Diseño de Computadores: Síntesis Lógica
Apuntes de Prácticas sobre
Diseño y Simulación con VHDL empleando el entorno de XILINX: ISE

Manuel J. Bellido

Octubre 2008

Contenidos

- Diseñando en Lenguaje VHDL y verificando mediante simulación
  - Estructura del código VHDL: packages, diseños y testbenchs
  - Diseño de Registros
  - Herramienta de Simulación de circuitos descritos en VHDL: Isesimulator
  - Diseño de bloques combinacionales
  - Diseño de Máquinas de Estados Finitos
  - Uniendo Componentes

Diseñando en Código VHDL

- -- DEFINICION DE LA ESTRUCTURA: ENTIDAD, ENTRADAS Y SALIDAS--

```
library ieee;           -- Definicion de librería a usar
use ieee.std_logic_1164.all; -- Definición de package a usar
use ieee.numeric_std.all;    -- Definición de package a usar

Entity <NOMBRE_ENTIDAD> is
port (
-- SEÑALES DE ENTRADA
-- SEÑALES DE SALIDA
 );
end <NOMBRE_ENTIDAD>;
```

Diseñando en Código VHDL

- -- DEFINICIÓN DEL COMPORTAMIENTO: ARQUITECTURA DE LA ENTIDAD

```
Architecture <NOMBRE_ARQUITEC> of <NOMBRE_ENTIDAD> is
<SENALES INTERNAS QUE SE NECESITAN>
begin
--PROCESO SECUENCIAL
<NOMBRE PROCESO>: process (<LISTA DE SENSIBILIDAD: Rst, CLK>)
        begin
    <cuerpo del proceso secuencial>
        end process;
--PROCESO COMBINACIONAL
<NOMBRE_PROC.>: process (<LISTA:todas las señales que sean leidas>)
        begin
    <cuerpo del proceso combinacional>
        end process;
end <NOMBRE_ARQUITEC>;
```

Diseñando en Código VHDL

- Packages: Es un elemento de VHDL que contiene declaraciones que pueden ser usadas por mas de un componente en el diseño.

- Declaraciones en un packages:
  - types, subtypes
  - constants
  - Components
  - subprograms (procedures and functions)

- -- DEFINICION DE LA ESTRUCTURA DE UN PACKAGE

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

package <NOMBRE_PAQUETE> is
 [type_decl]
  [subtype_decl]
  [constant_decl]
  [deferred_constant_decl]
  [subprogram_header_decl]
  [component_decl]
end <NOMBRE_PAQUETE>;
```

Diseñando en Código VHDL

- -- DEFINICIÓN DEL CUERPO DE PACKAGE

  package body **<NOMBRE_PAQUETE>** is

  [deferred_constant_value]

  [subprogram_body]

  end **<NOMBRE_PAQUETE>**;
- Inclusión de un packege en un diseño:
- Syntax:
- the default library is WORK

use [library_name.]package_name.all;

Si la librería no es work, hay que incluir una linea definiendo a la librería.

Diseñando en Código VHDL

- Guías básicas de diseño
- Proceso secuencial:
  - Lista sensibilidad: solo CLK y señal de inicialización ASÍNCRONA
  - Señal asíncrona mayor prioridad que el reloj
  - Flanco de reloj en VHDL:
    - CLK='1' and CLK'event :Flanco de subida
    - CLK='0' and CLK'event :Flanco de bajada
- Proceso Combinacional:
  - Lista de sensibilidad: todas las señales que van a ser leídas en el proceso (que son entradas en el bloque combinacional)
  - En caso de usar sentencias tipo “case” o “if” hay que estar seguros de cubrir todas las condiciones posibles
    - case: usar when others
    - if: usar else

Diseñando en Código VHDL

- Creación de Testbench para verificación del diseño mediante simulación
  - Se crea una Entidad vacia como testbench.
  - Se incluyen en la arquitectura una señal interna por cada entrada o salida de la entidad control.
  - Se incluye, el componente del diseño a verificar.
  - Por último, se crean patrones de entrada en las señales de entrada.
    - (Seguir las guías que se dan de creación de testbench en la Guía de codificación en VHDL para simulación y síntesis:

      http://www.dte.us.es/ing_inf/dise_comp/VHDLsimCMSG.pdf

)

Diseñando en Código VHDL

- Diseño de Registros

-- Si la salida del registro es condicional (con alta impedancia

--hay que definir la señal interna que define al registro:

```
signal <reg> : std_logic_vector( n downto 0);
begin
--PROCESO SECUENCIAL: Operaciones de cambio de valor
<NOMBRE PROCESO>: process (CLK)
        begin
            if clk='1' and clk'event then
                if write='1' then
                    <reg> <= <entrada_datos>;
                end if;
            end if;
        end process;
```

Condiciones de escritura en reg

Diseñando en Código VHDL

- Diseño de Registros

```
--PROCESO COMBINACIONAL

<NOMBRE_PROC.>: process (<señales de control lectura>, <reg>)

        begin
            if <señal_lectura>='1' then
                <salida_datos> <= <reg>;--LECTURA DEL REGISTRO
            else
                <salida_datos> <= (others => 'Z');--ALTA IMPEDANCIA
            end if;
        end process;
    end <NOMBRE_ARQUITEC>;
```

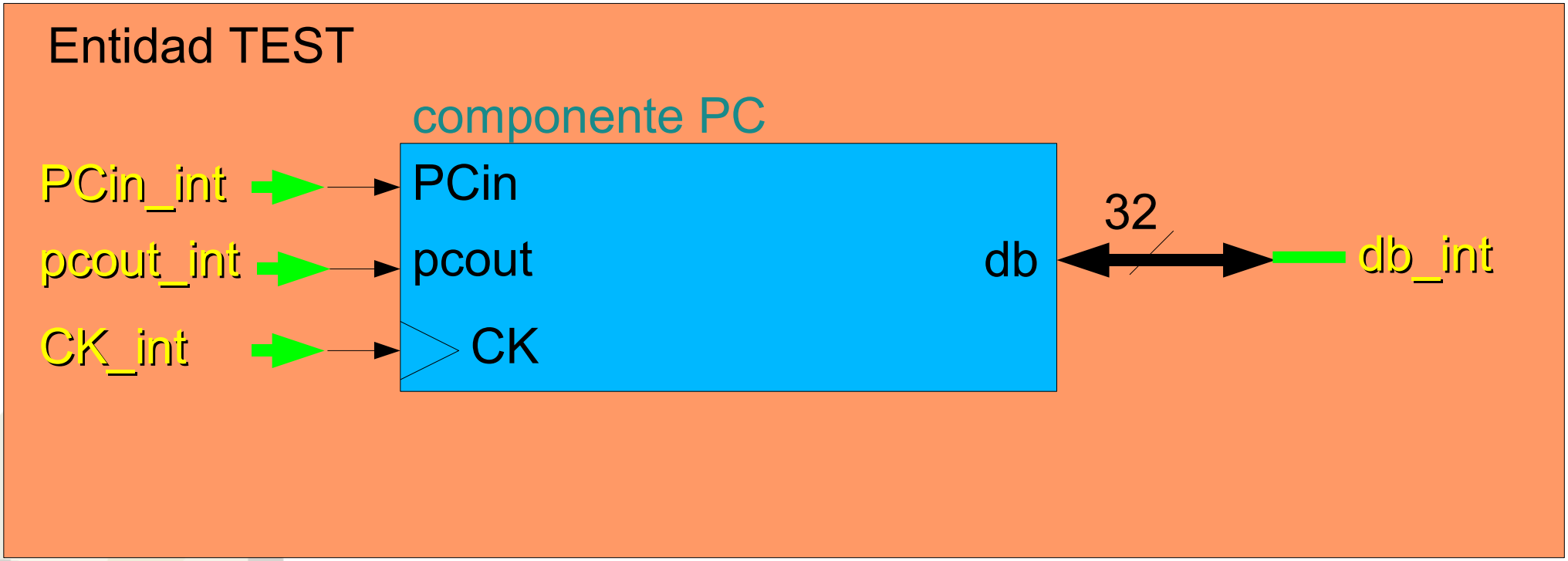
-

- Ejemplo de diseño de registro: Registro pc (enlace en la página web de la asignatura al fichero registro_pc.vhd)

Pcin	pcout	Oper.	db
0	0	PC<--PC	H.I.
0	1	PC<--PC	[PC]
1	0	PC<--IN	IN
1	1	-----	---

Diseñando en Código VHDL

- Creando código de test del registro PC (testbench) (enlace en la página web de la asignatura al fichero tb_pc.vhd)
  - 1.- Estructura del diseño:



  - 2.- Creación de patrones
- Simular y comprobar el funcionamiento mediante análisis de las formas de onda (waveforms)

Simulación con Isesimulator

- Herramienta empleada: Isesimulator de XILINX
  - Se encuentra dentro del entorno de ISE
- Recetario para simulación con Isesimulator (en coria.dte.us.es)
  - 1.- Inicializar variables: $ ent_xilinx
  - 2.- Crear un Nuevo proyecto para la simulacion
  - 3.- Durante la creación del proyecto conviene seleccionar las siguientes opciones:
    - Top-level source: HDL; Family: virtex2p; Device= XC2VP30; Package= FF896; Speed= -7; Synthesis Tool=XST; Simulator= Isesimulator; Preferred Language= VHDL
    - En la venta de create new source --> next
    - En la ventana de add source --> botón “add source” y añadir los fuentes del diseño a simular (incluyendo los packages necesarios) y del testbench.
  -

Simulación con Isesimulator

- 4.- Una vez creado el proyecto con los fuentes añadidos, seleccionar “behavioral simulation” en “Sources For:”. Debera aparecer el test-bench y, debajo de el, el diseño a simular.
- 5.- Seleccionar el fichero de testbench. En la ventana de “Procesess for”, desplegar “Xilinx ISE Simulator”.
- 6.- Doble click en “Simulate Behavioral Model”
- 7.- Corregir los errores (en caso de que los haya) o analizar las formas de onda para comprobar si el resultado es satisfactorio

- Ejercicio: Modificar el Registro PC para añadirle una señal síncrona de reset (reset_pc) y simular el diseño adecuadamente

Diseñando en Código VHDL

- Diseño de Arrays de Registros

```
-- Array de M Registros de n bits: Definición de tipo de señal
type ram is array (0 to M) of std_logic_vector(n downto 0);
signal <array_reg>  : ram;
begin

--PROCESO SECUENCIAL: Operaciones de cambio de valor
<NOMBRE PROCESO>: process (CLK)
        begin
            if clk='1' and clk'event then
                if write='1' then
                    <array_reg>(<Num_Entero>)<= <entrada_datos>;
            endif;
        end if;
    end process;
```

ATRIBUTOS MÁS UTILIZADOS

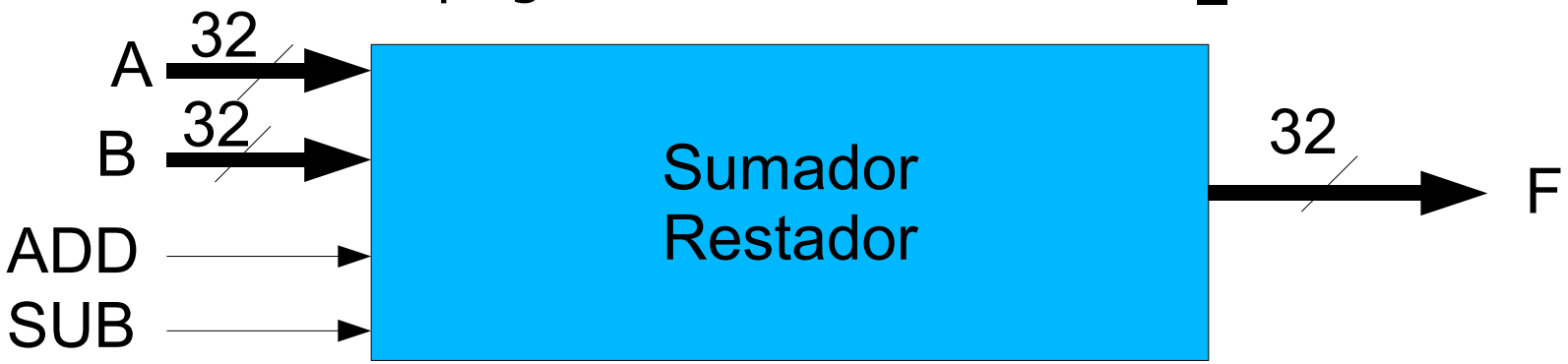
- atributos -> referencia a carácterísticas de tipos
- atributos que devuelven elementos
  - ‘left -> devuelve el elmento de la izquierda de la lista
  - ‘right -> devuelve el elemento de la derecha de la lista
  - ‘high -> devuelve el mayor elemento de la lista
  - ‘low -> devuelve el menor elemento de la lista
  - ‘succ(<ELEmento>) -> devuelve el siguiente elemento al indicado
  - ‘val(<índice>) -> devuelve el elemento correspondiente al índice
  - ‘pred(<elemento>) -> devuelve el elemento anterior al indicado
- Atributos que devuelven un valor
  - ‘pos(<elemento>) -> devuelve el índice correspondiente al elemento
  - ‘stable -> cierto cuando la no hay cambio en la señal
  - ‘event -> cierto cuando hay cambio en la señal
  - ‘range -> devuelve el rango del tipo
  - ‘length -> devuelve la longitud del tipo
- Atributos que devuelven una señal
  - ‘delayed(<N>) -> devuelve la misma señal retrasada n segundos

Diseñando en Código VHDL

- Diseño de Bloques combinacionales: ALU
- Circuitos con Operaciones Aritméticas.
  - Para manejar números con signo emplear **señales internas** tipo signed y para manejar número sin signo (magnitudes) emplearemos señales tipo unsigned que asociaremos a las entradas/salidas de la entidad que son tipo std_logic, si es necesario.
  - Ejemplo:
    - puertos de la entidad:
      - port (A : in std_logic_vector(n downto 0);.....)
    - Señales internas:
      - signal A_sig_int : signed(n downto 0);
      - signal A_uns_int : unsigned(n downto 0);
    - cuerpo de la arquitectura: Asociando A a las señales internas.
      - A_sig_int <= signed(A);
      - A_uns_int <= unsigned(A);
    - Conversión a std_logic: std_logic_vector(<nombre_signed>)

Diseñando en Código VHDL

- Ejemplo: Sumador-restador de números con signo C-2.
  - Enlace en la página web al fichero sum_res.vhd



ADD	SUB	Oper.
0	0	----
0	1	A - B
1	0	A + B
1	1	----

- Ejercicio: Crear testbench para el sumador-restador y verificar funcionalmente su operación.
- Ejercicio: A partir del sumador-restador, diseñar una alu que tenga, además de suma y resta, operaciones lógicas AND y OR de A y B

Diseñando en Código VHDL

- Diseño de Bloques combinacionales: ALU
- Circuitos con Operaciones de Desplazamiento.
  - Se emplean las funciones:
    - shift_right(B,A)
    - shift_left(B,A)
    - rotate_right(B,A)
    - rotate_left(B,A)
      - B dato tipo unsigned o signed que va a ser desplazado.
      - A dato tipo integer que indica el número de desplazamientos.
      - Si el dato B es tipo signed, el desplazamiento es Aritmético.
- Ejemplo: Desplazador Lógico a derecha e izquierda de números de 32 bits.
  - Enlace en la página web al fichero desplazador.vhd

- Conversiones de tipos en numeric_std:
  - A entero:
    - to_integer(<num_signed o num_unsigned>)
  - De entero a numero signed o unsigned
    - to_signed(<num_entero, tamaño del signed>)
    - to_unsigned(<num_entero, tamaño del unsigned>)
  - De signed o unsigned a std_logic:
    - std_logic_vector(<num_signed o unsigned>)
  - De std_logic_vector a signed o unsigned:
    - unsigned(<std_logic_vector>) signed(<std_logic_vector>)

Diseñando en Código VHDL

- Diseño de FSM:
  - --SENALES INTERNAS QUE SE NECESITAN:
    - tipo de estados y señales de estado actual y próximo:

```
type STATE_TYPE is (<definición de estados>);
signal CS, NS        : STATE_TYPE;
.....................................
```

  - --PROCESO SEC.: SEÑALES ASÍNCRONAS -Rst- Y LA SEÑAL DE RELOJ

```
<NOMBRE PROCESO>: process (Rst, CLK)
        begin
            if Rst = '0' then
                CS <== <estado inicial>;
            elsif CLK= '1' and CLK'event then
                CS <== NS;
            end if;
        end process;
```

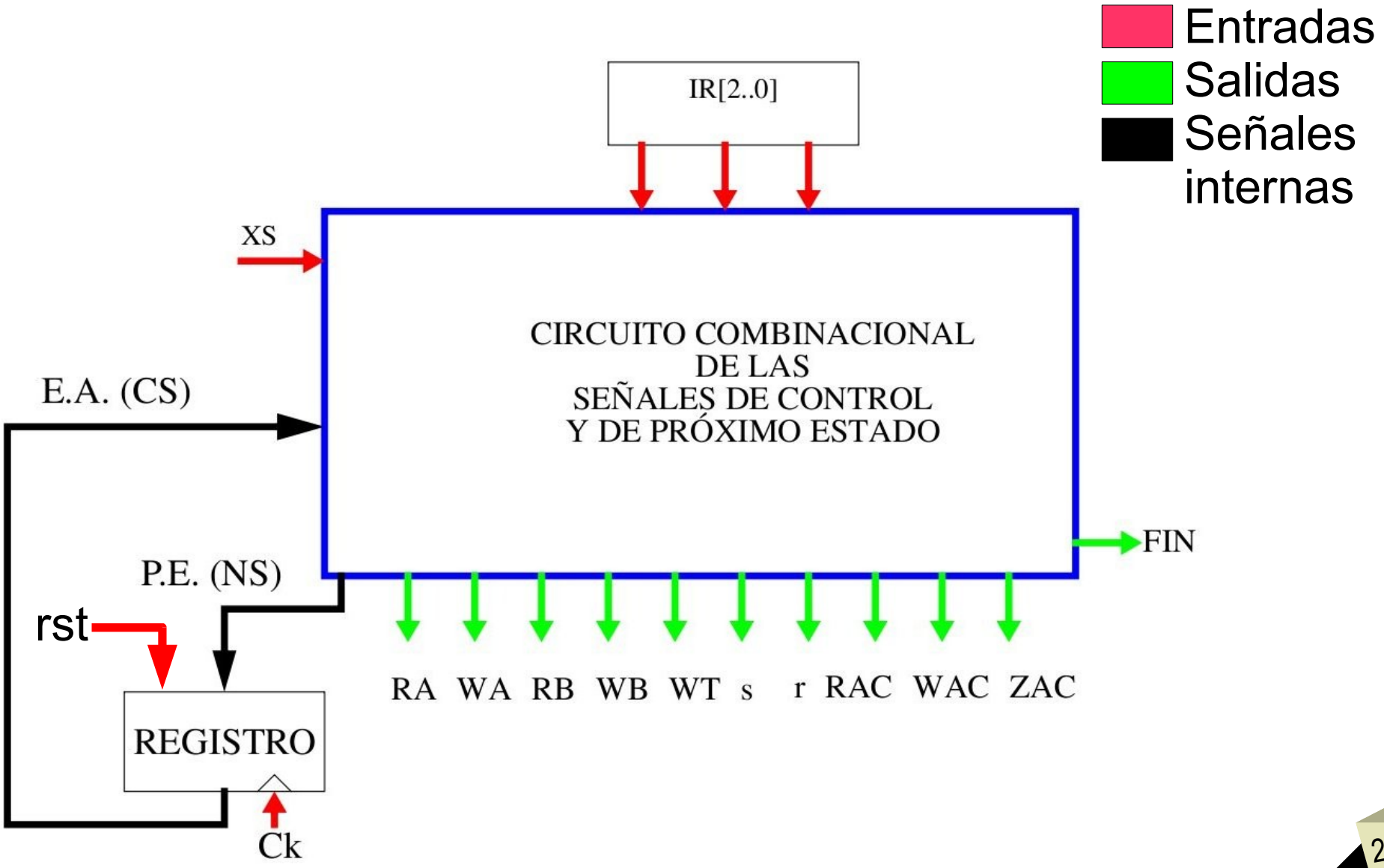
Diseñando en Código VHDL

- Diseño de FSM:
  - -- PROCESO COMBINACIONAL: OBTIENE EL VALOR DE LAS SALIDAS
    -- EN CADA ESTADO

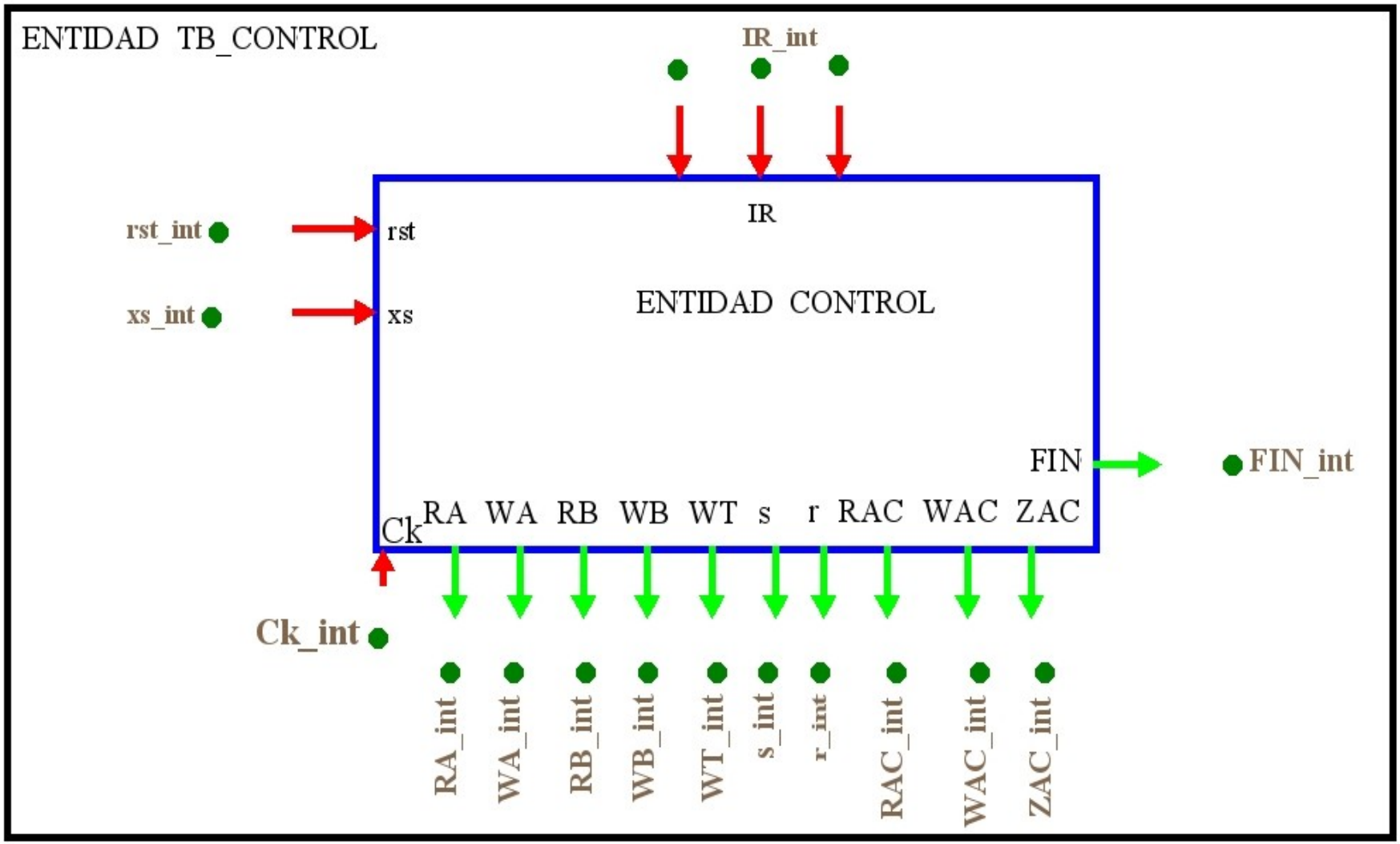
```
<NOMBRE PROCESO>: process (< todas las señales que aparezcan>)
        variables <todas_las_salidas>: std_logic;
          begin
          <variables_salidas> := <valor_defecto>;
              case CS is
                    when <actual> =>
                         <variables_salidas> := <valor_estado_actual>;
                          NS     <= <próximo_estado>;
              ..................
              end case;
              <señales de salida> <== <variables_salidas>;
          end process;
```

Diseñando en Código VHDL

- **Ejemplo: Diseño de la U.C de la calculadora del tema 1. Enlace en la web al fichero control_calculadora.vhd**



- **Ejemplo: Testbench para simular la unidad de control de la calculadora. Enlace en la web al fichero tb_control.vhd**



Diseñando en Código VHDL

- Diseño Estructural: Uniendo componentes
  - Se crea la entidad con los puertos de E/S
  - Se declaran los componentes y las señales internas que se necesiten para interconectar aquellos entradas/salidas de componentes que no aparezcan en la entidad en la arquitectura, antes de comenzar a definir el comportamiento de la misma (antes del “begin”)
  - Para definir el comportamiento de la arquitectura (tras el “begin”) se colocan los componentes, indicando que señal (interna o de la entidad) se conecta con cada señal del componente

Diseñando en Código VHDL

- Diseño Estructural: Uniendo componentes

```
Architecture <NOMBRE_ARQUITEC> of <NOMBRE_ENTIDAD> is
component <Nombre_componente>
port ( <entradas_salidas_del_componente> );
end component;
---------<resto de componentes>------
<señales_internas>
begin

<nombre_de_colocacion_componente>: <nombre_componente>
port(
<señal_1_compo> => <señal_entidad_o_interna_de_conexión>,
<señal_2_compo> => <señal_entidad_o_interna_de_conexión>,
--------------------);
----------<resto de componentes>---------
end <nombre_arquitectura>;
```

-

- **Ejemplo: Unión de los registros PC e IPC**
  - **Diseñar el Registro IPC**

IPcin	ipcout	Oper.	db
0	0	IPC<--IPC	H.I.
0	1	IPC<--IPC	[IPC]
1	0	IPC<--IN	IN
1	1	-----	---

- **Crear Testbench para IPC**
- **Unir PC e IPC (enlace al fichero union_pc_ipc.vhd de la web de la asignatura)**
- **Crear testbench y simular union_pc_ipc.vhd (tb_union_pc_ipc.vhd)**