



## Práctica 2

- Máquinas de estados finitas

## Práctica 2



- Diseño de una máquina de estados finita (FSM)
- Diseñar un sistema que mida los reflejos del usuario, la coordinación visomotriz es la habilidad de los ojos y manos para trabajar juntos para la realización de una tarea. Diseñar un circuito que mida cuán rápido puede la mano actuar en respuesta a un estímulo visual



## Práctica 2

- Podéis simular el correcto funcionamiento en casa con un testbench propio
- El módulo que tenéis que desarrollar tendrá la siguiente interfaz de entrada/salida:

```
entity reflejo is
port(clk, rst, boton, switch: in std_logic;
      luces: out std_logic_vector(4 downto 0));
end reflejo;
```

# Práctica 2



- El sistema tiene como entrada un push button de la FPGA. Estos botones tienen lógica inversa; es decir, cuando no se pulsan, devuelven '1', y si se pulsan, devuelven '0'
- El sistema tiene otra entrada que es switch y que servirá para poner el circuito en marcha
- Se utilizarán 5 LEDs como salida
  - Un LED que servirá como estímulo al usuario
  - 4 LEDs que indican la medida tomada
    - "1000" Error
    - "0100" Respuesta lenta
    - "0010" Respuesta media
    - "0001" Respuesta rápida

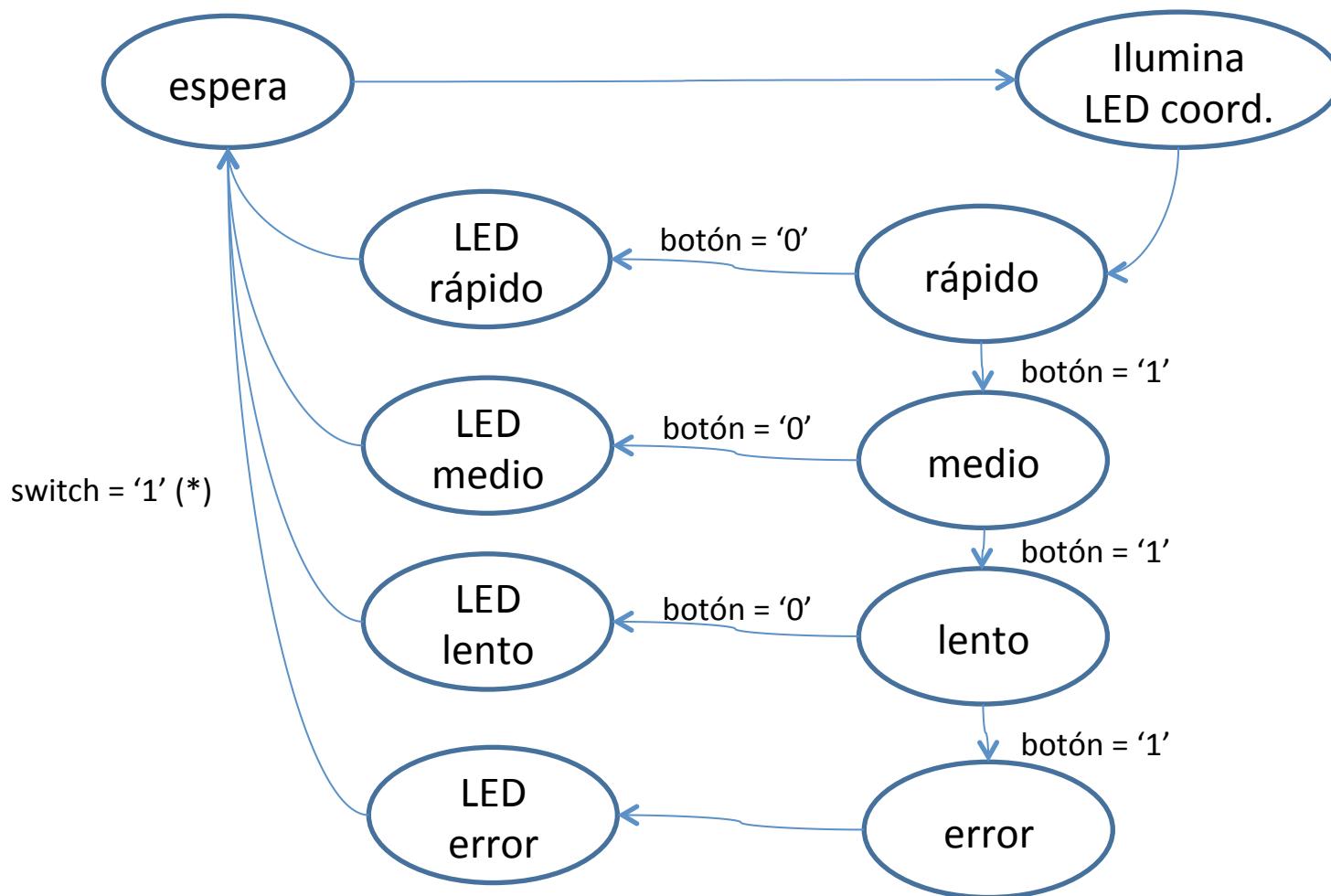
# Práctica 2



- Después de un reset, el sistema comienza en un estado de espera
- Desde ese estado se pasa automáticamente (en el siguiente ciclo de reloj) a un estado que enciende el LED de coordinación
- Desde este estado, se pasa secuencialmente a 4 estados. Se saldrá de esta secuencia si se pulsa el botón o si se llega al último estado
  - 1<sup>er</sup> estado: respuesta rápida
  - 2<sup>o</sup> estado: respuesta media
  - 3<sup>er</sup> estado: respuesta lenta
  - 4<sup>o</sup> estado: error
- Cada uno de los cuatro estados tiene asociado otro estado para indicar la iluminación de los otros 4 LEDs de salida de acuerdo a lo que se ha indicado en la transparencia anterior
- El sistema permanecerá en estos estados de iluminación hasta que la entrada switch pase a '1', desde donde pasará al estado inicial



# Diagrama de estados



(\*) Switch tiene que ser igual a '1' para que la FSM vuelva al estado inicial

# Práctica 2



- Diseñar el sistema como una máquina de estados tipo Moore.
  - Tenéis que seguir exactamente el mismo diagrama de la transparencia anterior y codificar la máquina de estados como se indicó en las transparencias 58 y 59 del tema 1.
- Después, generar el fichero de test para simulación.
  - Simular y comprobar su correcto funcionamiento.
- Finalmente, implementar sobre la FPGA.
  - Para ello, habrá que volver a añadir el divisor de frecuencias. Podéis consultar la solución de la práctica 1 (en el Campus Virtual) para ver cómo se hace, o leer las siguientes transparencias.
  - RECORDAD: Para simular, el divisor de frecuencias no se utiliza. Para implementación, sí se debe utilizar.

# Test bench de simulación



- Podéis generar el testbench de simulación haciendo click derecho en el proyecto, y después: *new source -> VHDL testbench*.
  - Llamarlo, por ejemplo, test.vhd.
- Se crea automáticamente un testbench, donde lo único que hay que modificar es el process `stim_proc`.
- Por ejemplo, así:

```
-- Stimulus process
stim_proc: process
begin
    rst <= '1';
    boton <= '1';
    switch <= '0';
    wait for 42 ns;

    rst <= '0';
    wait for 40 ns;

    boton <= '0';

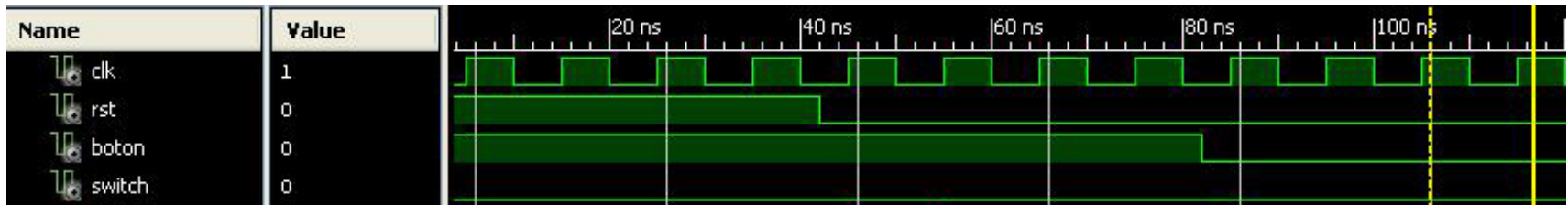
    wait;
end process;
```







# Test bench de simulación

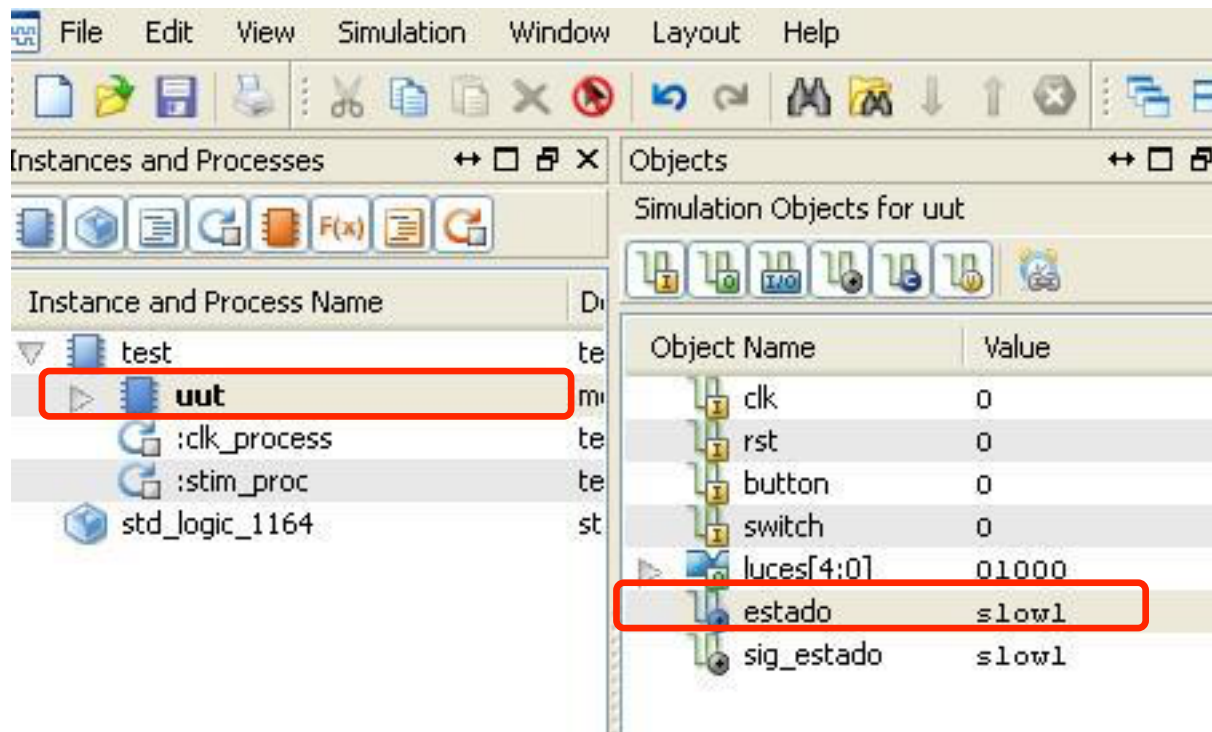
- Lo cual genera la siguiente forma de onda para rst, boton y switch:



# Test bench de simulación



Resulta muy útil comprobar los valores del estado. Para ello, se puede arrastrar y soltar la señal “estado” que se encuentra en la pestaña del simulador “*Instances and Processes -> uut*” a la ventana de simulación, reiniciar la simulación (click en el botón ) y volver a simular (  ).



The screenshot shows the simulation software interface. The 'Instances and Processes' window on the left lists the components: test, uut (highlighted with a red box), :clk\_process, :stim\_proc, and std\_logic\_1164. The 'Objects' window on the right shows the 'Simulation Objects for uut' table:

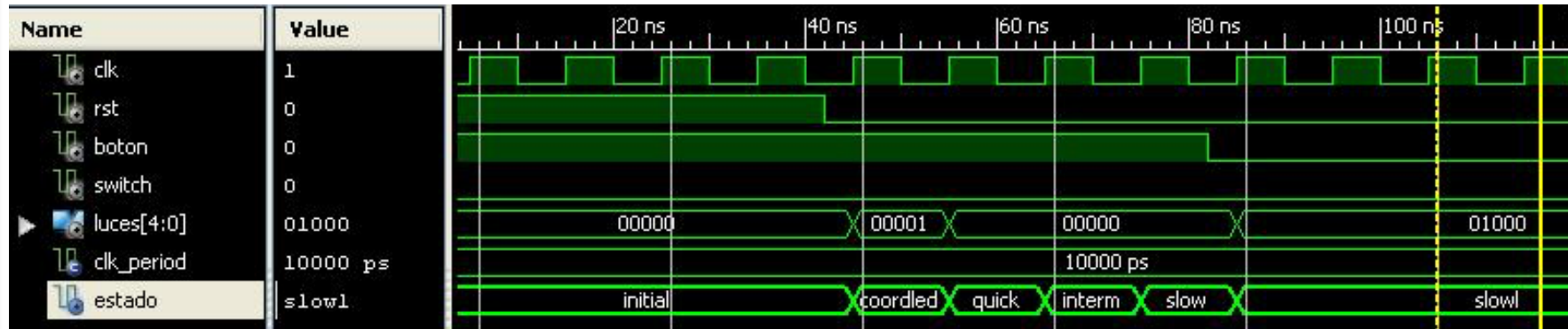
Object Name	Value
clk	0
rst	0
button	0
switch	0
luces[4:0]	01000
estado	slow1
sig_estado	slow1

The 'estado' row is highlighted with a red box.



# Test bench de simulación

- Tras hacer esto, la ventana de simulación permitirá ver la señal de estado:



- En este caso, la salida (luces) se ha implementado como un `std_logic_vector` (4 `downto` 0), aunque se podría haber implementado como 5 señales de tipo `std_logic`.
  - En cualquier caso, en vuestro código VHDL, no olvidéis dar valores a TODAS las salidas en todas las ramas del case!!! De lo contrario, se añaden latches.

# Implementación en la FPGA



- Es NECESARIO añadir el divisor de frecuencias para tener control sobre cómo se comporta la máquina de estados en la FPGA. Para ello:
  1. Añadir el fichero “divisor.vhd” (disponible en el Campus Virtual) a vuestro proyecto (file → add source).
  2. Agregarlo a vuestro diseño (entre architecture y begin de la entity reflejo) como un componente, y además añadir una señal intermedia clk\_intermediate:

```
component divider
    port(rst: in std_logic;
          clk_in: in std_logic;
          clk_out: out std_logic);
end component;

signal clk_intermediate: std_logic;
```

Esto mismo ya está hecho en la solución de la práctica 1 (en el Campus Virtual).



# Implementación en la FPGA

3. A continuación, añadir el siguiente código, tras el begin de la architecture:

```
-- COMMENT/UNCOMMENT FOR REMOVING/ADDING THE DIVIDER
divider_1: divider port map (rst => '0',
                             clk_in => clk,
                             clk_out => clk_intermediate);
```

```
-- COMMENT/UNCOMMENT FOR ADDING/REMOVING THE DIVIDER
-- clk_intermediate <= clk;
```

El reloj de entrada (clk) irá directamente asignado a clk\_intermediate si no queremos utilizar el divisor (SIMULACIÓN)

El componente se instanciará si se desea utilizar el divisor (IMPLEMENTACIÓN)

- En simulación, no se debe utilizar el divisor porque el reloj sería demasiado lento (1Hz) y tarda demasiado en completarse. Sin embargo, en implementación, es imprescindible utilizarlo.
- Hay que dejar descomentada sólo una de estas dos partes del código para así trabajar en modo simulación o implementación.

# Implementación en la FPGA



4. En cualquier caso, ahora el nuevo reloj que la máquina de estados utiliza internamente es `clk_intermediate`. Por tanto, hay que hacer al proceso SYNC sensible a este nuevo reloj:

```
SYNC: process (clk, reset)
begin
  if clk'event and clk = '1' then
    (...)
```



```
SYNC: process (clk_intermediate, reset)
begin
  if clk_intermediate'event and clk_intermediate = '1' then
    (...)
```

# Calificación



- El estudiante debe acudir al laboratorio con la práctica estudiada e IMPLEMENTADA desde casa
- El estudiante debe hacer funcionar la práctica 2.a en el laboratorio EN LA FPGA y mostrarla al profesor
  - Si funciona, +0.2 puntos
  - En caso de que no funcione, podéis enseñar una simulación, equivalente a la mitad de la nota (+0.1 puntos)
- La práctica 2 presenta una parte avanzada (+0.2 puntos), a realizar durante la sesión de laboratorio

# Calificación



- Simulación para calificación:
  - El reloj de simulación es de **100MHz**, aunque cuando se implemente en la FPGA se utiliza un reloj de 1Hz (igual que en la práctica anterior)
    - Si no hacéis esto, no podréis controlar absolutamente nada de vuestro sistema
  - Es recomendable sacar el reloj de 1Hz a otro LED más de la FPGA para así ver por dónde va. Esto os ayudará a depurar el funcionamiento del sistema