



Práctica 2

- Máquinas de estados finitas

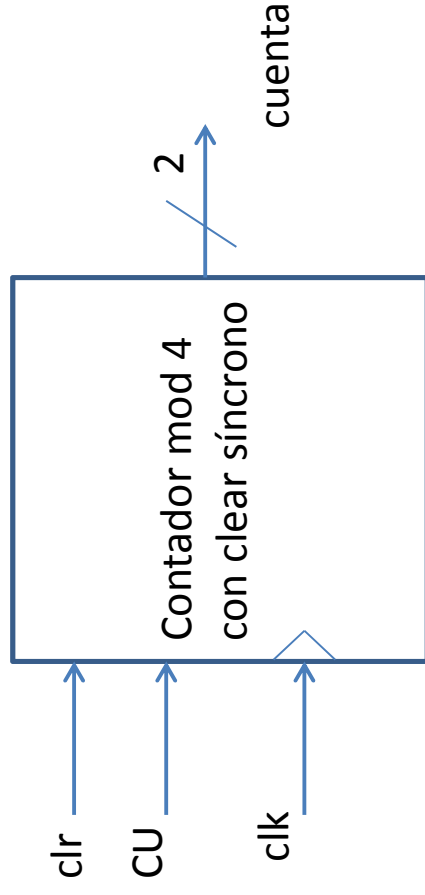
Práctica 2.a

- Diseñar un contador módulo 4 con *señal de cuenta (CU)* a partir del contador `contmod4.vhd` dado

```
➤ Si  y clr='1' ➡ cuenta <= "00";
```

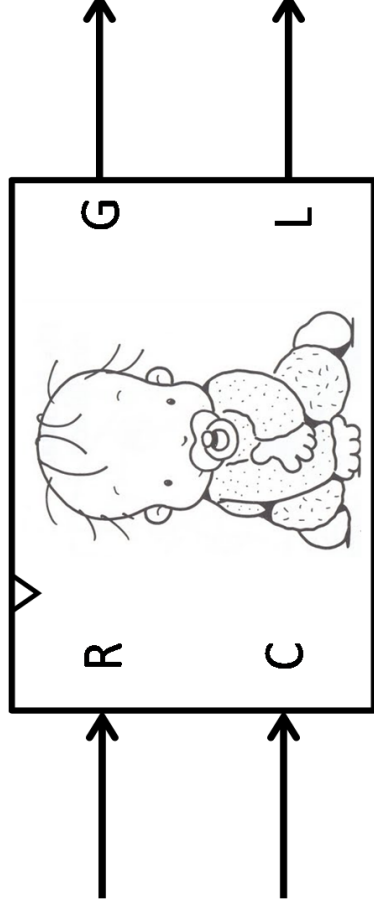
```
➤ Si  y clr='0' :
```

- Si `CU='1'` ➡ `cuenta<=cuenta+1`;
- Sino `cuenta <=cuenta`;



Práctica 2.b (I)

- Diseño de una máquina de estados finita (FSM)
- Diseñar el sistema de control de una muñeca interactiva



Práctica 2.b (I)

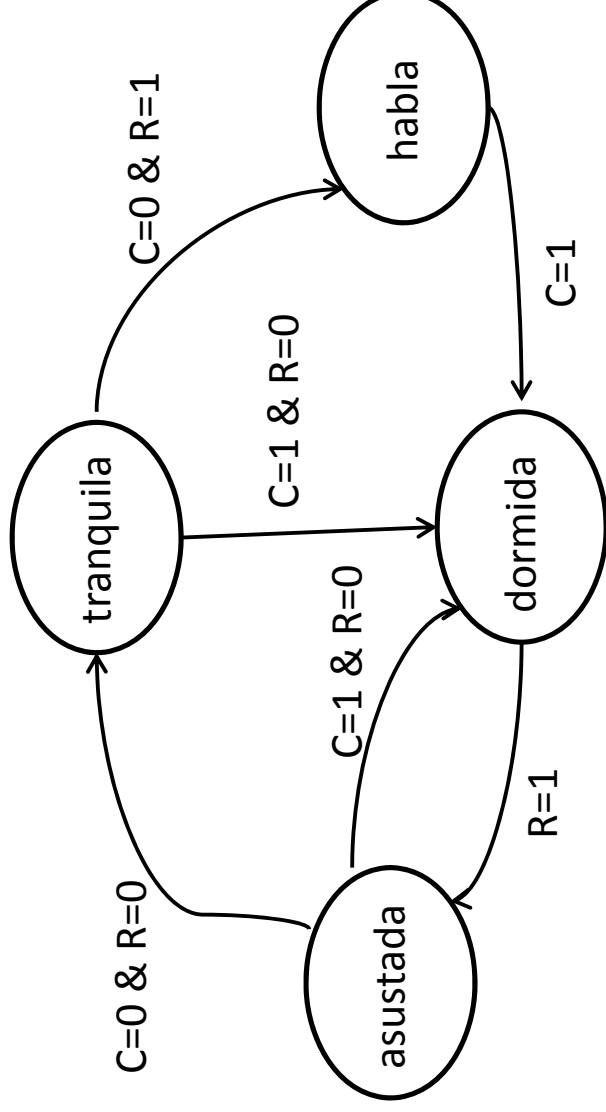
- El sistema tiene 2 entradas y 2 salidas
- La entrada R valdrá 1 cuando haya ruido
- La entrada C valdrá 1 cuando haya un chupete en la boca de la muñeca
- La salida G habilita un generador de sonidos
 - Si la muñeca hace ruido (habla o llora), $G=1$
- La salida L es igual a 1 cuando la muñeca llora y es igual a 0 cuando habla.

Práctica 2.b (III)

- Una vez encendida, la muñeca estará “tranquila”.
 - Ni habla, ni llora.
- Si está “tranquila”
 - Si se hace ruido, la muñeca “habla”.
 - Si se le pone el chupete pasará a estar “dormida”.
- Si “habla” y se le pone el chupete pasa a “dormida”
- Si está “dormida” no hace nada y permanecerá así hasta que se escuche un ruido.
 - En ese caso pasará al estado “asustada”.
- En el estado “asustada” la muñeca llorará
 - Cuando el ruido desaparezca pasará a estar “dormida” o “tranquila” en función de si tiene o no el chupete puesto.

Diagrama de estados

- Para otros valores de C y R no especificados en el diagrama el sistema permanecerá en el mismo estado en el que se encuentre

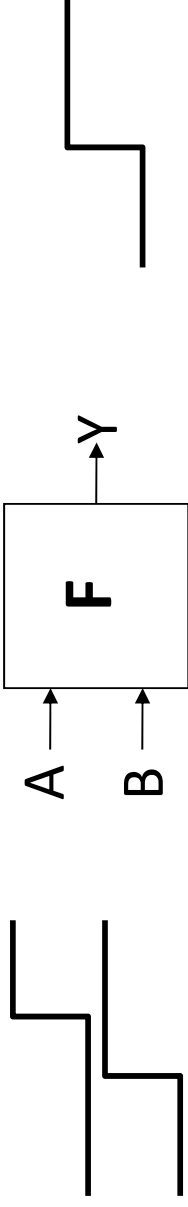


Práctica 2.b (III)

- Diseñar el sistema como una FSM tipo Moore
- Generar el fichero de test para simulación
- Simular y comprobar su correcto funcionamiento
- Implementar sobre la FPGA

Test-bench de simulación

- Para conocer si nuestro diseño funciona correctamente tendremos que introducir unos estímulos a las entradas y comprobar que las salidas obtenidas son las esperadas



Test-bench que reproduce los estímulos

```
entity test-bench
end test-bench;
...
```

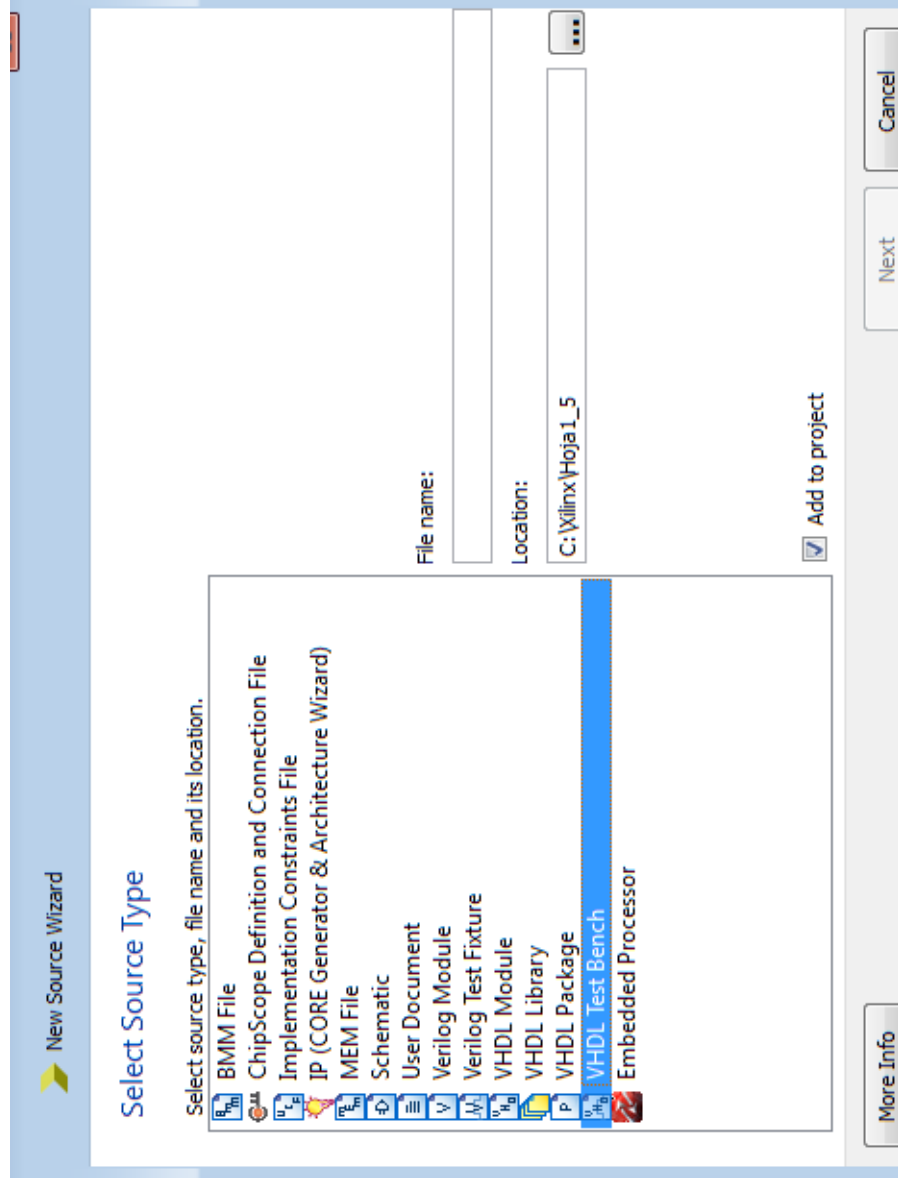
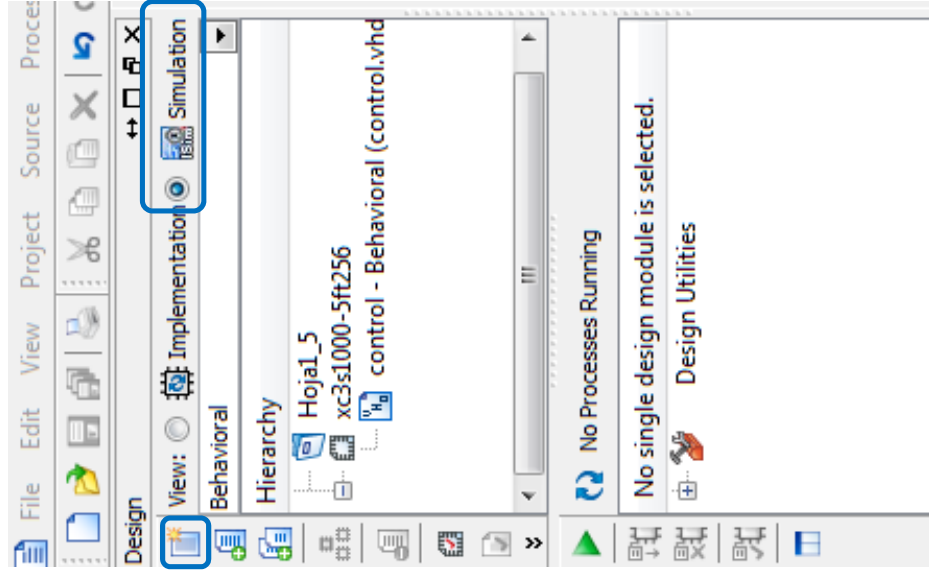
Diseño a verificar

```
entity F is
    port (A, ...)
end F;
...
```

Salida gráfica de simulación ofrecida por la herramienta

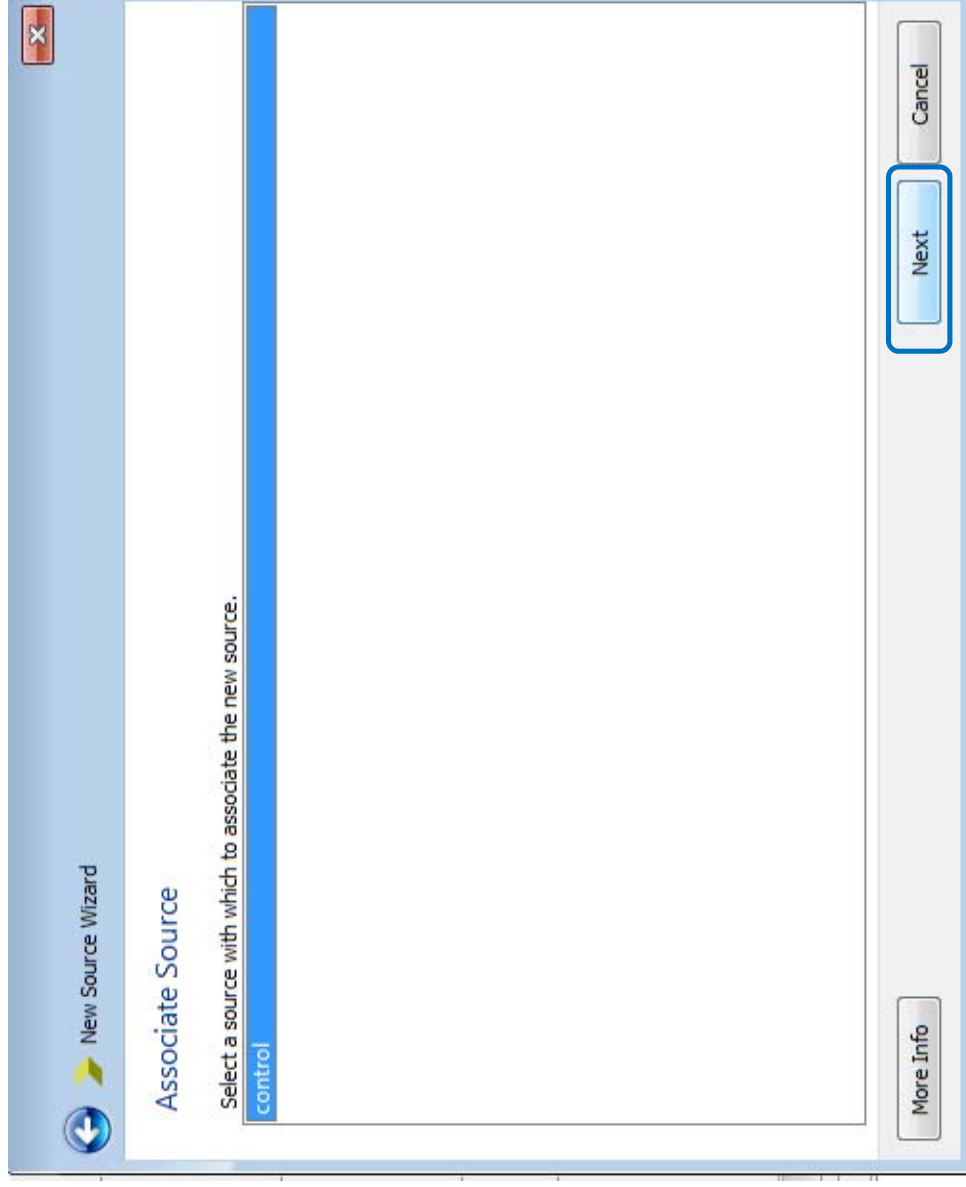
Test-bench de simulación

- Marcando la opción de Simulación, añadir *New Source*



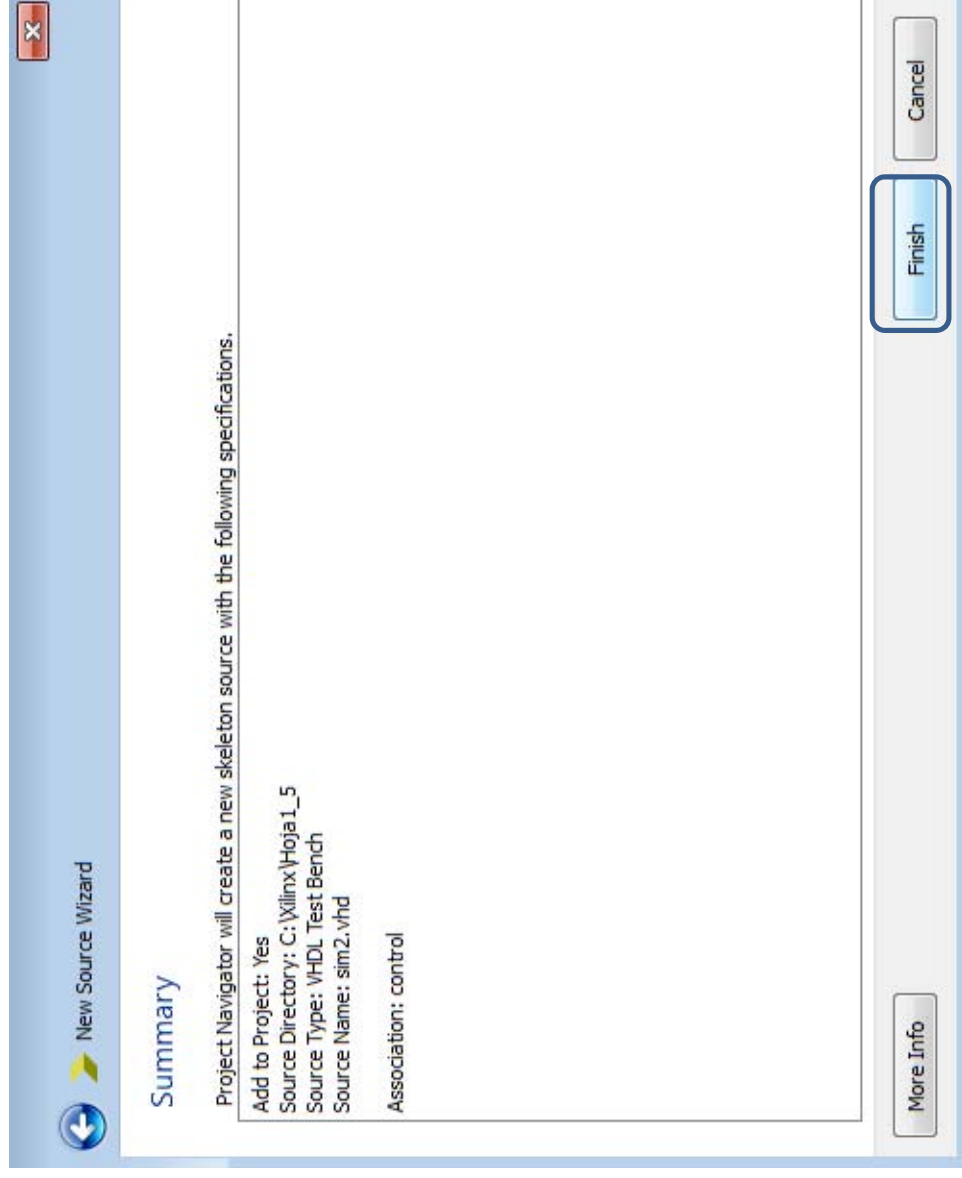
Test-bench de simulación

- Se abrirá la herramienta *New Source Wizard*



Test-bench de simulación

- Para finalizar



Test-bench de simulación

- La herramienta nos proporciona un fichero VHDL con la siguiente estructura básica

```
entity test-bench  
end test-bench;
```

i) Se crea una entidad de simulación sin puertos de entrada ni de salida

```
architecture behaviour of test_bench  
component F is
```

ii) Se añade como *component* la *entity* del diseño a verificar

```
    port (A, B: in bit; Y out bit);
```

```
component F;
```

```
signal A, B, Y: bit;
```

```
begin
```

```
...
```

iii) Se definen tantas señales como puertos de la *entity* del diseño a verificar

Test-bench de simulación

- Se distinguirán dos *process*: uno para definir los estímulos y otro para la señal de reloj

```
...  
begin
```

```
    uut: F port map(  
        A => A,  
        B => B,  
        Y => Y);
```

iv) Se instancia el *component* igualando las señales internas a las entradas

```
    tb : process  
    begin
```

v) Se crea el *process* de simulación. NO tiene lista de sensibilidad

```
        A<='0';  
        B<='0';  
        wait for 100 ns;  
        B<= '1';  
        wait for 100 ns;    -- instante 200 ns  
        A<='1';  
        wait for 100 ns;    -- instante 300 ns  
        ...  
        wait;  -- espera para siempre  
    end process tb;
```

vi) Se definen los valores iniciales de las entradas

vi) Se definen los valores de las entradas en los siguientes instantes de tiempo

```
end;
```

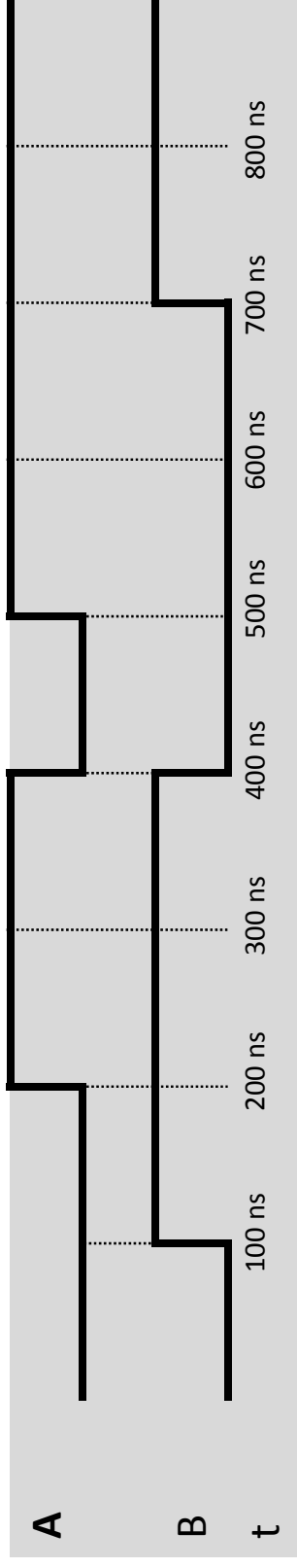
Test bench de simulación

- *Process* para la señal de reloj (clk)

```
clk_process :process
begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
end process;
```

- Como este *process* no acaba con un *wait* (sin *for*), se repetirá indefinidamente

Test bench de simulación



```
tb : process
begin
    A<='0';
    B<='0';
    wait for 100 ns;
    B<= '1';
    wait for 100 ns;      -- instante 200 ns
    A<='1';
    wait for 200 ns;      -- instante 400 ns
    A<='0';
    B<='0';
    wait for 100 ns;      -- instante 500 ns
    A<='1';
    wait for 200 ns;      -- instante 700 ns
    B<='1';
    wait for 100 ns;      -- instante 800 ns
    wait;
end process tb;
```