Fundamentos de Computadores I

VHDL - Introducción

curso 2019-20

Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

Introducción

- ¿Que es HDL?
  - Lenguaje específicamente creado 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?
  - Poder descubrir problemas en el diseño antes de su implementación física.
  - Dado que la complejidad de los sistemas electrónicos crece exponencialmente, son necesarias herramientas que faciliten el trabajo en equipo y la simplificación del diseño.

“Entity” y “Architecture”

```
entity nombre_entity is
port (lista de puertos de entrada y salida);
end nombre_entity;
```

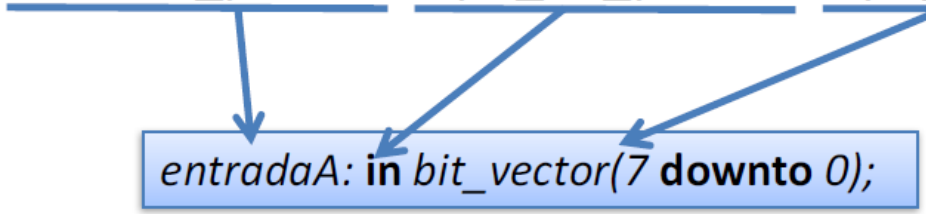
```
architecture circuito of nombre_entity is
-- señales
begin
-- programación
end architecture circuito;
```

Entity

```
entity nombre_entidad is
   port  (lista puertos de entrada y salida);
end nombre_entidad;
```

Lista puertos de entrada-salida

nombre_puerto: tipo_de_puerto tipo_de_señal; ...

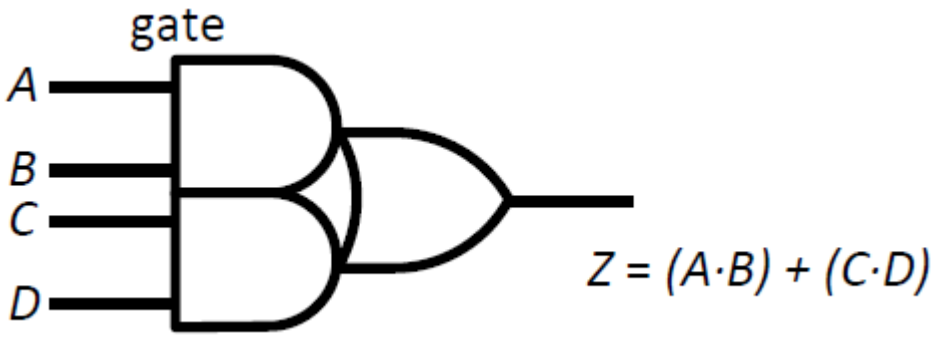


Descripción Estructural

- Esta descripción utiliza entidades descritas y compiladas previamente que llamaremos componentes. → "COMPONENT"
- de esta manera podemos reutilizar diseños previos, o realizar diseños para que se reutilicen en otros más complicados.
- Se declaran los componentes que se van a utilizar y después se realizan las conexiones entre los puertos.
- Las descripciones estructurales son útiles cuando se trata de diseños jerárquicos → Diseño "*bottom-up": empezamos diseñando los componentes más sencillos y los utilizamos como base para crear los más complejos.*

Ejemplo trivial

- Queremos implementar la siguiente puerta:



- Partimos de las definiciones de una puerta “AND” y “OR”.

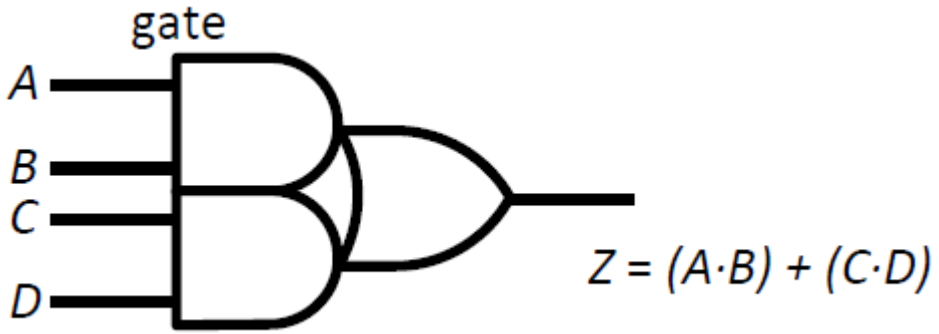
Componentes “AND” y “OR”

- De momento, sólo nos interesa los puertos en la definición de la entidad.

```
1
2  entity component_and is
3    port(
4      entrada1, entrada2: in bit;
5      salida: out bit
6    );
7  end entity;
8
9  architecture arch_and of component_and is
10 begin
11   salida <= '1' when entrada1 = '1' and entrada2 = '1' else '0';
12 end architecture;
13
14
15 entity component_or is
16   port(
17     entrada1, entrada2: in bit;
18     salida: out bit
19   );
20 end entity;
21
22 architecture arch_or of component_or is
23 begin
24   salida <= '1' when entrada1 = '1' or entrada2 = '1' else '0';
25 end architecture;
```

Diseño Bottom-Up

- Ahora construimos nuestro módulo “sumprod” (suma de productos) utilizando los componentes anteriores.
- Primero declaramos los puertos en la definición de la entidad:




```
27  entity sumprod is
28  port(
29    A, B, C, D: in bit;
30    Z: out bit
31  );
32  end entity;
```

Arquitectura

- En la arquitectura necesitamos declarar primero qué componentes vamos a usar y qué puertos tienen cada uno.

```
34  architecture arch_sumprod of sumprod is
35  component component_and
36  port(
37    entrada1, entrada2: in bit;
38    salida: out bit
39  );
40  end component;
41  component component_or
42  port(
43    entrada1, entrada2: in bit;
44    salida: out bit
45  );
46  end component;
47
48    signal sal_and1, sal_and2: bit;
```

Las salidas de las puertas “and” son señales internas de esta arquitectura.

Conexiones

- Finalmente, declaramos las conexiones entre los componentes y los puertos de nuestro módulo:

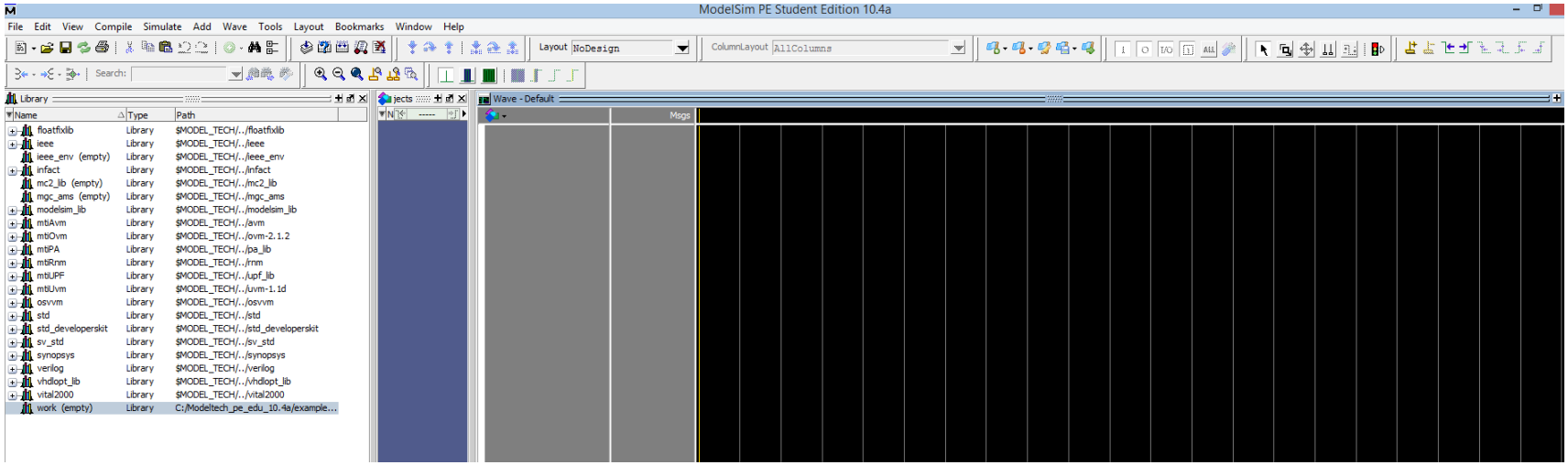
```
50  begin
51  and1_pm: component_and port map(
52    entrada1 => A,
53    entrada2 => B,
54    salida => sal_and1
55  );
56
57  and2_pm: component_and port map(
58    entrada1 => C,
59    entrada2 => D,
60    salida => sal_and2
61  );
62
63  or_pm: component_or port map(
64    entrada1 => sal_and1,
65    entrada2 => sal_and2,
66    salida => Z
67  );
68
69  end architecture;
```

Código Completo

```
27  entity sumprod is
28  port(
29    A, B, C, D: in bit;
30    Z: out bit
31  );
32  end entity;
33
34  architecture arch_sumprod of sumprod is
35  component component_and
36  port(
37    entrada1, entrada2: in bit;
38    salida: out bit
39  );
40  end component;
41  component component_or
42  port(
43    entrada1, entrada2: in bit;
44    salida: out bit
45  );
46  end component;
47
48    signal sal_and1, sal_and2: bit;
49
50  begin
51  and1_pm: component_and port map(
52    entrada1 => A,
53    entrada2 => B,
54    salida => sal_and1
55  );
56
57  and2_pm: component_and port map(
58    entrada1 => C,
59    entrada2 => D,
60    salida => sal_and2
61  );
62
63  or_pm: component_or port map(
64    entrada1 => sal_and1,
65    entrada2 => sal_and2,
66    salida => Z
67  );
68
69  end architecture;
```

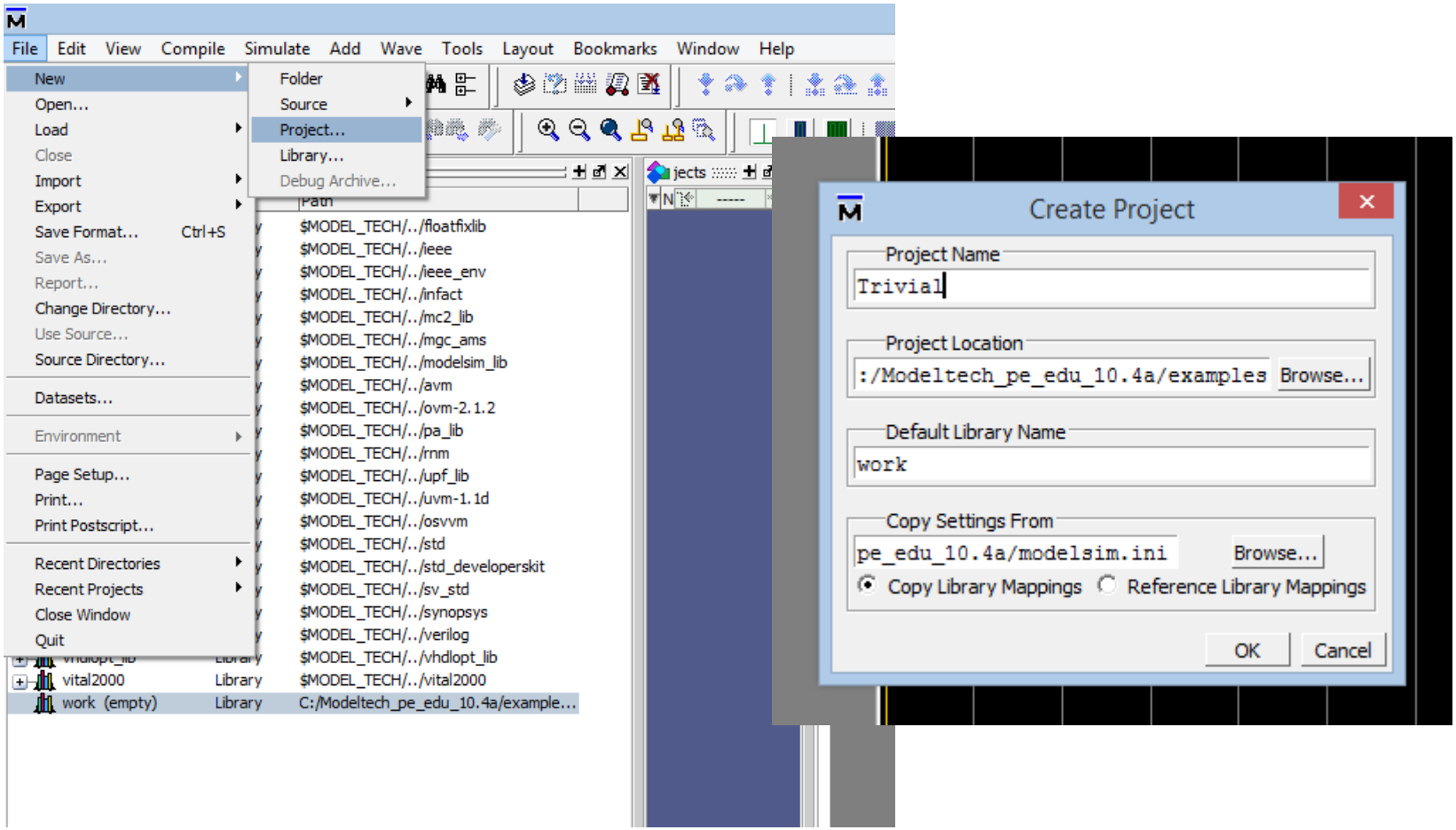
ModelSim

- La herramienta de simulación que vamos a utilizar se llama ModelSim:

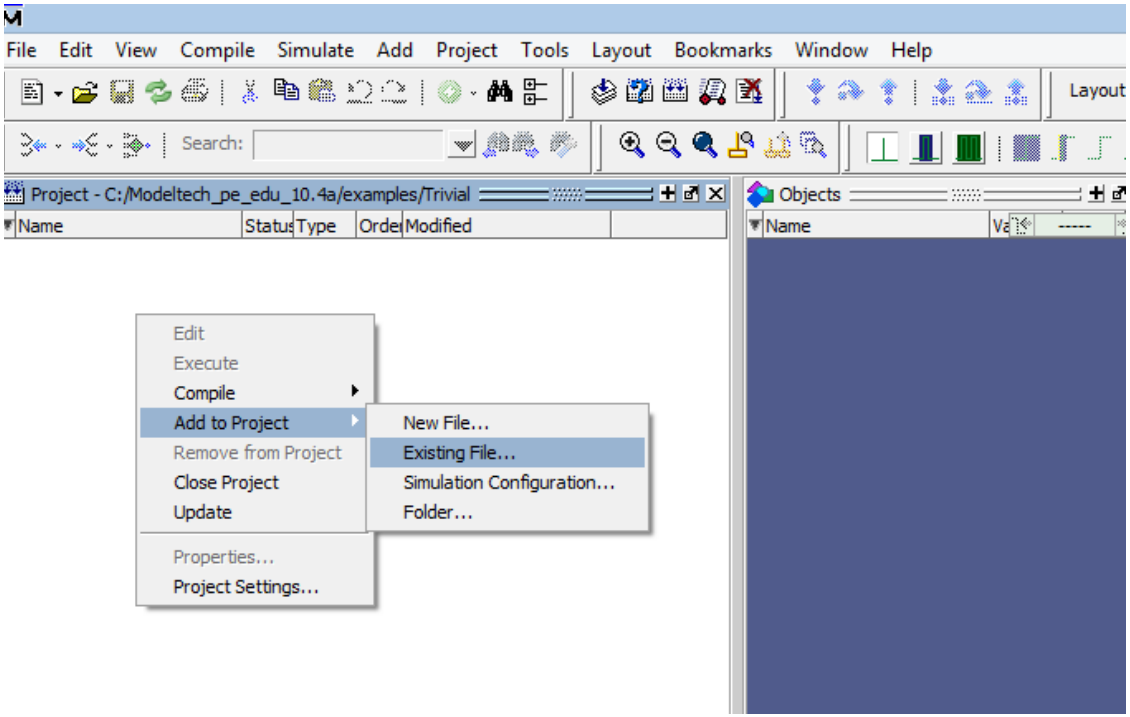


ModelSim

- Creamos un proyecto:

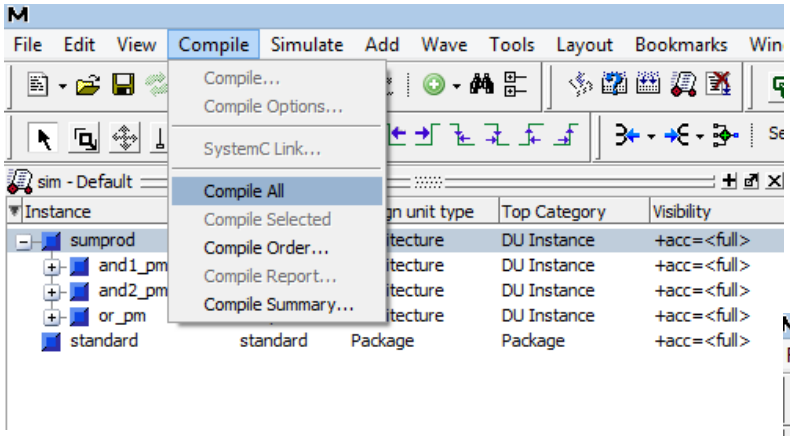


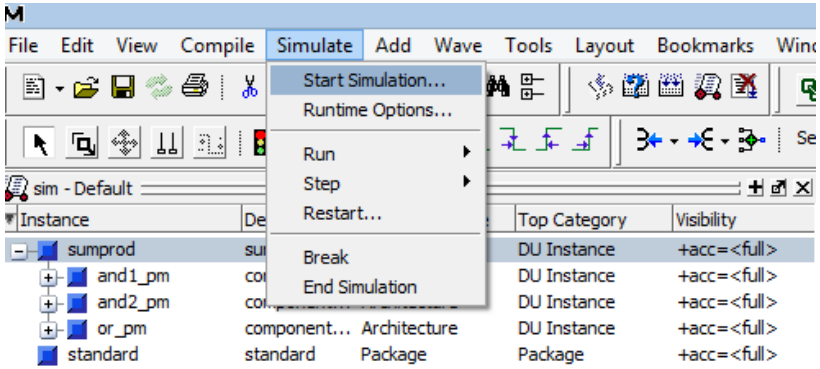
- Añadimos el fichero con nuestro código



Compilación

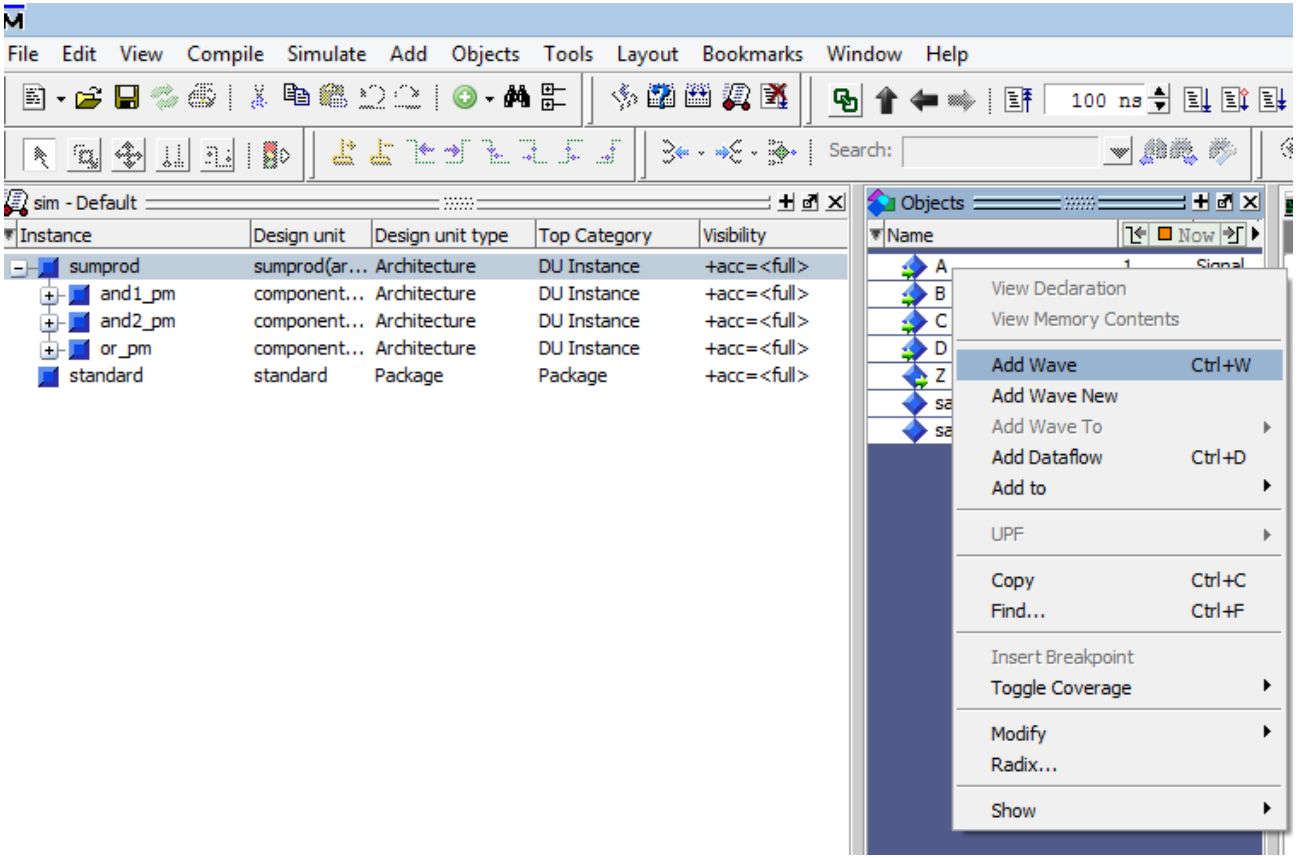
- Compilamos y empezamos la simulación:



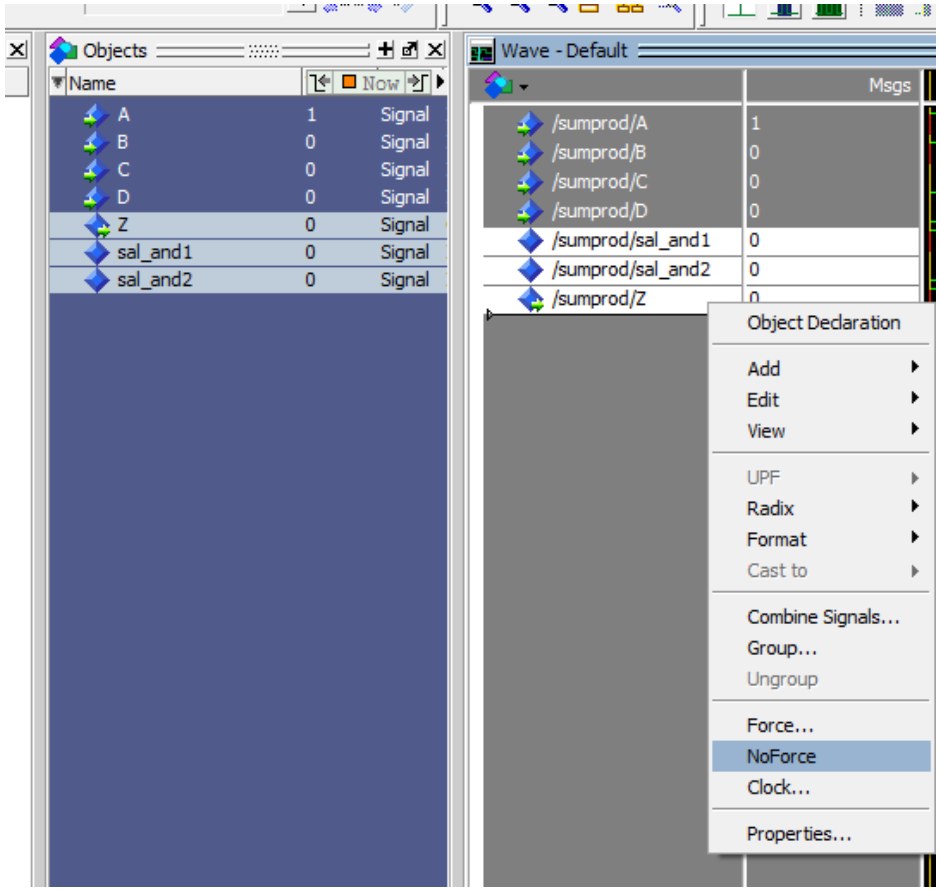


Simulación

- Añadimos las señales que queremos ver en la simulación. En este caso, todas.

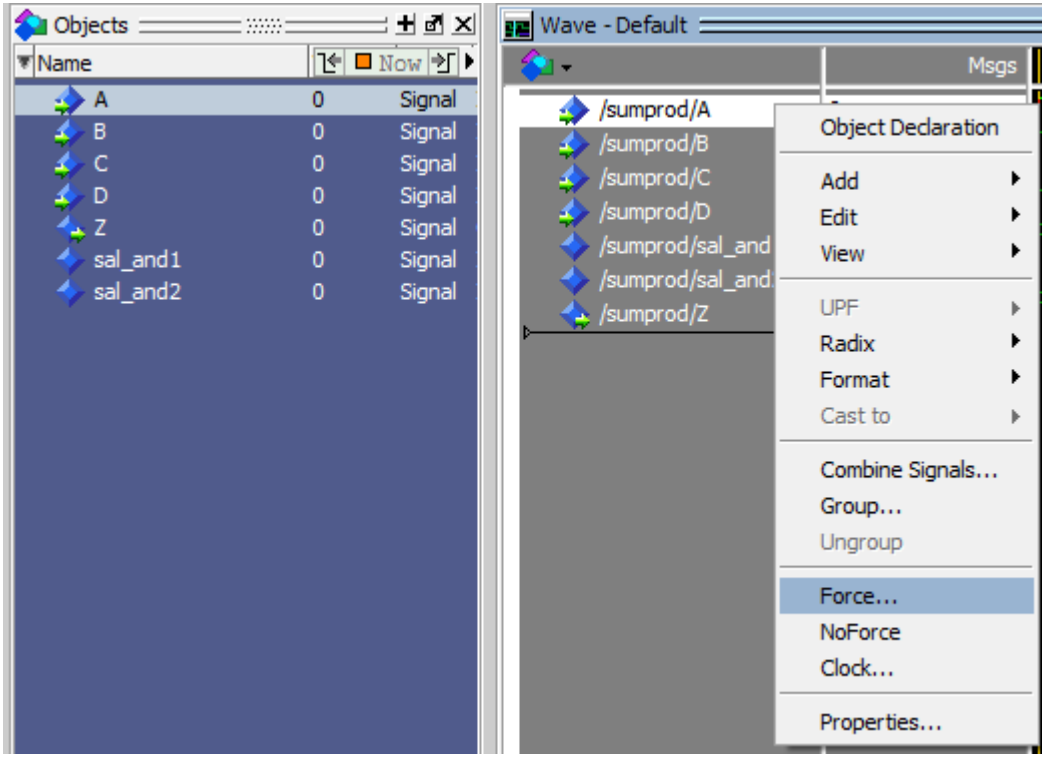


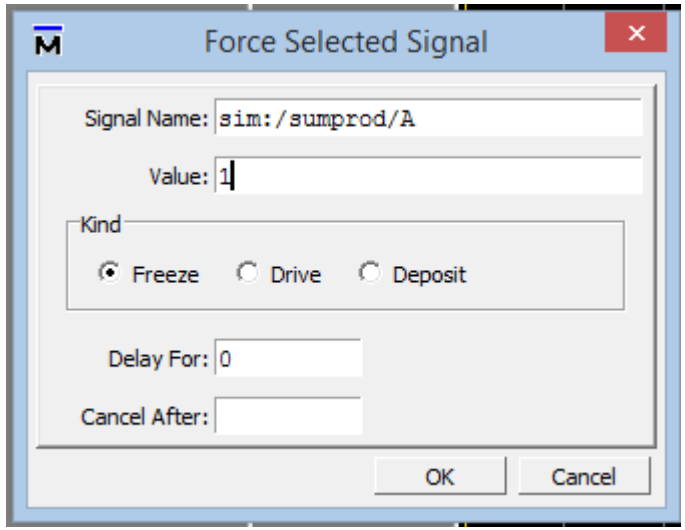
- No forzamos las salidas.



Entradas

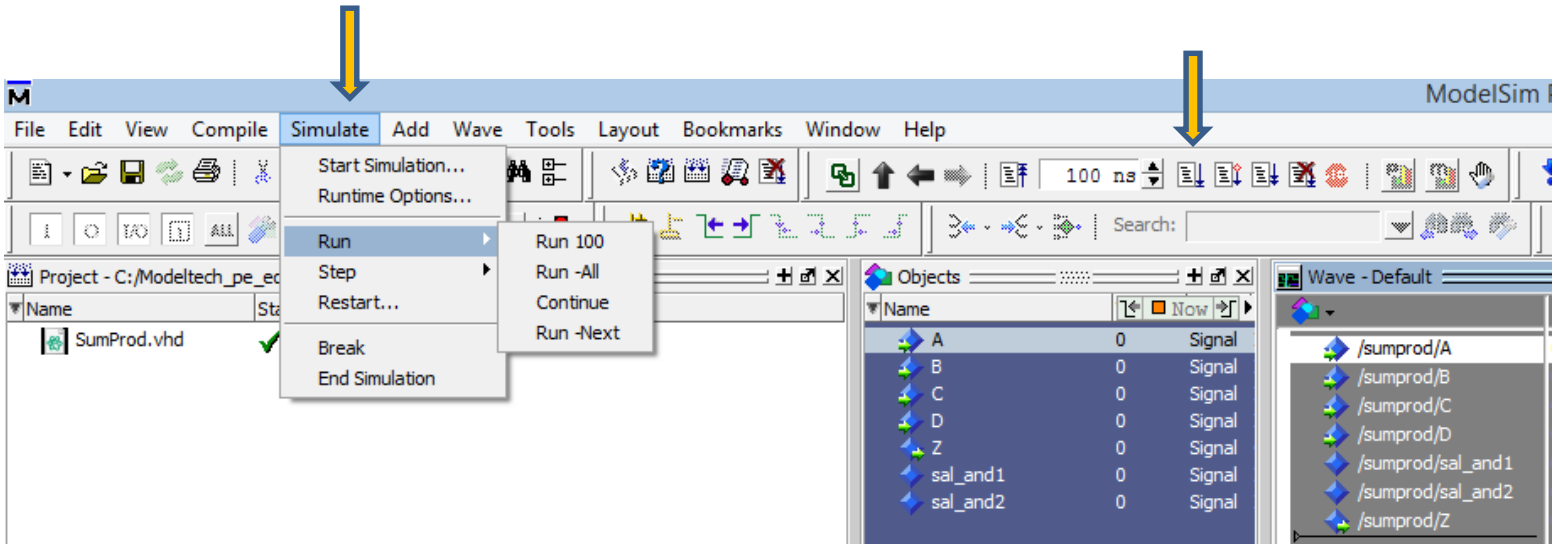
- Forzamos las entradas:
  - Por ejemplo, A = 1.





Ejecutar la Simulación

- Tenemos varias opciones para simular:



Simulación paso a paso

- Proceso:
  - Forzamos: A = 1, B = 1, C = 0, D = 0.
  - Run,
  - Forzamos el valor de B a 0
  - Run
  - Forzamos el valor de A a 0
  - Run
  - …