Repaso de VHDL para síntesis

Hipólito Guzmán Miranda
Profesor Contratado Doctor
Universidad de Sevilla
hguzman@us.es

Propuesta: empezamos hoy con Repaso de VHDL (extra)
A cambio terminamos la teoría 1 día más tarde,
el jueves de la 1a semana de prácticas

El VHDL que conocéis

- Estructura de un fichero VHDL
- Sección Library
- Sección Entity
- Sección Architecture
  - Antes del begin
  - Después del begin

VHDL = VHSIC HDL

VHSIC = Very High Speed Integrated Circuit

HDL = Hardware Description Language

VHDL = VHSIC Hardware Description Language

Lenguages informáticos

Lenguajes de programación
(C, python, matlab, C++, ...)

describen un programa

programa = secuencia de instrucciones

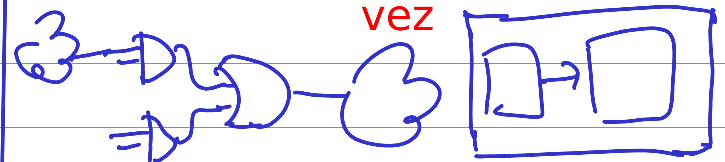
el ORDEN es importante

Lenguajes de descripción hardware

describen un circuito

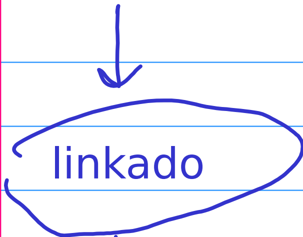
en un circuito, los diferentes elementos funcionan de manera concurrente

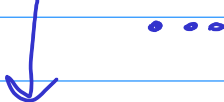
concurrente = que ocurre a la vez



código C
(programa)

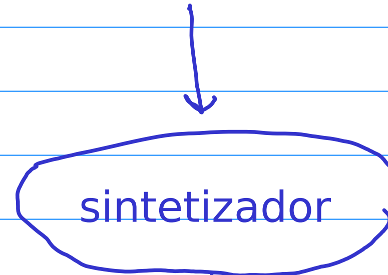
código objeto

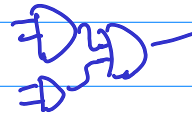




código máquina
(ejecutable)

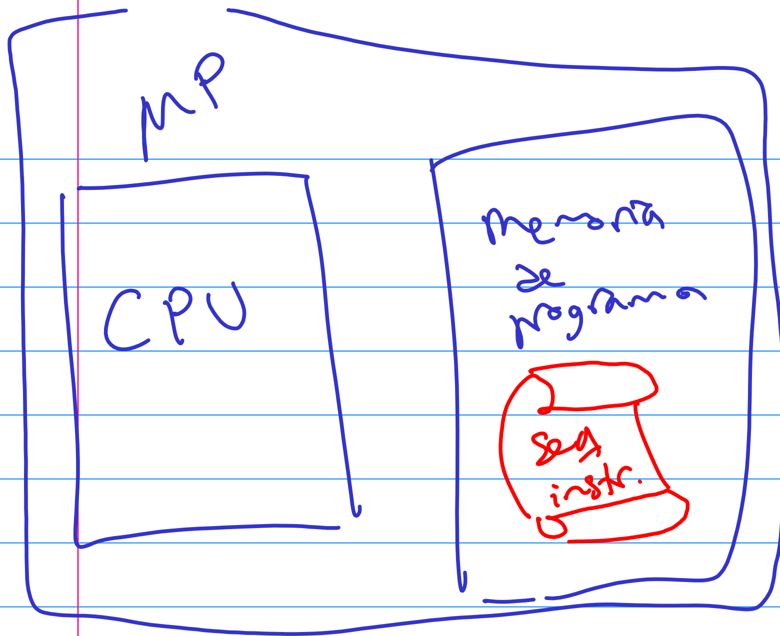
código VHDL
(circuito)





descripción a nivel de puerta

fichero de configuración de la FPGA
(o diseño a nivel físico para fabricación)



FPGA

Field Programmable Gate Array

+ PRESTACIONES
+ COSTE
+ ESFUERZO

la separación entre CPU y memoria de programa es lo que hace que el dispositivo sea programable

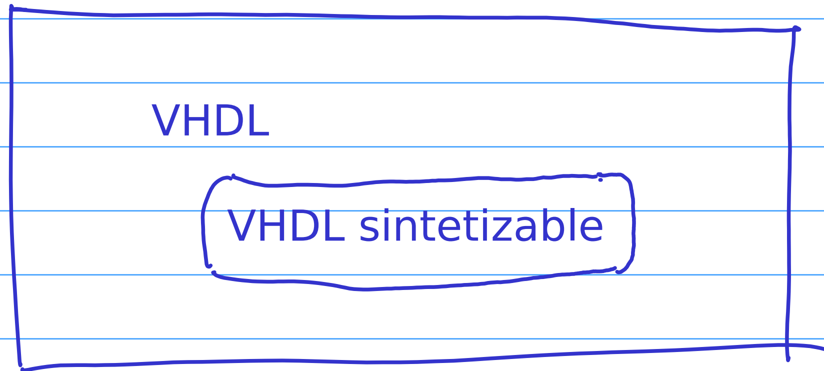
... pero el hardware que procesa las instrucciones es siempre el mismo

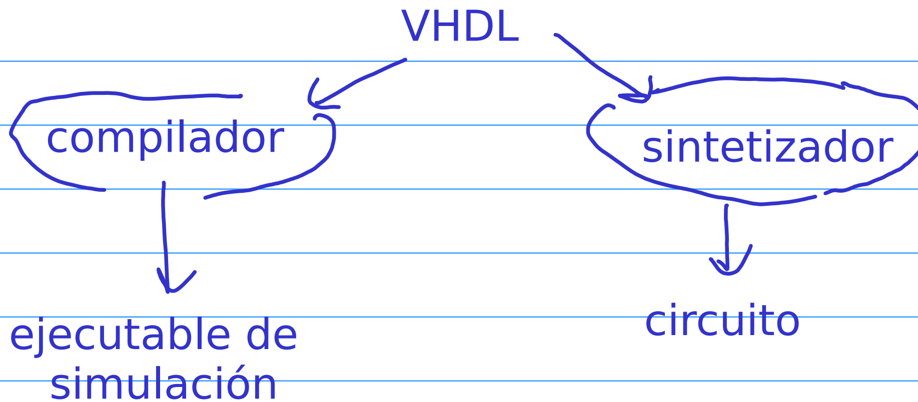
Verilog (otro HDL) se parece a C

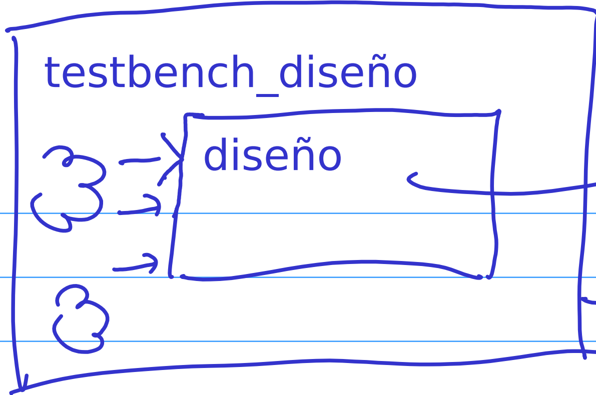
VHDL se parece a Ada

→ de tipado duro (a ambos lados de una asignación tiene que haber exactamente el mismo tipo de dato)

```
a <= b;
```







se sintetiza

se compila y se ejecuta para comprobar que tu diseño es correcto (verificación)

Secciones de un fichero VHDL

1) Library

2) Entity

3) Architecture

begin

antes del begin
1)
2)
3)

después del begin
1)
2)
3)

~~Configuration~~ (no se suele usar)

Sección Library

Library

Inclusión de librerías y paquetes con:

Tipos de datos, Funciones, Componentes, ...

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

Sección Library

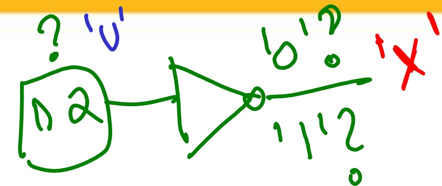
Ejemplo:

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

Sintaxis:

```
library lib_name;
use lib_name.package_name.all;
```

Sección Library



Paquetes a utilizar:

ieee.std_logic_1164.**all**;

el tipo de dato bit en VHDL no se debe usar para síntesis, ya que sólo puede tomar los valores '0' y '1'

el tipo de dato std_logic puede tomar 9 valores posibles:

ieee.numeric_std.**all**;

Sintaxis:

```
library lib_name;
use lib_name.package_name.all;
```

std_logic_1164 define
std_logic
std_logic_vector (MSB downto LSB)

numeric_std define
signed (MSB downto LSB)
unsigned (MSB downto LSB)
y operaciones aritméticas entre ellos

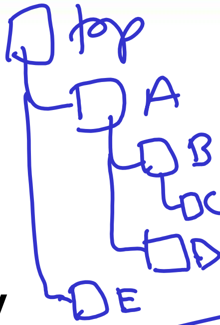
```
mi_std_logic_vector_4bits <= mi_unsigned_4_bits;
```

ERROR porque el tipo de dato no coincide
(lenguaje es de tipado duro)

solución: usar funciones de conversión

Sección Entity

en un lenguaje de programación, la unidad mínima de funcionalidad es la función
en VHDL, la unidad mínima de funcionalidad es la entidad



Entity

Descripción de 'caja negra': entradas, salidas y parámetros (generics)

```
entity counter is
  Generic (N : integer := 8);
  Port ( rst    : in  STD_LOGIC;
         clk    : in  STD_LOGIC;
         enable : in  STD_LOGIC;
         count  : out  STD_LOGIC_VECTOR (N-1 downto 0)
  );
end counter;
```

tipo de dato

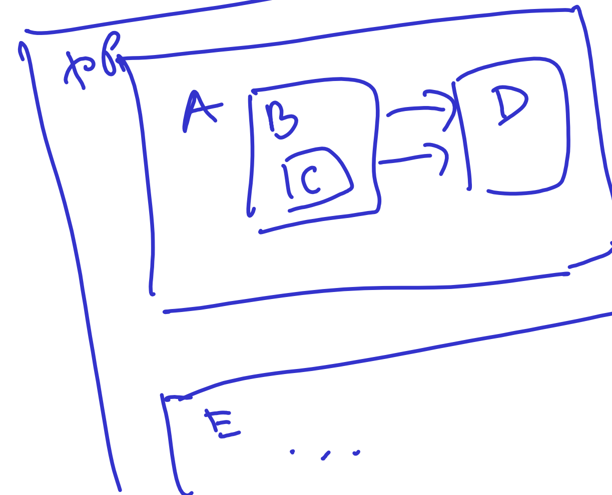
valor por defecto

nombre

dirección

tipo de dato

el último no lleva separador



Sección Entity

Sintaxis:

Direction debe ser in, out o bidir

```
entity entity_name is
   Generic (gen_name : data_type := default_value;
               <another generic>;
               <last port doesn’t have separating ;>
              );
   Port ( port_name  : direction data_type;
             <another port>;
             <last port doesn’t have separating ;>
          );
end entity_name;
```

Sección Architecture

Dos partes diferenciadas:

- Antes del `begin`

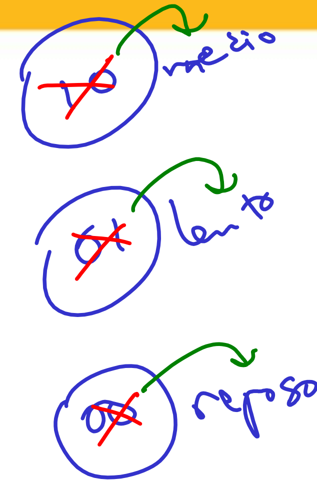
  declaración de elementos que utilizaremos después

- Después del `begin`

  comportamiento dinámico
  funcionalidad
  "todo lo que se parezca a un circuito funcionando"

Antes del begin

1) • Definición de tipos de dato
2) • Declaración de señales — cables que permiten conectar elementos
3) • Declaración de componentes



se usa para FSMs

tipo enumerado

```
1) type t_estado is (parada, lento, medio, rapido);
2) signal estado, p_estado: t_estado;
   signal cuenta, p_cuenta: std_logic_vector(7 downto 0);

1) type enum_data_type is (first, second, third, fourth);
2) signal signal_name: data_type;
   signal signal1, signal2: data_type;
```

Antes del begin

necesario para reutilizar entities descritos en otros ficheros

3) Declaración de componentes:

```
component counter is
  Generic (N : integer := 8;
           M : integer := 10);
  Port ( rst    : in  STD_LOGIC;
         clk    : in  STD_LOGIC;
         enable : in  STD_LOGIC;
         count  : out  STD_LOGIC_VECTOR
                       (N-1 downto 0));
end component;
```

Subsección Generic y Port son exactamente iguales a las de la entidad que estamos declarando como componente

Después del begin

1) - Sentencias concurrentes no escalan bien para funcionalidades complejas
2) - Process
3) - Instancias de componentes

poner (gastando área) una copia (instancia) del component que has declarado anteriormente

que ocurren a la vez (concurren)

Sentencias concurrentes

Sentencias concurrentes:

- Asignaciones: b `<= a;`
- Operaciones lógicas: c `<= a and (not b);`
- When… else
- With… select

y artiméticas si están declaradas

c <= expr (a, b, ...)

asignación

toma de decisiones

análogo al if

Sentencias concurrentes

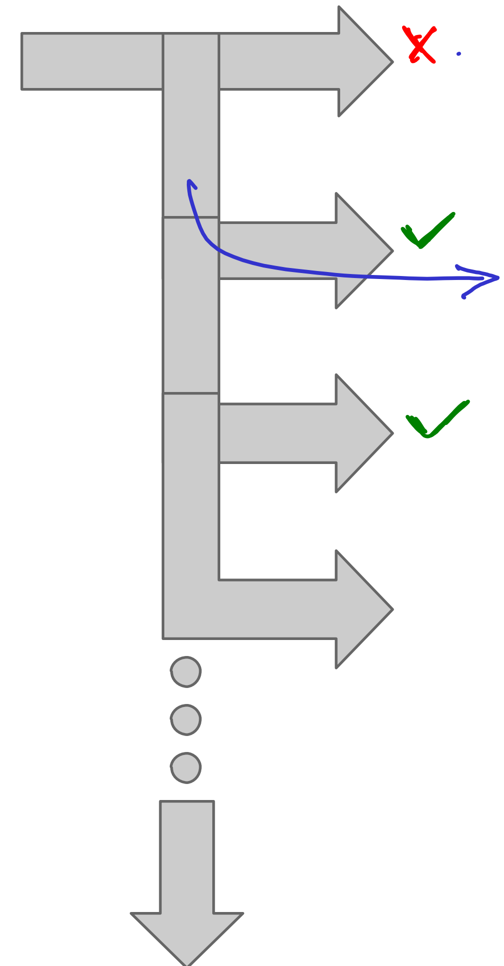
decisión con prioridad

When… else:

```
d <= (not a) when e="01" else
             b when e="10" else
             c;
```

asignación

```
sig1 <= expr1 when cond1 else
        expr2 when cond2 else
        <…>
        exprN;
```



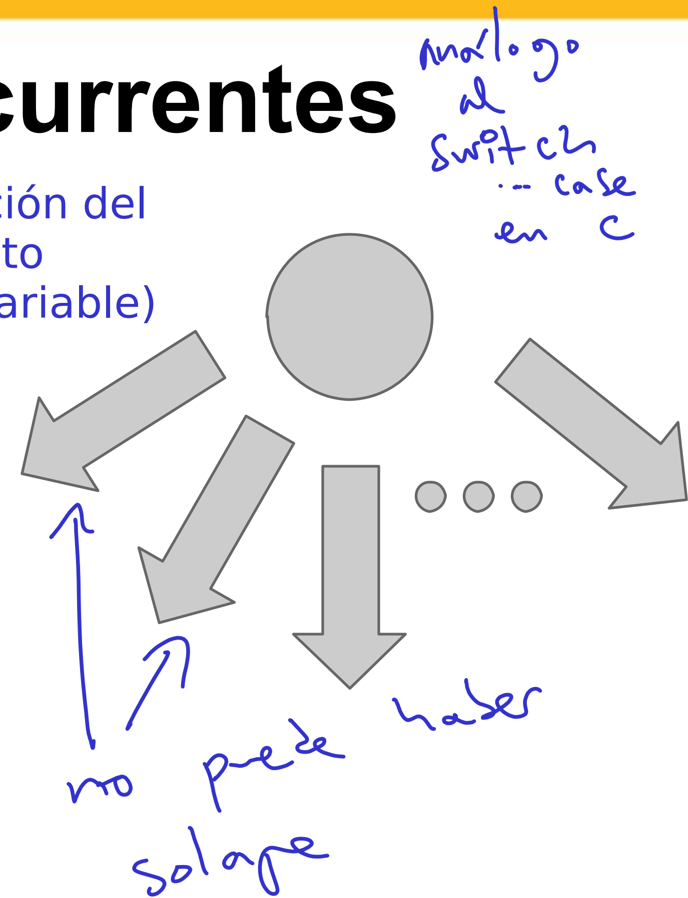
Sentencias concurrentes

With… select:

decisión en función del valor de un objeto (signal / port / variable)

```
with e select
   d <= not a when "01",
            b when "10",
            c when others;

with sig1 select
   sig2 <= expr1 when value1,
           expr2 when value2,
           <...>
           expr3 when others;
```



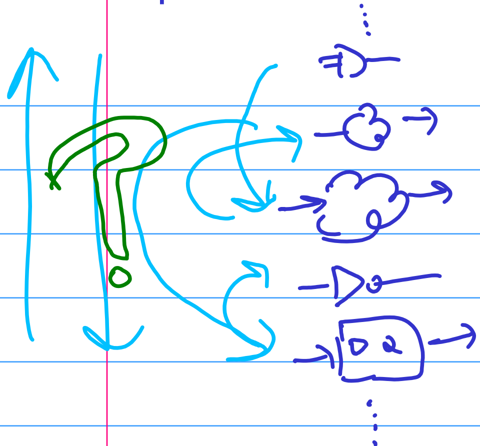
Después del begin

Procesos:

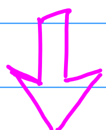
- Asignaciones: b `<= a;`
- Operaciones lógicas: c `<= a and (not b);`
  y aritméticas si están declaradas
- If… elsif… else
- Case… when

→ toma de decisiones

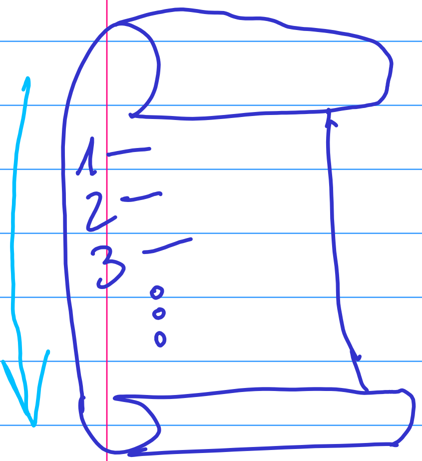
problema sentencias concurrentes:

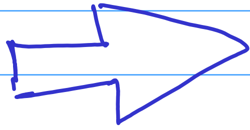


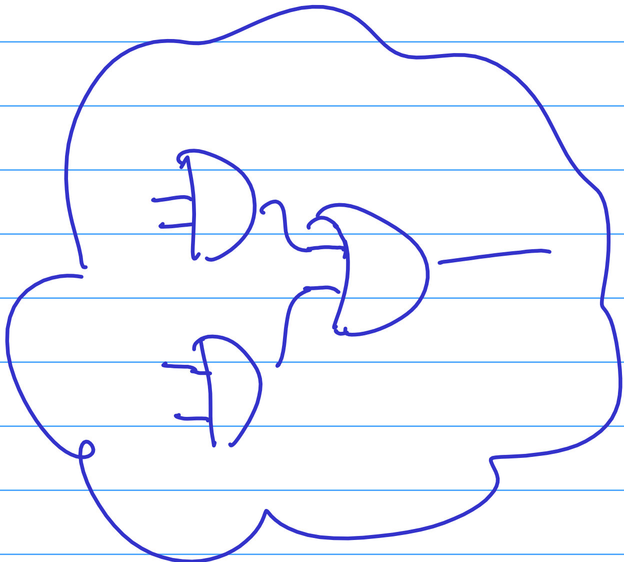
¿orden de lectura para ENTENDERLO?



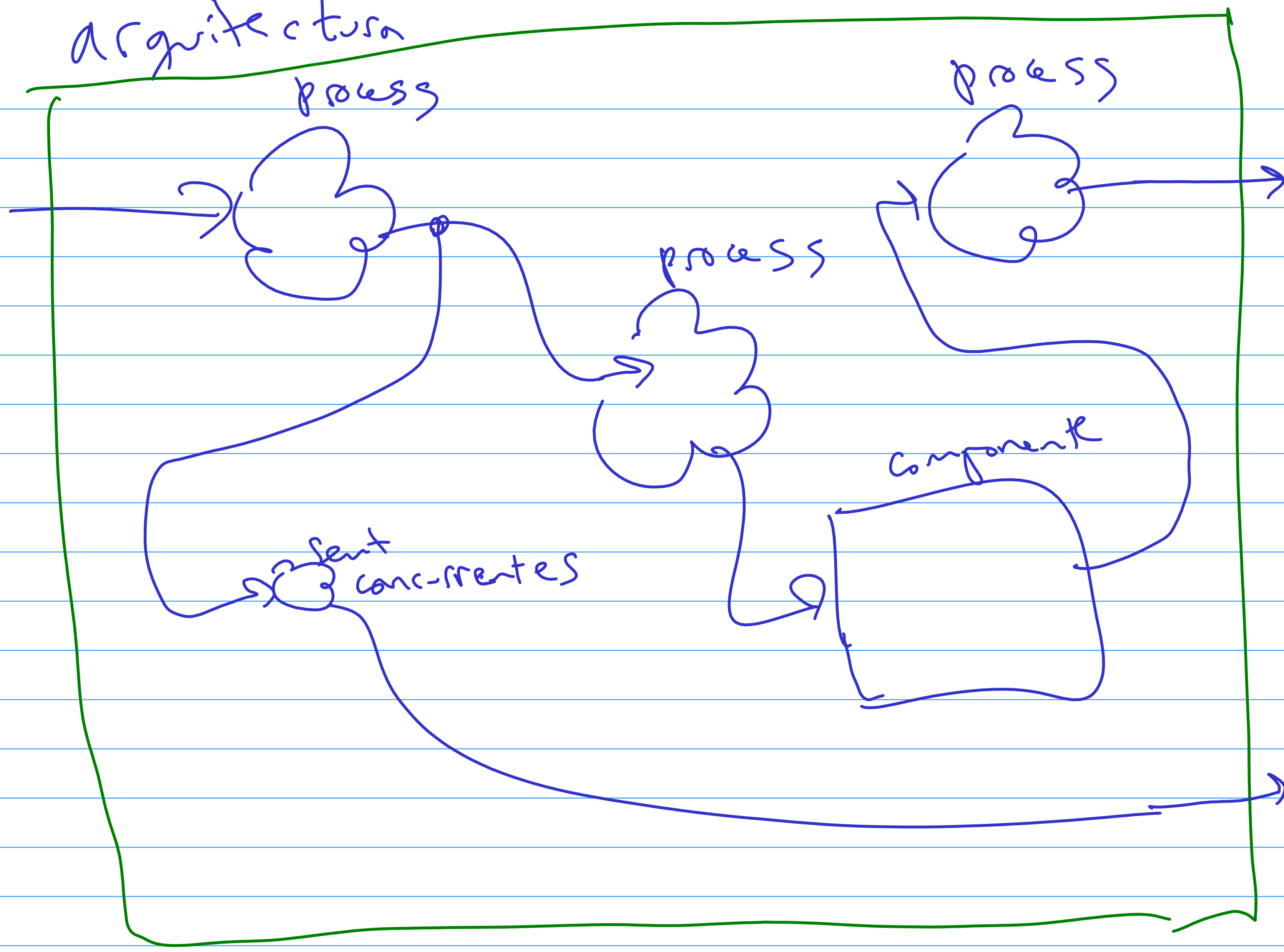
PROCESS







secuencia de instrucciones


arquitectura
process
process
process
componente
sem concorrentes

Procesos

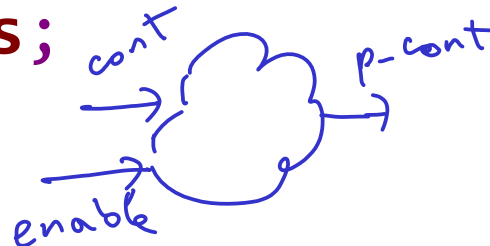
lista de sensibilidad

```
comb: process (cont, enable)
  begin
    if (enable = '1') then
      p_cont <= cont + 1;
    else
      p_cont <= cont;
    end if;
  end process;
```

todo objeto (signal o port) que influya en el resultado del process

1) todo lo que está dentro de la condición de un if (o el valor que se mira en un case)

2) todo lo que está a la derecha de una asignación

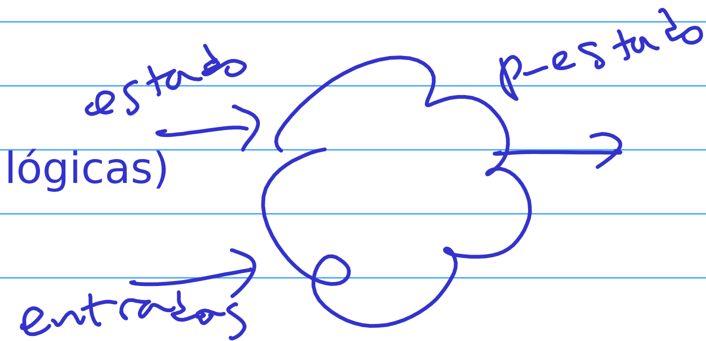


Procesos

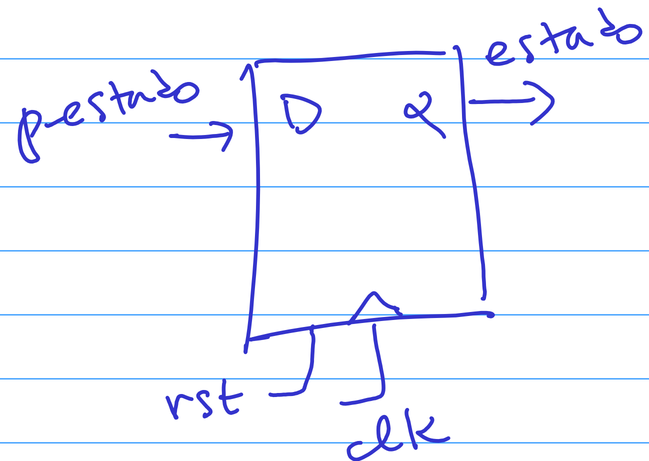
```
proc_name: process (lista_sensibilidad)
  begin
    <Sentencias>
  end process;
```

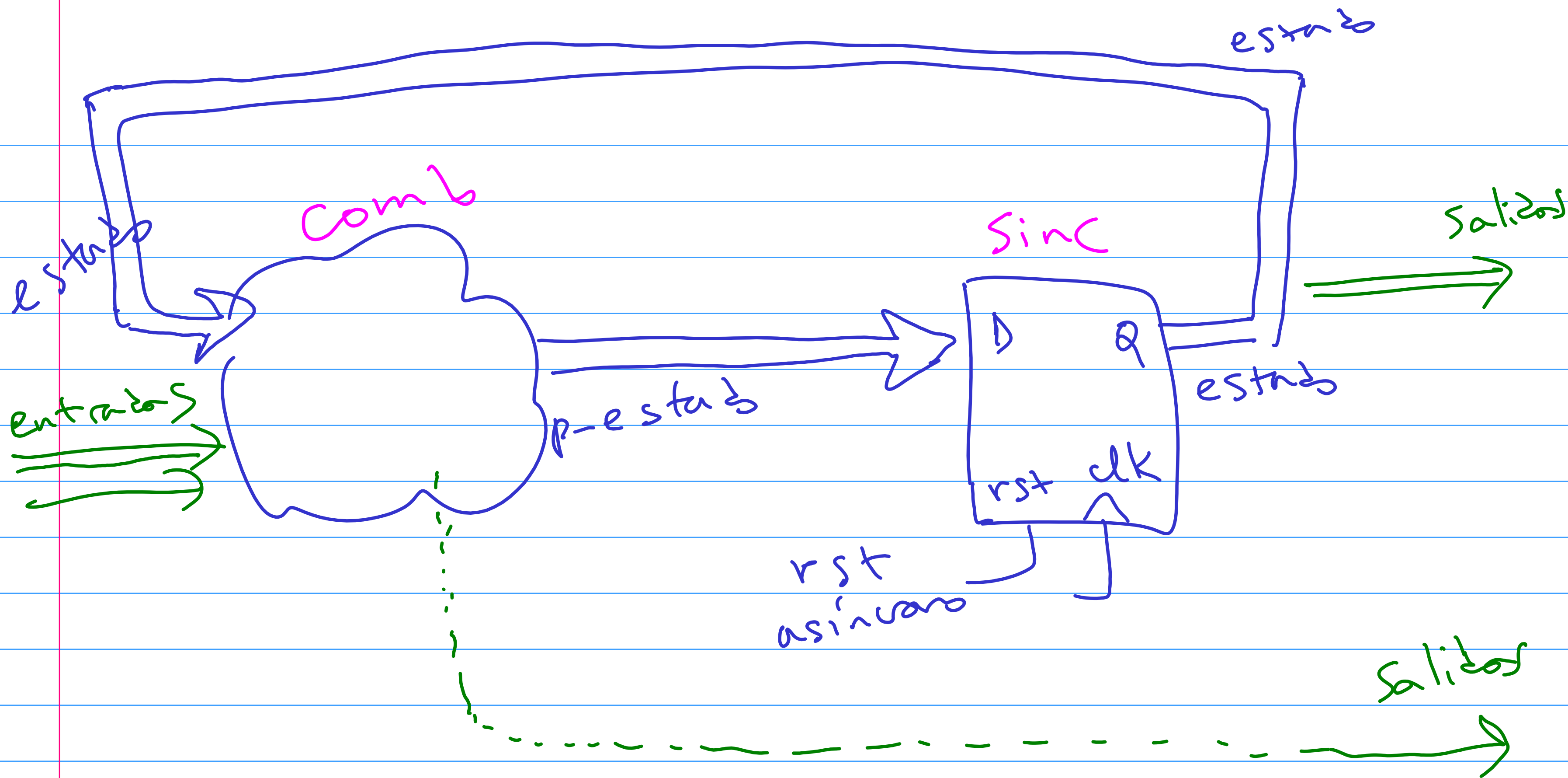
Diseño con dos procesos
- más enseñable / asequible
- más depurable (aparecen señales internas en la simulación)
- recomendado por la ESA
- más difícil equivocarse
- más control sobre el resultado final (sobre todo al principio)
- no es incompatible con entender diseños hechos con 1 process en el futuro

1 proceso combinacional
- toma de decisiones
- se convierte en funciones lógicas (puertas lógicas)
- if y case como queramos



1 proceso síncrono
- elementos de memoria
- se convierte en biestables activos por flanco (FF / flip-flops)
- siempre la misma "receta"




estado
comb
sinc
salidas
estado
entradas
D
Q
p_estado
estado
rst
clk
rst
asíncrono
salidas

Procesos síncronos

sólo es sensible al reset (si tiene) y al clk

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

sync

pone eg a TODOS los bits a 'ϕ' "000.....00"

si no hay rst='1' ni flanco de subida de clk ¿qué hace? -> almacena el valor anterior que es lo queremos que haga el biestable

Los procesos síncronos siempre se describen de la misma manera

Si en los procesos combinacionales no definimos qué valor se asigna a las salidas de dicho process en TODOS los casos posibles, se genera un LATCH, ya que las puertas lógicas no tienen memoria

Hay que evitar los latches, así como las listas de sensibilidad incompletas

El sintetizador avisa si hay latches y listas de sensibilidad incompletas

A B C	f
0 0 0	0
0 0 1	0
⋮	
1 1 0	<no definido>

→ mantiene el valor anterior
como el process comb no tiene clk, tiene que meter un elemento de memoria activo por nivel (Latch)

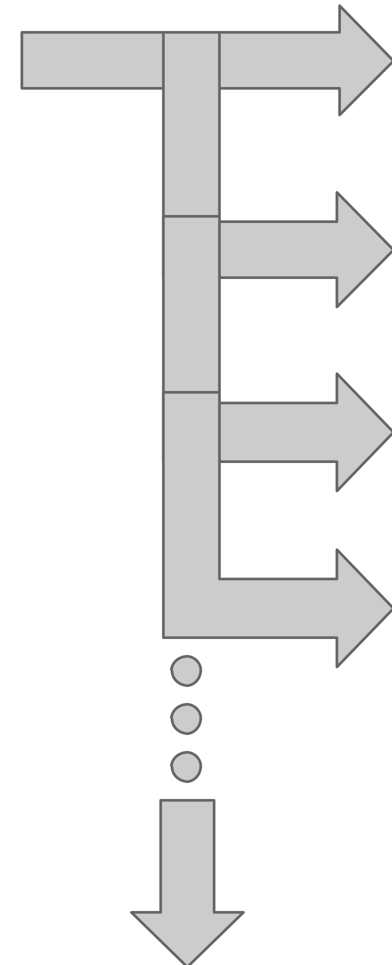
Los latches son una pesadilla a nivel de timing

Procesos

decisión con prioridad

If… elsif… else:

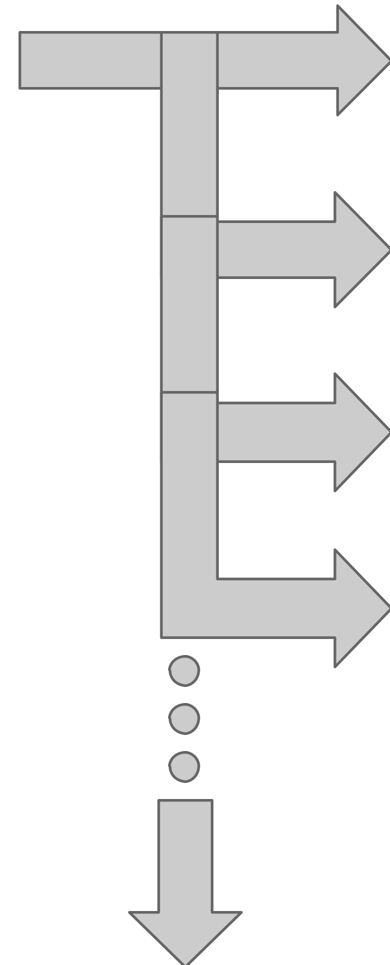
```
if (rst_sync = '1') then
   p_cont <= (others=>'0');
elsif (enable = '1') then
   p_cont <= cont + 1;
else
   p_cont <= cont;
end if;
```



Procesos

If… elsif… else:

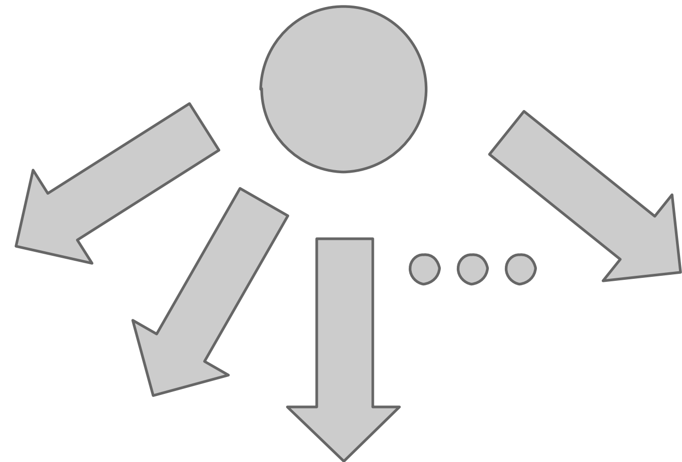
```
if (cond1) then
   <sentencias>
elsif (cond2) then
   <sentencias>
<... más elsif ...>
else
   <sentencias>
end if;
```



Procesos

Case... when:

```
case state is
     when idle =>
        <sentencias>
     when count =>
        <sentencias>
     when header =>
        <sentencias>
     when others =>
        <sentencias>
end case;
```

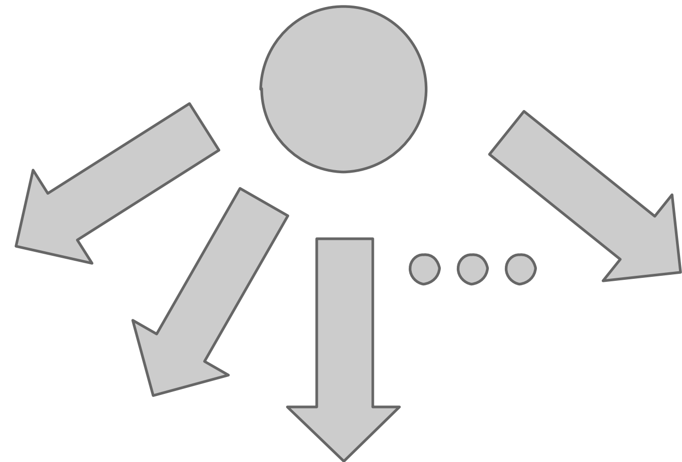


Muy utilizado para describir máquinas de estados

Procesos

Case... when:

```
case sig1 is
    when value1 =>
        <sentencias>
    when value2 =>
        <sentencias>
    when value3 =>
        <sentencias>
    when others =>
        <sentencias>
end case;
```



Muy utilizado para describir máquinas de estados

Instancias de componentes

```
cont_inst: counter
generic map ( N => 8, M => 10 )
port map (
      rst_high => sig_rst_high,
      enable => sig_enable,
      clk => clk,
      data_out => sig_data_out
    );
```

Instancias de componentes

```
inst_name: component_name
generic map ( gen1 => val1, gen2 => val2 )
port map (
        port1 => sig_top1,
        port2 => sig_top2,
        port3 => sig_top3,
        <...>
        portN => sig_topN
    );
```

Component_port => top_signal_or_port,

Ejercicio

Diseñemos y simulemos un contador de N bits (instanciado con N=8) con Xilinx ISE

Entradas:

Clk, rst, enable : std_logic;

Salidas;

Cuenta : std_logic_vector (7 downto 0);