Tema 5:
Verilog para diseñadores de VHDL

Hipólito Guzmán Miranda
Departamento de Ingeniería Electrónica
Universidad de Sevilla
hguzman@us.es

Contexto docente

BT02: Lenguajes de descripción hardware

- Tema 3: VHDL avanzado
- Tema 4: VHDL para procesado de señal
- Tema 5: Verilog para diseñadores de VHDL

Conocimientos previos requeridos:

- VHDL básico (SEC GITT / Complementos de Electrónica MUIT)
  - Diseño con dos procesos

Objetivos de aprendizaje

- Tener un primer contacto con el lenguaje Verilog
- Conocer las diferencias entre Verilog y VHDL
- Ser capaz de describir módulos sencillos en Verilog
- Ser capaz de comprender módulos descritos en Verilog por terceros

Repaso

VHDL para procesado de señal:

- Tipos de datos
- Punto fijo vs punto flotante
- Diseño de interfaces
- FIFOs y memorias

Contenido

- Motivación
- Introducción
- Versiones del estándar
- Tipado débil vs tipado fuerte
- Módulos
- Tipos de datos
- Asignaciones
- Operaciones aritméticas y lógicas
- Bloques always
- Bloques initial
- Assertions
- Instanciación de módulos
- Conclusiones
- Bibliografía

Contenido

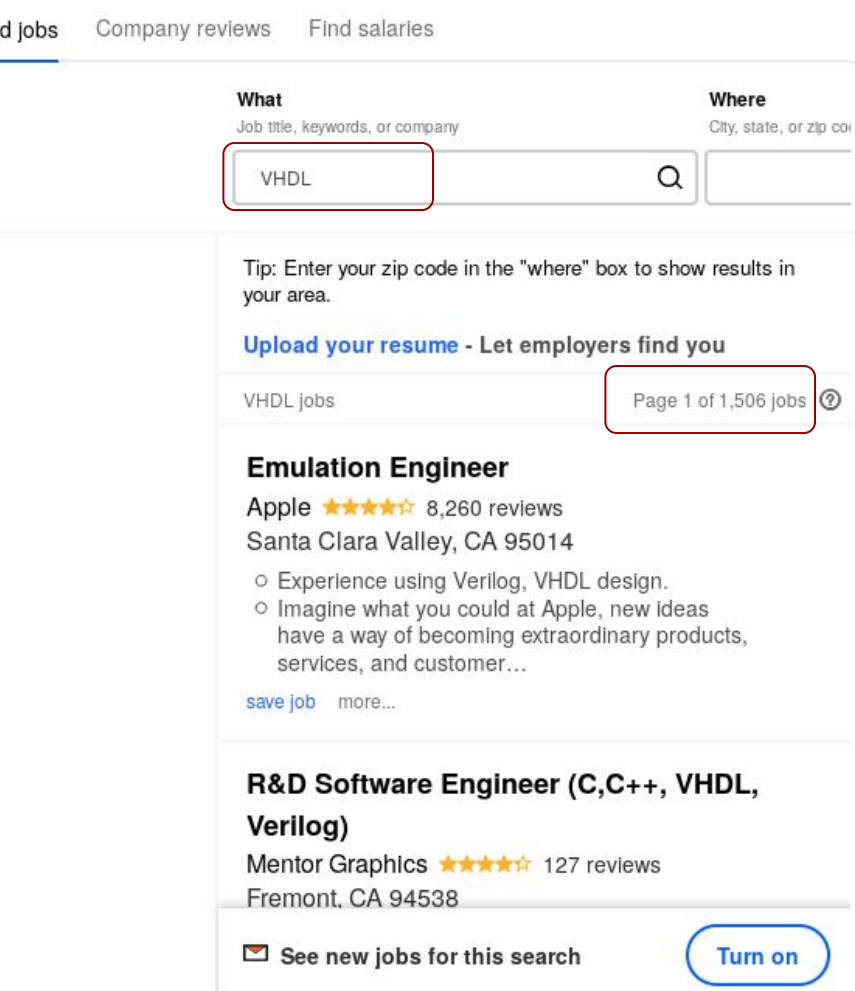
- Motivación
- Introducción
- Versiones del estándar
- Tipado débil vs tipado fuerte
- Módulos
- Tipos de datos
- Asignaciones
- Operaciones aritméticas y lógicas
- Bloques always
- Bloques initial
- Assertions
- Instanciación de módulos
- Conclusiones
- Bibliografía

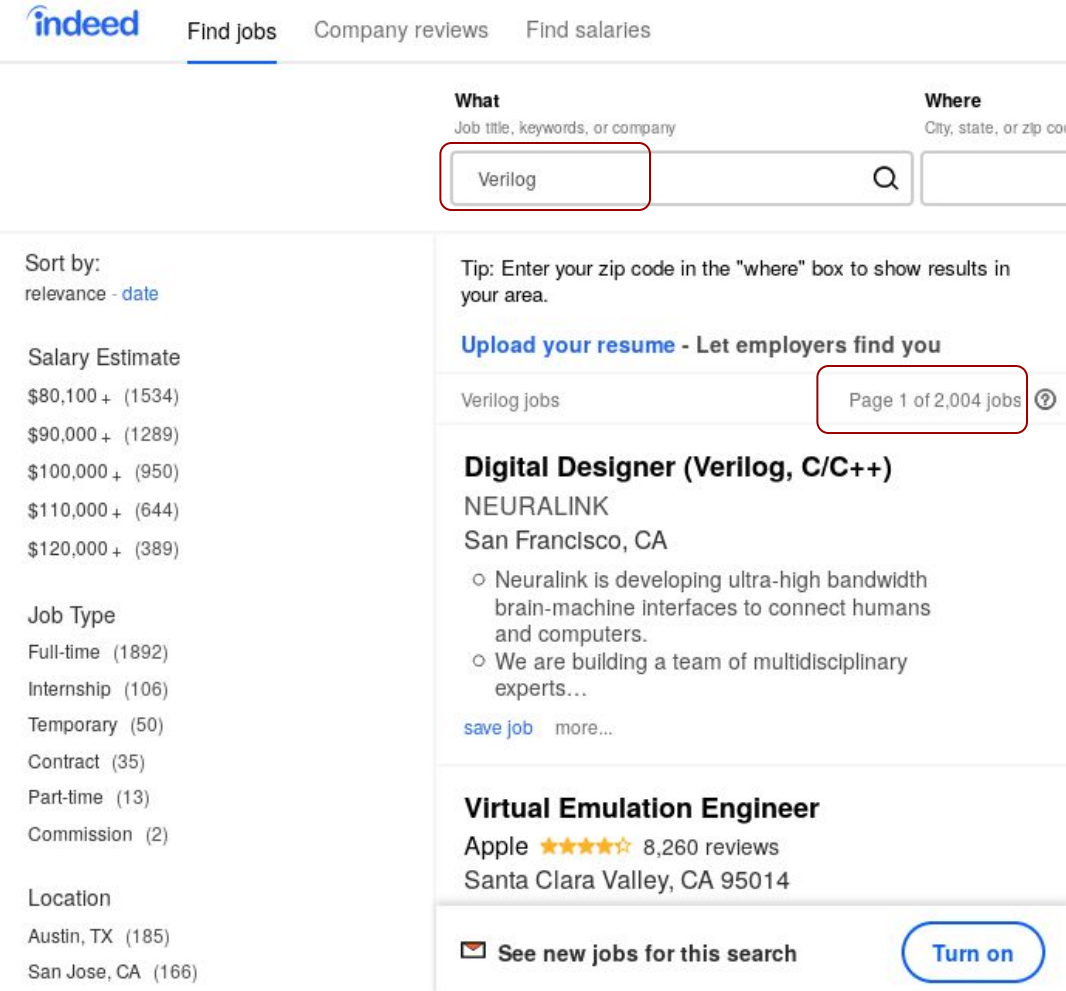
¿Por qué aprender Verilog?

- Muchas empresas utilizan Verilog en lugar de VHDL
  - Mejorar la empleabilidad (~2x)
  - VHDL domina en Europa, Verilog domina fuera
  - VHDL es preferido para aplicaciones críticas (aviónica, espacio, etc)
- Mucha verificación se hace utilizando Verilog o SystemVerilog (por ejemplo UVM)
  - También hay alternativas usando VHDL (UVVM)
  - Puede ser interesante hacer el diseño en un lenguaje y el testbench en otro

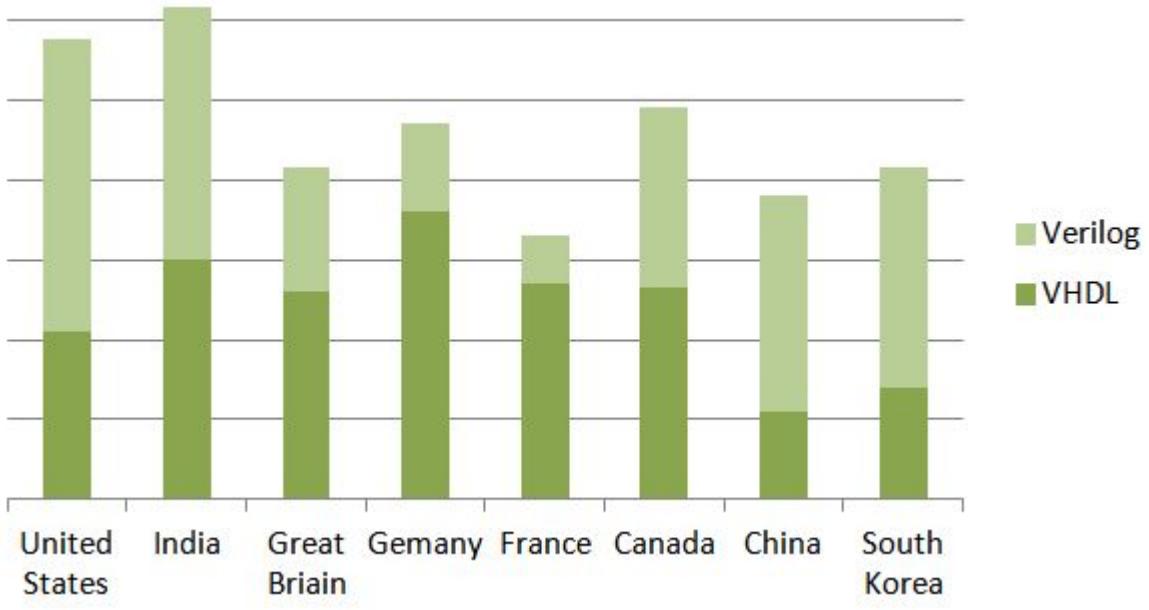
Motivación

¿Por qué aprender Verilog?





¿Por qué aprender Verilog?



Búsquedas de google Verilog vs VHDL (julio 2013 a julio 2014).

Fuente: nandland

¿Por qué aprender Verilog?

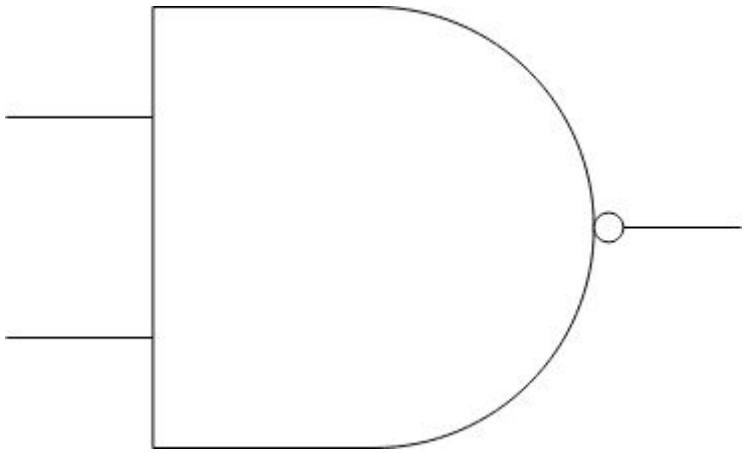
- Entender código generado por terceros
  - Otros diseñadores
  - Código auto-generado por herramientas
- Soportado por Yosys Open Synthesis Suite
  - VHDL soportado por el front-end propietario Verific (Yosys-vhdl disponible bajo licencia académica de Symbiotic EDA)
  - En el futuro, ghdlsynth-beta podría sustituir a este front-end
- Es factible aprenderlo si sabemos VHDL y nos enfocamos en las diferencias entre ambos

Contenido

Verilog es un HDL

- Al igual que con VHDL, no debemos perder de vista que las sentencias describen un hardware!

```
module nandexample
(
  input a,
  input b,
  output o
  );
assign o = ~(a & b);
endmodule
```



VHDL vs Verilog

- Tipado fuerte
- entity
- process
- sentencias concurrentes
- std_logic con 9 valores
- Un tipo de asignación
- No existen operadores de reducción

- Tipado débil
- module
- always
- assign
- Tipos multievaluados tienen cuatro valores (0,1,Z,X)
- Dos tipos de asignaciones (*blocking* y *nonblocking*)
- Operadores de reducción
- wire y reg
- Bloques initial
- begin y end opcionales
  - (si sólo hay una sentencia)

VHDL vs Verilog

- Sintaxis parecida a Ada
- Case-insensitive
- Comentarios de línea (--)
- Más errores encontrados en tiempo de síntesis / compilación

- Sintaxis parecida a C
- Case-sensitive
- Comentarios de línea (//) y bloque (/* … */)
- Directivas de preprocesador (ifdef, ifndef, …)
- Funciones de sistema ($dumpfile, $dumpvars, $finish, $display…)
- Código más compacto

En ambos se puede diseñar con uno o dos process/always, pero en Verilog está muy extendido el diseño con un único bloque always

VHDL vs Verilog:

contador (1 proceso)

VHDL

```vhdl
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity counter is
  port (clk, rst: in std_logic;
        Q: out unsigned(7 downto 0)
        );
end counter;

architecture behavioral of counter is
    signal count: unsigned(7 downto 0);
begin

    process(clk, rst)
    begin
        if (rst='1') then
            count <= (others=>'0');
        elsif (rising_edge(clk)) then
            count <= count + 1;
        end if;
    end process;

    Q <= count;

end behavioral;
```

Verilog

```verilog
module counter
(
  input clk, rst,
  output [7:0] Q
);

reg [7:0] count;

always @(posedge clk or posedge rst)
begin
        count <= count + 8'b1;
        if (rst) begin
          count <= 8'b0;
        end
end

assign Q = count;

endmodule
```

VHDL vs Verilog:

contador (2 procesos)

VHDL

```
[...]
  signal count:   unsigned(7 downto 0);
  signal p_count: unsigned(7 downto 0);

begin

    sinc: process(clk, rst)
    begin
        if (rst='1') then
            count <= (others=>'0');
        elsif (rising_edge(clk)) then
            count <= p_count;
        end if;
    end process;

    comb: process(count)
    begin
            p_count <= count + 1;
    end process;

    Q <= count;

end behavioral;
```

Verilog

```
[...]

reg [7:0] count;
reg [7:0] p_count;

always @(posedge clk or posedge rst )
begin
       count <= p_count;
       if (rst) begin
         count <= 8'b0;
       end
end

always @(count)
begin
        p_count <= count + 8'b1;
end

assign Q = count;

endmodule
```

Estructura de un fichero Verilog

```
module <nombremodulo>
(
  input <puerto1>, <puerto2>,
  input [N-1:0] <puerto3>,
  output [7:0] <puerto4>
);

reg [M-1:0] <reg1>;
reg <reg2>, <reg3>;
wire <wire1>;

always @(...)
begin
  <sentencias...>
end

assign <destino> = <expresión>;

endmodule
```

- reg y wire pueden ser declarados en cualquier punto de la “arquitectura”
  - Es buena práctica declararlos antes de usarlos!
- Todas las sentencias always y assign funcionan en paralelo
  - Al igual que process y sentencias concurrentes
- También puede haber instanciación de módulos
  - La veremos más adelante

Contenido

- Motivación
- Introducción
- Versiones del estándar
- Tipado débil vs tipado fuerte
- Módulos
- Tipos de datos
- Asignaciones
- Operaciones aritméticas y lógicas
- Bloques always
- Bloques initial
- Assertions
- Instanciación de módulos
- Conclusiones
- Bibliografía

Versiones del estándar

- Inicios (1980s)
  - Lenguaje propietario de Gateway/Cadence
- Verilog-95
  - Primera versión estandarizada
- Verilog 2001
  - Mejoras de usabilidad, sentencia generate
  - Versión con mayor soporte de los vendors
- Verilog 2005
  - Correciones menores
- SystemVerilog
  - Superset de Verilog 2005 que incluye múltiples funcionalidades para verificación y permite programación orientada a objetos

Contenido

- Motivación
- Introducción
- Versiones del estándar
- Tipado débil vs tipado fuerte
- Módulos
- Tipos de datos
- Asignaciones
- Operaciones aritméticas y lógicas
- Bloques always
- Bloques initial
- Assertions
- Instanciación de módulos
- Conclusiones
- Bibliografía

Tipado débil vs tipado fuerte

¿Qué implica?

a es vector de 8 bits

b es vector de 12 bits

VHDL (tipado fuerte)

`a <= b;` → ERROR

Verilog (tipado débil)

`a <= b;` → permitido

Perdemos datos: ¡buena suerte encontrando el error!

Contenido

- Motivación
- Introducción
- Versiones del estándar
- Tipado débil vs tipado fuerte
- Módulos
- Tipos de datos
- Asignaciones
- Operaciones aritméticas y lógicas
- Bloques always
- Bloques initial
- Assertions
- Instanciación de módulos
- Conclusiones
- Bibliografía

Unidad básica de funcionalidad

VHDL: entity

```
entity counter is
  port (clk, rst: in std_logic;
        Q: out unsigned(7 downto 0)
        );
end counter;
```

Verilog: module

```
module counter
(
  input clk, rst,
  output [7:0] Q
);
```

Ambos tienen puertos de entrada y salida, que se definen en una sección diferente a donde se define el funcionamiento

Contenido

- Motivación
- Introducción
- Versiones del estándar
- Tipado débil vs tipado fuerte
- Módulos
- Tipos de datos
- Asignaciones
- Operaciones aritméticas y lógicas
- Bloques always
- Bloques initial
- Assertions
- Instanciación de módulos
- Conclusiones
- Bibliografía

Conjuntos de valores

Los datos predefinidos en verilog tienen dos (0, 1) o cuatro (0, 1, X, Z) valores posibles:

- 0 : cero lógico fuerte
- 1 : uno lógico fuerte
- X : valor lógico desconocido
- Z : alta impedancia

Al igual que en VHDL, X y Z no se suelen asignar en síntesis (salvo el caso particular de describir puertos triestado)

Conjuntos de valores

4-state (0, 1, X, Z):

- reg: vector definido por el usuario
- wire: vector definido por el usuario
- integer: 32-bit, signed
- time: 64-bit, unsigned
- logic: vector definido por el usuario (SV)

2-state (0, 1) (SystemVerilog):

- shortint: 16-bit, signed
- int: 32-bit, signed
- longint: 64-bit, signed
- byte: 8-bit, signed
- bit: vector definido por el usuario

Modificadores:
signed
unsigned

Valores literales

[tamaño]'[signed][radix]valor

- tamaño en bits
- signed: `s` o `S` indica que el dato es con signo
- radix es la base:
  - `b` o `B` para binario
  - `o` u `O` para octal
  - `h` o `H` para hexadecimal
  - `d` o `D` para decimal

Los campos entre corchetes son opcionales

Valores literales

[tamaño]'[signed][radix]valor

Ejemplos:

```
2'b00, 32'hDEADBEEF, 4'b01XZ,
32'd10, 32'd0, 8'b0
```

reg vs wire

reg

- Representa un objeto que almacena un valor entre una asignación y la siguiente
- Puede leerse en un bloque secuencial (always, initial) y en un bloque assign
- Puede asignarse en un bloque secuencial
- No puede asignarse en un bloque assign (sentencia concurrente)
- No tiene por qué implicar la síntesis de un registro

wire

- Representa un cable que conecta puertas o módulos
- Puede leerse en un bloque secuencial (always, initial) y en un bloque assign
- No puede asignarse en un bloque secuencial
- Puede asignarse en un bloque assign (sentencia concurrente) o como salida de un módulo instanciado

reg vs wire

Ambos tipos pueden tomar 4 valores por bit (0, 1, X, Z)

La distinción realmente es en cómo se asignan (reg se asignan en always e initial, wire se asignan en assign y por submódulos instanciados)

¿Y… cómo asignamos los puertos?

- Puertos tipo `input` e `inout` se tratan como wire
- Puertos tipo `output` por defecto se tratan como wire, pero se pueden configurar como reg

```
output [7:0] Q          (wire)
output reg [7:0] Q      (reg)
```

Contenido

- Motivación
- Introducción
- Versiones del estándar
- Tipado débil vs tipado fuerte
- Módulos
- Tipos de datos
- Asignaciones
- Operaciones aritméticas y lógicas
- Bloques always
- Bloques initial
- Assertions
- Instanciación de módulos
- Conclusiones
- Bibliografía

Verilog tiene 2 tipos de asignaciones

Blocking =

Bloquean la ejecución de las siguientes sentencias en un bloque secuencial (ej: always, initial) hasta que no termina la asignación

En síntesis, normalmente las utilizaremos en los assign

Non-blocking <=

No bloquean la ejecución (se realizan en paralelo)

No pueden utilizarse en los assign

En síntesis, normalmente las utilizaremos en los bloques secuenciales (initial, always)

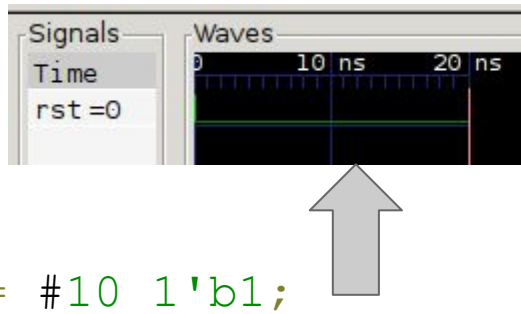
Ejemplo blocking vs non-blocking

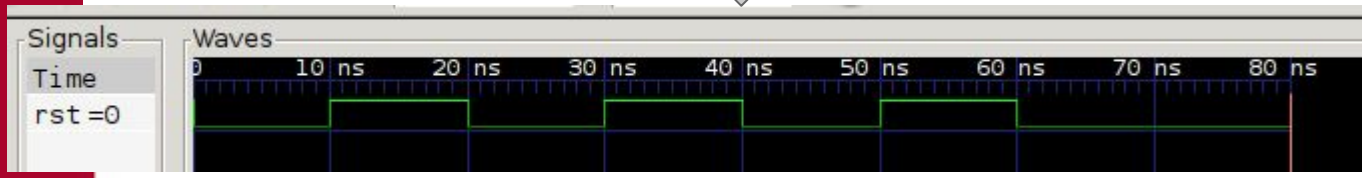
```
reg rst = 0;

initial begin
        rst = #10 1'b1;
        rst = #10 1'b0;
        rst = #10 1'b1;
        rst = #10 1'b0;
        rst = #10 1'b1;
        rst = #10 1'b0;
        # 20 $finish;
end
```

```
reg rst = 0;

initial begin
        rst <= #10 1'b1;
        rst <= #10 1'b0;
        rst <= #10 1'b1;
        rst <= #10 1'b0;
        rst <= #10 1'b1;
        rst <= #10 1'b0;
        # 20 $finish;
end
```





Contenido

- Motivación
- Introducción
- Versiones del estándar
- Tipado débil vs tipado fuerte
- Módulos
- Tipos de datos
- Asignaciones
- Operaciones aritméticas y lógicas
- Bloques always
- Bloques initial
- Assertions
- Instanciación de módulos
- Conclusiones
- Bibliografía

Operaciones

- Not: `~`
- And binaria: `&`
- Nand binaria: `~&`
- And “lógica”: `&&`
- Or binaria: `|`
- Nor binaria: `~|`
- Or “lógica”: `||`
- Xor: `^`
- Xnor: `~^`
- Operaciones aritméticas: `+, -, *, /`

Entre vectores:

- `vect_c <= vect_a & vect_b;`
- `vect_c <= vect_a | vect_b;`

Reducción:

(*and* de todos los bits de un vector, *or* de todos los bits, etc)

- `bit_b <= &vect_a;`
- `bit_b <= |vect_a;`

Operadores de comparación

- Igual: `==`
- Distinto: `!=`
- Mayor que: `>`
- Mayor or igual que: `>=`
- Menor que: `>`
- Menor o igual que: `<=`

- “Case equality”: `===`
- “Case inequality”: `!==`
  - Permiten comparar con X y Z, en simulación
  - Por ejemplo, `a === 1'bz`
  - No deben usarse en síntesis

Contenido

- Motivación
- Introducción
- Versiones del estándar
- Tipado débil vs tipado fuerte
- Módulos
- Tipos de datos
- Asignaciones
- Operaciones aritméticas y lógicas
- Bloques always
- Bloques initial
- Assertions
- Instanciación de módulos
- Conclusiones
- Bibliografía

Equivalentes a `process`

```
always @(evento)
begin
        <sentencias>
end


always @(evento)
        <sentencia>
```

- El evento es equivalente a la lista de sensibilidad
- En procesos síncronos:
  - `posedge clk` (si no tiene reset o tiene reset síncrono)
  - `posedge clk or posedge rst` (si tiene reset asíncrono)
- En procesos combinacionales:
  - `a or b or c` (lista de sensibilidad)
  - `*` (equivalente a '`all`' en VHDL-2008)

Contenido

- Motivación
- Introducción
- Versiones del estándar
- Tipado débil vs tipado fuerte
- Módulos
- Tipos de datos
- Asignaciones
- Operaciones aritméticas y lógicas
- Bloques always
- Bloques initial
- Assertions
- Instanciación de módulos
- Conclusiones
- Bibliografía

Bloques secuenciales que se ejecutan una única vez

Equivalente a un always que sólo se ejecuta una vez, en tiempo 0

Dos usos:

- En simulación, para generar estímulos
- Para asignar los INIT values de registros en FPGA
  - El valor de INIT de un biestable (el que toma cuando se programa el bistream) no tiene por qué coincidir con el de reset

Bloques initial

Dos usos:

- En simulación, para generar estímulos

```
/* Pulse reset and stop
simulation */
reg rst = 0;

initial begin
        # 5 rst = 1;
        # 20 rst = 0;
        # 2600 $finish;
end
```

- Para asignar los INIT values de registros en FPGA

```
initial begin
        count  <= 8'b0;
        count2 <= 8'b0;
end
```

Bloques secuenciales que se ejecutan una única vez

Si tenemos varios bloques initial en un módulo, ¿cuál se ejecuta antes?

- Pueden darse condiciones de carrera entre ellos

Ej: un bloque initial que asigna un valor a `A`, en paralelo a otro bloque initial que asigna `B <= ~A;`

¿Qué sentencias podemos utilizar para control de flujo?

if

```
if (condicion) begin
  <sentencias>
end
else begin
  <sentencias>
end
```

Verilog no tiene `elsif`, pero se puede usar `if` dentro de un `else`. `begin` y `end` son opcionales si sólo hay una sentencia

case

```
case (expresión)
  valor1 : <sentencia1>;
  valor2, valor3 : <sentencia2>;
  valor4 : begin
             <sentencias>
           end
  default : <sentencia3>;
endcase
```

¿Qué sentencias podemos utilizar para control de flujo?

if

```
if (enable) begin
  p_count <= count + 1;
end
else begin
  p_count <= count;
end
```

Verilog no tiene `elsif`, pero se puede usar `if` dentro de un `else`. `begin` y `end` son opcionales si sólo hay una sentencia

case

```
case (data)
  2'b00 : <sentencia1>;
  2'b01, 2'b10 : <sentencia2>;
  2'b11 : begin
            <sentencias>
          end
  default : <sentencia3>;
endcase
```

Contenido

- Motivación
- Introducción
- Versiones del estándar
- Tipado débil vs tipado fuerte
- Módulos
- Tipos de datos
- Asignaciones
- Operaciones aritméticas y lógicas
- Bloques always
- Bloques initial
- Assertions
- Instanciación de módulos
- Conclusiones
- Bibliografía

Equivalentes a las de VHDL

```
assert (condition);
```

```
assert (actual == expected);
assert (self_destruct == 1'b0);
assert (state != ERROR_STATE);
```

Utilizaremos los Assertions de Verilog para verificación formal.

SVA (SystemVerilog Assertions) tiene expresiones más potentes aún (implicación, propiedades, secuencias…), pero pocas herramientas las soportan (ej: QuestaSim).

Contenido

- Motivación
- Introducción
- Versiones del estándar
- Tipado débil vs tipado fuerte
- Módulos
- Tipos de datos
- Asignaciones
- Operaciones aritméticas y lógicas
- Bloques always
- Bloques initial
- Assertions
- Instanciación de módulos
- Conclusiones
- Bibliografía

Instanciación de módulos

No es necesario declararlos como componente

```
nombremodulo nombreinstancia (
   .puertomodulo    ( wire_del_top ),
   .puertomodulo2   ( wire_del_top2 ),
   [...]
   .puertomoduloN   ( wire_del_topN )
);

counter uut (
   .clk    ( clk ),
   .rst    ( rst ),
   .Q      ( value )
);
```

Contenido

- Motivación
- Introducción
- Versiones del estándar
- Tipado débil vs tipado fuerte
- Módulos
- Tipos de datos
- Asignaciones
- Operaciones aritméticas y lógicas
- Bloques always
- Bloques initial
- Assertions
- Instanciación de módulos
- Conclusiones
- Bibliografía

Conclusiones

- Verilog es un lenguaje de descripción hardware y permite hacer lo mismo que permite VHDL
- El tipado débil de Verilog hace que sea más fácil equivocarse pero evita tener que usar funciones de conversión
  - Cada diseñador tendrá sus preferencias
- Es importante prestar atención a las diferencias entre ambos lenguajes!
  - Cuidado con reg/wire, blocking/nonbloking, initial...
- El lenguaje tiene más funcionalidades de las que hemos visto aquí
  - Packed vs unpacked arrays, ifdef, $dumpvars, $readmem, ...

Recomendaciones

Que la primera línea de todos vuestros ficheros Verilog sea:

```
`default_nettype none
```

Para asegurar que las herramientas os dan un error si intentáis usar un reg/wire que no esté declarado (si no, suele ser complicado de depurar)

Si diseñáis con Verilog, existen ‘linters’ de Verilog que analizan vuestro código buscando errores comunes

Contenido

- Motivación
- Introducción
- Versiones del estándar
- Tipado débil vs tipado fuerte
- Módulos
- Tipos de datos
- Asignaciones
- Operaciones aritméticas y lógicas
- Bloques always
- Bloques initial
- Assertions
- Instanciación de módulos
- Conclusiones
- Bibliografía

Bibliografía

- Peter Wilson, *Design Recipes for FPGAs using Verilog and VHDL.* Elsevier, 2016
- Stuart Sutherland, Don Mills, *Verilog and SystemVerilog Gotchas: 101 Common Coding Errors and How to Avoid Them.* Springer, 2007
- Nazeih Botros, *HDL with Digital Design: VHDL and Verilog.* Mercury Learning and Information, 2015

Resultados de aprendizaje

- ¿Podrías enumerar las diferencias principales entre VHDL y Verilog?
- Dada una lista de palabras clave de Verilog, indicar a qué palabras clave de VHDL equivalen
- Qué son reg y wire y cuándo se pueden asignar
- Diferencias entre =, <=, ==, ===
- ¿Cuándo podemos prescindir de begin y end en sentencias Verilog?