Tema 6:

Especificación usando Verilog

Diseño automático de sistemas

José Manuel Mendías Cuadros
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

Verilog
evolución del estándar

- Verilog es un HDL nacido en 1984 de propiedad de Cadence Design System que pasó a dominio público en 1990:
  - Verilog 1364-1995: Primer estándar.
  - Verilog 1364-2001: Moderniza el lenguaje para hacerlo comparable a VHDL.
  - Verilog 1364-2005: Cambios menores del lenguaje.
  - SystemVerilog 1800-2005: Extensión de Verilog'05 para facilitar la verificación OO.
  - SytemVerilog 1800-2009: Se unen los estándares de Verilog y SystemVerilog.
  - SystemVerilog 1800-2012: Clarifica y corrige SystemVerilog'09 e incorpora mejoras para facilitar la simulación.
- El subconjunto de síntesis soportado por ISE está basado en Verilog'95/01
  - No soporta SystemVerilog.
- Nuevas herramientas como Vivado soportan también SystemVerilog'12
  - Pero con un subconjunto de síntesis análogo al soportado en VHLD'93 / Verilog'01 .

Tipos de datos

- Solo existe un tipo de dato relevante en síntesis:
  - Es atómico y puede tomar 4 valores: **0**, **1**, **Z** (alta impedancia) y **X** (don't care)
- Todo dato puede declararse como:
  - Atómico: para representar un bit
  - Vectorial: para representar un vector de bits que codifica un entero con o sin signo.
  - Array multidimensional
- Todo dato debe pertenecer a una clase:
  - Interconexión: transporta valores (usado para conectar partes de un diseño).
    - **wire**, **wor**, **wand**, **tri**…
  - Variable: transporta y almacena valores (en simulación equivale a **signal** de VHDL).
    - **reg**, **integer**, **real**…
  - En síntesis solo son relevantes las clases: **wire**, **reg** e **integer**.

```
wire a;
reg  b;
```

```
wire [7:0] c;
reg  [7:0] d;
wire signed [7:0] e;
reg  signed [7:0] f;
integer i;
```

```
wire [7:0] a [0:255];
reg  [7:0] a [0:255];
```

Tipos de datos

- En síntesis, la selección de la clase se hace principalmente por sintáxis
  - Los puertos de entrada de un módulo son solo y por defecto de clase **`wire`**
  - Los puertos de salida de un módulo pueden ser de clases **`wire`** (por defecto) o **`reg`**
  - Los puertos de salida de un modulo se instancian con señales de clase **`wire`**
  - Los puertos de entrada de un módulo se instancian con señales de clases **`wire`** o **`reg`**
  - En un bloque **`always`** solo pueden asignarse señales de clase **`reg`**
  - En una sentencia **`assign`** solo pueden asignarse señales de clase **`wire`**
  - Para especificar lógica combinacional pueden usarse señales de clases **`wire`** o **`reg`**
  - Para especificar lógica secuencial solo pueden usarse señales de clase **`reg`**

- Las señales, con independencia de su clase pueden asignarse:
  - No bloqueante (<=): en simulación equivale a (<=) de VHDL
    - Debe usarse para especificar lógica secuencial, solo en bloques **`always`**
  - Bloqueante (=): en simulación equivale a (:=) de VHDL
    - Debe usarse para especificar lógica combinacional, en bloques **`always`** o **`assign`**

Tipos de datos

- Verilog dispone de un conjunto de operadores similar a VHDL

VHDL	Verilog
not	!
and	&&
or	||

Lógicos

VHDL	Verilog
+	+
-	-
*	*
/	/
mod	%

Aritméticos

VHDL	Verilog
=	==
/=	!=
<	<
<=	<=
>	>
>=	>=

Relacionales

VHDL	Verilog
not	~
and	&
or	|
nand	~&
nor	~|
xor	^
xnor	~ ^

Bit a bit

VHDL	Verilog
a & b	{a , b}
sll	<<
srl	>>
sla	<<<
sra	>>>

Varios

- Los literales numéricos pueden indican codificación, tamaño y base:

58 'h3a	8'b00111010 8'd58 8'h3a 8'o72

-58 -'h3a	-8'b00111010 -8'd58 -8'h3a -8'o72	8'sb11000110 8'sd198 8'shc6 8'so306

Equivalencias básicas

declaración e instanciación

VHDL'93

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

entity adder is
  generic( N : integer := 8 );
  port(
    x : in  std_logic_vector(n-1 downto 0);
    y : in  std_logic_vector(n-1 downto 0);
    z : out std_logic_vector(n-1 downto 0) );
end adder;

architecture syn of adder is
begin

  ...

end syn;
```

Verilog'01

```verilog
module adder
#( parameter N = 8 )
(
  input  [n-1:0] x,
  input  [n-1:0] y,
  output [n-1:0] s
);

  ...

endmodule
```

```verilog
module adder (x, y, s);
  parameter N = 8;

  input  [n-1:0] x;
  input  [n-1:0] y;
  output [n-1:0] s;

  ...

endmodule
```

```vhdl
myAdder: adder
  generic map ( N => 8 )
  port map ( x => a, y => b, z => s );
```

```verilog
adder #( .N(8) ) myAdder
  ( .x(a), .y(b), .z(s) );
```

Equivalencias básicas

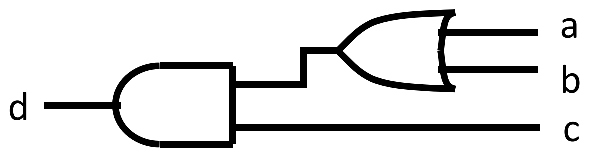
- El bloque **assign** equivale a la asignación concurrente de VHDL
  - En Verilog solo existen la incondicional y la de selección simple

VHDL'93

```
d <= ( a or b ) and c;
```

Verilog'01

```
assign d = ( a | b ) & c;
```

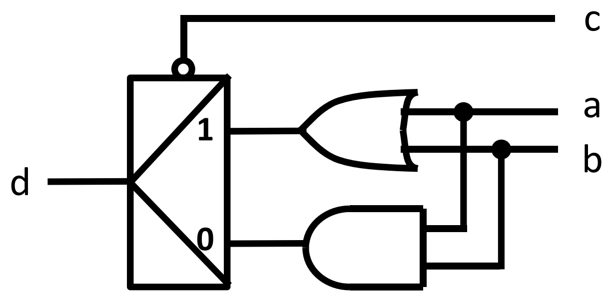


VHDL'93

```
d <= (a or b) when c='0' else (a and b);
```

Verilog'01

```
assign d = !c ? (a | b) : (a & b);
```



Equivalencias básicas

- El bloque **always** equivale al **process** VHDL
  - Puede usarse tanto para especificar lógica combinacional como secuencial

VHDL'93

```
process (a, b)
begin
  c <= a + b;
end process;
```

Verilog'01

```
always @(a or b)
  c = a + b;
```

```
always @(a, b)
  c = a + b;
```

```
always @(*)
  c = a + b;
```

VHDL'93

```
process (gate, d)
begin
  if gate='1' then
    q <= d;
  end if;
end process;
```

Verilog'01

```
always @(gate or d)
  if (gate)
    q <= d;
```

```
always @(gate, d)
  if (gate)
    q <= d;
```

```
always @(*)
  if (gate)
    q <= d;
```

VHDL'93

```
process (rst_n, clk)
begin
  if rst_n='0' then
    q <= '0';
  elsif rising_edge(clk) then
    q <= d;
  end if;
end process;
```

Verilog'01

```
always @(negedge rst_n or posedge clk)
 if (!rst_n)
    q <= 0;
 else
    q <= d;
```

```
always @(negedge rst_n, posedge clk)
 if (!rst_n)
    q <= 0;
 else
    q <= d;
```

Lógica secuencial

FSM (i)

```
stateGen:
process (currentState, input)
begin
  nextState <= currentState;
  case currentState is
    when ... =>
      if (input ...) then
        nextState <= ...;
      elsif (input ...) then
        ...
      else
        ...
      end if;
    ...
  end case;
end process;
```

VHDL'93

```
always @(*)
begin
  nextState = currentState;
  case (currentState)
    ... :
      if (input ...)
        nextState = ...;
      else if (input ...)
        ...
      else
        ...
    ...
  endcase
end
```

Verilog'01

```
state:
process (rst_n, clk)
begin
  if rst_n='0' then
    currentState <= ...;
  elsif risign_edge(clk) then
    currentState <= nextState;
  end if;
end process;
```

VHDL'93

```
always @(negedge rst_n, posedge clk)
  if (!rst_n)
    currentState <= ...;
  else
    currentState <= nextState;
```

Verilog'01

Lógica secuencial

FSM (ii)

```
mealyGen:
process (currentState, input)
begin
  mealyOutput <= ...;
  case currentState is
    when ... =>
      if (input ...) then
        mealyOutput <= ...;
      elsif (input ...) then
        ...
      else
        ...
      end if;
    ...
  end case;
end process;
```

VHDL'93

```
always @(*)
begin
  mealyOutput = ...;
  case (currentState)
    ... :
      if (input ...)
        mealyOutput = ...;
      else if (input ...)
        ...
      else
        ...
    ...
  endcase
end
```

Verilog'01

```
mooreGen:
process (currentState)
begin
  mooreOutput <= ...;
  case currentState is
    when ... =>
      mooreOutput <= ...;
    ...
  end case;
end process;
```

VHDL'93

```
always @(*)
begin
  mooreOutput = ...;
  case (currentState)
    ... :
      mooreOutput = ...;
    ...
  endcase
end
```

Verilog'01

J.M. Mendías 2017

Ejemplos

sumador genérico (i)

VHDL'93

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

entity adder is
  generic( n : integer := 8 );
  port(
    x : in  std_logic_vector(n-1 downto 0);
    y : in  std_logic_vector(n-1 downto 0);
    s : out std_logic_vector(n-1 downto 0) );
end adder;

library ieee;
use ieee.numeric_std.all;

architecture syn of adder is
begin

  s <= std_logic_vector(unsigned(x) + unsigned(y));

end syn;
```

Verilog'01

```
module adder
#( parameter n = 8 )
(
  input  [n-1:0] x, y,
  output [n-1:0] s
);

  assign s = x + y;

endmodule
```

Ejemplos

sumador genérico (ii)

VHDL'93

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

entity adder is
  generic( n : integer := 8 );
  port(
    x    : in  std_logic_vector(n-1 downto 0);
    y    : in  std_logic_vector(n-1 downto 0);
    cin  : in  std_logic;
    s    : out std_logic_vector(n-1 downto 0);
    cout : out std_logic );
end adder;

architecture syn of adder is
  signal cinAux : unsigned(0 downto 0);
  signal temp   : unsigned(n downto 0);
begin
  cinAux <= (0=>cin);
  temp <= to_unsigned( to_integer(unsigned(x))
    + to_integer(unsigned(y))
    + to_integer(cinAux), n+1);
  s <= std_logic_vector(temp(n-1 downto 0));
  cout <= std_logic(temp(n));
end syn;
```

Verilog'01

```
module adder
#( parameter n = 8 )
(
  input [n-1:0]  x, y,
  input          cin,
  output [n-1:0] s,
  output         cout
);

  assign
    {cout, s} = x + y + cin;

endmodule
```

J.M. Mendías
2017

Ejemplos

registro genérico

VHDL'93

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

entity reg is
  generic( n : integer := 8 );
  port(
    rst_n, clk, ld : in  std_logic;
    din   : in  std_logic_vector(n-1 downto 0);
    dout  : out std_logic_vector(n-1 downto 0)
  );
end reg;

architecture syn of reg is
begin
 process (rst_n, clk)
   begin
    if rst_n='0' then
      dout <= (others=>'0');
    elsif rising_edge(clk) then
      if ld='1' then
        dout <= din;
      end if;
    end if;
  end process;
end syn;
```

Verilog'01

```
module register
#( parameter N = 8 )
(
  input rst_n, clk, ld,
  input [N-1:0] din,
  output reg [N-1:0] dout
);

  always @(negedge rst_n, posedge clk)
    if (!rst_n)
      dout <= 0;
    else if (ld)
      dout <= din;

endmodule
```

Ejemplos

registro genérico con salida en alta impedancia

VHDL'93

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

entity triStateReg is
  generic( n : integer := 8 );
  port(
    rst_n, clk, ld, en : in  std_logic;
    din                : in  std_logic_vector( n-1 downto 0 );
    dout               : out std_logic_vector( n-1 downto 0 ) );
end triStateReg;

architecture syn of triStateReg is
  signal cs : std_logic_vector(n-1 downto 0);
begin

  process (rst_n, clk)
  begin
    if rst_n='0' then
      cs <= (others=>'0');
    elsif rising_edge(clk) then
      if ld='1' then
        cs <= din;
      end if;
    end if;
  end process;

  dout <= cs when en='1'
          else (others=>'Z');

end syn;
```

Verilog'01

```verilog
module triStateReg
#( parameter N = 8 )
(
  input rst_n, clk, ld, en,
  input  [N-1:0] din,
  output [N-1:0] dout
);
  reg [N-1:0] cs;

  always @(negedge rst_n, posedge clk)
    if (!rst_n)
      cs <= 0;
    else if (ld)
      cs <= din;

  assign dout = en ? cs : 'bZ;

endmodule
```

Ejemplos

registro de desplazamiento genérico

VHDL'93

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

entity shiftReg is
  generic( N : integer := 8 );
  port(
    rst_n, clk, sht, din : in std_logic;
    dout : out std_logic_vector(n-1 downto 0) );
end shiftReg;

architecture syn of shiftReg is
  signal cs : std_logic_vector(n-1 downto 0);
begin
 process (rst_n, clk)
  begin
    dout <= cs;
    if rst_n='0' then
      cs <= (others=>'0');
    elsif rising_edge(clk) then
      if sht='1' then
        for i in cs'high downto cs'low+1
        loop
          cs(i) <= cs(i-1);
        end loop;
        cs(0) <= din;
      end if;
    end if;
  end process;
end syn;
```

Verilog'01

```verilog
module shiftReg
#( parameter N = 8 )
(
  input rst_n, clk, sht, din,
  output reg [N-1:0] dout
);

  integer i;

  always @(negedge rst_n, posedge clk)
    if (!rst_n)
      dout <= 0;
    else if (sht) begin
      for( i=N-1; i<1; i=i-1 )
        dout[i] <= dout[i-1];
      dout[0] <= din;
    end
endmodule
```

Ejemplos

contador ascendente genérico (i)

VHDL'93

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

entity counter is
  generic( N : integer := 8 );
  port(
    rst_n, clk, ld, ce :  in std_logic;
    din   :  in std_logic_vector(n-1 downto 0);
    tc    : out std_logic;
    dout  : out std_logic_vector(n-1 downto 0))
end counter;

architecture syn of counter is
  signal cs: unsigned(n-1 downto 0);
begin
  process (rst_n, clk)
  begin
    dout <= std_logic_vector(cs);
    if rst_n='0' then
      cs <= (others=>'0');
    elsif rising_edge(clk) then
      if ld='1' then
        cs <= unsigned(din);
      elsif ce='1' then
        cs <= cs + 1;
    end if;
  end process;
  tc <= '1' when ce='1' and cs=2**N-1 else '0';
end syn;
```

Verilog'01

```
module modCounter
#( parameter N = 8 )
(
  input rst_n, clk, ld, ce,
  input [N-1:0] din,
  output tc,
  output reg [N-1:0] dout
);

  always @(negedge rst_n, posedge clk)
    if (!rst_n)
      dout <= 0;
    else if (ld)
      dout <= din;
    else if (ce)
      dout <= dout + 1;

  assign
    tc = (ce && dout==2**N-1) ? 1 : 0;

endmodule
```

J.M. Mendías
2017

Ejemplos

contador ascendente modulo-máximo genérico

```vhdl
library ieee; use ieee.std_logic_1164.all; use ieee.numeric_std.all;
entity counter is
  generic(
    N : integer := 4; MAX : integer := 10 );
  port(
    rst_n, clk, ce :  in std_logic;
    dout           : out std_logic_vector(n-1 downto 0));
end counter;

architecture syn of counter is
  signal cs : unsigned(n-1 downto 0);
begin
  process (rst_n, clk, ce, cs)
  begin
    dout <= std_logic_vector(cs);
    if rst_n='0' then
      cs <= (others=>'0');
    elsif rising_edge(clk) then
      if ce='1' then
        if cs=MAX then
          cs <= (others=>'0');
        else
          cs <= cs + 1;
        end if;
      end if;
    end if;
  end process;
end syn;
```

VHDL'93

```verilog
module counter
#( parameter N = 4, MAX = 10 )
(
  input rst_n, clk, ce,
  output reg [N-1:0] dout
);

  always @(negedge rst_n, posedge clk)
    if (!rst_n)
      dout <= 0;
    else if (ce)
      if (dout==MAX)
        dout <= 0;
      else
        dout <= dout + 1;

endmodule
```

Verilog'01

Ejemplos

RAM

VHDL'93

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

entity ram is
  port(
    clk : in std_logic;
    we  : in std_logic;
    a   : in std_logic_vector(7 downto 0);
    di  : in std_logic_vector(15 downto 0);
    do  : out std_logic_vector(15 downto 0) );
end ram;

architecture syn of ram is
  type ramType is array (0 to 255)
    of std_logic_vector(15 downto 0);
  signal ram : ramType;
begin
process (clk)
  begin
    if rising_edge(clk) then
      if we='1' then
        ram( to_integer( unsigned( address ) ) ) <= di;
      end if;
    end if;
  end process;
  do <= ram( to_integer( unsigned( address ) ) );
end;
```

Verilog'01

```
module ram
(
  input clk,
  input we,
  input [7:0] a,
  input [15:0] di,
  output [15:0] do
);
  reg [15:0] ram [255:0];

  always @(posedge clk)
    if (we)
      ram[a] <= di;

  assign do = ram[a];

endmodule
```

Ejemplos

FSM temporizadas: interfaz

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

entity controlSemaforo is
  generic( cRojo, cAmarillo, cVerde : natural );
  port(
    rst, clk : in std_logic;
    boton : in std_logic;
    coche, peaton : out std_logic_vector(2 downto 0)
  );
end controlSemaforo;

architecture syn of controlSemaforo is

  constant sRojo     : std_logic_vector(2 downto 0) := "100";
  constant sAmarillo : std_logic_vector(2 downto 0) := "010";
  constant sVerde    : std_logic_vector(2 downto 0) := "001";

  signal cargar, fin : std_logic;
  signal numCiclos, ciclos : natural;

  type estados_t is ( pVsR, pVsRCond, pAsR, pRsV, pRsA );
  signal estado : estados_t;

begin
  ...
end syn;
```

VHDL'93

```verilog
module controlSemaforos
#( parameter
  cRojo, cAmarillo, cVerde
)
(
  input rst, clk, boton,
  output reg [2:0] coche,
  output reg [2:0] peaton
);
  localparam cVpR     = 3'd0;
  localparam cVpRCond = 3'd1;
  localparam cApR     = 3'd2;
  localparam cRpV     = 3'd3;
  localparam cRpR     = 3'd4;

  localparam sRojo     = 3'b100;
  localparam sAmarillo = 3'b010;
  localparam sVerde    = 3'b001;

  reg cargar;
  wire fin;
  integer numCiclos, ciclos;
  reg [2:0] estado;

  ...

endmodule
```

Verilog'01

Ejemplos

FSM temporizadas: temporizador

VHDL'93

```
temporizador:
process (rst, clk)
begin
  if numCiclos=0 then
    fin <= '1';
  else
    fin <= '0';
  end if;
  if rst='1' then
    numCiclos <= cVerde;
  elsif rising_edge(clk) then
    if cargar='1' then
      numCiclos <= ciclos;
    elsif fin='0' then
      numCiclos <= numCiclos - 1;
    end if;
  end if;
end process;
```

Verilog'01

```
always @(posedge rst, posedge clk)
  if (rst)
    numCiclos <= cVerde;
  else if (cargar)
    numCiclos <= ciclos;
  else if (!fin)
    numCiclos <= numCiclos – 1;

assign fin = numCiclos ? 0 : 1;
```

Ejemplos

FSM temporizadas: controlador (i)

VHDL'93

```
controlador:
process (estado, boton, fin)
begin
  cargar <= '0'; ciclos <= cVerde;
  case estado is
    when cVpR =>
      coche <= sVerde; peaton <= sRojo;
    when cVpRCond =>
      coche <= sVerde; peaton <= sRojo;
      if boton='1' then
        cargar <= '1'; ciclos <= cAmarillo;
      end if;
    when cApR =>
      ...
    when cRpV =>
      ...
    when cRpR =>
      coche <= sRojo; peaton <= sRojo;
      if fin='1' then
        cargar <= '1'; ciclos <= cVerde;
      end if;
  end case;
  ...
```

Verilog'01

```
always @(*)
begin
  cargar = 0; ciclos = cVerde;
  case (csC)
    cVpR :
      begin
        coche = sVerde; peaton = sRojo;
      end
    cVpRCond :
      begin
        coche = sVerde; peaton = sRojo;
        if (boton) begin
          cargar = 1; ciclos = cAmarillo;
        end
      end
    cApR :
      ...
    cRpV :
      ...
    default :
      begin
        coche = sRojo; peaton = sRojo;
        if (fin) begin
          cargar = 1; ciclos = cVerde;
        end
      end
   endcase
end
```

Ejemplos

FSM temporizadas: controlador (ii)

```
process (rst, clk)
begin
 if rst = '1' then
    estado <= cVpR;
  elsif rising_edge(clk) then
    case estado is
      when cVpR =>
        if fin='1' then
          estado <= cVpRCond;
        end if;
      when cVpRCond =>
        if boton='1' then
          estado <= cApR;
        end if;
      when cApR =>
        if fin='1' then
          estado <= cRpV;
        end if;
      when cRpV =>
        if fin='1' then
          estado <= cRpR;
        end if;
      when cRpR =>
        if fin='1' then
          estado <= cVpR;
        end if;
    end case;
  end if;
end process;
```

VHDL'93

```
always @(posedge rst, posedge clk)
  if (rst)
    estado <= cVpR;
  else
    case (estado)
      cVpR :
        if (fin)
          estado <= cVpRCond;
      cVpRCond :
        if (boton)
          estado <= cApR;
      cApR :
        if (fin)
          estado <= cRpV;
      cRpV :
        if (fin)
          estado <= cRpR;
      default :
        if (fin)
          estado <= cVpR;
   endcase
```

Verilog'01

Ejemplos

FSMD con datapath implícito (i)

VHDL'93

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity multiplier is
  port
  (
    rst_n :  in std_logic;
    clk   :  in std_logic;
    start :  in std_logic;
    done  : out std_logic;
    a     :  in std_logic_vector(31 downto 0);
    b     :  in std_logic_vector(31 downto 0);
    r     : out std_logic_vector(63 downto 0)
  );
end multiplier;
architecture syn of multiplier is
  signal ra : unsigned(63 downto 0);
  signal rb : unsigned(31 downto 0);
  signal rr : unsigned(63 downto 0);
  signal rc : unsigned(4 downto 0);
  type states_t is ( s0, s1, s2, s3, s4 );
  signal cs : states_t;
begin
  ...
end syn;
```

Verilog'01

```
module multiplier
(
  input  rst_n, clk, start,
  output done,
  input [31:0] a, b,
  output [63:0] r
);
  localparam s0 = 3'd0;
  localparam s1 = 3'd1;
  localparam s2 = 3'd2;
  localparam s3 = 3'd3;
  localparam s4 = 3'd4;

  reg [63:0] ra;
  reg [31:0] rb;
  reg [63:0] rr;
  reg [4:0]  rc;

  reg [2:0] cs;

  ...

endmodule
```

Ejemplos

FSMD con datapath implícito (ii)

```
done <= '1' when cs=s0 else '0';

r <= std_logic_vector(rr);

process (rst_n, clk, cs)
begin
  if rst_n='0' then
    ra <= (others => '0');
    rb <= (others => '0');
    rr <= (others => '0');
    rc <= (others => '0');
    cs <= s0;
  elsif rising_edge(clk) then
    case cs is
      when s0 =>
        ra <= resize( unsigned(a), 64);
        rb <= unsigned(b);
        if start='1' then
          cs <= s1;
        end if;
      when s1 =>
        rr <= (others => '0');
        if rb(0)='1' then
          cs <= s3;
        else
          cs <= s2;
        end if;
...
```

VHDL'93

```
assign done = cs==s0 ? 1 : 0;

assign r = rr;

always @(negedge rst_n, posedge clk)
  if (!rst_n) begin
    ra <= 0;
    rb <= 0;
    rr <= 0;
    rc <= 0;
    cs <= s0;
  end else
    case (cs)
      s0 :
        begin
          ra <= a;
          rb <= b;
          if (start)
            cs <= s1;
        end
      s1 :
        begin
          rr <= 0;
          if (rb[0])
            cs <= s3;
          else
            cs <= s2;
        end
...
```

Verilog'01

Ejemplos

FSMD con datapath implícito (iii)

VHDL'93

```
...
      when s2 =>
        ra <= ra(62 downto 0) & '0';
        rb <= '0' & rb(31 downto 1);
        rc <= rc + 1;
        cs <= s4;
      when s3 =>
        rr <= ra + rr;
        ra <= ra(62 downto 0) & '0';
        rb <= '0' & rb(31 downto 1);
        rc <= rc + 1;
        cs <= s4;
      when s4 =>
        if rc=0 then
          cs <= s0;
        else
          if rb(0)='1' then
            cs <= s3;
          else
            cs <= s2;
          end if;
        end if;
    end case;
  end if;
end process;
```

Verilog'01

```
...
      s2 :
        begin
          ra <= ra << 1;
          rb <= rb >> 1;
          rc <= rc + 1;
          cs <= s4;
        end
      s3 :
        begin
          rr <= ra + rr;
          ra <= ra << 1;
          rb <= rb >> 1;
          rc <= rc + 1;
          cs <= s4;
        end
      default :
        if (rc==0)
          cs <= s0;
        else if (rb[0])
          cs <= s3;
        else
          cs <= s2;
    endcase
```

Acerca de *Creative Commons*

- Licencia CC (Creative Commons)
  - Ofrece algunos derechos a terceras personas bajo ciertas condiciones. Este documento tiene establecidas las siguientes:

Reconocimiento (*Attribution*):
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.

No comercial (*Non commercial*):
La explotación de la obra queda limitada a usos no comerciales.

Compartir igual (*Share alike*):
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Más información: https://creativecommons.org/licenses/by-nc-sa/4.0/