durante la fase de datos, este bus indica que byte o bytes en el bus *ad* son válidos.

par
Tipo: Tri-estado
Descripción: Paridad. Es el resultado de calcular la paridad de los bits bus *ad* y del bus *cben*. La paridad de los datos transferidos en una fase de datos es presentada en el flanco de reloj siguiente.

idsel
Tipo: Entrada
Nivel Activo: Alto
Descripción: La señal *idsel* le indica al dispositivo cuando se está realizado un ciclo de acceso a sus registros de configuración.

framen
Tipo: STS
Nivel Activo: Bajo
Descripción: Frame. La señal *framen* es manejada por el dispositivo master del bus en dicho instante, e indica el comienzo y la duración de una operación en el bus. Cuando *framen* está activa, la dirección y el comando están presentes en los buses *ad* y *cben*. La señal *framen* es mantenida activa durante la transferencia de datos y se desactiva para indicar el fin de un ciclo.

irdyn
Tipo: STS
Nivel Activo: Bajo
Descripción: Initiator ready. La señal *irdyn* es manejada por el dispositivo master del bus en dicho instante, e indica que éste puede completar la transacción de datos que se está realizando. En una transacción de escritura, *irdyn* indica que el bus *ad* tiene datos válidos. En una transacción de lectura, *irdny* indica que el dispositivo maestro está listo para aceptar los datos presentes en el bus *ad*.

devseln
Tipo: STS
Nivel Activo: Bajo
Descripción: Device select. El dispositivo target activa *devseln* cuando ha

decodificado su dirección y solicita la transacción.

trdyn
Tipo: STS
Nivel Activo: Bajo
Descripción: Target Ready. El dispositivo target activa *trdyn* para indicar que puede completar la transferencia de datos que se está realizando. En una operación de lectura, *trdyn* indica que el target está colocando datos válidos en el bus *ad*. En una operación de escritura, *trdyn* indica que el dispositivo target está listo para aceptar datos.

stopn
Tipo: STS
Nivel Activo: Bajo
Descripción: Stop. El dispositivo target activa *stopn* para indicar al dispositivo master que debe terminar la transacción en curso. La señal *stopn* se usa en conjunto con *trdyn* y *devseln* para indicar el tipo de terminación de transacción iniciada por el target.

perrn
Tipo: STS
Nivel Activo: Bajo
Descripción: Parity Error. La señal *perrn* indica que hubo un error de paridad en los datos. La señal *perrn* es activada un ciclo de reloj después de la señal *par*, o lo que es lo mismo dos ciclos de reloj luego de haber ocurrido un error de paridad en el bus.

serrn
Tipo: Colector abierto
Nivel Activo: Bajo
Descripción: System Error. La señal *serrn* indica un error del sistema y error de paridad en la dirección. Los dispositivos pci deben activar la señal *serrn* si detectan un error de paridad durante una fase de transferencia de direcciones.

Señales del interfaz Wishbone

A continuación se describen las señales del interfaz Whishbone utilizadas por el core PCI. Todas las señales son activas por nivel alto y la terminación *_I* y *_O* indica si son entradas o salidas al core respectivamente.

CLK_I
Tipo: Entrada

Descripción: Reloj. La señal de reloj coordina todas las actividades para la lógica dentro de un dispositivo Wishbone. Todas las señales de salida Wishbone son registradas en el flanco de subida de *CLK_I*. Todas las entradas Wishbone deben estar estables antes del flanco de subida de *CLK_I*.

DAT_I[31..0]
Tipo: Entrada
Descripción: El bus *DAT_I* es la entrada de datos. Su ancho de palabra es de 32 bits.

DAT_O[31..0]
Tipo: Salida
Descripción: El bus *DAT_O* es la salida datos. Su ancho de palabra es de 32 bits.

ACK_I
Tipo: Entrada
Descripción: Acknowledge Input. La señal *ACK_I* le indica al dispositivo master que se ha realizado una transferencia en forma exitosa.

ADR_O[31..0]
Tipo: Salida
Descripción: El bus *ADR_O* es manejado por el master y e indica la dirección de los datos que deben leerse o escribirse.

CYC_O
Tipo: Salida
Descripción: Cycle Output. La señal *CYC_O* indica que un ciclo válido está en progreso. La señal se mantiene activa por la duración de todo el ciclo. Por ejemplo, en una transferencia en bloque, la señal *CYC_O* se activa en la primer transferencia y se mantiene activa hasta la última.

RTY_I
Tipo: Entrada
Descripción: Retry Input. Indica que el interfaz no esta listo para aceptar o enviar datos, y que el ciclo debe ser reintentado más tarde. En este caso, el core PCI interrumpe el ciclo Wishbone y vuelve a comenzar un nuevo ciclo para terminar la transferencia de datos.

SEL_O[3..0]
Tipo: Salida

Descripción: Select Output. El bus *SEL_O* indica qué bytes del bus *DAT_O* contienen datos válidos, o en que bytes del bus *DAT_I* se esperan datos válidos. Estas señales se corresponden con las señales del bus PCI *cben*.

STB_O

Tipo: Salida

Descripción: Strobe Output. La señal *STB_O* indica un la presencia de un dato válido en *DAT_O* en un ciclo de escritura o que esta listo para recibir datos en un ciclo de lectura. El dispositivo esclavo Wishbone debe responder con alguna de las señales *ACK_I* o *RTY_I* frente a cada activación de la señal *STB_O*.

WE_O

Tipo: Salida

Descripción: Write Enable Output. La señal *WE_O* indica si el ciclo de bus corresponde a una lectura o a una escritura. La señal se activa durante los ciclos de escritura y se desactiva en los ciclos de lectura.

CTI_O[2..0]

Tipo: Salida

Nivel Activo: Alto

Descripción: Cycle Type Identifier. La señal CTI_O provee información adicional sobre el ciclo que se está realizando. Es parte de la especificación de Wishbone avanzado. El master le envía esta información al esclavo y este debe usarla para preparar la respuesta a dar en el ciclo siguiente.

La siguiente tabla muestra los posibles tipos de ciclos:

CTI_O[2..0]	Descripción
000	Ciclo clásico
001	Ciclo burst de dirección constante
010	Ciclo burst con incremento de direcciones
111	Fin del ciclo burst

BTE_O[1..0]

Tipo: Salida

Descripción: Burst Type Extension. La señal *BTE_O* provee información adicional sobre el ciclo que se está realizando. Esta información es relevante únicamente para ciclos burst con incremento de direcciones.

En la implementación del core PCI se incluye, ya que es mandatoria en la especificación Wishbone si se desea utilizar CTI_O, pero su valor es siempre 00. Esto significa que si se utilizan ciclos burst con incremento de direcciones, su incremento es lineal.

Jerarquía de los archivos VHDL

El siguiente esquema muestra la jerarquía de los archivos .vhd que componen el core PCI.

El orden de síntesis debe de ser del de menor al de mayor jerarquía.

Recursos utilizados

A modo de comparación, se sintetizó un core con

- 3 BARs
- tamaño BAR 0 (bits): 8095
- tamaño BAR 1 (bits): 1048576
- tamaño BAR 2 (bits): 4194304
- profundidad de FIFO: 7

Para el pasaje de VHDL a EDIF se utilizó SynplifyPRO 7.0.1 y para la síntesis final Max+PlusII 10.2.

Lógica	Optimization	No.BAR	Bits(%)	LCs	Max. reloj PCI	Max. reloj WB
EP1K100-1	área	3	1216(2%)	1137(22%)	50.76MHz	50.76MHz
	velocidad	3	1216(2%)	1276(25%)	56.81MHz	62.89MHz
EP1K100-2	área	3	1216(2%)	1137(22%)	38.91MHz	37.73MHz
	velocidad	5	1216(2%)	1276(25%)	42.73MHz	46.29MHz

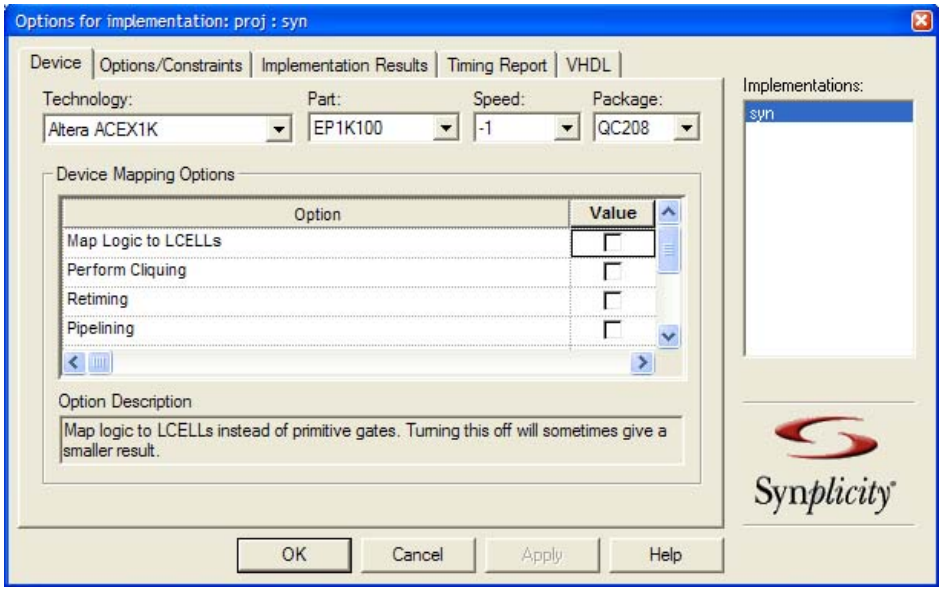
Los valores deben ser tomados como una referencia para comparar la implementación del core en cada FPGA, ya que el tamaño y velocidad final dependen fuertemente de la aplicación con la cual se sinteticen.

Herramientas recomendadas

VHDL a EDIF

El software sintetizador Max+PlusII soporta un sub-conjunto de VHDL muy limitado y los mensajes de error no son claros, lo que hace el trabajo muy difícil.
Ante esta situación se optó por utilizar algún software de síntesis que a partir de archivos VHDL generara archivos EDIF.
Los archivos EDIF son un estándar de la industria para el pasaje de diseños entre programas.
El software utilizado fue SynlifyPRO 7.0.1 de Synplicity (http://www.synplicity.com)

Durante las pruebas se consiguieron mejores resultados si no se seleccionaba el casillero Map Logic to LCELLs



Esta opción genera un EDIF con lógica mapeada a la estructura del FPGA en el que se va a sintetizar el diseño. Al no seleccionarse, solo se genera un archivo de nodos optimizado y el Max+PlusII se encarga finalmente de mapear la lógica a los recursos del FPGA de mejor forma.

Sintetizador

Una vez generado el EDIF se crea un proyecto con el archivo, automáticamente el

Max+PlusII identifica que el archivo fue creado con SynlifyPRO.

Las opciones de síntesis con las que se lograron diseños capaces de utilizar mayores velocidades de reloj fueron:

- En menú: *Global Project Logic Synthesis*
  - Optimización: 10 (speed)
  - *One-Hot State Machine Encoding* seleccionado
  - En sub-menú: *Define Synthesis Style*
    - Minimización: *Partial*
    - *Carry Chain*: Manual, Max. 8
    - *Cascade Chain*: Manual, Max. 8
    - Opciones Avanzadas: Todas las opciones seleccionadas

Simulador

Para las simulaciones se utilizó el simulador incluido en el software Max+PlusII.

Asignación de pines para placa IIE-PCI

En caso de utilizar el core en la placa PCI IIE-PCI, puede incluirse la siguiente asignación de pines en el archivo ACF del proyecto de Max+PlusII.

```
CHIP NOMBRE_DE_PROYECTO
BEGIN
   |wen      :   OUTPUT_PIN = 180;
   |reqn     :   BIDIR_PIN  = 41;
   |intan    :   BIDIR_PIN  = 31;
   |intbn    :   BIDIR_PIN  = 36;
   |intcn    :   BIDIR_PIN  = 37;
   |intdn    :   BIDIR_PIN  = 38;
   |cben0    :   BIDIR_PIN  = 100;
   |cben1    :   BIDIR_PIN  = 89;
   |cben2    :   BIDIR_PIN  = 70;
   |cben3    :   BIDIR_PIN  = 57;
   |devseln  :   BIDIR_PIN  = 75;
   |stopn    :   BIDIR_PIN  = 83;
   |trdyn    :   BIDIR_PIN  = 74;
   |serrn    :   BIDIR_PIN  = 87;
   |perrn    :   BIDIR_PIN  = 86;
   |irdyn    :   BIDIR_PIN  = 73;
   |framen   :   BIDIR_PIN  = 71;
   |IO_EXP30 :   BIDIR_PIN  = 176;
   |DQ0      :   BIDIR_PIN  = 196;
   |DQ1      :   BIDIR_PIN  = 195;
   |DQ2      :   BIDIR_PIN  = 193;
   |DQ3      :   BIDIR_PIN  = 192;
   |DQ4      :   BIDIR_PIN  = 191;
   |DQ5      :   BIDIR_PIN  = 190;
   |DQ6      :   BIDIR_PIN  = 189;
   |DQ7      :   BIDIR_PIN  = 187;
   |DQ8      :   BIDIR_PIN  = 164;
   |DQ9      :   BIDIR_PIN  = 166;
   |DQ10     :   BIDIR_PIN  = 167;
   |DQ11     :   BIDIR_PIN  = 168;
   |DQ12     :   BIDIR_PIN  = 169;
   |DQ13     :   BIDIR_PIN  = 170;
   |DQ14     :   BIDIR_PIN  = 172;
   |DQ15     :   BIDIR_PIN  = 173;
   |DQ16     :   BIDIR_PIN  = 126;
   |DQ17     :   BIDIR_PIN  = 125;
   |DQ18     :   BIDIR_PIN  = 122;
   |DQ19     :   BIDIR_PIN  = 121;
   |DQ20     :   BIDIR_PIN  = 120;
   |DQ21     :   BIDIR_PIN  = 119;
```

```
|DQ22     :   BIDIR_PIN  = 116;
|DQ23     :   BIDIR_PIN  = 115;
|DQ24     :   BIDIR_PIN  = 136;
|DQ25     :   BIDIR_PIN  = 139;
|DQ26     :   BIDIR_PIN  = 140;
|DQ27     :   BIDIR_PIN  = 141;
|DQ28     :   BIDIR_PIN  = 142;
|DQ29     :   BIDIR_PIN  = 143;
|DQ30     :   BIDIR_PIN  = 144;
|DQ31     :   BIDIR_PIN  = 147;
|A0       :   OUTPUT_PIN = 132;
|A1       :   OUTPUT_PIN = 131;
|A2       :   OUTPUT_PIN = 128;
|A3       :   OUTPUT_PIN = 149;
|A4       :   OUTPUT_PIN = 150;
|A5       :   OUTPUT_PIN = 157;
|A6       :   OUTPUT_PIN = 158;
|A7       :   OUTPUT_PIN = 159;
|A8       :   OUTPUT_PIN = 160;
|A9       :   OUTPUT_PIN = 161;
|A10      :   OUTPUT_PIN = 133;
|A11      :   OUTPUT_PIN = 174;
|BA0      :   OUTPUT_PIN = 135;
|BA1      :   OUTPUT_PIN = 134;
|DQM0     :   OUTPUT_PIN = 186;
|DQM1     :   OUTPUT_PIN = 163;
|DQM2     :   OUTPUT_PIN = 127;
|DQM3     :   OUTPUT_PIN = 148;
|RASn     :   OUTPUT_PIN = 177;
|CASn     :   OUTPUT_PIN = 179;
|ledazul  :   OUTPUT_PIN = 62;
|ledverde :   OUTPUT_PIN = 19;
|SW_DIP1  :   INPUT_PIN  = 182;
|SW_DIP2  :   INPUT_PIN  = 80;
|SW_DIP3  :   INPUT_PIN  = 184;
|IO_EXP29 :   BIDIR_PIN  = 197;
|IO_EXP28 :   BIDIR_PIN  = 198;
|IO_EXP27 :   BIDIR_PIN  = 199;
|IO_EXP26 :   BIDIR_PIN  = 200;
|IO_EXP25 :   BIDIR_PIN  = 202;
|IO_EXP24 :   BIDIR_PIN  = 203;
|IO_EXP23 :   BIDIR_PIN  = 204;
|IO_EXP22 :   BIDIR_PIN  = 205;
|IO_EXP21 :   BIDIR_PIN  = 206;
|IO_EXP20 :   BIDIR_PIN  = 207;
|IO_EXP19 :   BIDIR_PIN  = 208;
|IO_EXP18 :   BIDIR_PIN  = 7;
|IO_EXP17 :   BIDIR_PIN  = 8;
|IO_EXP16 :   BIDIR_PIN  = 9;
|IO_EXP15 :   BIDIR_PIN  = 10;
|IO_EXP14 :   BIDIR_PIN  = 11;
|IO_EXP13 :   BIDIR_PIN  = 12;
```

```
  |IO_EXP12 :   BIDIR_PIN  = 13;
  |IO_EXP11 :   BIDIR_PIN  = 14;
  |IO_EXP10 :   BIDIR_PIN  = 15;
  |IO_EXP9  :   BIDIR_PIN  = 16;
  |IO_EXP8  :   BIDIR_PIN  = 17;
  |IO_EXP7  :   BIDIR_PIN  = 18;
  |IO_EXP6  :   BIDIR_PIN  = 24;
  |IO_EXP5  :   BIDIR_PIN  = 25;
  |IO_EXP4  :   BIDIR_PIN  = 26;
  |IO_EXP3  :   BIDIR_PIN  = 27;
  |IO_EXP2  :   BIDIR_PIN  = 28;
  |IO_EXP1  :   BIDIR_PIN  = 29;
  |IO_EXP0  :   BIDIR_PIN  = 30;
  |lockn    :   INPUT_PIN  = 85;
  |ad0      :   BIDIR_PIN  = 114;
  |ad1      :   BIDIR_PIN  = 113;
  |ad2      :   BIDIR_PIN  = 112;
  |ad3      :   BIDIR_PIN  = 111;
  |ad4      :   BIDIR_PIN  = 104;
  |ad5      :   BIDIR_PIN  = 103;
  |ad6      :   BIDIR_PIN  = 102;
  |ad7      :   BIDIR_PIN  = 101;
  |ad8      :   BIDIR_PIN  = 99;
  |ad9      :   BIDIR_PIN  = 97;
  |ad10     :   BIDIR_PIN  = 96;
  |ad11     :   BIDIR_PIN  = 95;
  |ad12     :   BIDIR_PIN  = 94;
  |ad13     :   BIDIR_PIN  = 93;
  |ad14     :   BIDIR_PIN  = 92;
  |ad16     :   BIDIR_PIN  = 69;
  |ad17     :   BIDIR_PIN  = 68;
  |ad18     :   BIDIR_PIN  = 67;
  |ad19     :   BIDIR_PIN  = 65;
  |ad20     :   BIDIR_PIN  = 64;
  |ad21     :   BIDIR_PIN  = 63;
  |ad22     :   BIDIR_PIN  = 61;
  |ad23     :   BIDIR_PIN  = 60;
  |ad24     :   BIDIR_PIN  = 56;
  |ad25     :   BIDIR_PIN  = 55;
  |ad26     :   BIDIR_PIN  = 54;
  |ad27     :   BIDIR_PIN  = 53;
  |ad28     :   BIDIR_PIN  = 47;
  |ad29     :   BIDIR_PIN  = 46;
  |ad30     :   BIDIR_PIN  = 45;
  |ad31     :   BIDIR_PIN  = 44;
  |idsel    :   INPUT_PIN  = 58;
  |gntn     :   INPUT_PIN  = 40;
  |rstn     :   INPUT_PIN  = 39;
  |par      :   BIDIR_PIN  = 88;
  |cke      :   OUTPUT_PIN = 162;
  |csn      :   OUTPUT_PIN = 175;
  |clk      :   INPUT_PIN  = 183;
```

```
   |ad15     :   BIDIR_PIN  = 90;
   |SW_DIP4  :   INPUT_PIN  = 78;
   |gclk1    :   INPUT_PIN  = 79;
END;
```