UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

VHSIC Hardware Description Language

Técnicas de Modelado para Síntesis

Pedro P. Carballo
carballo@iuma.ulpgc.es

Jorge Monagas
jmonagas@iuma.ulpgc.es

Contenido

- Parte I: Recomendaciones de Codificación VHDL para la reutilización del diseño a nivel RTL.
- Parte II: Modelado RTL de componentes para Xilinx en VHDL.

Referencias:
1. Reuse methodology manual for system-on-a-chip designs. *by Michael Keating, Pierre Bricaud. Kluwer Academic Publishers, Boston (2002) - (3rd ed.) ISBN: 1402071418*
2. Xilinx Synthesis Tecnonology User Manual. Xilinx (www.xilinx.com).

Parte I:
Recomendaciones de Codificación VHDL para la reutilización del diseño a nivel RTL

Recomendaciones en el uso de código RTL

1. Reglas Generales
2. Portabilidad del Código
3. Señales de reloj y reset
4. Recomendaciones para Síntesis

1. Reglas Generales

- *Genéricas*
  - *Regla:* Elegir un nombre adecuado para el diseño que se use consistentemente.
  - *Recomendación:* Usar minúsculas para todos los nombres de señales, variables y puertos.
  - *Recomendación:* Usar mayúsculas para nombres de constantes y tipos definidos por el usuario.
  - *Recomendación:* Emplear nombres significativos para señales, puertos, funciones y parámetros.
  - *Recomendación:* Utilizar nombres descriptivos cortos a los parámetros del diseño, para evitar que las herramientas de síntesis generen nombres de unidades de diseño excesivamente largos.

1. Reglas generales

- *Genéricas*
  - *Recomendación:*
    - La señal de reloj se denominará *clk*.
    - Si se utiliza más de una señal de reloj en el diseño es necesario que todas comiencen con el prefijo *clk*.
    - Usar el mismo nombre para todas las señales de reloj que son controladas desde la misma fuente.
  - *Recomendación:*
    - Utilizar la misma nomenclatura para todas las señales activas a nivel bajo del diseño: un '_' seguido por una letra en minúscula.
    - Para estandarización, se recomienda emplear _n para indicar todas las señales activas a nivel bajo.
    - La señal de reset se denominará *rst*. Si esta señal es activa a nivel bajo: *rst_n*.

1. Reglas generales

- *Genéricas*
  - *Regla:*
    - Emplear un orden consistente de bits para describir buses. En VHDL: (y downto x) o (x to y).
    - Se recomienda usar en VHDL para señales multibits: (y downto x) para establecer un estándar entre múltiples diseños y equipos de diseño.
  - *Recomendación:*
    - Siempre que sea posible, usar el mismo nombre o nombres similares para puertos y señales que estén conectados.

1. Reglas Generales

- Convenios en la denominación para soporte VITAL
  - VITAL (VHDL Initative Towars ASIC Libraries):
    - Estándar para modelado a nivel de puertas de librerías VHDL (IEEE 1076.4). Especifica las restricciones en los convenios de elección de nombres (y otras características) de las declaraciones de los puertos del nivel superior de un elemento de la librería.
  - *Regla:*
    - Un puerto no estará en modo *linkage*.
  - *Regla:*
    - El puerto, en un declaración de un puerto de una entidad no puede ser un puerto protegido. Además, la declaración no puede imponer una restricción en el rango del puerto, ni puede alterar la resolución del puerto desde la definición dada en el paquete lógico estándar.

1. Reglas Generales

- Cabeceras código fuente
  - *Regla:*
    - Incluir una cabecera en la parte superior de cada fichero fuente, incluyendo los scripts de síntesis.
    - La cabecera debe contener: nombre del fichero, autor, descripción de la función y lista de las características claves del módulo, fecha de creación, histórico de modificaciones incluyendo día, nombre de la persona que realiza la modificación y descripción del cambio.
    - Ejemplo:

```
-- File : file.vhd
-- Author :Nombre <email>
-- Date : DD/MM/YYYY
-- Version : 0.1
-- Abstract : Este fichero contiene la entidad y arquitectura del módulo
--   X. A partir de las entradas Y Z
--   y E_D se obtendrán las …
--
-- Modification History:
-- Date By Version Change Description
--=================================================
--
--=================================================
```

Técnicas de Modelado para Síntesis 9

1. Reglas Generales

- *Uso de Comentarios*
  - *Regla:*
    - Usar comentarios adecuados para explicar todos los procesos, funciones y declaraciones de tipos y subtipos.
  - *Recomendación:*
    - Usar comentarios para describir puertos, señales y variables, o grupos de señales o variables, evitando comentarios obvios de funcionalidad.
- *Mantener sentencias en líneas separadas*
  - *Regla:*
    - Usar líneas separadas para cada declaración de HDL.
- *Longitud de las líneas*
  - *Recomendación:*
    - Mantener la longitud de las líneas en 72 caracteres o menos, utilizando la indentación en las próximas líneas para indicar que es una continuación de la línea anterior.

Técnicas de Modelado para Síntesis 10

1. Reglas Generales

- *Indentación*
  - *Regla:*
    - Usar la indentación para mejorar la lectura de líneas de código continuas y bucles anidados.
    - Se recomienda emplear dos espacios para la indentación puesto que valores mayores limitan la longitud de la línea cuando existen varios niveles de bucles, evitando el empleo de tabuladores.
- *No usar palabras reservadas del HDL*
  - *Regla:*
    - No usar palabras reservadas de VHDL o Verilog para nombres de cualquier elemento de los ficheros fuente porque los diseño de macros deben ser traducibles de VHDL a Verilog y viceversa.

1. Reglas Generales

- *Organización de los puertos*
  - *Regla:*
    - Declarar los puertos siguiendo un orden lógico y manteniéndolo consistente en todo el diseño, con un puerto por línea, seguido de un comentario.
  - *Recomendación:*
    - Declarar los puertos en el siguiente orden:
      - Entradas: reloj/es, reset/s, enable/s, otras señales de control, datos y líneas de direcciones.
      - Salidas: Reloj/es, reset/s, enable/s, otras señales de control y datos.
  - *Recomendación:*
    - Emplear comentarios para describir grupos de puertos.
  - *Recomendación:*
    - Dejar una línea en blanco entre los puertos de entrada y salida para mejorar la lectura.

1. Reglas Generales

- *Secciones de entidad, arquitectura y configuración en VHDL*
  - *Recomendación:*
    - Localizar las secciones de entidad, arquitectura y configuración del diseño VHDL en el mismo fichero, para que sea más fácil entender y mantener un diseño.
    - Si se incluye en un fichero fuente configuraciones de subdiseños con declaraciones de la entidad y arquitectura, se debe comentar para síntesis, empleando en el código VHDL los pseudo-comentarios pragma *translate_off* y pragma *translate_on* o aquellos que sean necesarios por la herramienta soportada.
- *Uso de funciones*
  - *Recomendación:*
    - Siempre que sea posible emplea funciones con sus correspondientes comentarios.
    - Generalizar funciones las hace reutilizables.

Técnicas de Modelado para Síntesis 13

1. Reglas Generales

- *Uso de bucles y arrays*
  - *Recomendación:*
    - Usar bucles y arrays para mejorar la lectura del código fuente.
  - *Recomendación:*
    - Los arrays son significativamente más rápidos para simular que los bucles *for*, por lo que siempre que se pueda es conveniente usar operaciones con vectores en arrays en vez de bucles *for*.
- *Uso de etiquetas significativas*
  - *Regla:*
    - Etiquetar cada proceso con un nombre significativo, resulta muy útil en el momento de la depuración.
  - *Regla:*
    - No duplicar nombres de señales con variables o con entidades.

Técnicas de Modelado para Síntesis 14

2. Portabilidad del Código

- *Usar sólo tipos estándar IEEE*
  - *Regla:*
    - Usar sólo tipos estándar IEEE. Se pueden crear tipos y subtipos adicionales pero todos basados en tipos estándar IEEE.
  - *Recomendación:*
    - Usar std_logic en vez de *std_ulogic*.
    - Igualmente *std_logic_vector* en vez de *std_ulogic_vector*.
    - Los tipos *std_logic* y *std_logic_vector* proporcionan las funciones de resolución requeridas para buses triestado, lo que NO ocurre con *std_ulogic* y *std_ulogic_vector*
  - *Recomendación:*
    - Hay que ser conservador en el número de subtipos que se crean. Usar demasiados subtipos hace que sea difícil de entender el código.

2. Portabilidad del Código

- *Usar sólo tipos estándar IEEE (cont)*
  - *Recomendación:*
    - No usar los tipos *bit* o *bit_vector*, puesto que muchos simuladores no aportan funciones aritméticas construidas para estos tipos.
  - *Recomendación:*
    - No usar valores numéricos codificados de forma fija sino emplear constantes que definan dichos valores en el diseño.
    - Como una excepción, se puede usar los valores 0 y 1 (pero no en combinación, como 1001).

2. Portabilidad del Código

- *Paquetes*
  - *Recomendación:*
    - Recoger todas las definiciones de funciones y valores de los parámetros para un diseño en un único fichero separado ("*package*") con el nombre DesignName_package.vhd.
- *Scripts de síntesis empotrados (dc_shell, …)*
  - No es recomendable el uso de comandos de síntesis empotrados directamente en el código fuente, puesto que si el diseño va a ser reutilizado en una nueva aplicación, los objetivos de síntesis pueden de ser diferentes.
  - Si el código fuente es reutilizado con nuevas versiones de la herramienta de síntesis (*Design Compiler por ejemplo),* el comando puede que sea obsoleto.
  - Hay diversas excepciones a esta regla. En particular, las directivas de síntesis *synthesis on* y *synthesis off* deben estar empotradas en un lugar adecuado del código.

Técnicas de Modelado para Síntesis 17

2. Portabilidad del Código

- *Usar librerías independientes de la tecnología*
  - *Recomendación:*
    - Emplear librerías independientes de la tecnología (*DesignWare* o equivalentes).
    - Estas librerías aportan arquitecturas para los componentes aritméticos básicos (sumadores, multiplicadores, comparadores, incrementadores, decrementadores, etc), así como otros más complejos (senos, cosenos, divisores, raíz cuadrada, desplazadores aritméticos, etc.).
    - También incluye componentes secuenciales, completamente portables, que pueden ahorrar un considerable tiempo de diseño (FIFOs, controladores de FIFO, ECC, CRC, componentes JTAG, etc.).

Técnicas de Modelado para Síntesis 18

3. Señales de reloj y reset

- Estructura del reloj
  - Una estructura simple de reloj es más fácil de entender, analizar y mantener, además de producir mejores resultados de síntesis.
  - La mejor estructura de reloj en un sencillo reloj global y *flip-flops* activos por flanco de subida.
  - *Evitar mezclar flancos de reloj (subida y bajada)*
- *Recomendación:*
  - Evitar usar *flip-flops* activos por flanco de subida y por flanco de bajada en el mismo diseño.
    - Mezclar flancos de reloj puede ser necesario en algunos diseños, donde por ejemplo, sea útil capturar datos en ambos flancos de reloj.
    - Crea diversos problemas, ya que el ciclo de trabajo del reloj se convierte en un parámetro crítico en el análisis temporal,
    - La mayoría de las metodologías de test basadas en scan requieren un manejo separado de *flip-flops* activos por flanco de subida y de bajada.

Técnicas de Modelado para Síntesis 19

3. Señales de reloj y reset

- *Evitar mezclar flancos de reloj (II)*
  - *Regla:*
    - Si se debe emplear un *flip-flop* activo por flanco positivo y negativo en el diseño, hay que estar seguro de modelar el peor caso del ciclo de trabajo del reloj con precisión en los análisis de síntesis y temporización e indicar en la documentación de usuario el ciclo de trabajo asumido.
  - *Recomendación:*
    - Si se debe usar un gran número de *flip-flops* activos por flanco positivo y negativo en el diseño, puede ser útil separarlos en diferentes módulos.
      - Facilita la identificación de los flip-flops activos por flanco negativo y la partición en diferentes cadenas de scan

Técnicas de Modelado para Síntesis 20

3. Señales de reloj y reset

- *Evitar buffers de reloj*
  - *Recomendación:*
    - Evitar el manejo de buffers de reloj en el código RTL.
      - Normalmente, los buffers de reloj se insertan después de la síntesis como parte del diseño físico.
      - En el código RTL sintetizable, las nets de reloj se consideran normalmente nets ideales, sin retardos.
      - Durante el diseño físico, la herramienta de inserción de árboles de reloj inserta la estructura adecuada para crearlo tan próximo a un ideal como sea posible, redes de distribución de reloj balanceado.

Técnicas de Modelado para Síntesis 21

3. Señales de reloj y reset

- *Evitar el empleo de relojes atacados por puertas*
  - *Recomendación:*
    - Evitar el empleo de relojes atacados por puertas en el código RTL.
      - Estos circuitos tienden a ser específicos de la tecnología y dependiente del retardo.
      - Un esquema temporal inadecuado de un reloj atacado por puerta puede generar un falso reloj o *glitch*, causando que un flip-flop tome un dato incorrecto.
      - El *skew* de diferentes relojes locales pueden causar violaciones en el tiempo de retención (*hold*).
      - Se pueden crear problemas de testeabilidad debido a que esta lógica no puede formar parte de una cadena *scan*.

Técnicas de Modelado para Síntesis 22

3. Señales de reloj y reset

- *Evitar los relojes generados internamente*
  - *Recomendación:*
    - Evitar usar los relojes que han sido generados internamente porque esta lógica no puede formar parte de una cadena *scan* y hacen más difícil (normalmente no es posible) restringir el diseño para la síntesis.
  - *Recomendación:*
    - Si se necesita un señal de reloj controlada por puertas o un reloj o reset generado internamente, es mejor mantener el circuito de generación del reloj y/o reset como un módulo separado en el más alto nivel de jerarquía del diseño.
    - Dividir el diseño de tal forma que toda la lógica de un único módulo usa un reloj y un reset.

3. Señales de reloj y reset

- *Evitar resets generados internamente*
  - *Recomendación:*
    - En la medida que sea posible evitar resets generados internamente.
    - Generalmente, todos los registros deberían ser reseteados al mismo tiempo. Esta restricción hace que el diseño sea mucho más sencillo y más fácil de controlar.
  - *Recomendación:*
    - Si se requiere un reset condicional, es necesario crear una señal separada para la señal de reset, y aislar la lógica de reset condicional en un módulo separado.
      - Esta recomendación permite un código más legible y mejora los resultados de síntesis.

4. Recomendaciones para Síntesis

- *Obtención de registros*
  - *Recomendación:*
    - Los principales elementos que se utilizan en la lógica secuencial son los registros (*flip-flops*).
    - Para mantener la consistencia y asegurar una correcta síntesis, es conveniente utilizar procesos que infieren registros independientes de la tecnología .

```
-- proceso con reset síncrono
PROC: process(clk)
 begin
  if (clk'event and clk='1') then
   if rst_n='0' then
     ....
   else
     ...
   end if;
  end if;
 end process PROC;
```

```
-- proceso con reset asíncrono
PROC: process(clk,rst_n)
 begin
  if (rst_n='0') then
    ....
  elsif (clk'event and clk='1') then
    ....
  end if;
 end if;
end process PROC;
```

Técnicas de Modelado para Síntesis 25

4. Recomendaciones para Síntesis

- *Evitar latches*
  - *Regla:*
    - Evitar usar *latches* en el diseño.
  - *Recomendación:*
    - Se puede evitar la inferencia de *latches* usando cualquiera de las siguientes técnicas de código:
      - Asignar valores por defecto al comienzo de un proceso.
      - Asignar salidas para todas las condiciones de entrada.
      - Usar else (en vez de elsif) para la bifurcación final de prioridad.
- *Evitar realimentaciones combinacionales*
  - *Recomendación:*
    - Evitar la realimentación combinacional en los bucles de los procesos combinacionales.

Técnicas de Modelado para Síntesis 26

4. Recomendaciones para Síntesis

- *Especificar las listas sensibles completas para los procesos*
  - *Regla:*
    - Incluir una lista sensible completa en cada uno de los procesos (VHDL).
      - Si no se usa una lista sensible completa, el comportamiento del diseño a nivel RTL y lógico puede diferir.
    - *Bloques combinacionales:*
      - Para bloques combinacionales (bloques que no contienen registros ni *latches*), la lista de sensibilidad debe incluir cada señal que es leída por el proceso. En general, esto significa, cada señal que aparece en el lado derecho de una asignación (<=) o en una expresión condicional.
    - *Bloques secuenciales:*
      - Para bloques secuenciales, la lista sensible debe incluir la señal de reloj que será leída por el proceso. Si el bloque del proceso secuencial utiliza también una señal de reset asíncrono, es necesario incluir ésta en la lista.
- *Lista sensitiva y funcionamiento de la simulación.*
  - *Recomendación:*
    - Hay que estar seguro de que las señales incluidas en la lista sensible son las necesarias.
      - Añadir más señales de las que se necesitan hace más lenta la simulación.

4. Recomendaciones para Síntesis

- *Asignaciones de señales vs. variables (VHDL)*
  - En la simulación con VHDL, las asignaciones de señales son programadas para la ejecución en el próximo ciclo de simulación mientras que las asignaciones de variable tienen lugar inmediatamente donde aparece en el código.
  - *Recomendación:*
    - Cuando se escribe código sintetizable, se sugiere emplear señales frente a variables para asegurar que el comportamiento de la simulación del diseño presíntesis sea igual que la netlist postsíntesis.
- *Declaraciones case vs. declaraciones if-then-else*
  - En VHDL, una declaración case supone un multiplexor de un único nivel, mientras que una declaración *if-then-else* supone un decodificador con prioridad.
  - *Recomendación:*
    - El multiplexor es un circuito más rápido. Si no se requiere una estructura de un decodificador con prioridad, se recomienda el empleo de declaraciones case frente a la *if-then-else*.

4. Recomendaciones para Síntesis

- *Declaraciones case vs. declaraciones if-then-else*
  - En VHDL, una declaración case supone un multiplexor de un único nivel, mientras que una declaración *if-then-else* supone un decodificador con prioridad.
  - *Recomendación:*
    - El multiplexor es un circuito más rápido. Si no se requiere una estructura de decodificación con prioridad, se recomienda el empleo de declaraciones case frente a la *if-then-else*.
      - En un simulador basado en ciclo, la declaración *case* se simula más rápido que la declaración *if-then-else*.
      - Una declaración de señal condicional puede también ser usada para obtener un multiplexor.
      - Para grandes multiplexores, una declaración case se simulará más rápido que una declaración condicional en la mayoría de los simuladores, y especialmente en simuladores basados en ciclos.
      - Para pequeños multiplexores, la velocidad relativa de las dos construcciones varía con diferentes simuladores.

4. Recomendaciones para Síntesis

- Las siguientes recomendaciones se utilizan para hacer un correcto particionado para síntesis,
  - Aporta ventajas como:
    - mejoras en los resultados de síntesis,
    - tiempo de compilación más rápido, etc.
- *Registrar todas las salidas*
  - *Recomendación:*
    - Para cada bloque de un diseño jerárquico, registrar todas las señales de salida desde el bloque.
      - Simplifica el proceso de síntesis porque hace que las salidas sean fuertemente controlables y los retardos de las entradas predecibles.
      - Todas las entradas de cada bloque llegan con el mismo retardo relativo.
- *Localizar la lógica combinacional en un único módulo*
  - *Recomendación:*
    - Mantener la lógica combinacional relacionada en el mismo módulo.
      - Las herramientas de síntesis *producen mejores resultados* cuando la lógica combinacional relativa está localizada en el mismo módulo ya que generalmente no es recomendable mover lógica a través de los límites jerárquicos del módulo.
      - Mantener la lógica combinacional relacionada en el mismo módulo permite una simulación más rápida.

4. Recomendaciones para Síntesis

- *Lógica asíncrona*
  - *Recomendación:*
    - Evitar la lógica asíncrona.
      - La lógica asíncrona es más difícil para diseñar correctamente y hacer verificaciones de un diseño.
      - Su temporización y funcionalidad correcta puede ser dependiente de la tecnología lo que limita la portabilidad del diseño.
  - *Recomendación:*
    - Si se requiere lógica asíncrona en el diseño, partir la lógica asíncrona en un módulo separado al de la lógica síncrona,
      - Aislar dicha lógica en un módulo separado hace mucho más fácil la inspección del código.

Técnicas de Modelado para Síntesis 31

Técnicas de Modelado para Síntesis 32

Parte II:
Modelado RTL de componentes para Xilinx en VHDL

Modelado de componentes para Xilinx

- Introducción
- Uso de tipos Signed y Unsigned
- Uso de constantes y parámetros
- Translate_on, translate_off
- Registros
- Triestados
- Contadores
- Acumuladores
- Registros de desplazamiento
- Registros dinámicos
- Multiplexores
- Decodificadores
- Codificadores con prioridad
- Desplazamiento lógico
- Operaciones aritméticas
- MAC
- Divisores por 2
- RAMS
- Máquinas de estado
- Soporte para “Cajas Negras”

Introducción

- Durante la fase de síntesis se trata de reconocer las construcciones del lenguaje que permitan reconocer estructuras hardware combinacionales y secuenciales tales como flip-flops, sumadores, contadores, maquinas de estado finito (FSM), RAMS, etc.)
- A partir de aquí dichas construcciones se optimizan dependiendo de la arquitectura de la FPGA de referencia.
- Es posible tener control sobre la lógica inferida mediante el uso de restricciones de diseño

Técnicas de Modelado para Síntesis 35

Soporte de tipos Signed/Unsigned

- En VHDL, dependiendo del tipo de operación y de operandos, es necesario incluir diferentes librerías (signed, unsigned, numeric …)
- Por ejemplo:
  - Para crear un sumador sin signo:

PACKAGE	TYPE
numeric_std	unsigned
std_logic_arith	unsigned
std_logic_unsigned	std_logic_vector

  - Para crear un sumador con signo:

PACKAGE	TYPE
numeric_std	signed
std_logic_arith	signed
std_logic_signed	std_logic_vector

Técnicas de Modelado para Síntesis 36

Uso de Constantes y Parámetros

- El uso de constantes y parámetros permite clarificar el código haciéndolo más portable y facilitando su lectura
- Puede facilitar la documentación sobre el propio código
- Ejemplo:

```
constant ZERO : STD_LOGIC_VECTOR (1 downto 0):="00";
constant A_AND_B: STD_LOGIC_VECTOR (1 downto 0):="01";
constant A_OR_B : STD_LOGIC_VECTOR (1 downto 0):="10";
constant ONE : STD_LOGIC_VECTOR (1 downto 0):="11";

process (OPCODE, A, B)
 begin
          if (OPCODE = A_AND_B)then OP_OUT <= A and B;
          elsif (OPCODE = A_OR_B) then
            OP_OUT <= A or B;
          elsif (OPCODE = ONE) then
            OP_OUT <= '1';
          else
            OP_OUT <= '0';
          end if;
        end process;
```

Técnicas de Modelado para Síntesis 37

Uso de Constantes y Parámetros

- Es posible especificar parámetros para crear módulos parametrizables mediante el uso de genéricos en VHDL, ya sea para puertos o señales
- Es posible especificar valores por defecto de los parámetros lo que facilita la reutilización del código
- Ejemplo:

```
-- FIFO_WIDTH data width (number of bits)
-- FIFO_DEPTH by number of address bits for the FIFO RAM i.e. 9 -> 2**9 -> 512 words
-- FIFO_RAM_TYPE: BLOCKRAM or DISTRIBUTED_RAM
-- Note: DISTRIBUTED_RAM suggested for FIFO_DEPTH of 5 or less
entity async_fifo is
   generic (FIFO_WIDTH: integer := 16;)
             FIFO_DEPTH: integer := 9;
             FIFO_RAM_TYPE: string := "BLOCKRAM");
   port ( din : in std_logic_vector(FIFO_WIDTH-1 downto 0);
          rd_clk, rd_en, ainit, wr_clk, wr_en : in std_logic;
          dout : out std_logic_vector(FIFO_WIDTH-1 downto 0) := (others=>'0');
          empty, almost_empty : out std_logic := '1';
          full, almost_full : out std_logic := '0');
end async_fifo;
architecture BEHAVIORAL of async_fifo is
   type ram_type is array ((2**FIFO_DEPTH)-1 downto 0) of std_logic_vector (FIFO_WIDTH-1 downto 0);
```

Técnicas de Modelado para Síntesis 38

Translate_on translate_off

- TRANSLATE_OFF y TRANSLATE_ON puede utilizarse para incluir código de simulacíón en ficheros sintetizables.

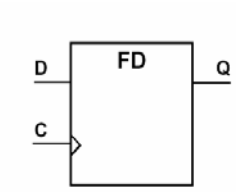
Registros

- Es posible inferir registros a partir de l descripción VHDL con las siguientes señales de control:
  - Set/Clear asíncrono
  - Set/Clear síncrono
  - Clock Enable
- Ejemplo de fichero de información de resultados de síntesis

```
...
Synthesizing Unit <flop>.
Related source file is ff_1.vhd.
Found 1-bit register for signal <q>.
Summary:
inferred 1 D-type flip-flop(s).
Unit <flop> synthesized.
...
=============================
HDL Synthesis Report
Macro Statistics
# Registers : 1
1-bit register : 1
=============================
...
```

Flip-flop con flanco de reloj positivo

```
library IEEE;
use IEEE.std_logic_1164.all;
entity flipflop is
  port(C, D : in std_logic;
       Q : out std_logic);
end flipflop;
architecture rtl of flop is
 begin
   process (C)
     begin
      if (C'event and C='1') then
        Q <= D;
      end if;
   end process;
end rtl;
```



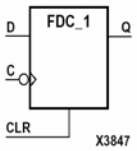
IO Pins	Description
D	Data Input
C	Positive Edge Clock
Q	Data Output

OPCIONES:
- Para flanco de reloj positivo, (C'event and C='1') es equivalente a (rising_edge(C))
- Para flanco de reloj negativo, (falling_edge(C)) es equivalente a C'event and C='0'

Técnicas de Modelado para Síntesis 41

Flip-flop con flanco de reloj negativo con clear asíncrono

```
library ieee;
use ieee.std_logic_1164.all;
entity flipflop is
  port(C, CLR, D : in std_logic;
       Q : out std_logic);
end flipflop;
architecture rtl of flop is
 begin
   process (C)
     begin
      if (CLR = '1') then
        Q <= '0';
      elsif (C'event and C='0')then
        Q <= D;
      end if;
   end process;
end rtl;
```



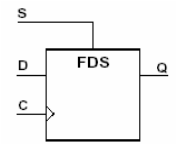
IO Pins	Description
D	Data Input
C	Negative-Edge Clock
CLR	Asynchronous Clear (active High)
Q	Data Output

OPCIONES:
- Para flanco de reloj positivo, (C'event and C='1') es equivalente a (rising_edge(C))
- Para flanco de bajada, (falling_edge(C)) es equivalente a C'event and C='0'

Técnicas de Modelado para Síntesis 42

Flip-flop con flanco de reloj positivo y set síncrono

```
library ieee;
use ieee.std_logic_1164.all;
entity flop is
  port(C, D, S : in std_logic;
       Q : out std_logic);
  end flop;
architecture archi of flop is
  begin
    process (C)
      begin
        if (C'event and C='1') then
          if (S='1') then
            Q <= '1';
          else
            Q <= D;
          end if;
        end if;
  end process;
end archi;
```



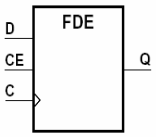
IO Pins	Description
D	Data Input
C	Positive-Edge Clock
S	Synchronous Set (active High)
Q	Data Output

OPCIONES:
- Para flanco de reloj positivo, (C'event and C='1') es equivalente a (rising_edge(C))
- Para flanco de bajada, (falling_edge(C)) es equivalente a C'event and C='0'

Técnicas de Modelado para Síntesis 43

Flip-flop con flanco de reloj positivo y clock enable

```
library ieee;
use ieee.std_logic_1164.all;
entity flop is
    port(C, D, CE : in std_logic;
         Q : out std_logic);
end flop;
architecture archi of flop is
  begin
    process (C)
      begin
        if (C'event and C='1') then
          if (CE='1') then
            Q <= D;
          end if;
        end if;
  end process;
end archi;
```



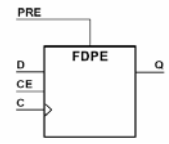
IO Pins	Description
D	Data Input
C	Positive-Edge Clock
CE	Clock Enable (active High)
Q	Data Output

OPCIONES:
- Para flanco de reloj positivo, (C'event and C='1') es equivalente a (rising_edge(C))
- Para flanco de bajada, (falling_edge(C)) es equivalente a C'event and C='0'

Técnicas de Modelado para Síntesis 44

Registro de 4-bits con flanco de subida del reloj, set asíncrono y Clock Enable

```
library ieee;
use ieee.std_logic_1164.all;
entity flop is
  port(C, CE, PRE : in std_logic;
    D : in std_logic_vector(3 downto 0);
    Q : out std_logic_vector(3 downto 0));
end flop;
architecture archi of flop is
  begin
    process (C, PRE)
      begin
        if (PRE='1') then
          Q <= "1111";
        elsif (C'event and C='1') then
          if (CE='1') then
            Q <= D;
          end if;
        end if;
    end process;
end archi;
```



IO Pins	Description
D[3:0]	Data Input
C	Positive-Edge Clock
PRE	Asynchronous Set (active High)

OPCIONES:
- Para flanco de reloj positivo, (C'event and C='1') es equivalente a (rising_edge(C))
- Para flanco de bajada, (falling_edge(C)) es equivalente a C'event and C='0'

Técnicas de Modelado para Síntesis 45

Latches

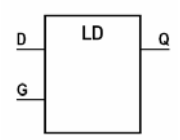
- Es posible inferir latches con set y clear asíncrono.
- Los latches se pueden describir usando:
  - Procesos en VHDL sin señales de reloj
  - Procesos sin construcción del tipo clk'event and clk='1' (o '0').
  - Asignación concurrente de estados.
- Ejemplo de fichero LOG

```
...
Synthesizing Unit <latch>.
Related source file is latch_1.vhd.
WARNING:Xst:737 - Found 1-bit latch for signal <q>.
Summary:
inferred 1 Latch(s).
Unit <latch> synthesized.
=========================
HDL Synthesis Report
Macro Statistics
# Latches : 1
1-bit latch : 1
=========================
...
```

Técnicas de Modelado para Síntesis 46

Latch con control positivo

```
library ieee;
use ieee.std_logic_1164.all;

entity latch is
  port(G, D : in std_logic;
       Q : out std_logic);
end latch;
architecture archi of latch is
 begin
   process (G, D)
     begin
       if (G='1') then
         Q <= D;
       end if;
   end process;
end archi;
```

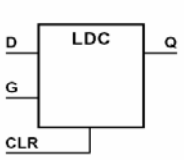


IO Pins	Description
D	Data Input
G	Positive Gate
Q	Data Output

Técnicas de Modelado para Síntesis 47

Latch con control positivo y clear asíncrono

```
library ieee;
use ieee.std_logic_1164.all;
entity latch is
  port(G, D, CLR : in std_logic;
       Q : out std_logic);
end latch;

architecture archi of latch is
 begin
   process (CLR, D, G)
     begin
       if (CLR='1') then
         Q <= '0';
       elsif (G='1') then
         Q <= D;
       end if;
   end process;
end archi;
```

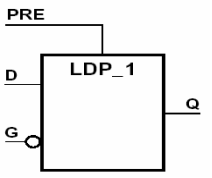


IO Pins	Description
D	Data Input
G	Positive Gate
CLR	Asynchronous Clear (active High)
Q	Data Output

Técnicas de Modelado para Síntesis 48

Latch de 4-bits con entrada de control invertida y preset asíncrono

```
library ieee;
use ieee.std_logic_1164.all;
entity latch is
  port(
    D : in std_logic_vector(3 downto 0);
    G, PRE : in std_logic;
    Q : out std_logic_vector(3 downto 0));
end latch;
architecture archi of latch is
  begin
    process (PRE, G)
      begin
        if (PRE='1') then
          Q <= "1111";
        elsif (G='0') then
          Q <= D;
        end if;
    end process;
end archi;
```



IO Pins	Description
D[3:0]	Data Input
G	Inverted Gate
PRE	Asynchronous Preset (active High)
Q[3:0]	Data Output

Técnicas de Modelado para Síntesis 49

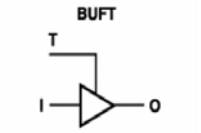
Control Tri-estado

- El control triestado facilita el control de buses de datos con operaciones de lectura y escritura compartidos entre diferentes dispositivos.
- En VHDL es posible describirlos mediante:
  - Un proceso combinacional
  - Una asignación concurrente
- Ejemplo de fichero LOG:

```
...
Synthesizing Unit <three_st>.
Related source file is tristates_1.vhd.
Found 1-bit tristate buffer for signal <o>.
Summary:
inferred 1 Tristate(s).
Unit <three_st> synthesized.
=============================
HDL Synthesis Report
Macro Statistics
# Tristates : 1
1-bit tristate buffer : 1
=============================
...
```

Técnicas de Modelado para Síntesis 50

Control Tri-estado



IO Pins	Description
I	Data Input
T	Output Enable (active Low)
O	Data Output

• Asignación concurrente

```
library ieee;
use ieee.std_logic_1164.all;
entity three_st is
  port(T : in std_logic;
    I: in std_logic;
   O: out std_logic);
  end three_st;
architecture archi of three_st is
  begin
    O <= I when (T='0') else 'Z';
end archi;
```

• Proceso combinacional

```
library ieee;
use ieee.std_logic_1164.all;
entity three_st is
  port(T : in std_logic;
    I : in std_logic;
   O : out std_logic);
end three_st;

architecture archi of three_st is
  begin
    process (I, T)
      begin
        if (T='0') then
          O <= I;
        else
          O <= 'Z';
        end if;
    end process;
end archi;
```

Técnicas de Modelado para Síntesis 51

Contadores

- Es posible obtener contadores con las siguientes señales de control:
  - Set/Clear Asíncrono
  - Set/Clear Síncrono
  - Carga de datos Asíncrona/Síncrona (señal o contante)
  - Clock Enable
  - Modos (Up, Down, Up/Down)
  - Mezcla de todas las posiblidades anteriores
- Las técnicas de codificación VHDL son similares a las descritas para los registros.
- Está soportada la posiblidad de inferir contadores con signo y sin él.

Fichero LOG

```
...
Synthesizing Unit <counter>.
Related source file is counters_1.vhd.
Found 4-bit up counter for signal <tmp>.
Summary:
inferred 1 Counter(s).
Unit <counter> synthesized.
=============================
HDL Synthesis Report
Macro Statistics
# Counters : 1
4-bit up counter : 1
=============================
...
```

Si no existe una señal de LOAD síncrona la herramienta de síntesis puede descomponer el contador en registros y sumadores. En este caso en el report aparecerán inferidos únicamente sumadores y registros.

Técnicas de Modelado para Síntesis 52

Contador de 4 bits con Clear asíncrono y cuenta ascendente

IO Pins	Description
C	Positive-Edge Clock
CLR	Asynchronous Clear (active High)
Q[3:0]	Data Output

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity counter is
  port(C, CLR : in std_logic;
    Q : out std_logic_vector(3 downto 0));
end counter;

architecture archi of counter is
  signal tmp: std_logic_vector(3 downto 0);
begin
  process (C, CLR)
    begin
      if (CLR='1') then
        tmp <= "0000";
    elsif (C'event and C='1') then
        tmp <= tmp + 1;
    end if;
  end process;
  Q <= tmp;
end archi;
```

Técnicas de Modelado para Síntesis 53

Contador de 4 bits con Set síncrono y cuenta descendente

IO Pins	Description
C	Positive-Edge Clock
S	Synchronous Set (active High)
Q[3:0]	Data Output

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity counter is
  port(C, S : in std_logic;
        Q : out std_logic_vector(3 downto 0));
end counter;
architecture archi of counter is
  signal tmp: std_logic_vector(3 downto 0);
  begin
    process (C)
      begin
        if (C'event and C='1') then
          if (S='1') then
            tmp <= "1111";
          else
            tmp <= tmp - 1;
          end if;
        end if;
      end process;
  Q <= tmp;
end archi;
```

Técnicas de Modelado para Síntesis 54

Contador de 4 bits con carga asíncrona desde la entrada primaria y cuenta ascendente

IO Pins	Description
C	Positive-Edge Clock
ALOAD	Asynchronous Load (active High)
D[3:0]	Data Input
Q[3:0]	Data Output

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity counter is
  port(C, ALOAD : in std_logic;
       D : in std_logic_vector(3 downto 0);
       Q : out std_logic_vector(3 downto 0));
end counter;
architecture archi of counter is
  signal tmp: std_logic_vector(3 downto 0);
  begin
    process (C, ALOAD, D)
      begin
        if (ALOAD='1') then
          tmp <= D;
        elsif (C'event and C='1') then
          tmp <= tmp + 1;
        end if;
    end process;
  Q <= tmp;
end archi;
```

Técnicas de Modelado para Síntesis 55

Contador de 4 bits con clear, clock enable y cuenta ascendente

IO Pins	Description
C	Positive-Edge Clock
CLR	Asynchronous Clear (active High)
CE	Clock Enable
Q[3:0]	Data Output

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity counter is
  port(C, CLR, CE : in std_logic;
       Q : out std_logic_vector(3 downto 0));
end counter;
architecture archi of counter is
  signal tmp: std_logic_vector(3 downto 0);
  begin
    process (C, CLR)
      begin
        if (CLR='1') then
          tmp <= "0000";
        elsif (C'event and C='1') then
          if (CE='1') then
            tmp <= tmp + 1;
          end if;
        end if;
    end process;
  Q <= tmp;
end archi;
```

Técnicas de Modelado para Síntesis 56

Contador de 4 bits con clear asíncrono y dirección de cuenta programable

IO Pins	Description
C	Positive-Edge Clock
CLR	Asynchronous Clear (active High)
UP_DOWN	up/down count mode selector
Q[3:0]	Data Output

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity counter is
  port(C, CLR, UP_DOWN : in std_logic;
       Q : out std_logic_vector(3 downto 0));
end counter;
architecture archi of counter is
  signal tmp: std_logic_vector(3 downto 0);
  begin
    process (C, CLR)
      begin
        if (CLR='1') then
          tmp <= "0000";
        elsif (C'event and C='1') then
          if (UP_DOWN='1') then
            tmp <= tmp + 1;
          else
            tmp <= tmp - 1;
          end if;
        end if;
      end process;
    Q <= tmp;
end archi;
```


Acumuladores

- En un acumulador:
  - el primer operando y el destino puesde ser una variable o señal
  - El segundo operando puede ser también una variable o señal o una contante (≠ 1)
- Un acumulador puede ser inferido con el mismo conjunto de señales de control de un registro
- Un acumulador puede inferirse tanto para acumulación positiva (↑) como negativa (↓)

Fichero LOG

```
...
Synthesizing Unit <accum>.
  Related source file is accumulators_1.vhd.
  Found 4-bit up accumulator for signal <tmp>.
  Summary:
    inferred 1 Accumulator(s).
Unit <accum> synthesized.
=============================
  HDL Synthesis Report
  Macro Statistics
  # Accumulators : 1
    4-bit up accumulator : 1
=============================
...
```

Al igual que ocurría con los contadores, si no existe una señal de LOAD síncrona la herramienta de síntesis puede descomponer el acumulador en registros y sumadores. En este caso en el report aparecerán inferidos únicamente sumadores y registros.

Acumulador de 4 bits con clear asíncrono

IO Pins	Description
C	Positive-Edge Clock
CLR	Asynchronous Clear (active High)
D[3:0]	Data Input
Q[3:0]	Data Output

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity accum is
port(C, CLR : in std_logic;
     D : in std_logic_vector(3 downto 0);
     Q : out std_logic_vector(3 downto 0));
end accum;

architecture archi of accum is
  signal tmp: std_logic_vector(3 downto 0);
  begin
    process (C, CLR)
      begin
        if (CLR='1') then
          tmp <= "0000";
        elsif (C'event and C='1') then
          tmp <= tmp + D;
        end if;
    end process;
  Q <= tmp;
end archi;
```

Técnicas de Modelado para Síntesis 59

Registros de desplazamiento

- Un registro de desplazamiento (RD) se caracteriza por las siguientes señales de control:
  - Clock
  - Entrada serial
  - Set/Reset Asíncrono
  - Set/Reset Síncrono
  - Carga paralela síncrona/asíncrona
  - Clock enable
  - Salida serial o paralela. Los modos de salida pueden ser:
    - Serie: únicamente se tiene acceso a la salida del último F/F
    - Paralela: Se accede a cualquiera de los otros F/Fs
  - Modos de desplazamiento: izquierda, derecha, etc.
- Existen diferentes modos de describir un RD:
  - Mediante el operador concatenación:

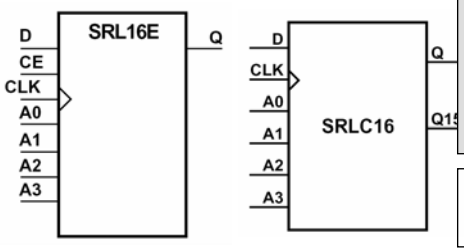
    ```
    shreg <= shreg (6 downto 0) & SI;
    ```
  - Mediante bucles FOR:

    ```
    for i in 0 to 6 loop
      shreg(i+1) <= shreg(i);
    end loop;
    shreg(0) <= SI;
    ```
  - Mediante los operadores predefinidos en el lenguaje (sll, srl, sla, ...)

Técnicas de Modelado para Síntesis 60

Registros de desplazamiento

- Restricciones especificas para las FPGAs
  - En algunas familias de XILINX existen recursos hardware específicos para implementar registros de desplazamiento



Fichero LOG

```
...
Synthesizing Unit <shift>.
    Related source file is shift_registers_1.vhd.
    Found 8-bit shift register for signal <tmp<7>>.
    Summary:
      inferred 1 Shift register(s).
Unit <shift> synthesized.
=============================
HDL Synthesis Report
  Macro Statistics
    # Shift Registers : 1
     8-bit shift register : 1
=============================
...
```

El parámetro shreg_extract permite controlar la extracción de registros de desplazamiento durante la síntesis

Registros de desplazamiento. Ejemplos

IO Pins	Description
C	Positive-Edge Clock
SI	Serial In
SO	Serial Output

```
-- 8-bit Shift-Left Register with Positive-Edge Clock,
-- Serial In, and Serial Out
-- Se infiere la macro SRL16.
library ieee;
use ieee.std_logic_1164.all;
entity shift is
   port(C, SI : in std_logic;
        SO : out std_logic);
end shift;
architecture archi of shift is
  signal tmp: std_logic_vector(7 downto 0);
  begin
    process (C)
      begin
        if (C'event and C='1') then
          for i in 0 to 6 loop
            tmp(i+1) <= tmp(i);
          end loop;
          tmp(0) <= SI;
        end if;
    end process;
   SO <= tmp(7);
end archi;
```

Registros de desplazamiento. Ejemplos

IO Pins	Description
C	Negative-Edge Clock
SI	Serial In
CE	Clock Enable (active High)
SO	Serial Output

```
-- 8-bit Shift-Left Register with
-- Negative-Edge Clock,
-- Clock Enable, Serial In, and Serial Out
-- Se infiere la macro SRL16E_1.

library ieee;
use ieee.std_logic_1164.all;
entity shift is
  port(C, SI, CE : in std_logic;
       SO : out std_logic);
end shift;
```

```
architecture archi of shift is
  signal tmp: std_logic_vector(7 downto 0);
  begin
    process (C)
      begin
        if (C'event and C='0') then
          if (CE='1') then
            for i in 0 to 6 loop
              tmp(i+1) <= tmp(i);
            end loop;
            tmp(0) <= SI;
          end if;
        end if;
    end process;
  SO <= tmp(7);
end archi;
```

Técnicas de Modelado para Síntesis 63

Registros de desplazamiento. Ejemplos

IO Pins	Description
C	Positive-Edge Clock
SI	Serial In
CLR	Asynchronous Clear (active High)
SO	Serial Output

```
-- 8-bit Shift-Left Register with
-- Negative-Edge Clock,
-- Clock Enable, Serial In, and Serial Out
-- Se infiere la macro SRL16E_1.

library ieee;
use ieee.std_logic_1164.all;
entity shift is
  port(C, SI, CE : in std_logic;
       SO : out std_logic);
end shift;
```

```
architecture archi of shift is
  signal tmp: std_logic_vector(7 downto 0);
  begin
    process (C)
      begin
        if (C'event and C='0') then
          if (CE='1') then
            for i in 0 to 6 loop
              tmp(i+1) <= tmp(i);
            end loop;
            tmp(0) <= SI;
          end if;
        end if;
    end process;
  SO <= tmp(7);
end archi;
```

Técnicas de Modelado para Síntesis 64

Registros de desplazamiento. Ejemplos

IO Pins	Description
C	Positive-Edge Clock
SI	Serial In
S	Synchronous Set (active High)
SO	Serial Output

```
-- 8-bit Shift-Left Register with
-- Positive-Edge Clock,
-- Synchronous Set, Serial In, and Serial Out
library ieee;
use ieee.std_logic_1164.all;
entity shift is
  port(C, SI, S : in std_logic;
       SO : out std_logic);
  end shift;
```

```
architecture archi of shift is
  signal tmp: std_logic_vector(7 downto 0);
  begin
    process (C, S)
      begin
        if (C'event and C='1') then
          if (S='1') then
            tmp <= (others => '1');
          else
            tmp <= tmp(6 downto 0) & SI;
          end if;
        end if;
    end process;
  SO <= tmp(7);
end archi;
```


Registros de desplazamiento. Ejemplos

IO Pins	Description
C	Positive-Edge Clock
SI	Serial In
PO[7:0]	Parallel Output

```
-- 8-bit shift-left register with a positive-edge clock,
-- serial in, and parallel out.
library ieee;
use ieee.std_logic_1164.all;

entity shift is
  port(C, SI : in std_logic;
       PO : out std_logic_vector(7 downto 0));
end shift;

architecture archi of shift is
  signal tmp: std_logic_vector(7 downto 0);
  begin
    process (C)
      begin
        if (C'event and C='1') then
          tmp <= tmp(6 downto 0)& SI;
        end if;
    end process;
  PO <= tmp;
end archi;
```

Registros de desplazamiento. Ejemplos

IO Pins	Description
C	Positive-Edge Clock
SI	Serial In
ALOAD	Asynchronous Parallel Load (active High)
D[7:0]	Data Input
SO	Serial Output

```
-- 8-bit shift-left register with a positive edge,
-- asynchronous parallel load, serial in, and
serial out.
library IEEE;
use ieee.std_logic_1164.all;
entity shift is
port(C, SI, ALOAD : in std_logic;
     D : in std_logic_vector(7 downto 0);
     SO : out std_logic);
end shift;
architecture archi of shift is
  signal tmp: std_logic_vector(7 downto 0);
  begin
    process (C, ALOAD, D)
      begin
        if (ALOAD='1') then
          tmp <= D;
        elsif (C'event and C='1') then
          tmp <= tmp(6 downto 0) & SI;
        end if;
    end process;
  SO <= tmp(7);
end archi;
```

Técnicas de Modelado para Síntesis 67

Registros de desplazamiento. Ejemplos

IO Pins	Description
C	Positive-Edge Clock
SI	Serial In
ALOAD	Asynchronous Parallel Load (active High)
D[7:0]	Data Input
SO	Serial Output

```
-- 8-bit shift-left register with a positive edge,
-- asynchronous parallel load, serial in, serial
out.
library ieee;
use ieee.std_logic_1164.all;
entity shift is
port(C, SI, ALOAD : in std_logic;
     D : in std_logic_vector(7 downto 0);
     SO : out std_logic);
end shift;
architecture archi of shift is
  signal tmp: std_logic_vector(7 downto 0);
  begin
    process (C, ALOAD, D)
      begin
        if (ALOAD='1') then
          tmp <= D;
        elsif (C'event and C='1') then
          tmp <= tmp(6 downto 0) & SI;
        end if;
    end process;
  SO <= tmp(7);
end archi;
```

Técnicas de Modelado para Síntesis 68

Registros de desplazamiento. Ejemplos

IO Pins	Description
C	Positive-Edge Clock
SI	Serial In
SLOAD	Synchronous Parallel Load (active High)
D[7:0]	Data Input
SO	Serial Output

```
-- 8-bit shift-left register with a positive-edge clock,
-- synchronous parallel load, serial in, and serial out.
library ieee;
use ieee.std_logic_1164.all;
entity shift is
  port(C, SI, SLOAD : in std_logic;
       D : in std_logic_vector(7 downto 0);
       SO : out std_logic);
end shift;
architecture archi of shift is
  signal tmp: std_logic_vector(7 downto 0);
  begin
    process (C)
      begin
        if (C'event and C='1') then
          if (SLOAD='1') then
            tmp <= D;
          else
            tmp <= tmp(6 downto 0) & SI;
          end if;
        end if;
    end process;
  SO <= tmp(7);
end archi;
```

Técnicas de Modelado para Síntesis 69

Registros de desplazamiento. Ejemplos

IO Pins	Description
C	Positive-Edge Clock
SI	Serial In
LEFT_RIGHT	Left/right shift mode selector
PO[7:0]	Parallel Output

```
-- 8-bit shift-left/shift-right register with a
-- positive-edge clock, serial in, and serial out.
library ieee;
use ieee.std_logic_1164.all;
entity shift is
  port(C, SI, LEFT_RIGHT : in std_logic;
       PO : out std_logic_vector(7 downto 0));
end shift;
architecture archi of shift is
  signal tmp: std_logic_vector(7 downto 0);
  begin
    process (C)
      begin
        if (C'event and C='1') then
          if (LEFT_RIGHT='0') then
            tmp <= tmp(6 downto 0) & SI;
          else
            tmp <= SI & tmp(7 downto 1);
          end if;
        end if;
    end process;
  PO <= tmp;
end archi;
```

Técnicas de Modelado para Síntesis 70

Registro de desplazamiento parametrizable

IO Pins	Description
Clk	Positive-Edge Clock
SI	Serial In
AClr	Asynchronous Clear (optional)
SClr	Synchronous Clear (optional)
SLoad	Synchronous Parallel Load (optional)
Data	Parallel Data Input Port (optional)
ClkEn	Clock Enable (optional)
LeftRight	Direction selection (optional)
SerialInRight	Serial Input Right for Bidirectional Shift Register (optional)
PSO[x:0]	Serial or Parallel Output

Fichero LOG

```
...
Synthesizing Unit <dynamic_srl>.
Related source file is dynamic_srl.vhd.
 Found 1-bit 16-to-1 multiplexer for signal <Q>.
 Found 16-bit register for signal <data>.
Summary:
  inferred 16 D-type flip-flop(s).
  inferred 1 Multiplexer(s).
 Unit <dynamic_srl> synthesized.
...
Macro Statistics
 # Shift Registers : 1
 # 16-bit dynamic shift register : 1
...
```

```
-- 16-bit dynamic shift register.
-- Opciones:  serial or parallel, left, right o parallel;
-- Clear sincrono o asincrono.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity shiftregluts is
  port(CLK, DATA, CE : in std_logic;
       A : in std_logic_vector(3 downto 0);
       Q : out std_logic);
end shiftregluts;
architecture rtl of shiftregluts is
  constant DEPTH_WIDTH : integer := 16;
  type SRL_ARRAY is array (0 to DEPTH_WIDTH-1) of std_logic;
  signal SRL_SIG : SRL_ARRAY;
  begin
    PROC_SRL16 : process (CLK)
      begin
        if (CLK'event and CLK = '1') then
          if (CE = '1') then
            SRL_SIG <= DATA & SRL_SIG(0 to DEPTH_WIDTH-2);
          end if;
        end if;
    end process;
  Q <= SRL_SIG(conv_integer(A));
end rtl;
```


Multiplexores

- Existen diferentes estilos para describir un multiplexor:
  - If-Then-Else
  - Case.
- A tener en cuenta...
  - Si se utiliza una sentencia Case sin especificar todos los valores del selector se inferirán latches en vez de un multiplexor.
  - Es posible utilizar valore “don't cares” para los valores del selector
- Es posible controlar la extracción de muxes utilizando las restricciones mux_extract y mux_style.

Fichero LOG

```
...
Synthesizing Unit <mux>.
Related source file is multiplexers_1.vhd.
Found 1-bit 4-to-1 multiplexer for signal <o>.
Summary:
inferred 1 Multiplexer(s).
Unit <mux> synthesized.
=========================
HDL Synthesis Report
Macro Statistics
# Multiplexers : 1
1-bit 4-to-1 multiplexer : 1
=========================
...
```

Multiplexores

IO Pins	Description
a, b, c, d	Data Inputs
s[1:0]	MUX selector
o	Data Output

```
-- 4-to-1 1-bit MUX using an If statement.
library ieee;
use ieee.std_logic_1164.all;
entity mux is
  port (a, b, c, d : in std_logic;
        s : in std_logic_vector (1 downto 0);
        o : out std_logic);
end mux;
architecture archi of mux is
  begin
    process (a, b, c, d, s)
      begin
        if (s = "00") then o <= a;
        elsif (s = "01") then o <= b;
        elsif (s = "10") then o <= c;
        else o <= d;
        end if;
  end process;
end archi;
```

```
-- 4-to-1 1-bit MUX using a Case statement.

library ieee;
use ieee.std_logic_1164.all;

entity mux is
  port (a, b, c, d : in std_logic;
        s : in std_logic_vector (1 downto 0);
        o : out std_logic);
end mux;
architecture archi of mux is
  begin
    process (a, b, c, d, s)
      begin
        case s is
          when "00" => o <= a;
          when "01" => o <= b;
          when "10" => o <= c;
          when others => o <= d;
        end case;
  end process;
end archi;
```

Técnicas de Modelado para Síntesis 73

Multiplexores

IO Pins	Description
a, b, c, d	Data Inputs
s[1:0]	MUX selector
o	Data Output

```
-- 4-to-1 1-bit MUX using tristate buffers.

library ieee;
use ieee.std_logic_1164.all;

entity mux is
  port (a, b, c, d : in std_logic;
        s : in std_logic_vector (3 downto 0);
        o : out std_logic);
end mux;

architecture archi of mux is
  begin
    o <= a when (s(0)='0') else 'Z';
    o <= b when (s(1)='0') else 'Z';
    o <= c when (s(2)='0') else 'Z';
    o <= d when (s(3)='0') else 'Z';
end archi;
```

```
--  3-to-1 1-bit MUX with a 1-bit latch.
-- No genera un multiplexor


library ieee;
use ieee.std_logic_1164.all;
entity mux is
  port (a, b, c, d : in std_logic;
        s : in std_logic_vector (1 downto 0);
        o : out std_logic);
end mux;
architecture archi of mux is
  begin
    process (a, b, c, d, s)
      begin
        if (s = "00") then o <= a;
        elsif (s = "01") then o <= b;
        elsif (s = "10") then o <= c;
        end if;
  end process;
end archi
```

Técnicas de Modelado para Síntesis 74

Decodificadores

- Existen diferentes estilos para describir un multiplexor:
  - If-Then-Else
  - Case.
- A tener en cuenta...
  - Si se utiliza una sentencia Case sin especificar todos los valores del selector se inferirán latches en vez de un multiplexor.
  - Es posible utilizar valore “don't cares” para los valores del selector
- Es posible controlar la extracción de muxes utilizando las restricciones mux_extract y mux_style.

Fichero LOG

```
...
Synthesizing Unit <mux>.
Related source file is multiplexers_1.vhd.
Found 1-bit 4-to-1 multiplexer for signal <o>.
Summary:
inferred 1 Multiplexer(s).
Unit <mux> synthesized.
==========================
HDL Synthesis Report
Macro Statistics
# Multiplexers : 1
1-bit 4-to-1 multiplexer : 1
==========================
...
```

Técnicas de Modelado para Síntesis 75

Decodificadores

IO pins	Description
s[2:0]	Selector
res	Data Output

```
Synthesizing Unit <dec>.
    Related source file is decoders_1.vhd.
    Found 1-of-8 decoder for signal <res>.
    Summary:
        inferred   1 Decoder(s).
Unit <dec> synthesized.
==========================
HDL Synthesis Report

Macro Statistics
# Decoders                  : 1
  1-of-8 decoder            : 1
==========================
...
```

-- One-Hot 1-of-8 decoder.

```
library ieee;
use ieee.std_logic_1164.all;
entity dec is
  port (sel: in std_logic_vector (2 downto 0);
        res: out std_logic_vector (7 downto 0));
end dec;
architecture archi of dec is
  begin
    res <= "00000001" when sel = "000" else
           "00000010" when sel = "001" else
           "00000100" when sel = "010" else
           "00001000" when sel = "011" else
           "00010000" when sel = "100" else
           "00100000" when sel = "101" else
           "01000000" when sel = "110" else
           "10000000";
end archi;
```

-- One-Cold 1-of-8 decoder.

```
library ieee;
use ieee.std_logic_1164.all;
entity dec is
  port (sel: in std_logic_vector (2 downto 0);
        res: out std_logic_vector (7 downto 0));
end dec;
architecture archi of dec is
  begin
    res <= "11111110" when sel = "000" else
           "11111101" when sel = "001" else
           "11111011" when sel = "010" else
           "11110111" when sel = "011" else
           "11101111" when sel = "100" else
           "11011111" when sel = "101" else
           "10111111" when sel = "110" else
           "01111111";
end archi;
```

Técnicas de Modelado para Síntesis 76

Codificadores con prioridad

IO pins	Description
s[2:0]	Selector
res	Data Output

```
...
Synthesizing Unit <priority>.
    Related source file is priority_encoders_1.vhd.
    Found 3-bit 1-of-9 priority encoder for signal <code>.
    Summary:
        inferred   3 Priority encoder(s).
Unit <priority> synthesized.

=============================
HDL Synthesis Report

Macro Statistics
# Priority Encoders               : 1
  3-bit 1-of-9 priority encoder    : 1
=============================
...
```

```
-- 3-bit 1-of-9 Priority Encoder.
-- Utilizar la directiva priority_extract para
-- forzar su inferencia

library ieee;
use ieee.std_logic_1164.all;
entity priority is
  port ( sel : in std_logic_vector (7 downto 0);
         code :out std_logic_vector (2 downto 0));
end priority;
architecture archi of priority is
  begin
    code <= "000" when sel(0) = '1' else
            "001" when sel(1) = '1' else
            "010" when sel(2) = '1' else
            "011" when sel(3) = '1' else
            "100" when sel(4) = '1' else
            "101" when sel(5) = '1' else
            "110" when sel(6) = '1' else
            "111" when sel(7) = '1' else
            "---";
end archi;
```

Técnicas de Modelado para Síntesis 77

Desplazadores Lógicos

- Circuito combinacional con dos entradas y una salida
  - Dato a desplazar
  - Distancia de desplazamiento
  - Salida de resultado de la operación
- Todas las E/S deben estar presentes
- Restricciones:
  - Usar únicamente operaciones de desplazamiento lógico, aritmético y rotación
  - Únicamente es posible usar los operadores de desplazamiento predefinidos: (sll, srl, rol, ror, rla, sra) o concatenación (&)
  - Usar únicamente un tipo de desplazamiento.
  - El valor n en puede ser aumentado únicamente en 1 para diferentes valores consecutivos del selector (ver ejemplos)
  - El valor de n deber ser positivo
  - Todos los valores del selector deben estar presentes.

Fichero LOG

```
...
Synthesizing Unit <lshift>.
Related source file is Logical_Shifters_1.vhd.
Found 8-bit shifter logical left for signal <so>.
Summary:
inferred 1 Combinational logic shifter(s).
Unit <lshift> synthesized.
...
==============================
HDL Synthesis Report
Macro Statistics
# Logic shifters : 1
8-bit shifter logical left : 1
==============================
...
```

El parámetro shift_extract permite controlar la inferencia de unidades de desplazamiento durante la síntesis

Técnicas de Modelado para Síntesis 78

Desplazadores Lógicos

IO pins	Description
D[7:0]	Data Input
SEL	shift distance selector
SO[7:0]	Data Output

```
-- Desplazador Logico

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity lshift is
  port(DI : in unsigned(7 downto 0);
       SEL : in unsigned(1 downto 0);
       SO : out unsigned(7 downto 0));
end lshift;
architecture archi of lshift is
  begin
    with SEL select
      SO <= DI when "00",
            DI sll 1 when "01",
            DI sll 2 when "10",
            DI sll 3 when others;
end archi;
```

```
-- Ejemplo donde NO se infiere un
-- Desplazador Logico ya que no todos
-- los valores del selector estan presentes

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity lshift is
  port(DI : in unsigned(7 downto 0);
       SEL : in unsigned(1 downto 0);
       SO : out unsigned(7 downto 0));
end lshift;
architecture archi of lshift is
  begin
    with SEL select
      SO <= DI when "00",
            DI sll 1 when "01",
            DI sll 2 when others;
end archi;
```

Técnicas de Modelado para Síntesis 79

Desplazadores Lógicos

IO pins	Description
D[7:0]	Data Input
SEL	shift distance selector
SO[7:0]	Data Output

```
-- Desplazador Logico

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity lshift is
  port(DI : in unsigned(7 downto 0);
       SEL : in unsigned(1 downto 0);
       SO : out unsigned(7 downto 0));
end lshift;
architecture archi of lshift is
  begin
    with SEL select
      SO <= DI when "00",
            DI sll 1 when "01",
            DI sll 2 when "10",
            DI sll 3 when others;
end archi;
```

```
-- Ejemplo donde NO se infiere un
-- Desplazador Logico ya que no todos
-- los valores del selector se incrementan
-- en una unidad

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity lshift is
  port(DI : in unsigned(7 downto 0);
       SEL : in unsigned(1 downto 0);
       SO : out unsigned(7 downto 0));
end lshift;
architecture archi of lshift is
  begin
    with SEL select
      SO <= DI when "00",
            DI sll 1 when "01",
            DI sll 3 when "10",
            DI sll 2 when others;
end archi;
```

Técnicas de Modelado para Síntesis 80

Operaciones aritméticas

- Operaciones soportadas:
  - Suma (con carry de entrada, de salida y de entrada y salida) ,
  - Resta,
  - Sumadores/restadores,
  - Comparadores (=, /=, <, <=, >, >=), Multiplicadores y divisores,
  - Operadores con signo o sin signo.
    - Paquetes:
      - IEEE.std_logic_unsigned
      - IEEE.std_logic_signed
- Optimización de recursos (*sharing*) para los operadores aritméticos

Sumadores, Restadores y Sumadores/restadores

IO pins	Description
A[7:0], B[7:0]	Add Operands
SUM[7:0]	Add Result

```
-- Sumador de 8 bits sin signo
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity adder is
  port(A,B : in std_logic_vector(7 downto 0);
       SUM : out std_logic_vector(7 downto
0));
end adder;
architecture archi of adder is
  begin
    SUM <= A + B;
end archi;
```

```
..
Synthesizing Unit <adder>.
    Related source file is arithmetic_operations_1.vhd.
    Found 8-bit adder for signal <sum>.
    Summary:
        inferred   1 Adder/Subtracter(s).
Unit <adder> synthesized.

=========================
HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                : 1
  8-bit adder                       : 1
=========================
```

IO pins	Description
A[7:0], B[7:0]	Add Operands
CI	Carry In
SUM[7:0]	Add Result

```
-- Sumador de 8 bits sin signo con
-- entrada de carry
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity adder is
  port(A,B : in std_logic_vector(7
downto 0);
       CI: in std_logic;
      SUM : out std_logic_vector(7
downto 0));
end adder;
architecture archi of adder is
  begin
    SUM <= A + B + CI;
end archi;
```

Sumadores, Restadores y Sumadores/restadores

- Operaciones con Carry de salida
  - La suma con carry se realiza añadiendo un bit al resultado. Dicho bit representa el carry de salida
  - Es necesario mantener los tipos de los operandos y del resultado
  - Si definimos:

    signal res : std_logic_vector(8 downto 0); -- resultado: carry & suma
    signal a, b: std_logic_vector(7 downto 0) -- operandos
  - No es posible realizar utilizando el paquete std_logic_unsigned la operación res <= b + b ya que el tamaño del resultado para la suma "+" es igual al tamaño del operando mayor (8 bits). Soluciones:
    - Solución1. Ajustar el tamaño de los operando mediante la concatenación a operandos de 9 bits:

      res <= ('0' & a) + ('0' & b);
    - Solución2: Otra solución consiste en convertir los operando a enteros y volver a convertir el resultado a std_logic_vector, especificando el tamaño del vector como 9:

      res <= conv>std_logic_vector(conv_integer(a)+conv_integer(b), 9)
  - En ambos casos:

    Carry <= res(8);
    Suma <= res(7 downto 0);

Sumadores, Restadores y Sumadores/restadores

IO pins	Description
A[7:0], B[7:0]	Add Operands
CI	Carry In
SUM[7:0]	Add Result
CO	Carry Out

```
-- Sumador de 8 bits con carry de entrada
-- y carry  de salida
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity adder is
  port(A,B : in std_logic_vector(7 downto 0);
       CI : in std_logic;
       SUM : out std_logic_vector(7 downto 0);
       CO : out std_logic);
end adder;
architecture archi of adder is
  signal tmp: std_logic_vector(8 downto 0);
  begin
    tmp <= conv_std_logic_vector(
         (conv_integer(A) +
          conv_integer(B) +
          conv_integer(CI)),9);
    SUM <= tmp(7 downto 0);
    CO <= tmp(8);
end archi;;
```

Sumadores, Restadores y Sumadores/restadores

IO pins	Description
A[7:0], B[7:0]	Add Operands
SUM[7:0]	Add Result

```
-- Sumador de 8 bits CON signo

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
entity adder is
  port(A,B : in std_logic_vector(7 downto 0);
       SUM : out std_logic_vector(7 downto 0));
end adder;
architecture archi of adder is
  begin
    SUM <= A + B;
end archi;
```

IO pins	Description
A[7:0], B[7:0]	Sub Operands
RES[7:0]	Sub Result

```
-- Restador de 8 bits

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity subtr is
  port(A,B : in std_logic_vector(7 downto 0);
       RES : out std_logic_vector(7 downto 0));
end subtr;
architecture archi of subtr is
  begin
    SUM <= A - B;
end archi;
```

Técnicas de Modelado para Síntesis 85

Sumadores, Restadores y Sumadores/restadores

IO pins	Description
A[7:0], B[7:0]	Add/Sub Operands
OPER	Add/Sub Select
SUM[7:0]	Add/Sub Result

```
-- Sumador / Restador
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity addsub is
  port(A,B : in std_logic_vector(7 downto 0);
       OPER: in std_logic;
       RES : out std_logic_vector(7 downto 0));
end addsub;
architecture archi of addsub is
  begin
   RES <= A + B when OPER='0'
             else A - B;
end archi;
```

Técnicas de Modelado para Síntesis 86

Comparators (=, /=,<, <=, >, >=)

IO pins	Description
A[7:0], B[7:0]	Add/Sub Operands
CMP	Comparison Result

```
-- Comparador de mayor o igual

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity compar is
  port(A,B : in std_logic_vector(7 downto 0);
       CMP : out std_logic);
end compar;
architecture archi of compar is
  begin
  CMP <= '1' when A >= B
              else '0';
end archi;
```

```
...
Synthesizing Unit <compar>.
  Related source file is comparators_1.vhd.
  Found 8-bit comparator greatequal for signal <$n0000> created at
line 10.
    Summary:
        inferred   1 Comparator(s).
 Unit <compar> synthesized.

 ============================
 HDL Synthesis Report

 Macro Statistics
 # Comparators                    : 1
   8-bit comparator greatequal    : 1
============================
...
```


Multiplicadores

- Cuando se implementa un multiplicador, el tamaño del resultado es igual a la suma de los tamaños de los operandos:
  - A (8-bit) * B (4-bit) → P (12-bit)
- En Virtex-II and Virtex-II Pro es posible implementar multiplicadores de 18x18 bits usando los bloques multiplicadores presentes en el dispositivo.
- También es posible generar multiplicadores registrados
  - En los siguientes casos se usará un multiplicador + registro:
    - La salida del multiplicador va a otro componente diferente de un registro
    - La selección del tipo de multiplicador es LUT
    - El multiplicador es asíncrono
    - El multiplicador tiene señales de control diferentes de un reset síncrono y un clock enable
    - El multiplicador no se puede mapear a un único módulo de 18x18 bits
  - Las siguientes E/S son opcionales para un multiplicador registrado:
    - Clock enable, Reset síncrono y asíncrono, reset y puerto de carga

Multiplicadores

IO pins	Description
A[7:0], B[3:0]	MULT Operands
RES[7:0]	MULT Result

```
-- Multiplicador 8x4 sin signo
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
entity mult is
  port(A : in std_logic_vector(7 downto 0);
       B : in std_logic_vector(3 downto 0);
     RES : out std_logic_vector(11 downto 0));
end mult;
architecture archi of mult is
  begin
    RES <= A * B;
end archi;
```

Fichero LOG

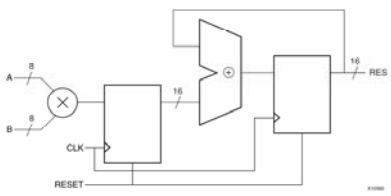
```
...
Synthesizing Unit <mult>.
Related source file is multipliers_1.vhd.
Found 8x4-bit multiplier for signal <res>.
Summary:
inferred 1 Multiplier(s).
Unit <mult> synthesized.
=============================
HDL Synthesis Report
Macro Statistics
# Multipliers : 1
8x4-bit multiplier : 1
=============================
...
```


Multiplicadores acumuladores

- Se trata de un bloque complejo compuesto por varias macros tales como multiplicadores, acumuladores y registros
- Da soporte a las macros DSP48 disponbles en los dispositivos VIRTEX 4 y 5.

```
=========================================================================
* HDL Synthesis *
=========================================================================
...
Synthesizing Unit <multipliers_7a>.
Related source file is "multipliers_7a.vhd".
Found 8x8-bit multiplier for signal <$n0002> created at line 28.
Found 16-bit up accumulator for signal <accum>.
Found 16-bit register for signal <mult>.
Summary:
inferred 1 Accumulator(s).
inferred 16 D-type flip-flop(s).
inferred 1 Multiplier(s).
Unit <multipliers_7a> synthesized....
=========================================================================
* Advanced HDL Synthesis *
=========================================================================
...
Synthesizing (advanced) Unit <Mmult__n0002>.
Multiplier <Mmult__n0002> in block <multipliers_7a> and
accumulator <accum> in block <multipliers_7a> are combined into a
MAC<Mmac_accum>.
The following registers are also absorbed by the MAC: <mult> in
block <multipliers_7a>.
Unit <Mmult__n0002> synthesized (advanced).
=========================================================================
HDL Synthesis Report
Macro Statistics
# MACs : 1
8x8-to-16-bit MAC : 1
=========================================================================
```

Multiplicadores acumuladores



IO Pins	Description
clk	Positive-Edge Clock
reset	Synchronous Reset
A, B	MAC Operands
RES	MAC Result

```
-- Multiplicador y acumulador con registro después
-- de la multiplicación
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity mac is
  generic (p_width: integer:=8);
  port (clk, reset: in std_logic;
        A, B: in std_logic_vector(p_width-1 downto 0);
        RES: out std_logic_vector(p_width*2-1 downto 0));
end mac;
architecture beh of mac is
signal mult, accum: std_logic_vector(p_width*2-1 downto 0);
begin
  process (clk)
    begin
      if (clk'event and clk='1') then
        if (reset = '1') then
          accum <= (others => '0');
          mult <= (others => '0');
        else
          accum <= accum + mult;
          mult <= A * B;
        end if;
      end if;
  end process;
  RES <= accum;
end beh;
```


Divisores

- La síntesis de la división está soportada únicamente cuando el divisor es una constante potencia de 2: implementación como un desplazador.

IO pins	Description
DI[7:0]	DIV Operands
DO[7:0]	DIV Result

```
-- Divisor por 2
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity divider is
  port(DI : in unsigned(7 downto 0);
       DO : out unsigned(7 downto 0));
end divider;
architecture archi of divider is
  begin
    DO <= DI / 2;
end archi;
```

Compartición de recursos

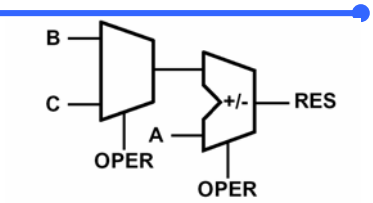
- El objetivo es minimizar el número de operadores y por tanto la lógica asociada en el diseño sintetizado
- Dos operaciones aritméticas similares se pueden implementarse en una única unidad funcional si no se están usando al mismo tiempo
- Para ello se utilizan multiplexores para seleccionar los datos a utilizar en cada momento.
- Esta capacidad está soportada para sumadores, restadores, sumadores/restadores y multiplicadores.

Fichero LOG

```
...
Synthesizing Unit <addsub>.
Related source file is resource_sharing_1.vhd.
Found 8-bit addsub for signal <res>.
Found 8 1-bit 2-to-1 multiplexers.
Summary:
inferred 1 Adder/Subtracter(s).
inferred 8 Multiplexer(s).
Unit <addsub> synthesized.
=============================
HDL Synthesis Report
Macro Statistics
# Multiplexers : 1
2-to-1 multiplexer : 1
# Adders/Subtractors : 1
8-bit addsub : 1
=============================
...
```


Compartición de recursos

IO pins	Description
A[7:0], B[7:0], B[7:0]	DIV Operands
OPER	Operation Selector
RES[7:0]	Data Output




```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity addsub is
  port(A,B,C : in std_logic_vector(7 downto 0);
       OPER : in std_logic;
       RES : out std_logic_vector(7 downto 0));
end addsub;
architecture archi of addsub is
  begin
    RES <= A + B when OPER='0'
     else A - C;
end archi;
```

Fichero LOG

```
...
Synthesizing Unit <addsub>.
Related source file is resource_sharing_1.vhd.
Found 8-bit addsub for signal <res>.
Found 8 1-bit 2-to-1 multiplexers.
Summary:
inferred 1 Adder/Subtracter(s).
inferred 8 Multiplexer(s).
Unit <addsub> synthesized.
=============================
HDL Synthesis Report
Macro Statistics
# Multiplexers : 1
2-to-1 multiplexer : 1
# Adders/Subtractors : 1
8-bit addsub : 1
=============================
...
```

Introducción

- Durante la fase de síntesis se trata de reconocer las construcciones del lenguaje que permitan reconocer estructuras hardware combinacionales y secuenciales tales como flip-flops, sumadores, contadores, maquinas de estado finito (FSM), RAMS, etc.
- A partir de aquí dichas construcciones se optimizan dependiendo de la arquitectura de la FPGA de referencia.
- Es posible tener control sobre la lógica inferida mediante el uso de restricciones de diseño (directivas).

Técnicas de Modelado para Síntesis 95

RAMs

- Se puede inferir memoria RAM distribuida o de bloque según su descripción…
  - Descripción con lectura asíncrona… RAM distribuida.
  - Descripción con lectura síncrona… RAM en bloque (*block-RAM*).
- Características de las memorias descritas…
  - Escritura síncrona.
  - Habilitación de escritura y de dispositivo.
  - Lectura asíncrona o síncrona.
  - Reset de la salida de datos y de sus latches.
  - Lectura de puerto único, doble y multipuerto.
  - Escritura de puerto único.
- Directivas relacionadas…
  - RAM_extract … habiltia o no el reconocimiento de RAM.
  - RAM_style … selecciona el tipo de memoria a usar (distribuida o bloque).

Técnicas de Modelado para Síntesis 96

RAMs : Fichero LOG

```
...
Synthesizing Unit <raminfr>.
Related source file is rams_1.vhd.
Found 128-bit single-port distributed RAM for signal <ram>.
-----------------------------------------------------------
| aspect ratio   | 32-word x 4-bit                |       |
| clock          | connected to signal <clk>      | rise  |
| write enable   | connected to signal <we>       | high  |
| address        | connected to signal <a>        |       |
| data in        | connected to signal <di>       |       |
| data out       | connected to signal <do>       |       |
| ram_style      | Auto                           |       |
-----------------------------------------------------------
INFO:Xst - For optimized device usage and improved timings, you
may take advantage of available block RAM resources by
registering the read address.
Summary:
inferred 1 RAM(s).
Unit <raminfr> synthesized.
=========================================
HDL Synthesis Report
Macro Statistics
# RAMs : 1
128-bit single-port distributed RAM : 1
=========================================
...
```

Cuando la herramienta de síntesis reconoce la presencia de una RAM, incluye en el informe la descripción de la misma como tipo y tamaño, así como la relativa a todos sus puertos de E/S.

RAMs : Modos de Lectura/Escritura

- La memoria tipo block-RAM dispone de diferentes modos de sincronización de lectura/escritura.
- Los modos disponibles según la familia del dispositivo...

Family	Inferred Modes	Behavior
Virtex-II, Virtex-II Pro	write-first, read-first, no-change	• Macro inference and generation • Attach adequate WRITE_MODE, WRITE_MODE_A, WRITE_MODE_B constraints to generated block RAMs in NCF
Virtex, Virtex-E, Spartan-II Spartan-IIE	write-first	• Macro inference and generation • No constraint to attach on generated block RAMs
CPLD	none	RAM inference completely disabled

- En caso de RAM multipuerto, cada puerto puede usar un modo distinto.

RAMs : Modos de Lectura/Escritura – *Read First*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity raminfr is
    port (clk : in std_logic;
          we : in std_logic;
          en : in std_logic;
          addr : in std_logic_vector(4 downto 0);
          di : in std_logic_vector(3 downto 0);
          do : out std_logic_vector(3 downto 0));
end raminfr;

architecture syn of raminfr is
    type ram_type is array (31 downto 0)
          of std_logic_vector (3 downto 0);
    signal RAM: ram_type;
begin
    process (clk)
    begin
          if clk'event and clk = '1' then
               if en = '1' then
                          if we = '1' then
                          RAM(conv_integer(addr)) <= di;
                          end if;
                          do <= RAM(conv_integer(addr)) ;
               end if;
          end if;
    end process;
end syn;
```

En este modo, cuando se realiza una escritura sobre una posición de la memoria RAM, a la salida de ésta se muestra el contenido que tenía la posición seleccionada ***antes*** de ser escrita con el nuevo valor.

P. Ej., supongamos que en la posición 20 está el valor 5 y se accede en escritura a esta posición para poner un 10. La RAM modifica el contenido de su posición 20 al valor 10 y muestra en su salida el valor de la posición 20 previo a la escritura, es decir, 5.

RAMs : Modos de Lectura/Escritura – *Write First*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity raminfr is
    port (clk : in std_logic;
          we : in std_logic;
          en : in std_logic;
          addr : in std_logic_vector(4 downto 0);
          di : in std_logic_vector(3 downto 0);
          do : out std_logic_vector(3 downto 0));
end raminfr;

architecture syn of raminfr is
    type ram_type is array (31 downto 0)
          of std_logic_vector (3 downto 0);
    signal RAM : ram_type;
begin
    process (clk)
    begin
          if clk'event and clk = '1' then
               if en = '1' then
                    if we = '1' then
                          RAM(conv_integer(addr)) <= di;
                          do <= di;
                    else
                          do <= RAM( conv_integer(addr));
                    end if;
               end if;
          end if;
    end process;
end syn;
```

En este modo, cuando se realiza una escritura sobre una posición de la memoria RAM, a la salida de ésta se muestra el contenido de la posición seleccionada ***después*** de ser escrita con el nuevo valor.

P. Ej., supongamos que en la posición 20 está el valor 5 y se accede en escritura a esta posición para poner un 10. La RAM modifica el contenido de su posición 20 al valor 10 y muestra en su salida el valor de la posición 20 tras la escritura, es decir, 10.

RAMs : Modos de Lectura/Escritura – *No Change*

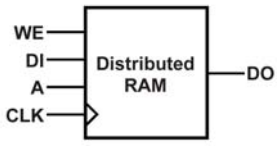
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity raminfr is
   port (clk : in std_logic;
         we : in std_logic;
         en : in std_logic;
         addr : in std_logic_vector(4 downto 0);
         di : in std_logic_vector(3 downto 0);
         do : out std_logic_vector(3 downto 0));
end raminfr;

architecture syn of raminfr is
   type ram_type is array (31 downto 0)
        of std_logic_vector (3 downto 0);
   signal RAM : ram_type;
begin
   process (clk)
   begin
        if clk'event and clk = '1' then
             if en = '1' then
                  if we = '1' then
                       RAM(conv_integer(addr)) <= di;
                  else
                       do <= RAM( conv_integer(addr));
                  end if;
             end if;
        end if;
   end process;
end syn;
```

En este modo, cuando se realiza una escritura sobre una posición de la memoria RAM, el valor presente en la salida de la memoria no se modifica.

P. Ej., supongamos que en la posición 20 está el valor 5, en la 21 está el valor 8, se accede en lectura a la posición 21 y en escritura a la posición 20 para poner un 10. Tras el acceso de lectura, la RAM muestra en su salida el valor 8 y tras el acceso de escritura modifica el contenido de su posición 20 al valor 10, pero su salida sigue mostrando el valor 8.

RAMs : Single-Port RAM con lectura asíncrona

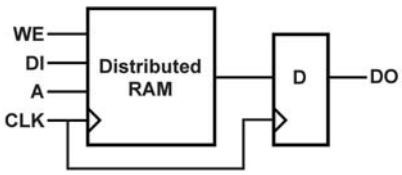
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity raminfr is
   port (clk : in std_logic;
         we : in std_logic;
         a : in std_logic_vector(4 downto 0);
         di : in std_logic_vector(3 downto 0);
         do : out std_logic_vector(3 downto 0));
end raminfr;

architecture syn of raminfr is
   type ram_type is array (31 downto 0)
     of std_logic_vector (3 downto 0);
   signal RAM : ram_type;
begin
   process (clk)
   begin
      if (clk'event and clk = '1') then
         if (we = '1') then
           RAM(conv_integer(a)) <= di;
         end if;
      end if;
   end process;
   do <= RAM(conv_integer(a));
end syn;
```



IO Pins	Description
clk	Positive-Edge Clock
we	Synchronous Write Enable (active High)
a	Read/Write Address
di	Data Input
do	Data Output

RAMs : Single-Port RAM con lectura síncrona 'falsa'

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity raminfr is
   port (clk : in std_logic;
         we : in std_logic;
         a : in std_logic_vector(4 downto 0);
         di : in std_logic_vector(3 downto 0);
         do : out std_logic_vector(3 downto 0));
end raminfr;

architecture syn of raminfr is
   type ram_type is array (31 downto 0)
         of std_logic_vector (3 downto 0);
   signal RAM : ram_type;
begin
   process (clk)
   begin
      if (clk'event and clk = '1') then
         if (we = '1') then
            RAM(conv_integer(a)) <= di;
         end if;
         do <= RAM(conv_integer(a));
      end if;
   end process;
end syn;
```

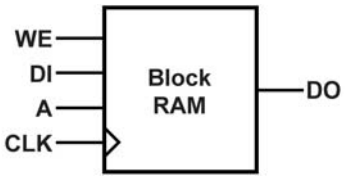


IO Pins	Description
clk	Positive-Edge Clock
we	Synchronous Write Enable (active High)
a	Read/Write Address
di	Data Input
do	Data Output

Técnicas de Modelado para Síntesis 103

RAMs : Single-Port RAM con lectura síncrona

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity raminfr is
   port (clk : in std_logic;
         we : in std_logic;
         a : in std_logic_vector(4 downto 0);
         di : in std_logic_vector(3 downto 0);
         do : out std_logic_vector(3 downto 0));
end raminfr;

architecture syn of raminfr is
   type ram_type is array (31 downto 0)
         of std_logic_vector (3 downto 0);
   signal RAM : ram_type;
   signal read_a : std_logic_vector(4 downto 0);
begin
   process (clk)
   begin
      if (clk'event and clk = '1') then
         if (we = '1') then
            RAM(conv_integer(a)) <= di;
         end if;
         read_a <= a;
      end if;
   end process;
   do <= RAM(conv_integer(read_a));
end syn;
```

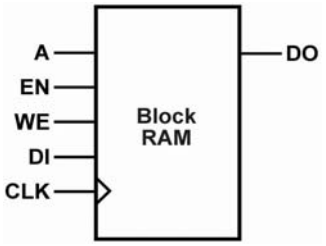


IO pins	Description
clk	Positive-Edge Clock
we	Synchronous Write Enable (active High)
a	Read/Write Address
di	Data Input
do	Data Output

Técnicas de Modelado para Síntesis 104

RAMs : Single-Port RAM con habilitación

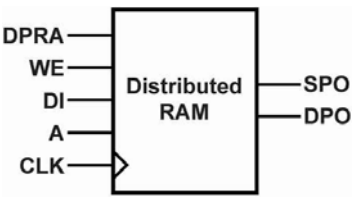
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity raminfr is
   port (clk, en, wen : in std_logic;
         a : in std_logic_vector(4 downto 0);
        di : in std_logic_vector(3 downto 0);
        do : out std_logic_vector(3 downto 0));
end raminfr;
architecture syn of raminfr is
   type ram_type is array (31 downto 0)
        of std_logic_vector (3 downto 0);
   signal RAM : ram_type;
   signal read_a : std_logic_vector(4 downto 0);
begin
   process (clk)
   begin
      if (clk'event and clk = '1') then
         if (en = '1') then
            if (we = '1') then
               RAM(conv_integer(a)) <= di;
            end if;
            read_a <= a;
         end if;
      end if;
   end process;
   do <= RAM(conv_integer(read_a));
end syn;
```



IO pins	Description
clk	Positive-Edge Clock
en	Global Enable
we	Synchronous Write Enable (active High)
a	Read/Write Address
di	Data Input
do	Data Output

RAMs : Dual-Port RAM con lectura asíncrona

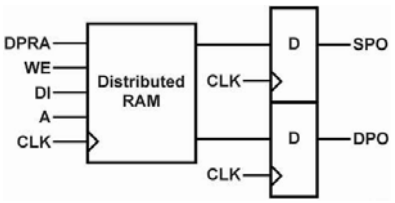
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity raminfr is
   port (clk, we : in std_logic;
         a : in std_logic_vector(4 downto 0);
         dpra : in std_logic_vector(4 downto 0);
         di : in std_logic_vector(3 downto 0);
         spo, dpo : out std_logic_vector(3 downto 0));
end raminfr;

architecture syn of raminfr is
   type ram_type is array (31 downto 0)
        of std_logic_vector (3 downto 0);
   signal RAM : ram_type;
begin
   process (clk)
   begin
      if (clk'event and clk = '1') then
         if (we = '1') then
            RAM(conv_integer(a)) <= di;
         end if;
      end if;
   end process;
   spo <= RAM(conv_integer(a));
   dpo <= RAM(conv_integer(dpra));
end syn;
```



IO pins	Description
clk	Positive-Edge Clock
we	Synchronous Write Enable (active High)
a	Write Address/Primary Read Address
dpra	Dual Read Address
di	Data Input
spo	Primary Output Port
dpo	Dual Output Port

RAMs : Dual-Port RAM con lectura síncrona 'falsa'

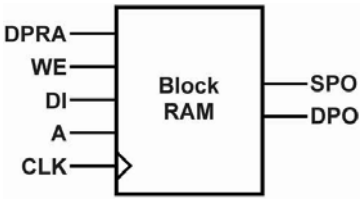
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity raminfr is
   port (clk, we : in std_logic;
            a : in std_logic_vector(4 downto 0);
         dpra : in std_logic_vector(4 downto 0);
         di : in std_logic_vector(3 downto 0);
         spo, dpo : out std_logic_vector(3 downto 0));
end raminfr;

architecture syn of raminfr is
   type ram_type is array (31 downto 0)
        of std_logic_vector (3 downto 0);
   signal RAM : ram_type;
begin
   process (clk)
   begin
      if (clk'event and clk = '1') then
         if (we = '1') then
            RAM(conv_integer(a)) <= di;
         end if;
         spo <= RAM(conv_integer(a));
         dpo <= RAM(conv_integer(dpra));
      end if;
   end process;
end syn;
```



IO Pins	Description
clk	Positive-Edge Clock
we	Synchronous Write Enable (active High)
a	Write Address/Primary Read Address
dpra	Dual Read Address
di	Data Input
spo	Primary Output Port
dpo	Dual Output Port

RAMs : Dual-Port RAM con lectura síncrona

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity raminfr is
   port (clk, we : in std_logic;
            a : in std_logic_vector(4 downto 0);
         dpra : in std_logic_vector(4 downto 0);
         di : in std_logic_vector(3 downto 0);
         spo, dpo : out std_logic_vector(3 downto 0));
end raminfr;
architecture syn of raminfr is
   type ram_type is array (31 downto 0)
        of std_logic_vector (3 downto 0);
   signal RAM : ram_type;
   signal read_a : std_logic_vector(4 downto 0);
   signal read_dpra : std_logic_vector(4 downto 0);
begin
   process (clk)
   begin
      if (clk'event and clk = '1') then
         if (we = '1') then
            RAM(conv_integer(a)) <= di;
         end if;
         read_a <= a;
         read_dpra <= dpra;
      end if;
   end process;
   spo <= RAM(conv_integer(read_a));
   dpo <= RAM(conv_integer(read_dpra));
end syn;
```



IO Pins	Description
clk	Positive-Edge Clock
we	Synchronous Write Enable (active High)
a	Write Address/Primary Read Address
dpra	Dual Read Address
di	Data Input
spo	Primary Output Port
dpo	Dual Output Port

RAMs : Dual-Port RAM con lectura síncrona (cont.)

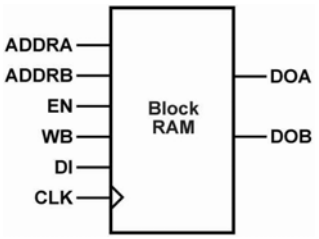
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity raminfr is
   port (clk1, clk2, we : in std_logic;
         add1, add2 : in std_logic_vector(4 downto 0);
            di : in std_logic_vector(3 downto 0);
         do1, do2 : out std_logic_vector(3 downto 0));
   end raminfr;

architecture syn of raminfr is
   type ram_type is array (31 downto 0)
         of std_logic_vector (3 downto 0);
   signal RAM : ram_type;
   signal read_add1 : std_logic_vector(4 downto 0);
   signal read_add2 : std_logic_vector(4 downto 0);
begin
   process (clk1)
   begin
      if (clk1'event and clk1 = '1') then
         if (we = '1') then
            RAM(conv_integer(add1)) <= di;
         end if;
         read_add1 <= add1;
      end if;
   end process;
   do1 <= RAM(conv_integer(read_add1));
   process (clk2)
   begin
      if (clk2'event and clk2 = '1') then
         read_add2 <= add2;
      end if;
   end process;
   do2 <= RAM(conv_integer(read_add2));
end syn;
```

Cuando ambos puertos de la RAM están sincronizados con distintas señales de reloj…

IO pins	Description
clk1	Positive-Edge Write/Primary Read Clock
clk2	Positive-Edge Dual Read Clock
we	Synchronous Write Enable (active High)
add1	Write/Primary Read Address
add2	Dual Read Address
di	Data Input
do1	Primary Output Port
do2	Dual Output Port

RAMs : Dual-Port RAM con habilitación común

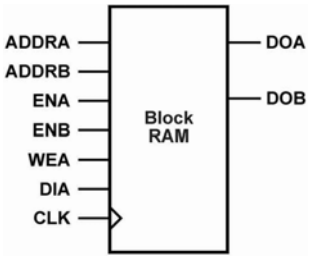
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity raminfr is
   port (clk, en, we : in std_logic;
addra: in std_logic_vector(4 downto 0);
        addrb: in std_logic_vector(4 downto 0);
        di : in std_logic_vector(3 downto 0);
        doa, dpb : out std_logic_vector(3 downto 0));
end raminfr;

architecture syn of raminfr is
   type ram_type is array (31 downto 0)
         of std_logic_vector (3 downto 0);
   signal RAM : ram_type;
   signal read_addra : std_logic_vector(4 downto 0);
   signal read_addrb : std_logic_vector(4 downto 0);
begin
   process (clk)
   begin
      if (clk'event and clk = '1') then
         if (en = '1') then
            if (we = '1') then
               RAM(conv_integer(addra)) <= di;
            end if;
            read_addra <= addra;
            read_addrb <= addrb;
         end if;
      end if;
   end process;
   doa <= RAM(conv_integer(read_addra));
   dob <= RAM(conv_integer(read_addrb));
end syn;
```



IO Pins	Description
clk	Positive-Edge Clock
en	Primary Global Enable (active High)
we	Primary Synchronous Write Enable (active High)
addra	Write Address/Primary Read Address
addrb	Dual Read Address
di	Primary Data Input
doa	Primary Output Port
dob	Dual Output Port

RAMs : Dual-Port RAM con habilitación independiente

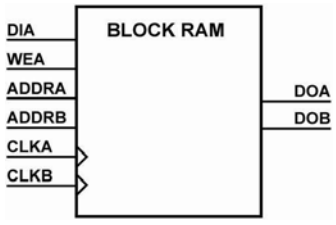
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity raminfr is
   port (clk, ena, enb, wea : in std_logic;
         addra, addrb: in std_logic_vector(4 downto 0);
          dia : in std_logic_vector(3 downto 0);
         doa, dob : out std_logic_vector(3 downto 0));
end raminfr;

architecture syn of raminfr is
   type ram_type is array (31 downto 0)
         of std_logic_vector (3 downto 0);
   signal RAM : ram_type;
   signal read_addra : std_logic_vector(4 downto 0);
   signal read_addrb : std_logic_vector(4 downto 0);
begin
   process (clk)
   begin
      if (clk'event and clk = '1') then
         if (ena = '1') then
            if (wea = '1') then
               RAM (conv_integer(addra)) <= dia;
            end if;
            read_addra <= addra;
         end if;
         if (enb = '1') then
            read_addrb <= addrb;
         end if;
      end if;
   end process;
   doa <= RAM(conv_integer(read_addra));
   dob <= RAM(conv_integer(read_addrb));
end syn;
```



IO Pins	Description
clk	Positive-Edge Clock
ena	Primary Global Enable (active High)
enb	Dual Global Enable (active High)
wea	Primary Synchronous Write Enable (active High)
addra	Write Address/Primary Read Address
addrb	Dual Read Address
dia	Primary Data Input
doa	Primary Output Port
dob	Dual Output Port

RAMs : Dual-Port RAM con reloj independiente

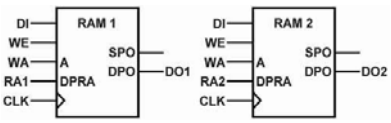
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity raminfr is
   port (clka, clkb, wea : in std_logic;
         addra, addrb: in std_logic_vector(4 downto 0);
          dia : in std_logic_vector(3 downto 0);
         doa, dob : out std_logic_vector(3 downto 0));
end raminfr;

architecture syn of raminfr is
   type ram_type is array (31 downto 0)
         of std_logic_vector (3 downto 0);
   signal RAM: ram_type;
   signal read_addra : std_logic_vector(4 downto 0);
   signal read_addrb : std_logic_vector(4 downto 0);
begin
   process (clka)
   begin
      if (clka'event and clka = '1') then
         if (wea = '1') then
            RAM(conv_integer(addra)) <= dia;
         end if;
         read_addra <= addra;
      end if;
   end process;
   process (clkb)
   begin
      if (clkb'event and clkb = '1') then
         read_addrb <= addrb;
      end if;
   end process;
   doa <= RAM(conv_integer(read_addra));
   dob <= RAM(conv_integer(read_addrb));
end syn;
```



IO Pins	Description
clka	Positive-Edge Clock
clkb	Positive-Edge Clock
wea	Primary Synchronous Write Enable (active High)
addra	Write Address/Primary Read Address
addrb	Dual Read Address
dia	Primary Data Input
doa	Primary Output Port
dob	Dual Output Port

RAMs : Multiple-Port RAM

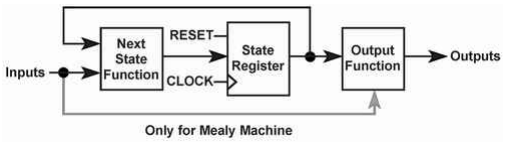
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity raminfr is
   port (clk : in std_logic;
         we : in std_logic;
         wa : in std_logic_vector(4 downto 0);
         ra1 : in std_logic_vector(4 downto 0);
         ra2 : in std_logic_vector(4 downto 0);
         di : in std_logic_vector(3 downto 0);
         do1 : out std_logic_vector(3 downto 0);
         do2 : out std_logic_vector(3 downto 0));
end raminfr;

architecture syn of raminfr is
   type ram_type is array (31 downto 0)
        of std_logic_vector (3 downto 0);
   signal RAM : ram_type;
Begin

   process (clk)
   begin
     if (clk'event and clk = '1') then
        if (we = '1') then
           RAM(conv_integer(wa)) <= di;
        end if;
     end if;
   end process;

   do1 <= RAM(conv_integer(ra1));
   do2 <= RAM(conv_integer(ra2));

end syn;
```



IO pins	Description
clk	Positive-Edge Clock
we	Synchronous Write Enable (active High)
wa	Write Address
ra1	Read Address of the first RAM
ra2	Read Address of the second RAM
di	Data Input
do1	First RAM Output Port
do2	Second RAM Output Port

Máquina de Estados Finitos (FSM)

- Sólo se reconocen las FSM síncronas.
- La codificación de los estados puede alterarse para obtener implementaciones optimizadas según diversos parámetros.
- Esquema típico…



- Típicamente se puede describir con 1, 2 ó 3 procesos.
- Directivas relacionadas…
  - FSM_extract … habilita o no el reconocimiento de FSMs.
  - FSM_encoding … selecciona el tipo de codificación.
  - FSM_fftype … selecciona el tipo de flip-flops para el registro de estado
  - ENUM_encoding… valores binarios para los estados enumerados.

FSM : Secciones

- Registros de estado…
  - *Deben* ser inicializados con un reset (asíncrono o no).
  - Su tipo suele definirse como una enumeración que incluye todos los posibles estados.
- Función de estado siguiente…
  - Puede definirse en el mismo procedimiento secuencial o en un procedimiento combinacional separado (y su lista sensible debe mostrar la señal de estado y todas las entradas).
- Salidas…
  - Las registradas deben ir dentro del mismo proceso secuencial.
  - Las no registradas en proceso combinacional o sentencias concurrentes.
- Entradas…
  - Señales internas asignadas dentro del proceso secuencial.

FSM : Técnicas de Codificación de Estados

- Auto…
  - La herramienta selecciona automáticamente el método de codificación óptimo para la máquina descrita.
- One-Hot…
  - Asocia un bit de código (1 flip-flop) a cada estado (001, 010, 100).
  - Apropiado en FPGAs con elevado número de flip-flops.
  - Optimiza la velocidad y reduce la disipación de potencia.
- Gray…
  - Garantiza el cambio de una única variable de estado cuando se produce una transición entre estados consecutivos.
  - Adecuado para FSM con muchos estados en línea y sin saltos.
  - Reduce la presencia de glitches.
  - Resultados óptimos cuando el registro de estado usa flip-flops tipo T.

FSM : Técnicas de Codificación de Estados (cont.)

- Compacto…
  - Minimiza el número de variables de estado (y flip-flops).
  - Adecuado para la optimización de área.
- Johnson…
  - Adecuado para muchos estados en línea y sin saltos.
- Secuencial…
  - Identifica trayectos lineales largos y asigna códigos binarios sucesivos a sus estados.
  - Minimiza las ecuaciones de *estado siguiente*.
- Usuario…
  - Usa la codificación indicada por el usuario en el código fuente.
  - Si se define un tipo enumerado se combina con la directiva *enum_encoding* para asignar valores binarios concretos a cada estado.

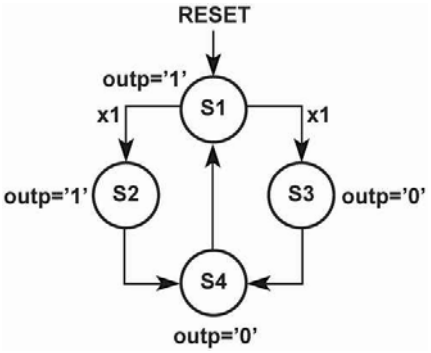
FSM : Fichero LOG

```
...
Synthesizing Unit <fsm>.
Related source file is state_machines_1.vhd.
Found finite state machine <FSM_0> for signal <state>.
-----------------------------------------------------------
| States           | 4                                    |
| Transitions      | 5                                    |
| Inputs           | 1                                    |
| Outputs          | 1                                    |
| Reset type       | asynchronous                         |
| Encoding         | automatic                            |
| State register   | D flip-flops                         |
-----------------------------------------------------------
...
Summary:
inferred 1 Finite State Machine(s).
...
Unit <fsm> synthesized.
=========================================================================
HDL Synthesis Report
Macro Statistics
# FSMs : 1
# Registers : 1
1-bit register : 1
=========================================================================
...
Optimizing FSM <FSM_0> with One-Hot encoding and D flip-flops. ...
...
```

Cuando la herramienta de síntesis reconoce la presencia de una FSM, incluye en el informe toda la información acerca de ésta, incluso la técnica de codificación seleccionada en caso de haber sido indicado el modo *Auto*.

FSM : Ejemplo a describir

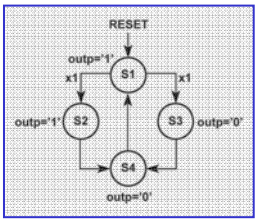
- Sea una máquina de Moore con reset asíncrono y las siguientes condiciones y grafo…
  - 4 estados
  - 5 transiciones
  - 1 entrada (x1)
  - 1 salida (outp)



FSM : Ejemplo con 1 proceso

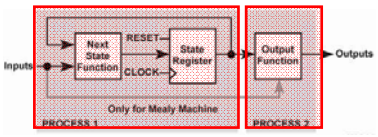
```
library IEEE;
use IEEE.std_logic_1164.all;

entity fsm is
   port (clk, reset, x1 : IN std_logic;
         outp : OUT std_logic);
end entity;

architecture beh1 of fsm is
   type state_type is (s1,s2,s3,s4);
   signal state: state_type;
Begin
   process (clk,reset)
   begin
      if (reset ='1') then
         state <= s1;
         outp <= '1';
      elsif (clk='1' and clk'event) then
         case state is
            when s1 => if x1='1' then state <= s2;
                       else state <= s3;
                       end if;
                       outp <= '1';
            when s2 => state <= s4; outp <= '1';
            when s3 => state <= s4; outp <= '0';
            when s4 => state <= s1; outp <= '0';
         end case;
      end if;
   end process;

end beh1;
```

Al ser la salida síncrona con el flanco de reloj, ya que se asigna dentro del proceso, la señal *outp* de salida se implementa con registro.

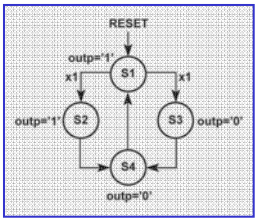


FSM : Ejemplo con 2 procesos

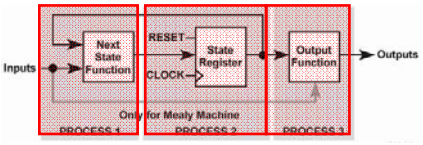
```
library IEEE;
use IEEE.std_logic_1164.all;

entity fsm is
   port (clk, reset, x1 : IN std_logic;
         outp : OUT std_logic);
end entity;

architecture beh1 of fsm is
   type state_type is (s1,s2,s3,s4);
   signal state: state_type;
begin
   process1: process (clk,reset)
   begin
      if (reset ='1') then state <=s1;
      elsif (clk='1' and clk'Event) then
         case state is
            when s1 => if x1='1' then state <= s2;
                       else state <= s3;
                       end if;
            when s2 => state <= s4;
            when s3 => state <= s4;
            when s4 => state <= s1;
         end case;
      end if;
   end process process1;
   process2 : process (state)
   begin
      case state is
         when s1 => outp <= '1';
         when s2 => outp <= '1';
         when s3 => outp <= '0';
         when s4 => outp <= '0';
      end case;
   end process process2;
end beh1;
```

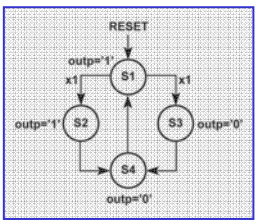


La descripción separa en un segundo proceso la función de salida de la máquina, eliminando así el registro de la salida.



FSM : Ejemplo con 3 procesos






```
library IEEE;
use IEEE.std_logic_1164.all;

entity fsm is
   port (clk, reset, x1 : IN std_logic;
         outp : OUT std_logic);
end entity;

architecture beh1 of fsm is
   type state_type is (s1,s2,s3,s4);
   signal state, next_state:
state_type;

begin
   process1: process (clk,reset)
   begin
      if (reset ='1') then
         state <=s1;
      elsif (clk='1' and clk'Event) then
         state <= next_state;
      end if;
   end process process1;
```

```
   process2 : process (state, x1)
   begin
      case state is
         when s1 => if x1='1' then
                       next_state <= s2;
                    else
                       next_state <= s3;
                    end if;
         when s2 => next_state <= s4;
         when s3 => next_state <= s4;
         when s4 => next_state <= s1;
      end case;
   end process process2;

   process3 : process (state)
   begin
      case state is
         when s1 => outp <= '1';
         when s2 => outp <= '1';
         when s3 => outp <= '0';
         when s4 => outp <= '0';
      end case;
   end process process3;

end beh1;
```

Soporte a componentes prediseñados (Black Box)

- El diseño puede contener módulos previamente optimizados, ya sea en la propia herramienta o en herramientas externas, generalmente en formato EDIF o NGC.
- Estos módulos se deben instanciar como componentes en el código VHDL. En estos casos la herramienta de síntesis no optimizará el netlist sino que lo incluirá y propagará tal cual en el diseño
- La información generada en el fichero log es la siguiente:

```
...
Analyzing Entity <black_b> (Architecture <archi>).
WARNING:Xst:766 - black_box_1.vhd (Line 15). Generating
        a Black Box
for component <my_block>.
Entity <black_b> analyzed. Unit <black_b> generated
....
```

- Ejemplo:

```
--
-- Black Box
--
library ieee;
use ieee.std_logic_1164.all;
entity black_box_1 is
  port(DI_1, DI_2 : in std_logic;
       DOUT : out std_logic);
end black_box_1;

architecture archi of black_box_1 is
 component my_block
   port (I1 : in std_logic;
         I2 : in std_logic;
         O : out std_logic);
   end component;
begin
   inst: my_block port map
           (I1=>DI_1,I2=>DI_2,O=>DOUT);
end archi;
```


Discusión

- Exponer las ideas y cuestiones de interés