# DISEÑO HARDWARE CON VHDL

- ✓ **Lenguaje de Descripción Hardware**
- ✓ **Descripción Hardware en VHDL**
- ✓ **Estructura de un Modelo VHDL**
- ✓ **Elementos Básicos de VHDL**
- ✓ **Ejemplo: Máquina de Estados Finita**

Javier Resano, ***GAZ: Grupo de Arquitectura de Computadores***


Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza

# Lenguaje de Descripción Hardware

- ¿Qué es HDL?
  - Lenguajes creados para el diseño de circuitos:
    - » Nivel de puerta (gate level).
    - » Nivel de comportamiento (behavioural level).
  - La estructura del lenguaje sugiere el diseño hardware.
- ¿Por qué usar HDL?
  - El diseño es mucho más rápido.
  - Permite simular antes de la implementación física.
  - Las herramientas de diseño se encargan de los pasos automatizables.

*No es lo mismo describir **hardware** que programar **software***

# Lenguaje de Descripción Hardware

- VHDL
  - VHSIC (Gobierno de EE.UU. 1980).
  - IEEE VHDL'87.
  - *www.vhdl.org*
- Verilog
  - Desarrollado por CADENCE.
  - IEEE 1364.
  - *www.eda.org*
- A partir de C:
  - Handel C
  - System C
  - C/C++ (síntesis de alto nivel)

# Lenguaje de Descripción Hardware

- VHDL
  - Descripción de la estructura del circuito
    - Descomposición en sub-circuitos
    - Interconexión de sub-circuitos
    - Comportamiento
    - Estructural
  - Permite la especificación de la funcionalidad de un circuito utilizando formas familiares en los lenguajes de programación
  - Permite la simulación del circuito antes de su fabricación
    - Testear y comparar alternativas sin necesidad de prototipos hardware

# Lenguaje de Descripción Hardware

- Verilog:
  - Circuitos Integrados + Sistemas Analógicos + Señales *(comportamiento o estructural)*
  - Analógico:
    - Bloques Analógicos
    - Operadores Integrales y Derivada
    - Transformada de Laplace
  - Señales:
    - Módulo de señales Analógico/Digital
    - Reglas para convolución de señales
    - Conversores A-D y D-A

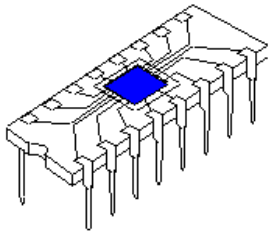
# Lenguajes de Descripción Hardware

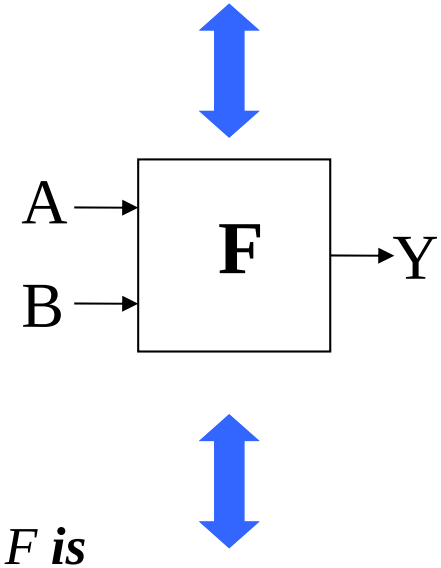
- ¿ Por qué VHDL?
  - Lenguaje **más utilizado** para el diseño de CI
  - Gran cantidad de **compiladores** de eficacia probada para su implementación en FPGA o sobre silicio. (Synopsis, Xilinx, Mentor Graphics, Cadence, hp ...)
  - Verilog necesita descripciones a un nivel más bajo. Es más utilizado para diseñar circuitos de tratamiento de señales. Es el más utilizado en América.

# Descripción Hardware en VHDL

- Descripción de un Sistema Digital:

Un sistema digital está descrito por sus entradas y sus salidas, donde las salidas dependen de las entradas






```
entity F is
    port (A, B: in bit; Y out bit);
end F
```

# Estructura de un Modelo VHDL

Los modelos VHDL están formado por 2 partes:

- ***Entity***
  - Define externamente al circuito o subcircuito
  - Nombre y  número de puertos, tipos de datos de entrada y salida
  - Tienes toda la información necesaria para conectar tu circuito a otros circuitos
- ***Architecture***
  - Define internamente el circuito
  - Señales internas, funciones, procedimientos, constantes …
- Puede haber varias arquitecturas para una entidad

# Elementos Básicos de VHDL

```
entity F is
    port (A, B: in bit; Y out bit);
end F

architecture circuito of F is
signal D, E: bit_vector(1 downto 0);
signal H: bit;
begin
(…)
end architecture circuito;
```

Entradas y salidas

Señales internas

funcionalidad

# Elementos Básicos de VHDL

```
-- comment
ENTITY name IS
        GENERIC (parameter list);
        PORT (io list);
END name;
```

parameter list:
parameter_i : type_name; …

*retardo: time; error: boolean*

io list:
port_i : io_type type_name; …

io_type
**IN | OUT | INOUT |BUFFER**

*entradaA: in bit_vector(7 downto 0);*
*salidaC: out std_logic;*

# Elementos Básicos de VHDL

## type_name

**BIT** {'0','1'} – **STD_LOGIC**
**BIT_VECTOR** (range) – **STD_LOGIC_VECTOR** (range)
**BOOLEAN** {TRUE,FALSE}
**CHARACTER** {ascii}
**STRING** {ascii}
**SEVERITY_LEVEL** {NOTE, WARNING, ERROR, FALURE}
**INTEGER** [range]
**NATURAL** [range]
**POSITIVE** [range]
**REAL** [range]
**TIME**

## range

n_min **TO** n_max  n_max **DOWNTO** n_min

# Elementos Básicos de VHDL

- Señales:
  - Representan elementos de memoria o conexiones
  - Los puertos de una entidad
  - En la arquitectura antes del BEGIN, lo cual nos permite realizar conexiones entre diferentes módulos

  señal **<=** valor
- Variables
  - Se utilizan como índices (instrucciones de bucle o modelar componentes)
  - Las variables NO representan conexiones o estados de memoria

  variable **:=** valor

**No las vamos a usar**

# Elementos Básicos de VHDL

- Descripcion funcional:

```
ARCHTECTURE arch_name OF entity_name IS
signal_i;
BEGIN
-- sentencias concurrentes
   AND, OR, NOT, NAND, XOR, +, - …
   WAIT
-- procedural (sequencial ) assertions
PROCESS (sensitivity list)
variable_j;
BEGIN
-- asignaciones, sentencias condicionales, bucles
END PROCESS;
END arch_name;
```

# Elementos Básicos de VHDL

- Dos métodos para representar la funcionalidad:
  - Processes:
    - Conjunto de sentencias secuenciales. El orden de las sentencias puede afectar al resultado.
  - Sentencias concurrentes:
    - Sentencias e invocaciones a procesos que se ejecutan en paralelo. Da igual en qué orden se escriban.

# Elementos Básicos de VHDL

```
Architecture ver1 of test is
begin
      c <= a and b;
      z <= c when e =‘1’ else ‘Z’;
seq: process (a, b)
      begin
          if a = b then
              f <= a or b;
          end if;
          If a = “1000” then
              d <= f;
          end if;
      end process;
end ver1;
```

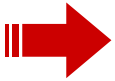
Sentencias concurrentes

Sentencias secuenciales:
Sólo dentro de los procesos

# Elementos Básicos de VHDL

- Operadores basicos:

*abs*
**, /, mod, rem*
+ (sig.), - (sig)
+, -, &
and, or, nand, nor …



```
y<=(x1 and x2) or d(0);
y(1) <= x1 and not x2;
```

```
y: bit_vector (1 downto 0);
   y <= x1&x2;
```

# Elementos Básicos de VHDL

- Sentencias Concurrentes (I):

```
signal_name <= valor_1 when condición1 else
               valor_2 when condición2 else
                  ...
               valor_i when condicióni else
               otro_valor;
```

```
salida <= "00" when entrada = "0001" else
          "01" when entrada = "0010" else
          "10" when entrada = "0100" else
          "11";
```

# Elementos Básicos de VHDL

- Sentencias Concurrentes (II):

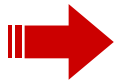
```
with identificador select
signal_name <= valor_1 when valor_identificador1,
               valor_2 when valor_identificador2,
                    ...
               valor_i when valor_identificadori,
               otro_valor when others;
```

```
with entrada select
salida <= "00" when  "0001",
          "01" when  "0010",
          "10" when "0100",
          "11" when others;
```

# Elementos Básicos de VHDL

- Sentencias secuenciales condicionales (I):

```
if cond_I then
...      --sequential statements
elsif cond-n then
...      --sequential statements
else   ... --sequential statements
end if;
```



```
if a = b then
    c <= a or b;
elsif a<b then
    c <= b;
else
    c <= “0000”;
end if;
```

# Elementos Básicos de VHDL

- Sentencias secuenciales condicionales (II):

```
case expression is
    when choice_l =>   ...  --seq. statements;
    when choice_n =>   ... --seq. statements;
    when others =>  ... --seq. statements;
end case;
```

```
case RGB is
   when “111” => r <= 0;
   when “100” => r <= 1;
   when “110” => r <= 1;
   when others => r <= 0;
end case;
```

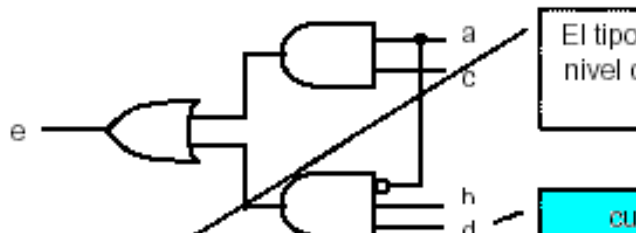
# Síntesis de Sentencias Secuenciales

- Simulación de procesos:
  - Un proceso se simula si cambia alguna señal de su lista de sensibilidad
  - Si un proceso representa lógica combinacional su lista debería incluir todas las entradas
  - Si un proceso representa lógica secuencial (un registro o un contador), su lista sólo debe incluir la señal de reloj y el reset o clear en caso de que sea asíncrono

# Síntesis de Sentencias Secuenciales

- Implementación de procesos:

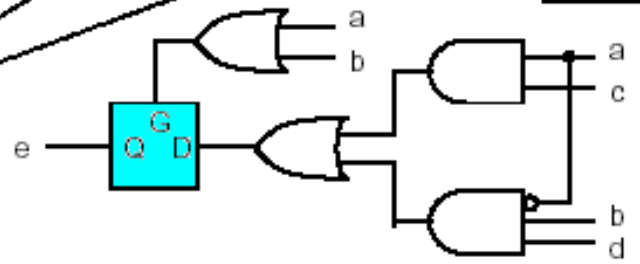
```
process( a, b, c, d )
begin
  if (a = '1') then
    e <= c;
  elsif (b = '1') then
    e <= d;
  else
    e <= '0';
  end if;
end process;
```




El tipo de expresiones utilizadas (de nivel o de flanco) determinan el tipo de HW secuencial

cualquier sentencia *if* o *case* incompeltamente especificada implica la necesidad de almacenar los valores no modificados

```
process( a, b, c, d )
begin
  if (a = '1') then
    e <= c;
  elsif (b = '1' ) then
    e <= d;
  end if;
end process;
```



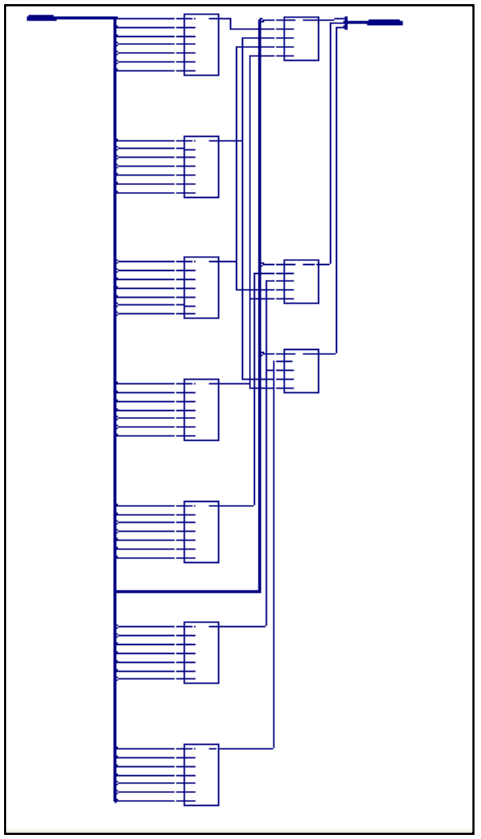
# Síntesis de Sentencias Secuenciales

- Implementación de procesos: registro de 32 bit

```
Reg: process (clk)
  begin
    if (clk'event and clk = '1') then
      if (reset = '1') then
        Dout <= "00000000000000000000000000000000";
      else
        if (load='1') then
            Dout <= Din;
        end if;
      end if;
    end if;
  end process;
```

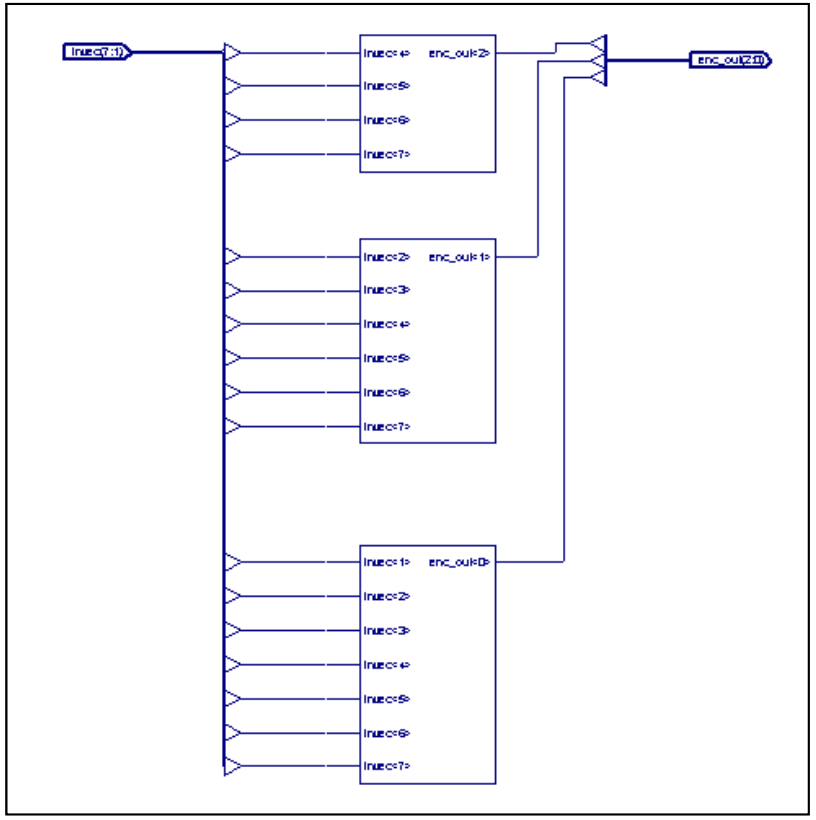
# Elementos Básicos de VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity encoder is
        port (invec: in std_logic_vector(7 downto 0); enc_out: out std_logic_vector(2 downto 0));
end encoder;
architecture rtl of encoder is
begin
  encode: process (invec)
 begin
    case invec is
      when "00000001" =>        enc_out <= "000";
      when "00000010" =>        enc_out <= "001";
      when "00000100" =>        enc_out <= "010";
      when "00001000" =>        enc_out <= "011";
      when "00010000" =>        enc_out <= "100";
      when "00100000" =>        enc_out <= "101";
      when "01000000" =>        enc_out <= "110";
      when "10000000" =>        enc_out <= "111";
      when others =>        enc_out <= "000";
      end case;
    end process;
end rtl;
```



# Elementos Básicos de VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity encoder is
  port (invec:in std_logic_vector(7 downto 0); enc_out:out  std_logic_vector(2 downto 0));
end encoder;
architecture rtl of encoder is
begin
  process (invec)
    begin
      if invec(7) = '1' then        enc_out <= "111";
      elsif invec(6) = '1' then        enc_out <= "110";
      elsif invec(5) = '1' then        enc_out <= "101";
      elsif invec(4) = '1' then        enc_out <= "100";
      elsif invec(3) = '1' then        enc_out <= "011";
      elsif invec(2) = '1' then        enc_out <= "010";
      elsif invec(1) = '1' then        enc_out <= "001";
      elsif invec(0) = '1' then        enc_out <= "000";
      else         enc_out <= "000";
      end if;
     end process;
end rtl;
```



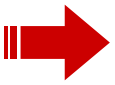
# Elementos Básicos de VHDL

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

# Elementos Básicos de VHDL

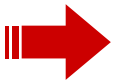
- Bucles (sólo dentro de los procesos):

```
for loop_var in range loop
...   --sequential statements
end loop;
```



```
for i in 3 downto 0 loop
-- i es una variable q no necesita
-- ser declarada
    d(i) <= a(i+1);
end loop;
```

```
while condición loop
  ...   --sequential statements
end loop;
```



```
while contador > 0 loop
contador := contador-1;
resultado := resultado+data;
end loop;
```

**No las vamos a usar**

# Elementos Básicos de VHDL

- Expresiones secuenciales:

**wait**;
**wait on** signal_list;
**wait for** time_expression;
**wait until** condition;

**Para esperar 10ns en una simulación**
**wait for 10ns**;

signal‘**event**
signal‘**last_event**
signal‘**last_value**

**Para que un registro haga algo sólo en el flanco de subida:**

**if** clock**’event and** clock = ‘1’;

# Elementos Básicos de VHDL

- Descripción estructural: *(estructural)*
  - Describe las interconexiones entre distintos módulos
  - Estos módulos pueden a su vez tener un modelo estructural o de comportamiento
  - Normalmente esta es una descripción a más alto nivel

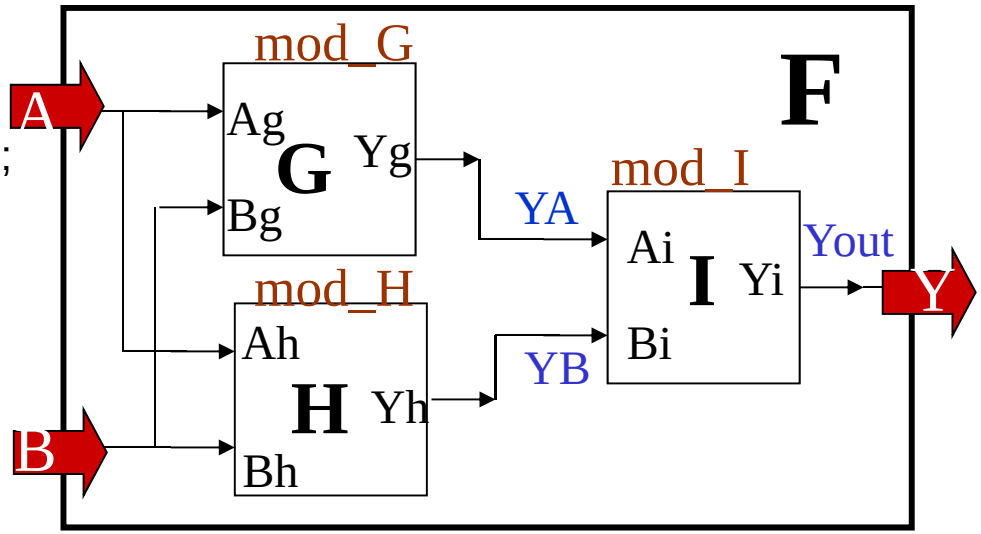
# Elementos Básicos de VHDL

- Descripción Estructural:

```
ARCHTECTURE arch_name OF entity_name IS
   COMPONENT component_name
        GENERIC (…);
        PORT (…);
   END COMPONENT;
signal declarations;
BEGIN
   component_i: component_name
      GENERIC MAP (parameter_value)
      PORT MAP (io_name)
END arch_name;
```

# Elementos Básicos de VHDL

» Ejemplo

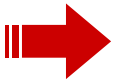



```
architecture structure of  F is
      component G
            port (Ag, Bg: in bit; Yg: out bit);
      end component;
      component H
            port (Ah, Bh: in bit; Yh: out bit);
      end component;
      component I
            port (Ai, Bi: in bit;Yi: out bit);
      end component;
      signal YA, YB, Yout: bit;
begin
      mod_G: G port map (Ag=>A; Bg=>B; Yg=>YA);
      mod_H: H port map (Ah=>A; Bh=>B; Yh=>YB);
      mod_I : I port map (Ai=>YA; Bi=>YB; Yi=>Yout);
      Y<=Yout;
end structure;
```

# Elementos Básicos de VHDL

- **GENERATE**: Las secuencias de generación de componentes permiten crear una o más copias de un conjunto de interconexiones, lo cual facilita el diseño de circuitos mediante descripciones estructurales.

```
for indice in rango generate
  -- instrucciones concurrentes
end generate;
```

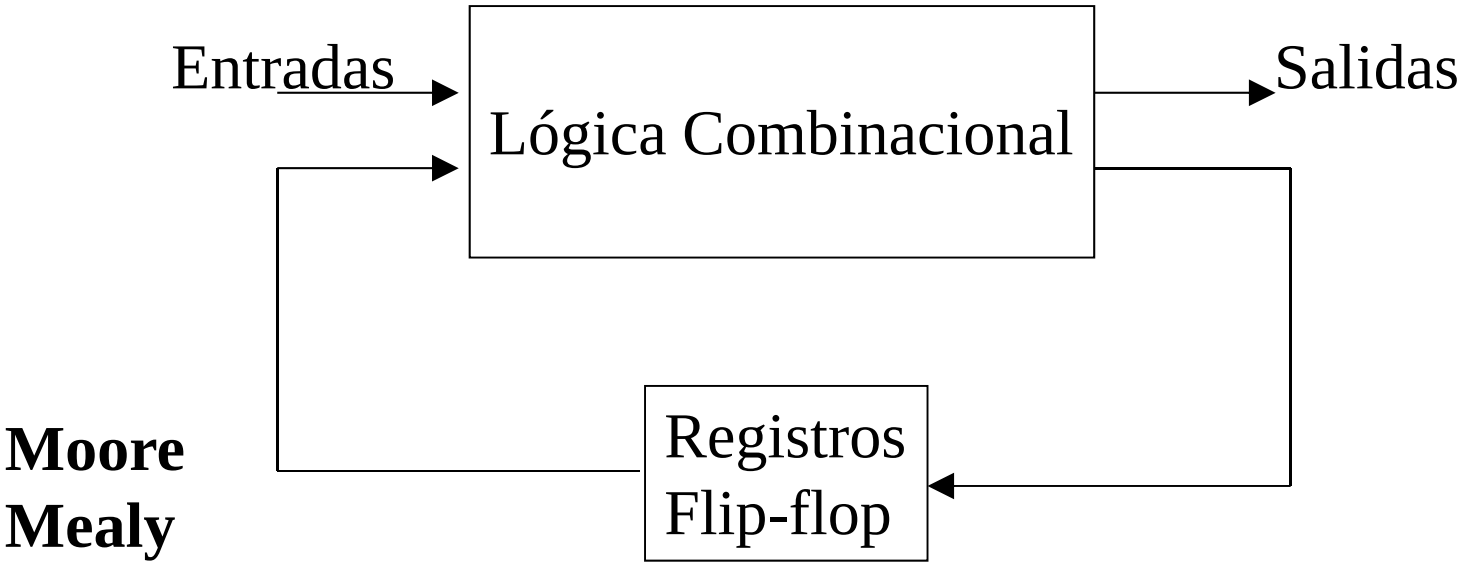


```
component comp
    port( x: in bit; y: out bit);
end comp
...
signal a, b: bit_vector(0 to 7)
...
gen: for i in 0 to 7 generate
    u: comp port map (a(i),b(i));
end generate gen;
```

**No lo vamos a usar**

# Máquina de Estados Finita (FSM)

- Todo circuito secuencial se divide en un circuito combinacional que implementa la salida del circuito y la transición al siguiente estado y en unos elementos de almacenamiento.




**Moore**
**Mealy**

# Máquina de Estados Finita (FSM)

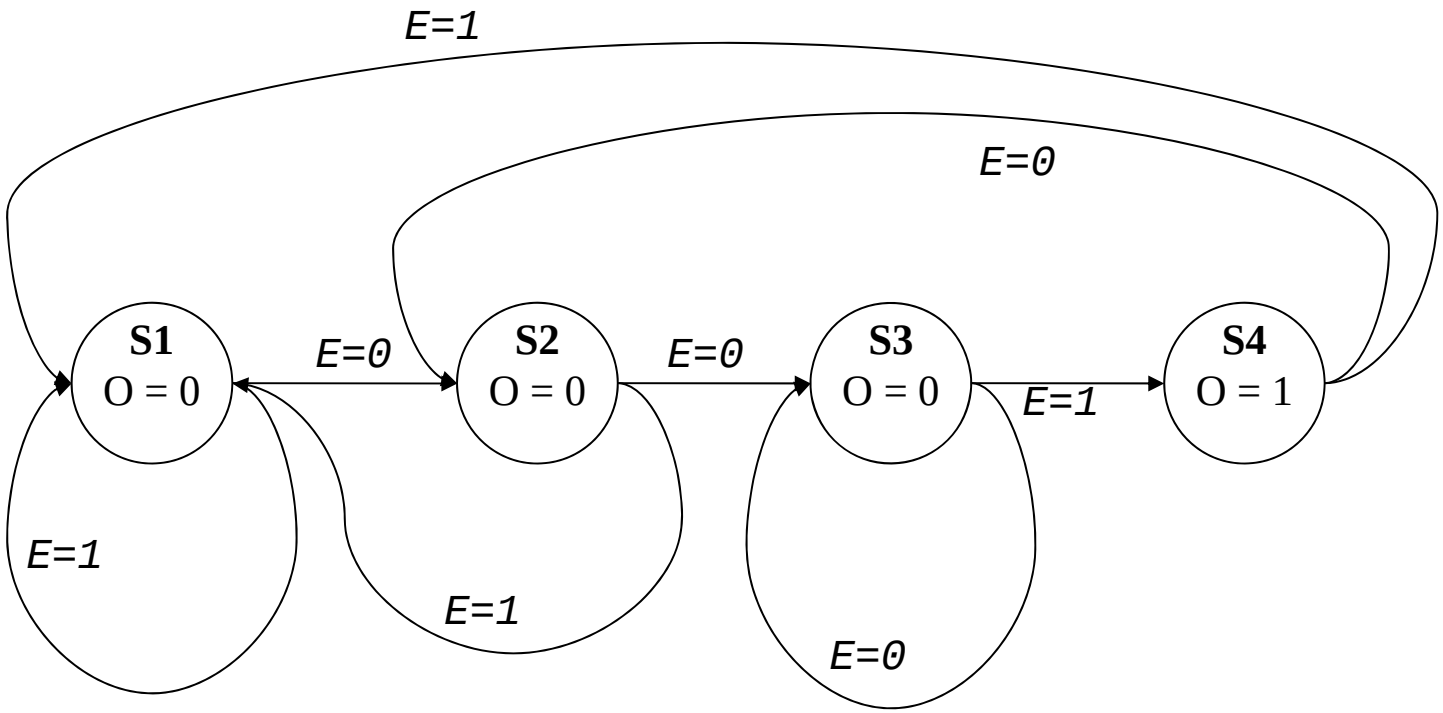
**architecture** comportamiento **of** *FSM* **is**
*internal signal declarations;*
**begin**

síncrono: **process** ( *clock, reset*)
**begin**
    *vhdl indicando los flip-flops*
**end process** síncrono;

comb: **process** ( *sensitivity list*)
**begin**
*vhdl indicando la lógica combinacional*
**end process** comb;

end comportamiento;

# Máquina de Estados Finita (FSM)



# Máquina de Estados Finita (FSM)

```
library IEEE;
use IEEE.std_logic_1164.all;
entity FSM is
        port(reset, E, clk: in bit; O: out bit);
end FSM;
architecture ARCH of FSM is
type ESTADOS is (S1, S2, S3,S4);
signal ESTADO, SIG_ESTADO: ESTADOS;

begin
```

```
SINCRONO: process(clk,reset)
begin
        if reset ='1' then
                ESTADO<=S1;
        elsif clk'event and clk='1' then
                ESTADO<= SIG_ESTADO;
         end if;
end process SINCRONO;
```

```
COMB: process(ESTADO,E)
begin
        case ESTADO is
        when S1 =>
                O <= '0';
                if (E='0') then SIG_ESTADO<=S2;
                else SIG_ESTADO<=S1;
                end if;
        when S2 =>
                O <= '0';
                if (E='0') then SIG_ESTADO<=S3;
                else SIG_ESTADO<=S1;
                end if;
        when S3 =>
                O <= '0';
                if (E='0') then SIG_ESTADO<=S3;
                else SIG_ESTADO<=S4;
                end if;
        when S4 =>
                O <= '1';
                if (E='0') then SIG_ESTADO<=S2;
                else SIG_ESTADO<=S1;
                end if;
end case;
end process control;
```

```
end ARCH;
```

# Máquina de Estados Finita (FSM)

```
library IEEE;
use IEEE.std_logic_1164.all;
entity FSM is
        port(reset, E, clk: in bit; O: out bit);
end FSM;
architecture ARCH of FSM is
type ESTADOS is (S1, S2, S3,S4);
signal ESTADO, SIG_ESTADO: ESTADOS;

begin
```

```
SINCRONO: process(clk,reset)
begin
        if reset ='1' then
                ESTADO<=S1;
        elsif clk'event and clk='1' then
                ESTADO<= SIG_ESTADO;
         end if;
end process SINCRONO;
```

```
COMB: process(ESTADO,E)
begin
        case ESTADO is
        when S1 =>
                O <= '0';
                if (E='0') then SIG_ESTADO<=S2;
                else SIG_ESTADO<=S1;
                end if;
        when S2 =>
                O <= '0';
                if (E='0') then SIG_ESTADO<=S3;
                else SIG_ESTADO<=S1;
                end if;
        when S3 =>
                O <= '0';
                if (E='0') then SIG_ESTADO<=S3;
                else SIG_ESTADO<=S4;
                end if;
        when S4 =>
                O <= '1';
                if (E='0') then SIG_ESTADO<=S2;
                else SIG_ESTADO<=S1;
                end if;
end case;
end process control;
```

```
end ARCH;
```