

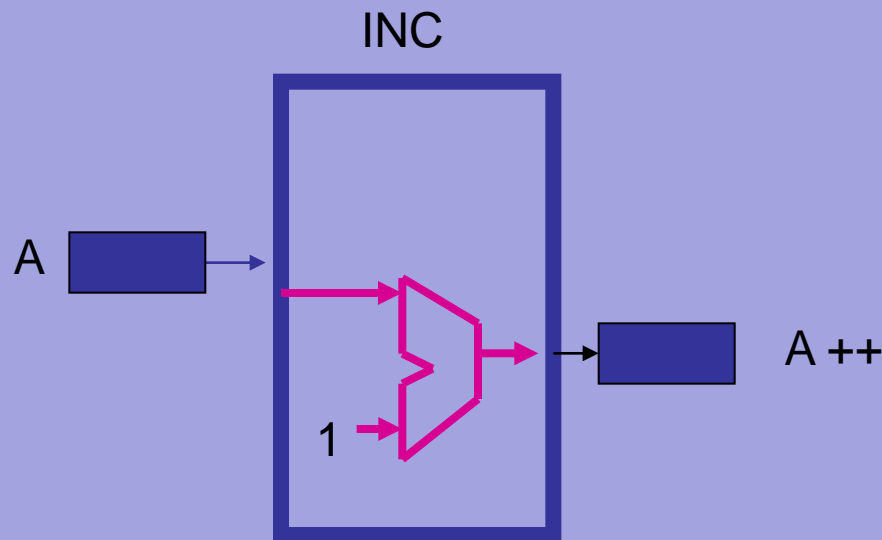
SISTEMAS DIGITALES VHDL

Fredy Hernán Riascos Campiño

Practica 1a:

Implementación de un incrementador:

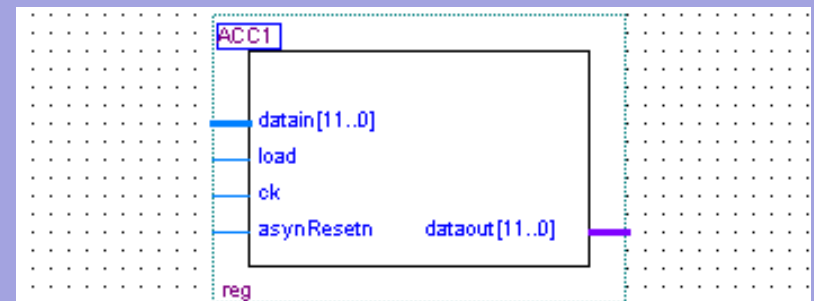
- Se he de implementar un modulo incrementador (INC), Este modulo no es mas que un sumador y un registro **y un sumador.**



Practica 1b:

Implementación de un registro:

- El registro carga con flanco de subida y debe tener las siguientes señales:
 - **asynResetn**: Raset activo bajo.
 - **load**: Señal de carga activa alta.
 - **ckl**: Señal de reloj
 - **datain**: Dato de entrada
 - **dataout**: dato de salida



Practica 1c:

Implementación de unidad aritmético lógica ALU:

- La ALU debe hacer las operaciones que aparecen en la tabla.
- Debe tener un registro de estado de 3 bits, que indique si el resultado de la operación es cero (z), negativo (N) o hay un desborde (C)

Practica 1c:

Implementación de unidad aritmético lógica ALU:

Operación (Sel)	Operacion
000	$C \leq B$
001	$C \leq A+B$
010	$C \leq A+B$
011	$C \leq B \& FF$
100	$C \leq B+1$
101	$C \leq A \gg 1$
110	$C \leq A \ll 1$

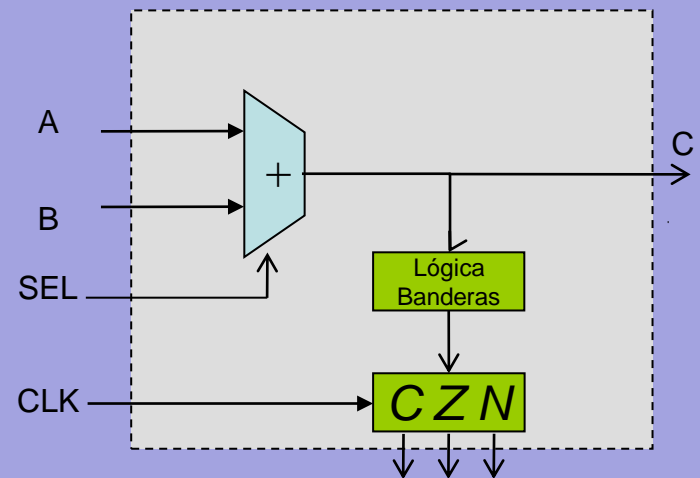


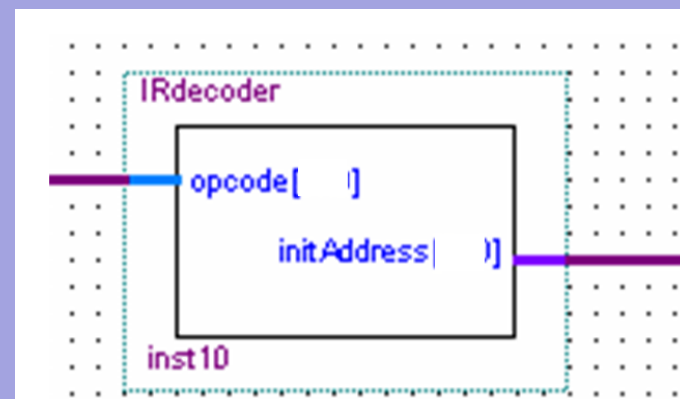
Diagrama de bloques ALU

Practica 1d:

Implementación de un IRcoder:

- Es un circuito combinacional cuya función lógica esta dada por la siguiente tabla de verdad:

opcode	initAddress
0000	"001000"
0001	"001101"
0010	"010010"
0011	"010101"
0100	"011001"
0101	"011100"
Otro caso	"XXXXXX"



Introducción al VHDL

- **VHDL** es el acrónimo que representa la combinación de **VHSIC** y **HDL**
- **VHSIC** es el acrónimo de *Very High Speed Integrated Circuit* y **HDL** es a su vez el acrónimo de *Hardware Description Language*.
- En términos generales VHDL es un lenguaje de descripción de hardware (Circuitos digitales)

Introducción al VHDL

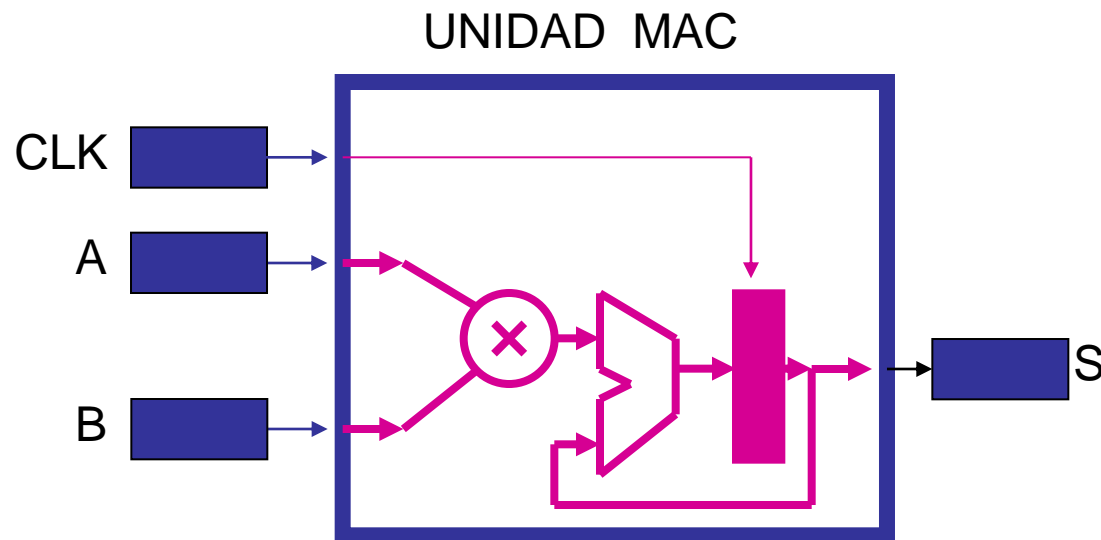
- **VHDL** Inicial mente fue impulsado por el departamento de defensa de los estados unidos en la década de los 80s
- Mas tarde se convirtió en un estándar y periódicamente se ha actualizado
 - IEEE VHDL-87
 - IEEE VHDL-93
 - IEEE VHDL-2001

Introducción al VHDL

- El lenguaje VHDL se utiliza para describir hardware.
- VHDL permite Implementar un diseño hardware a través de la descripción de su estructura.
- VHDL permite Implementar un diseño hardware a través de la descripción de su comportamiento.
- Su semántica permite describir aspectos como: entidad, conexiones, concurrencia, tiempos.
- No todo el código escrito en VHDL es sintetizable.

Entidades y Arquitecturas

- Entidad 
- Arquitectura 



Entidades

```
ENTITY mac IS
```

```
    GENERIC (ancho_bits :integer:=8 );
```

```
PORT(
```

```
    clk :IN          STD_LOGIC;
```

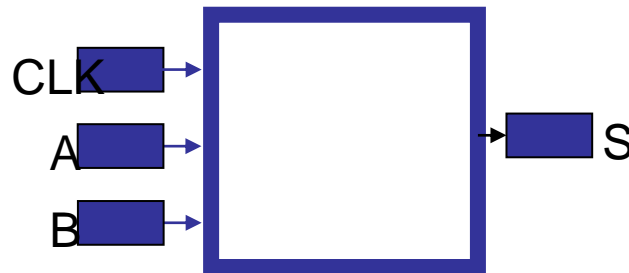
```
    a : IN          STD_LOGIC_VECTOR(ancho_bits-1 downto 0);
```

```
    b : IN          STD_LOGIC_VECTOR(ancho_bits-1 downto 0);
```

```
    s : OUT         STD_LOGIC_VECTOR(ancho_bits*2-1 downto 0)
```

```
);
```

```
END mac;
```



Arquitectura

Hay dos tipos de descripción para la arquitectura:

- **Estructural** : la descripción se basa en la interconexión de los elementos que componen el dispositivo. **Se utilizan componentes y/o paquetes**
- **Funcional (o de comportamiento)**: La descripción se basa en la tarea o funcionamiento del dispositivo. **Se utilizan procedimientos**

Arquitectura

- Sea cual sea el tipo de descripción el código se escribe en medio de las palabras `Architecture` y `end`, como se muestra abajo.

```
ARCHITECTURE a OF mac IS
```

```
    BEGIN
```

```
        -- Descripción del dispositivo
```

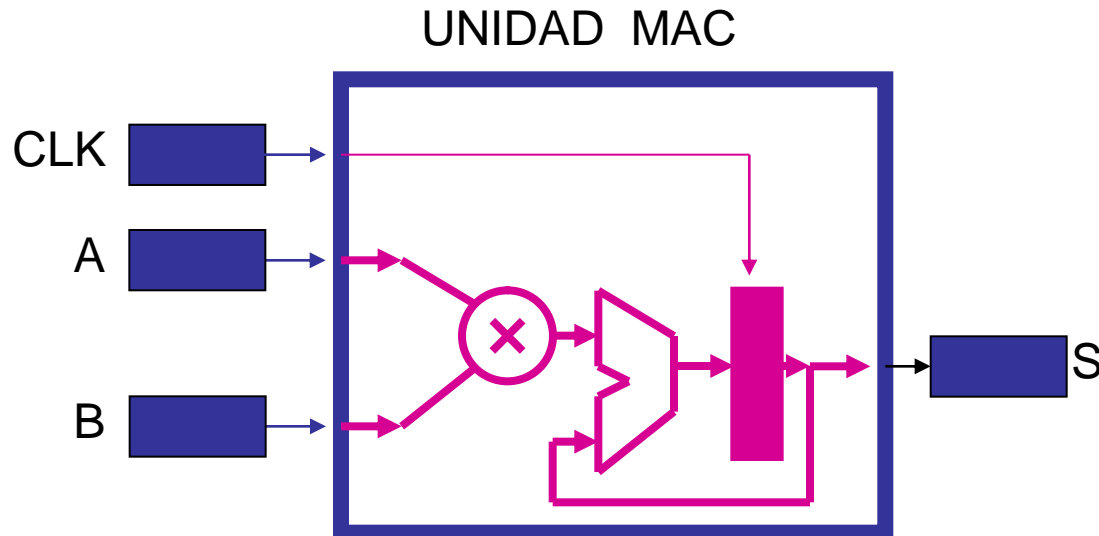
```
    END a;
```

Unidad MAC

EJEMPLO:

Implemente la arquitectura en VHDL para una unidad MAC:

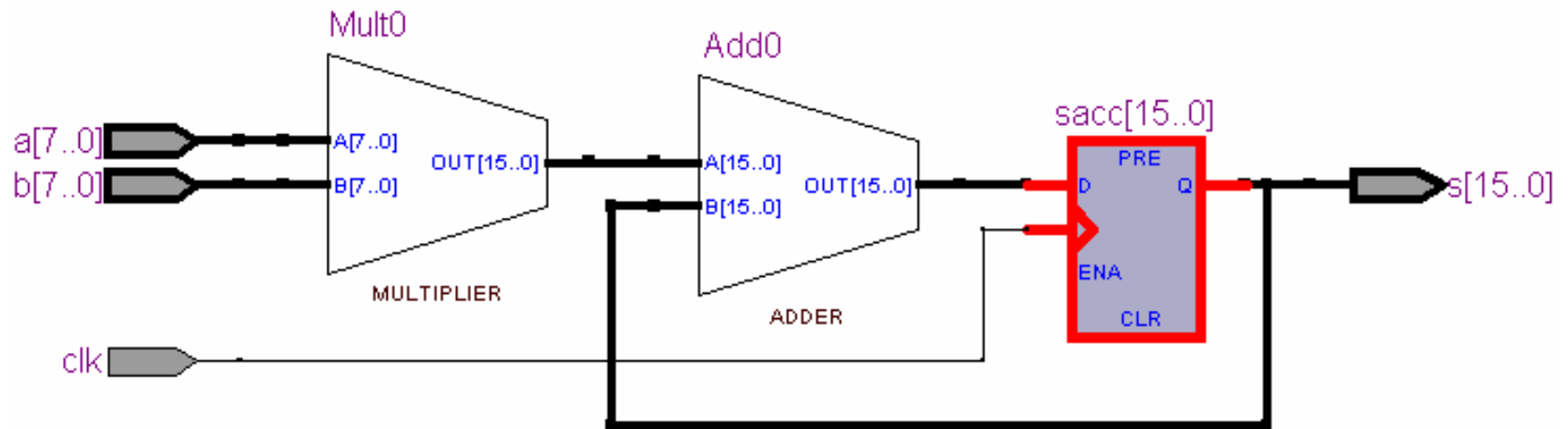
1. *Enfoque funcional* (proyecto mac1)
2. *Enfoque estructural* (proyecto mac2)



Arquitectura funcional unidad MAC1-VHDL

```
23 ARCHITECTURE funcional OF mac1 IS
24 -- CONSTANTES, SEÑALES
25 SIGNAL sacc,sadd: STD_LOGIC_VECTOR(ancho_bits*2-1 downto 0);
26
27 BEGIN
28 unidad_mac: PROCESS (clk,a,b,sacc,sadd)
29 -- VARIABLES
30
31 BEGIN
32     if clk'event and clk='1' then
33         sacc<= std_logic_vector(signed(a)*signed(b) + signed(sacc));
34     end if;
35     s<=sacc;
36 END PROCESS unidad_mac;
37
38 END funcional;
```

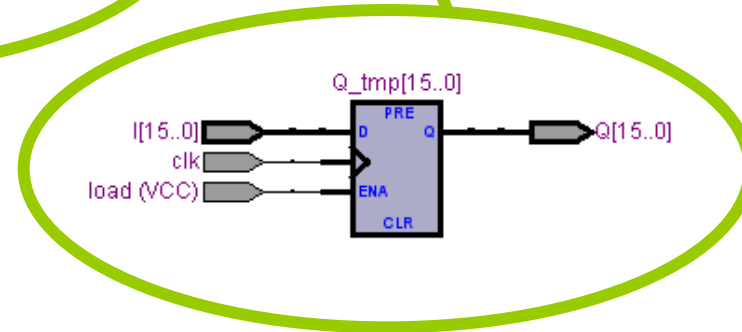
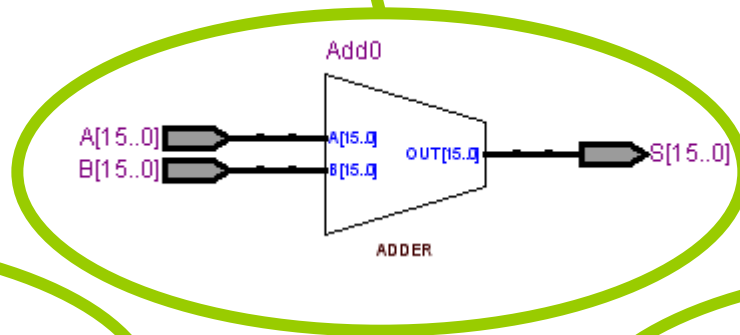
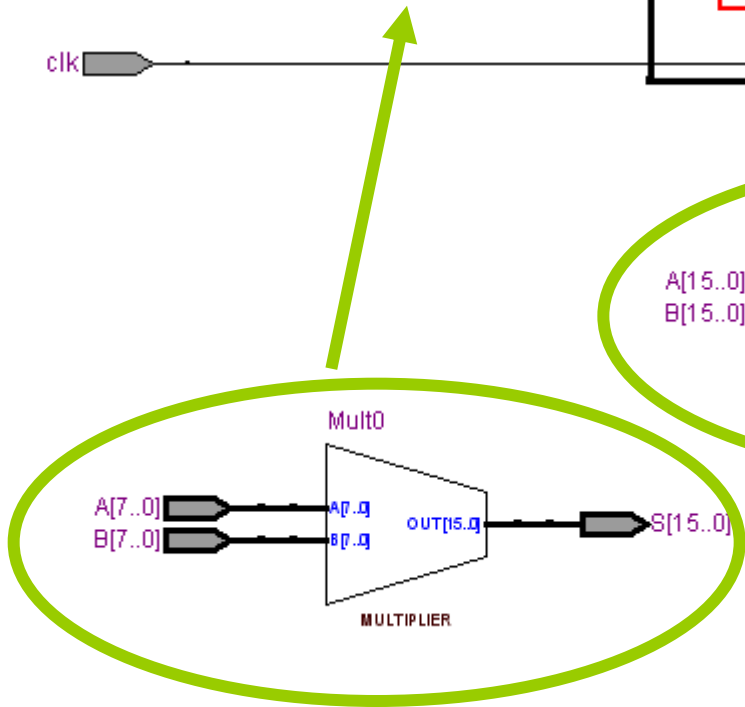
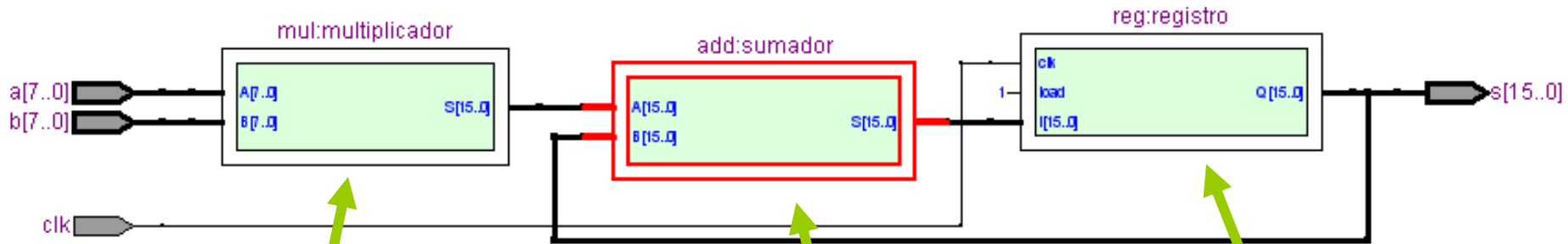
Arquitectura funcional unidad MAC1-RTL síntesis



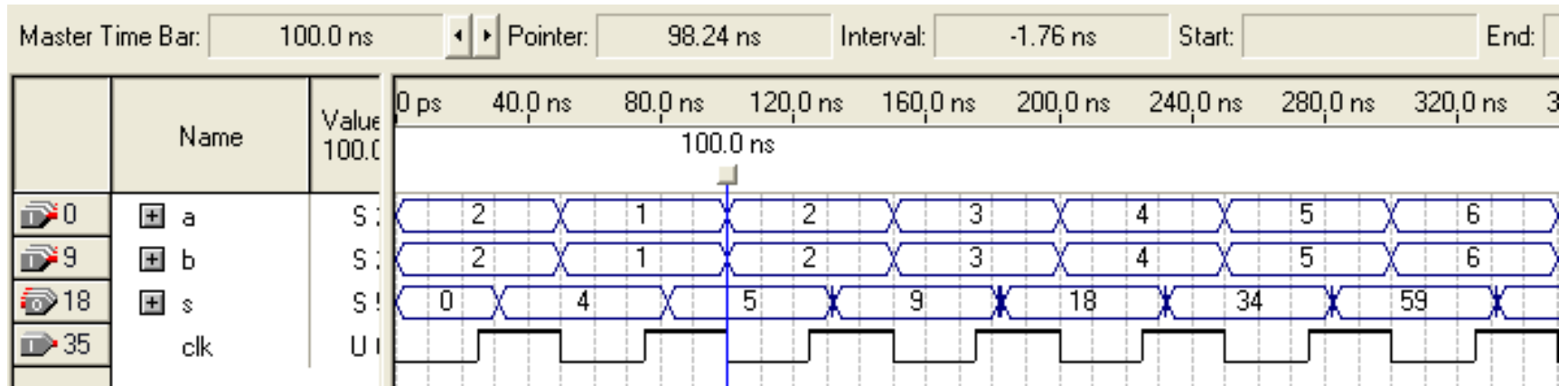
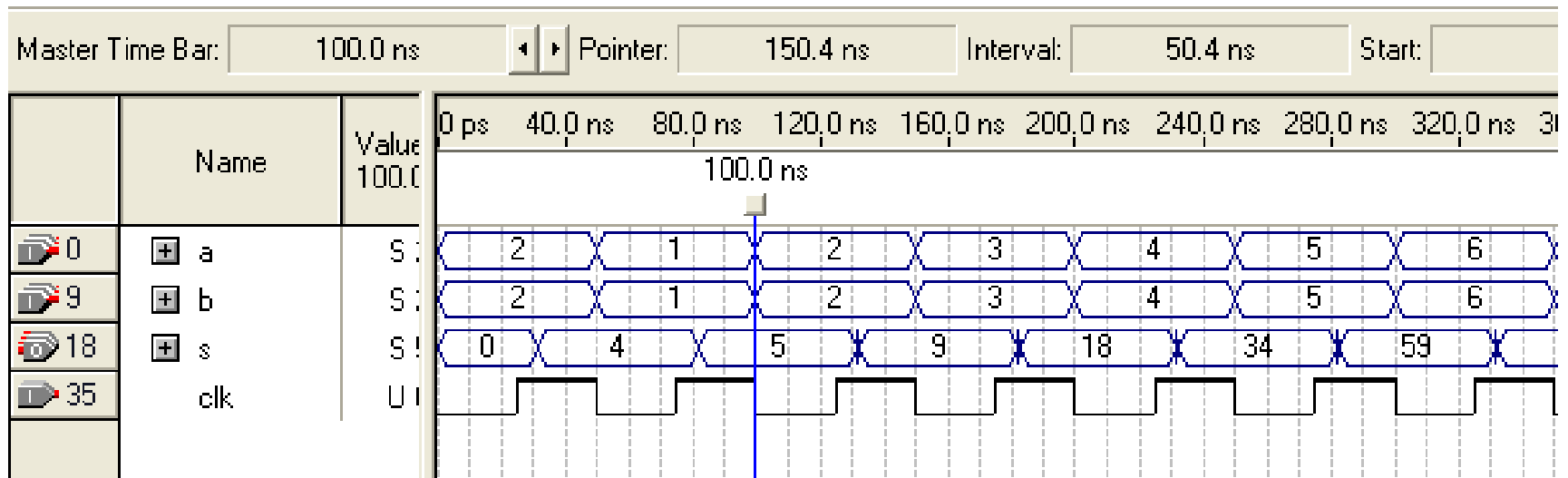
Arquitectura estructural unidad MAC2-VHDL

```
24 ARCHITECTURE estructural OF mac2 IS
25 -- CONSTANTES, SEÑALES
26 SIGNAL smul,sadd, sreg: STD_LOGIC_VECTOR(ancho_bits*2-1 downto 0);
27 BEGIN
28
29 -- CONEXION MULTIPLICADOR
30 multiplicador: mul
31 GENERIC MAP(width_dataa =>ancho_bits,width_datab =>ancho_bits)
32
33 PORT MAP (A=>a, B=>b, S=>smul);
34
35 -- CONEXION SUMADOR
36 sumador: add
37 GENERIC MAP(width_dataa =>ancho_bits*2,width_datab =>ancho_bits*2)
38
39 PORT MAP (A=>smul, B=>sreg, S=>sadd);
40
41 --CONEXION REGISTRO ACUMULADOR
42 registro: reg
43 GENERIC MAP(n=>ancho_bits*2)
44 PORT MAP (I=>sadd, clk=>clk, load=>'1', Q=>sreg);
45
46 s<=sreg;
47
48 END estructural;
```

Arquitectura estructural unidad MAC2-VHDL



Simulaciones MAC1 y MAC2



1. Objetos y tipos

Objetos y tipos

- **Objeto:** Un objeto en VHDL es un elemento que guarda el valor de un tipo de dato determinado.
 - **Constantes**
 - **Variables**
 - **Señales**
 - **Archivo** (*No sintetizable*)

Objetos y tipos

- Un tipo de dato es definido por:
 - Un conjunto de valores que pueden ser asignados al objeto
 - Un conjunto de operaciones que pueden realizadas con el objeto.
- Existen tipos predefinidos
- Se puede crear nuevos tipos
- Se puede crear subconjuntos de tipos.

Tipos predefinidos en VHDL

- **integer**: números $-(2^{31} - 1)$ a $2^{31} - 1$
- **boolean**: definido como (**false**, true).
- **bit**: ('1', '0')
- **bit_vector**: es un arreglo unidimensional de tipos bit

Crear nuevos tipos VHDL

- Para crear un nuevo tipo hay que realizar una declaración
- La declaración está formada por la palabra reservada **type**, un **identificador** para el nuevo tipo de dato y la **descripción del conjunto de valores** del tipo
- Los tipos pueden ser clasificados según las características de lo que van a determinar, (**descripción del conjunto de valores**):
 - Tipos enteros/reales
 - Tipos enumerados
 - Tipos array

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

- En este paquete se define nuevos tipos de datos. Mejora la descripción hardware
 - std_logic
 - std_logic_vector
- Toman los valores: ('1', '0', 'U', 'X', 'Z', 'W', 'L', 'H', ' - ')

```
library ieee;
```

```
use ieee.numeric_std.all;
```

- En este paquete se define dos nuevos tipos
 - `signed`
 - `Unsigned`
- Estos nuevos tipos soportan los operadores aritméticos `+`, `-`, `*`, `/`.
- Para usar estos operadores con tipos `std_logic_vector`, se debe convertir estos tipos a `signed` o `unsigned` a través de las funciones `unsigned()` y `signed()`.
- De esta forma se tiene control sobre la longitud de palabra de las operaciones.

Tipos de datos aplicables a las diferentes operaciones

Operator	Description	Data type of operand a	Data type of operand b	Data type of result
a ** b	exponentiation	integer	integer	integer
abs a	absolute value	integer		integer
not a	negation	boolean, bit, bit_vector		boolean, bit, bit_vector
a * b	multiplication	integer	integer	integer
a / b	division			
a mod b	modulo			
a rem b	remainder			
+ a	identity	integer		integer
- a	negation			
a + b	addition	integer	integer	integer
a - b	subtraction			
a & b	concatenation	1-D array, element	1-D array, element	1-D array

Tipos de datos aplicables a las diferentes operaciones

a sll b	shift-left logical	bit_vector	integer	bit_vector
a srl b	shift-right logical			
a sla b	shift-left arithmetic			
a sra b	shift-right arithmetic			
a rol b	rotate left			
a ror b	rotate right			
a = b	equal to	any	same as a	boolean
a /= b	not equal to			
a < b	less than	scalar or 1-D array	same as a	boolean
a <= b	less than or equal to			
a > b	greater than			
a >= b	greater than or equal to			
a and b	and	boolean, bit,	same as a	same as a
a or b	or	bit_vector		
a xor b	xor			
a nand b	nand			
a nor b	nor			
a xnor b	xnor			

OBJETOS: SEÑALES

- Las señales se utilizan como buses que interconectan diferentes módulos VHDL, sobretodo en el esquema estructural.
- También funcionan como registros, sobretodo cuando se utiliza el esquema funcional y en procedimientos secuenciales.
- Dentro de un procedimiento las señales sólo se actualizan al terminar el proceso en el que se usan
- Las señales deben declararse entre `architecture` y `begin`. También pueden declararse en un paquete.

OBJETOS: SEÑALES

```
ARCHITECTURE a OF __entity_name IS
    signal operando1 : int8:=0;
    signal operando2 : std_logic_vector(7 downto 0);

BEGIN
-- Definicion de la arquitectura

END a;
```

OBJETOS: VARIABLES

- Las variables se utilizan en los procedimientos almacenar valores, contadores, temporales.
- Las variables se actualizan instantáneamente, es decir, su valor cambia en el momento de la asignación.
- Las variables deben declararse entre `process` y `begin`.

OBJETOS: VARIABLES

```
process_label:  
PROCESS (signal_name, signal_name)  
    variable contador : int8 := 0;  
    variable vector : arreglo_int8;  
BEGIN  
  
-- Definición del procedimiento  
  
END PROCESS process_label;
```

OBJETOS

Diferencias entre variables y señales se profundisara en c3-vhdl

- Ejemplo: *sig_var*
- Ejemplo: *constantes*

2. Asignación concurrente y Circuitos Combinacionales

Asignación concurrente

- Hay tres formas para realizar la asignación concurrente de señales.
 - Simple
 - Condicional
 - Selectiva
- En general este tipo de instrucciones se utiliza en circuitos puramente **combinacionales**.

Asignación simple

Se utiliza el operador asignación:

```
status <= '1';
```

```
even <= (p1 and p2) or (p3 and p4);
```

```
arithout <= a + b + c - 1;
```

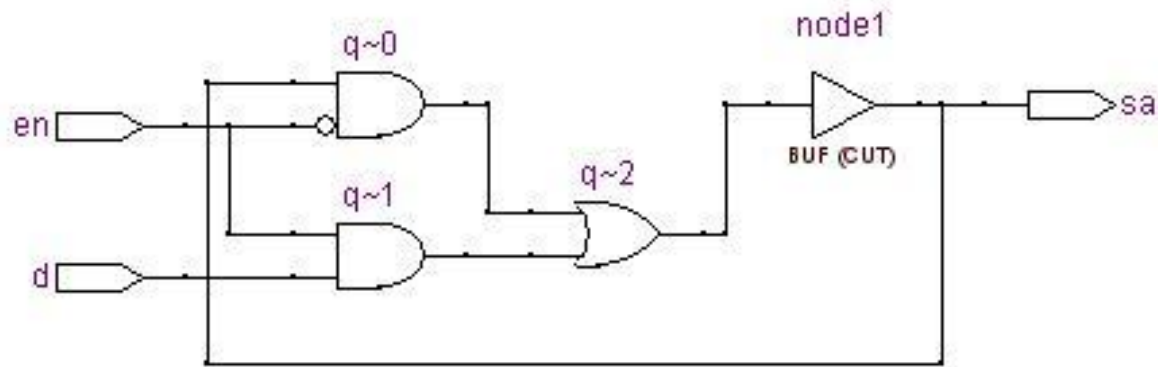
Asignación simple

- No debe permitirse retroalimentaciones de señales, pues el circuito deja de ser combinacional.

$q \leq (q \text{ and } (\text{not } en)) \text{ or } (d \text{ and } en);$

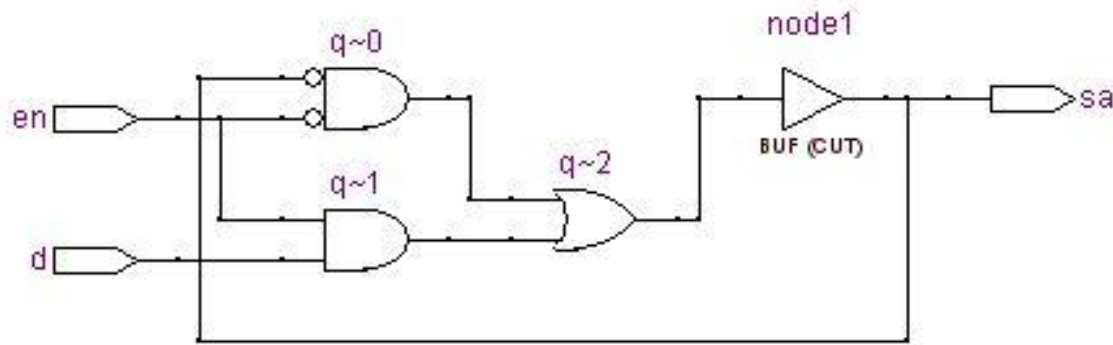
Asignación simple

- La salida Sa depende del estado anterior $q \sim 0$



Asignación simple

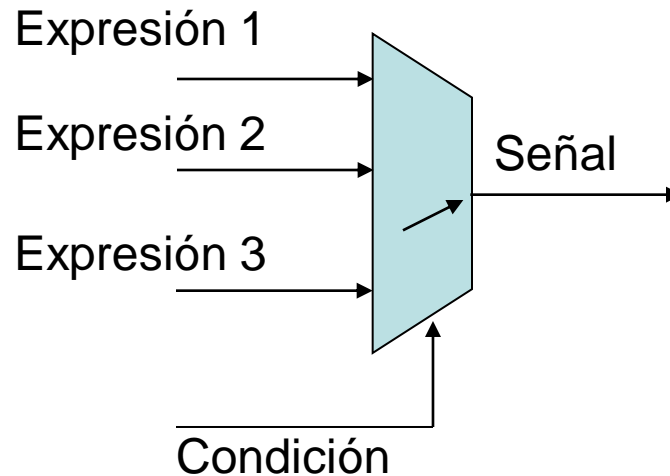
$q \leq ((\text{not } q) \text{ and } (\text{not } \text{en})) \text{ or } (\text{d and } \text{en});$



Si $\text{en} = '0'$ la señal de salida es inestable.

Asignación condicional WHEN-ELSE

```
Signal <= expresión WHEN expresion_booleana ELSE  
expresión WHEN expresion_booleana ELSE  
expresión;
```

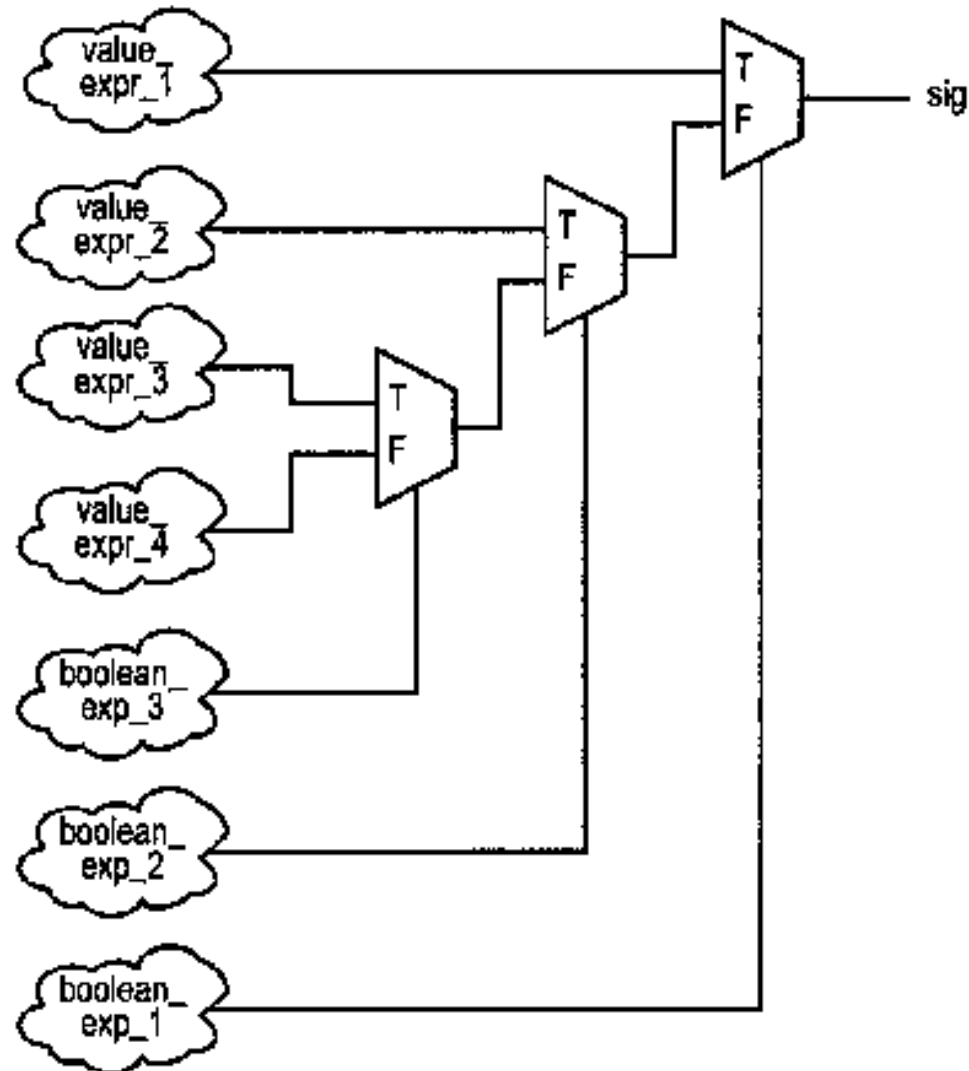


Asignación WHEN-ELSE

Un multiplexor de 4 entradas a una salida

```
x <= a when ( S ="00") else  
     b when ( S ="01") else  
     c when ( S ="10") else  
     d ;
```

Asignación WHEN-ELSE



Asignación WHEN-ELSE

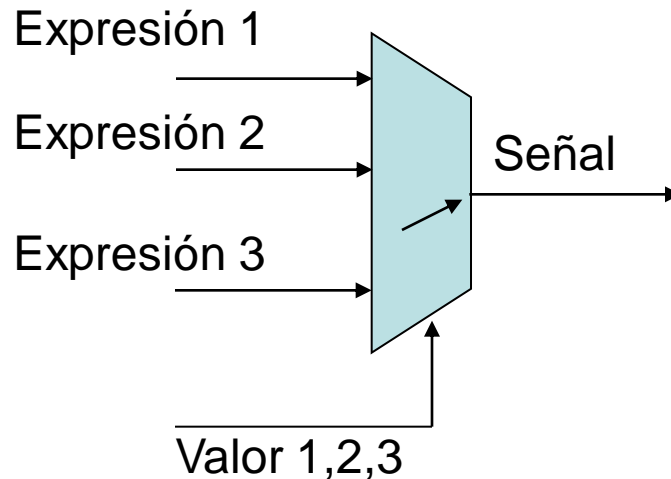
Una unidad aritmético lógica ALU, con: +, -,
, incremento, and, or, xor:



Asignación WITH-SELECT

WITH expresión SELECT

Señal <= expresión WHEN valor_constante,
expresión1 WHEN valor_constante,
expresión2 WHEN valor_constante,
expresión3 WHEN valor_constante;

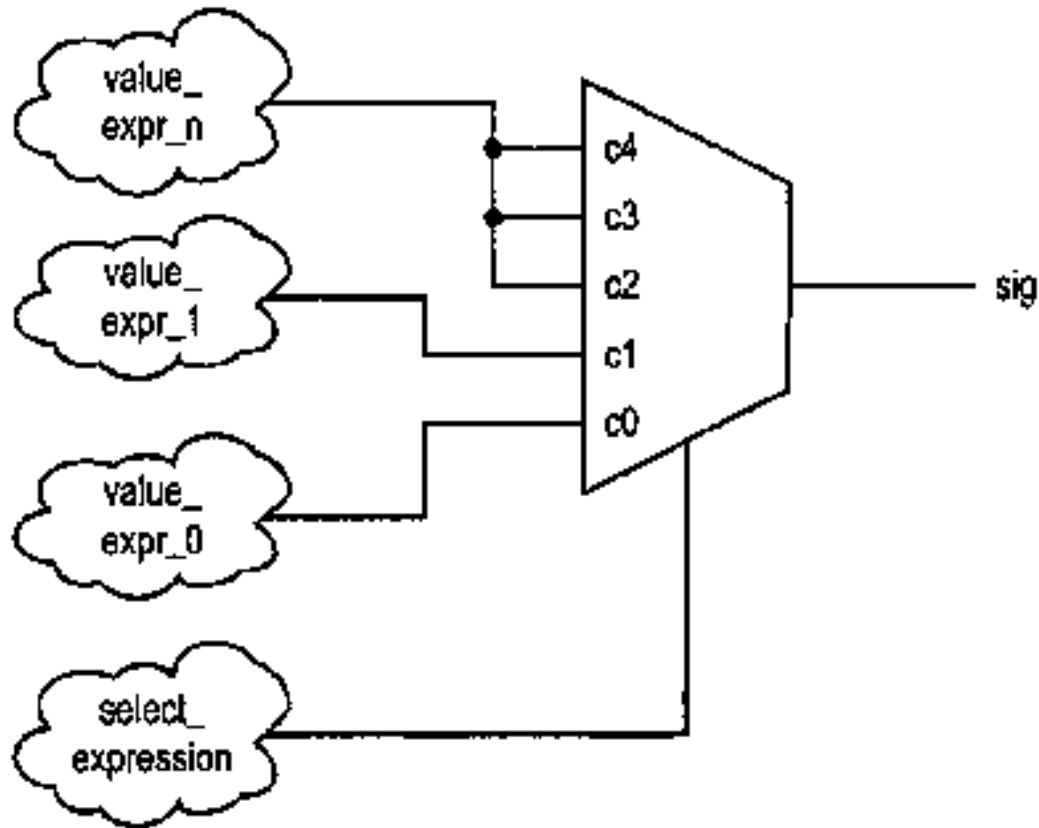


Asignación WITH-SELECT

withs s select

```
x <= a when "00" ,  
      b when "01" ,  
      c when "10" ,  
      d when others;
```

Asignación WITH-SELECT



Asignación WITH-SELECT

Una unidad aritmético lógica ALU, con: +, -,
, incremento, and, or, xor:



Consideraciones generales

- Evite losos cerrados en la asignación concurrente de señales
- Piense en las asignaciones como conexiones.
- Asignación **WHEN-ELSE**: Prioridad
- Asignación **WITH-SELECT**: Multiplexor

3. Sentencias secuenciales en VHDL

Sentencias secuenciales en VHDL

- Este tipo de sentencias son similares a las instrucciones de un lenguaje de software.
- El principal objetivo de este tipo de sentencias es modelar un circuito a través de su **comportamiento**.
- Las instrucciones secuenciales se agrupan en un bloque de código llamado **process**.

Process en VHDL

- Un **process** contiene un conjunto de instrucciones que se ejecutan en forma secuencial.
- Sin embargo varios **process** se ejecutan en forma concurrente.
- Hay diferentes sentencias que se pueden utilizar dentro del proces.

Sentencias ejecutables en process

Sentencia if-else-elsif

Sentencia Case

Sentencia ciclo for

Otros.....

Process en VHDL

- Un **process** tiene una lista de sensibilidad compuesta por un conjunto de señales.
- Cuando una de las señales de la lista de sensibilidad cambia, las instrucciones dentro del *process* se ejecuta en forma secuencial.
- Un process **no es como una función que se invoca**, simplemente se activa o desactiva según la lista de sensibilidad.
- Las sentencias de asignación concurrente no tiene sentido dentro del process. (**when-else** y **with-select**)

ESQUEMA PROCESS

Declaración
de variables

Lista de
sensibilidad

```
process ( a , b , c )
```

```
-- Definición de variables
```

```
begin
```

```
y <= a and b and c ;
```

```
end process ;
```

Asignación de señales en un process

- En un **process** la asignación de una señales es tratada en forma diferente.
- Dentro de un process a una señal se le pueden asignar múltiples ítems. Solo la ultima asignación tiene efecto.
- En un process, si a una señal se le asigna una expresión, esta solo tiene efecto hasta que el process se active nuevamente.

EJEMPLO PROCESS

- Dentro de un process a una señal se le pueden asignar múltiples ítems. Solo la ultima asignación tiene efecto.

```
process ( a , b , c )  
-- Definición de variables  
begin  
y <= a and b;  
y <= c or s;  
end process;
```

Sentencia if-else-elseif

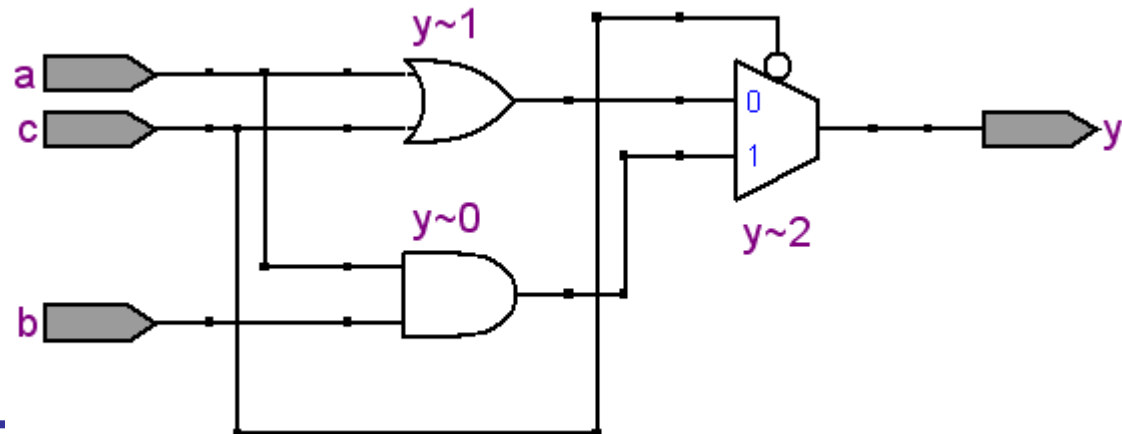
```
ARCHITECTURE funcional OF sif IS
BEGIN

process (a,b,c)
-- Definición de variables
begin

if c='0' then
    y<= a and b;
else
    y<= a or c;
end if;

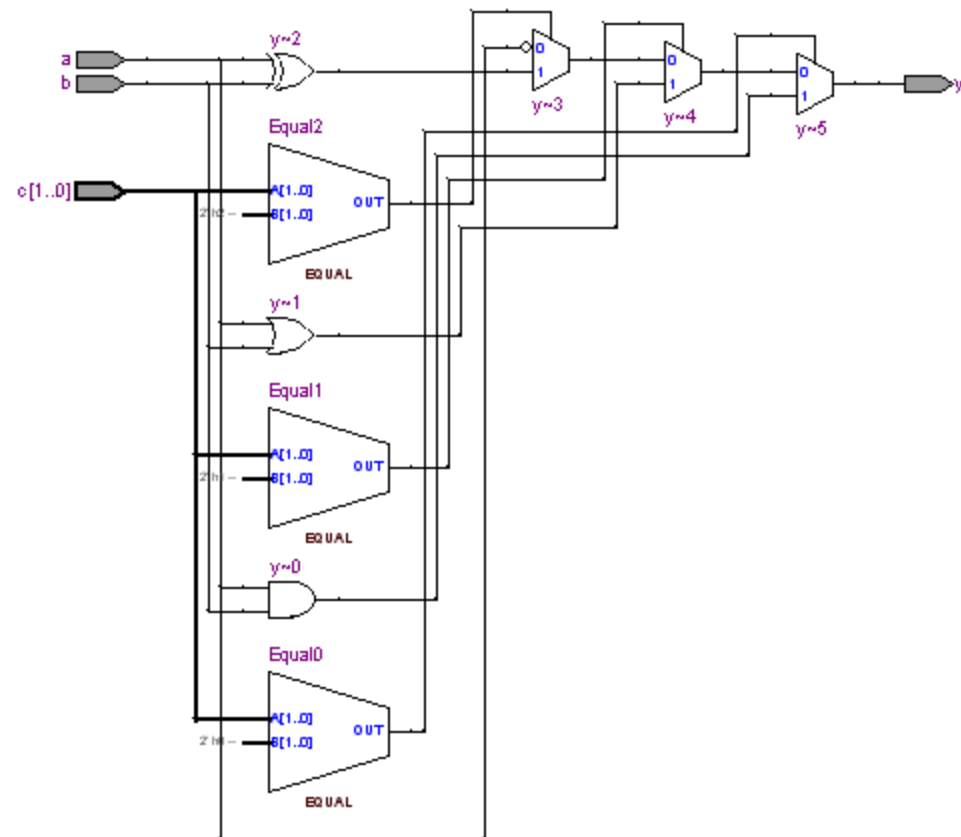
end process;
|
END funcional;
```

- Funciona como las sentencia condicional tradicional del lenguaje software



Sentencia if-else-elseif

```
16 ARCHITECTURE funcional OF sif IS
17 BEGIN
18
19 process (a,b,c)
20 -- Definición de variables
21 begin
22
23     if c="00" then
24         y<= a and b;
25     elsif c="01" then
26         y<= a or b;
27     elsif c="10" then
28         y<= a xor b;
29     else
30         y<= not a;
31     end if;
32 end process;
33 END funcional;
```



Sentencia if incompleta

Incompleto

Retroalimentación

Correcto

```
process (a ,b)
begin
  If ( a = b ) then
    eq <='1';
  end if;
end process;
```



```
process (a ,b)
begin
  If ( a = b ) then
    eq <='1';
  else
    eq<=eq;
  end if;
end process;
```



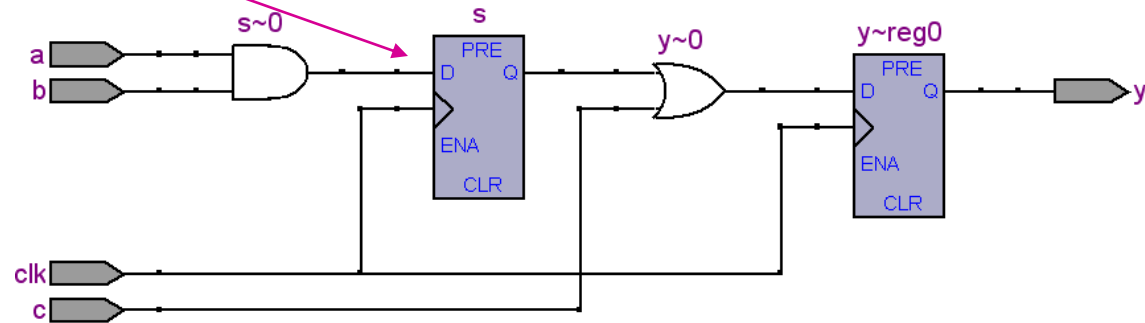
```
process (a ,b)
begin
  If ( a = b ) then
    eq <='1';
  else
    eq<='0';
  end if;
end process;
```



Asignación de señales en un process

```
15 ARCHITECTURE funcional OF sig IS
16 SIGNAL s : STD_LOGIC;
17 BEGIN
18
19 process ( a , b , c,clk )
20 begin
21
22     if clk'event and clk='1' then
23         s <= a and b;
24         y <= s or c;
25     end if;
26
27 end process;
28
29 END funcional;
```

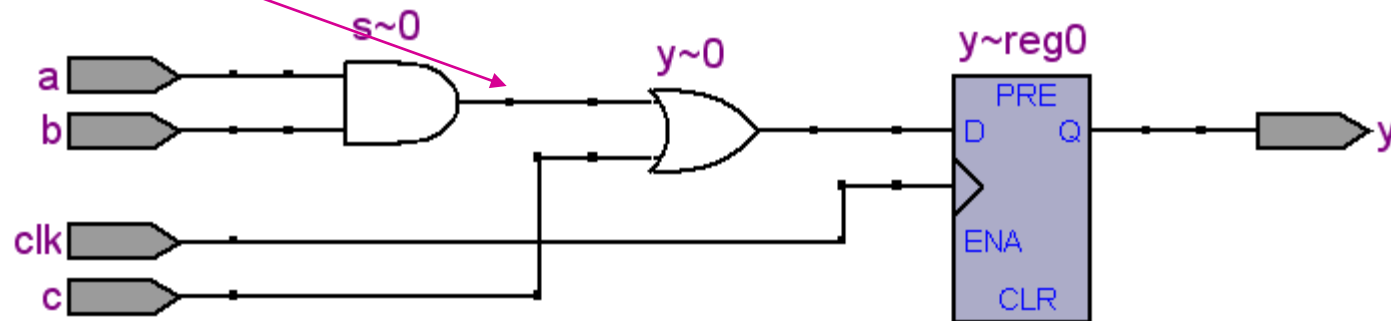
La asignación de una señal en una descripción sincrónica infiere la síntesis de un registro. Por lo que el valor solo estará disponible en el siguiente ciclo de reloj.



Asignación de señales en un process

```
15 ARCHITECTURE funcional OF sig IS
16 BEGIN
17
18 process ( a , b , c,clk )
19 -- Definición de variables
20 variable s:STD_LOGIC;
21 begin
22
23 if clk'event and clk='1' then
24     s := a and b;
25     y <= s or c;
26 end if;
27
28 end process;
29
30 END funcional;
```

Si se utilizan variables,
se elimina la posibilidad
de incluir registros

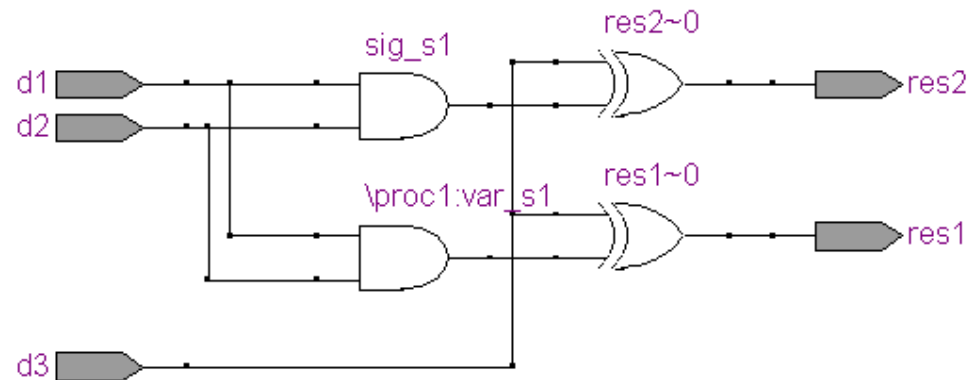


PROCESS VARIABLES-SEÑALES

- Para un process que describe un circuito combinacional el compilador trata variables y señales de igual forma.

```
proc1: process (d1,d2,d3)
variable var_s1: std_logic;
begin
    var_s1 := d1 and d2;
    res1 <= var_s1 xor d3;
end process;
```

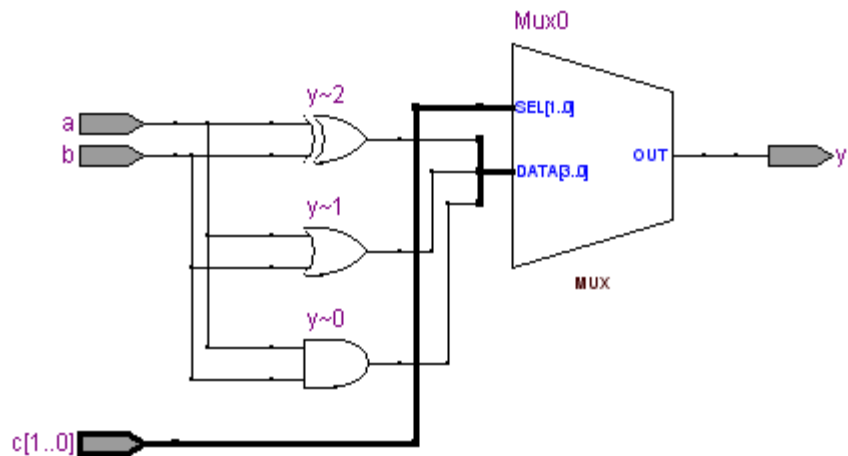
```
proc2: process (d1,d2,d3)
begin
    sig_s1 <= d1 and d2;
    res2 <= sig_s1 xor d3;
end process;
```



Sentencia case-when

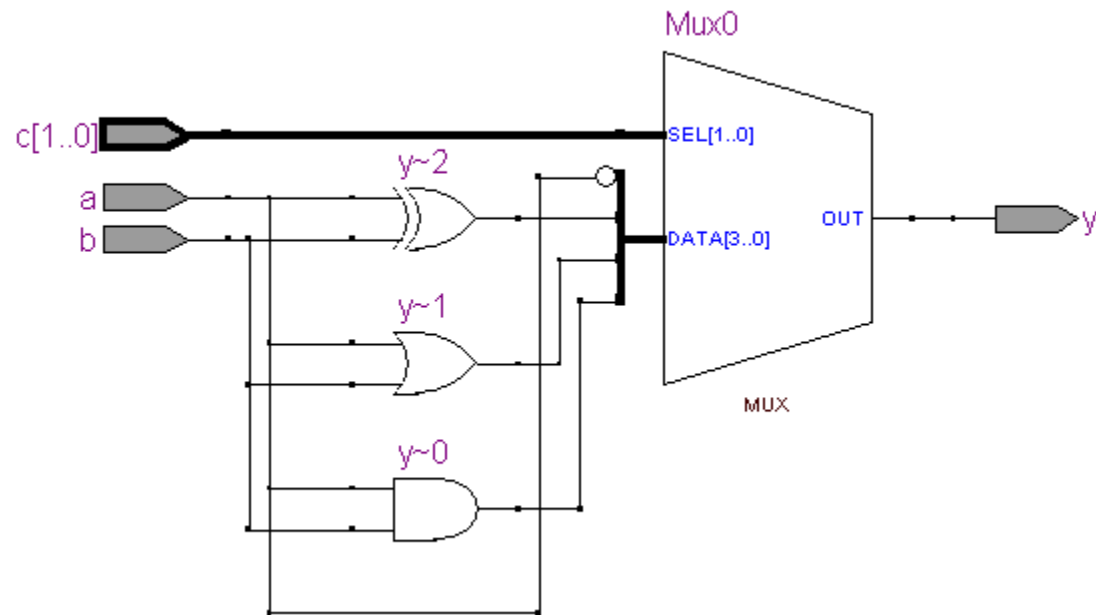
```
14 ARCHITECTURE funcional OF scase IS
15 BEGIN
16
17 process (a,b,c)
18 begin
19 case c is
20     when "00" =>
21         y<= a and b;
22     when "01" =>
23         y<= a or b;
24     when others =>
25         y<= a xor b;|
26     end case;
27
28 end process;
29 END funcional;
```

- Funciona como la sentencia switch-case del lenguaje C



Sentencia case-when

```
15 ARCHITECTURE funcional OF scase IS
16 BEGIN
17
18 process (a,b,c)
19 begin
20
21 case c is
22     when "00" =>
23         y<= a and b;
24     when "01" =>
25         y<= a or b;
26     when "10" =>
27         y<= a xor b;
28     when "11" =>
29         y<= not a;
30     end case;
31
32 end process;
33 END funcional;
```



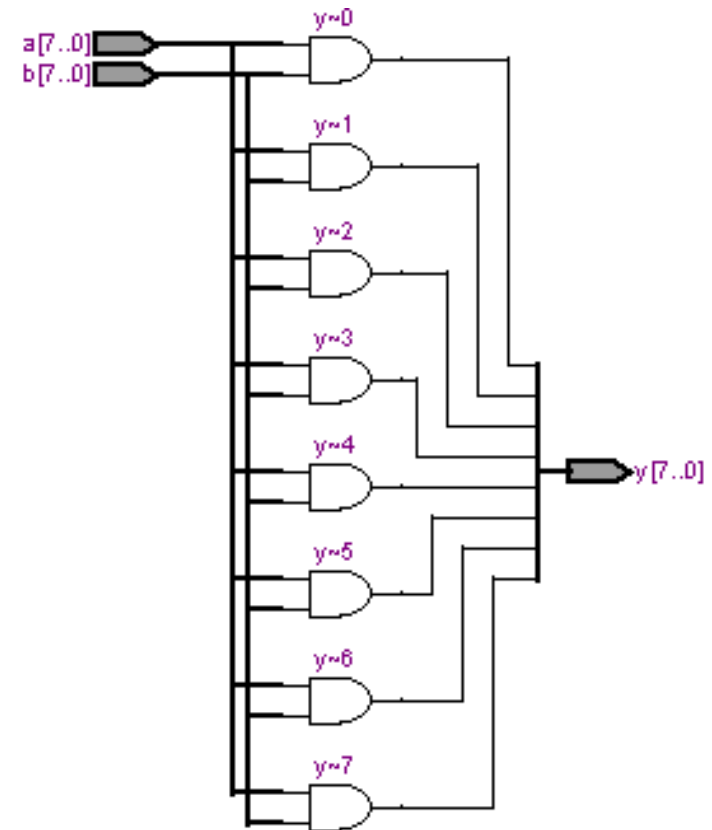
CICLO FOR

- Hay que ser cuidadoso en la manipulación de esta sentencia, su utilidad se restringe a la reproducción genérica de hardware.
- No se puede pretender utilizar un ciclo **for** de VHDL como si se tratara de un ciclo tradicional de un lenguaje software.

CICLO- FOR

- Modelado genérico de una compuerta and (hace lo mismo que $y \leq a \text{ and } b$)

```
19 ARCHITECTURE funcional OF loop1 IS
20 BEGIN
21
22 process (a,b)
23 begin
24
25 and_N:|
26 FOR k IN N-1 DOWNTO 0 LOOP
27     y(k) <= a(k) and b(k);
28 END LOOP and_N;
29
30 end process;
31
32 END funcional;
```



EJEMPLO: CICLO- FOR

```
2 library ieee;
3 use ieee.std_logic_1164.all;
4
5 ENTITY loop1 IS
6     GENERIC(
7         N: integer:=8
8     );
9
10    PORT(
11        a,b : IN  STD_LOGIC_VECTOR(N-1 downto 0);
12        y   : OUT STD_LOGIC_VECTOR(N-1 downto 0)
13    );
14 END loop1;
15
16 ARCHITECTURE funcional OF loop1 IS
17 BEGIN
18
19    process (a,b)
20    begin
21        FOR k IN 0 TO N-1 LOOP
22            y(k) <= a(k) and b(k);
23        END LOOP;
24    end process;
25
26 END funcional;
```

**Constantes genéricas,
favorece el modelamiento
parametrizado**

**Variable de control,
No hay que declararla**

**La cuenta es ascendente,
si fuera descendente
se utiliza *downto***

4. DEFINICIÓN DE TIEMPOS

Tiempo en: Circuitos combinacionales

- EL TIEMPO TPD (pin-to-pin delay)
 - Es el tiempo de propagación de un pin a otro en un circuito combinacional. El peor caso, se conoce generalmente como ruta critica.

Timing Analyzer Summary									
	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1	Worst-case tpd	N/A	None	11.429 ns	B[0]	M[7]	--	--	0
2	Total number of failed paths								0

Tiempo: circuitos secueanciales

- $T_{cq} = T_{co}$ (*clock to output delay*)
 - Tiempo en que tarda una señal en propagarse a la salida, después de que ha ocurrido el flanco de reloj
- $T_{setup} = T_{su}$ (*clock setup time*)
 - Tiempo que debe estar una señal a la entrada del registro antes del flanco de reloj.
- $T_{hold} = T_h$ (*clock hold time*)
 - Tiempo que debe estar una señal a la entrada del registro después del flanco de reloj.

Timing Analyzer Summary				
	Type	Slack	Required Time	Actual Time
1	Worst-case tsu	N/A	None	-0.049 ns
2	Worst-case tco	N/A	None	6.378 ns
3	Worst-case th	N/A	None	1.129 ns
4	Total number of failed paths			