**Informe Práctica 3 SED: Máquinas de Estado**

Autores:

* Francisco Javier Perea Vanguelov (52545).
* Lydia Vega Ochoa (52654).
* Alberto Martínez Trapiello (52713).

Grupo: A-408 (Grupo B).

**Introducción:**

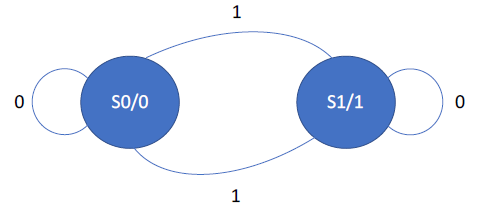
Una máquina de estado (FSM: Finite State Machine) es una máquina abstracta que puede estar en uno de un número finito de estados definidos por el usuario. La máquina puede estar en un solo estado a la vez. El estado en que se encuentra en un momento dado se llama el estado actual. Se cambia de un estado a otro cuando se produce un evento o condición desencadenante. Esto se llama una transición.

Para una mayor simplicidad, trabajaremos normalmente con máquinas de Moore (aquellas cuya salida depende únicamente del estado actual y no de las entradas).

En la presente práctica, realizamos un ejemplo de máquina de estado de Moore:

Sistema que comprueba la paridad de una secuencia de datos binarios.

El modelo de estados es el siguiente:



**Tarea 1:** **Implemente el circuito anterior en VIVADO. Desarrolle un banco de pruebas y verifique el modelo a través de una simulación de comportamiento.**

Como se especifica en el guion de la práctica, el código VHDL es el siguiente:

**TYPE** state\_type **IS (**S0**,** S1**);**

**SIGNAL** state**,** next\_state**:** state\_type**;**

*--Suponer señales reset, x y parity ya definidas*

**BEGIN**

SYNC\_PROC**: PROCESS (**clk**)**

**BEGIN**

**IF** rising\_edge**(**clk**) THEN**

**IF (**reset **=** '1'**) THEN**

state **<=** S0**;**

**ELSE**

state **<=** next\_state**;**

**END IF;**

**END IF;**

**END PROCESS;**

OUTPUT\_DECODE**: PROCESS (**state**)**

**BEGIN**

**CASE (**state**) is**

**WHEN** S0 **=>** parity **<=** '0'**;**

**WHEN** S1 **=>** parity **<=** '1'**;**

**WHEN OTHERS =>** parity **<=** '0'**;**

**END CASE;**

**END PROCESS;**

NEXT\_STATE\_DECODE**: PROCESS (**state**,** x**)**

**BEGIN**

next\_state **<=** S0**;**

**CASE (**state**) is**

**WHEN** S0 **=>**

**IF (**x **=** '1'**) THEN**

next\_state **<=** S1**;**

**END IF;**

**WHEN** S1 **=>**

**IF (**x **=** '0'**) THEN**

next\_state **<=** S1**;**

**END IF;**

**WHEN OTHERS =>** next\_state **<=** S0**;**

**END CASE;**

**END PROCESS;**

**Código Testbench:**

**LIBRARY** IEEE;

**USE** IEEE.STD\_LOGIC\_1164.ALL;

**ENTITY** state\_machine\_tb **IS**

-- Port ( );

**END** state\_machine\_tb;

-------------------------------------------------------------------------------------

**ARCHITECTURE** Behavioral **OF** state\_machine\_tb **IS**

**COMPONENT** parity\_machine **IS**

**PORT** (reset, x, clk: **IN** STD\_LOGIC;

parity: **OUT** STD\_LOGIC);

**END** **COMPONENT**;

**SIGNAL** reset\_in, x\_in, clk\_in, parity\_out: STD\_LOGIC;

**CONSTANT** k: TIME := 5 ns;

**BEGIN**

comp1: parity\_machine **PORT MAP**(

reset => reset\_in,

x => x\_in,

clk => clk\_in,

parity => parity\_out);

p1: **PROCESS**

**BEGIN**

clk\_in <= '0';

**WAIT FOR** 2\*k;

clk\_in <= '1';

**WAIT FOR** 2\*k;

**END PROCESS**;

p2: **PROCESS**

**BEGIN**

x\_in <= '1';

**WAIT FOR** 1.5\*k;

x\_in <= '0';

**WAIT FOR** 1.5\*k;

**END PROCESS**;

p3: **PROCESS**

**BEGIN**

**WAIT UNTIL** x\_in <= '0';

reset\_in <= '1';

**WAIT FOR** 2\*k;

reset\_in<='0';

**WAIT FOR** 2\*k;

**END PROCESS**;

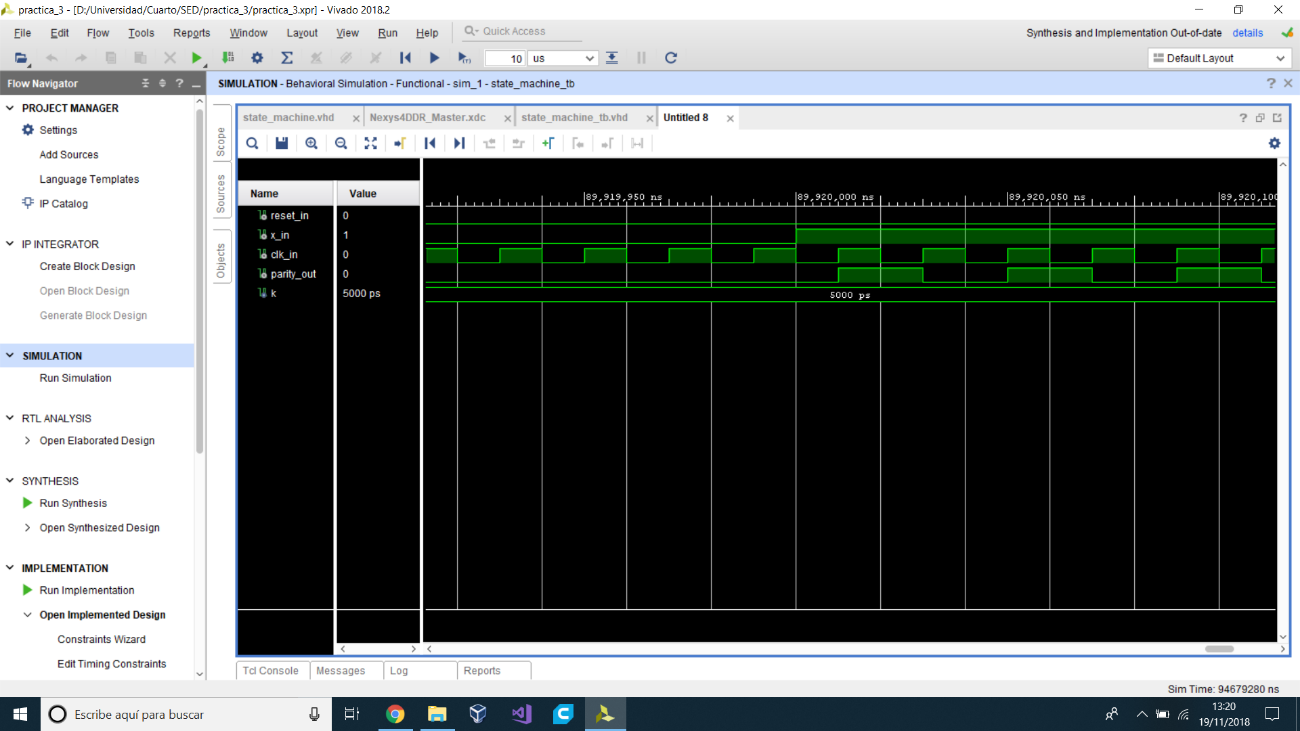
**ASSERT** false

**REPORT** "Éxito. Simulación finalizada."

**SEVERITY** failure;

**END** Behavioral;

-----------------------------------------------------------------------------------------



La simulación nos arroja información clara sobre el funcionamiento del detector de paridad. Tras un breve análisis se determina que con cada flanco de reloj se va analizando la entrada, de modo que si encuentra un 0 la paridad será par y por tanto la salida será 0. Pero en cuanto llegue un 1 la paridad se volverá impar y por tanto para que el resultado sea par se introduce el uno de la salida.

En este testbench se puede ver claramente que mientras le lleguen ceros la salida no cambia, pero en cuanto e le empiezan a introducir unos empieza a cambiar la salida ya que con cada flanco de reloj se les va introduciendo unos, los cuales obligan a cambiar la paridad. Por ello mientras la secuencia de unos se prolongue en el tiempo, la salida irá cambiando consecuentemente.

Cuando reset vale 1, el detector de paridad no funciona.

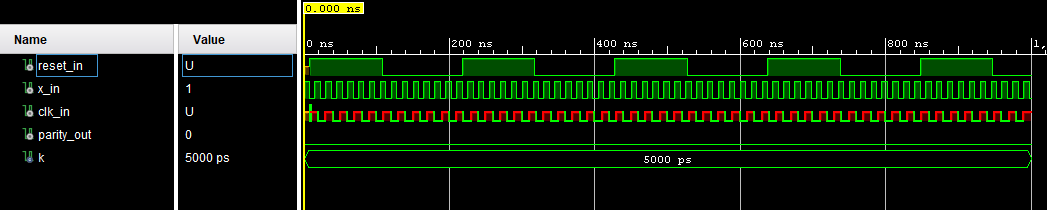
**Tarea 2: Implemente el circuito en la placa Nexys4DDR. Use la señal de reloj de la placa, SW0 como la entrada de bits “x”, el botón BTNU como reset, y LED0 como la salida “parity”. Realice el flujo de diseño, genere el bitstream y descárguelo en la placa Nexys4 DDR. Verifique la funcionalidad.**

Al implementar el código en la FPGA, se observa con sorpresa inicial que cuando la entrada la ponemos a uno, la salida ilumina con menor nivel mientras que al cambiar a cero la salida a veces se ilumina completamente y a veces se queda apagada.

Este comportamiento nos extrañó en un primer momento hasta descubrir el motivo de que iluminara con menor intensidad. Esto se debía a que al estar recibiendo unos por la entrada a la frecuencia del reloj, la salida estaba constantemente cambiando el bit ya que a cada uno que le llegaba cambiaba, lógicamente, el bit de paridad. Por otra parte, el motivo de que al cambiar a cero a veces diera uno y a veces diera cero deducimos que se debía a que según cuántos unos hubiera recibido por la entrada (fundamentalmente si estos son pares o impares) cambiaría la salida. Y posteriormente dado que se le introducen ceros, nunca cambiaría la salida ya que no afectan a la paridad.

Al estar tomando las entradas a la frecuencia de la entrada no hay manera de controlar lo que le entra ya que no podemos cambiar la entrada a la velocidad a la que cambia el reloj. Por lo que decidimos que podíamos o bien cambiar el reloj por un interruptor que controláramos a mano o bien más sencillo usando un divisor de frecuencia conseguir reducir la frecuencia a algo más manejable.

**Nota:** Se ha modificado el código y el testbench, añadiendo un divisor de frecuencia, que ya realizamos durante las clases teóricas, (para corregir el led parpadeante y que parity esté activa más tiempo seguido). Sin embargo, hemos encontrado problemas a la hora de realizar la simulación ya que, tal y como se muestra en la siguiente imagen, la señal de reloj no se muestra a nivel alto, sino que queda indeterminada.



Se muestra el código introducido a continuación:

**Código Principal:**

----------------------------------------------------------------------------------

**LIBRARY** IEEE;

**USE** IEEE.STD\_LOGIC\_1164.ALL;

**LIBRARY** UNISIM;

**USE** UNISIM.VComponents.all;

-----------------------------------------------------------------------------------

**ARCHITECTURE** parity\_machine **OF** parity\_machine **IS**

**TYPE** state\_type **IS** (S0, S1);

**SIGNAL** state, next\_state: state\_type;

**BEGIN**

SYNC\_PROC: **PROCESS** (clk)

**BEGIN**

**IF** rising\_edge(clk) **THEN**

**IF** (reset='1') **THEN**

state <= S0;

**ELSE**

state <= next\_state;

**END** **IF**;

**END IF**;

**END PROCESS**;

OUTPUT\_DECODE: **PROCESS**(state)

**BEGIN**

**CASE** (state) **IS**

**WHEN** S0 => parity <= '0';

**WHEN** S1 => parity <= '1';

**WHEN** **OTHERS** => parity <= '0';

**END CASE;**

**END PROCESS**;

NEXT\_STATE\_DECODE: **PROCESS** (state, x)

**BEGIN**

next\_state <= S0;

**CASE** (state) **IS**

**WHEN** S0 =>

**IF** (x='1') **THEN**

next\_state <= S1;

**END** **IF**;

**WHEN** S1 =>

**IF** (x = '0') **THEN**

next\_state <= S1;

**END** **IF**;

**WHEN** **OTHERS** => next\_state <= S0;

**END** **CASE**;

**END** **PROCESS**;

**END** parity\_machine;

--------------------------------------------------------------------------------------

--------------------------------------------------------------------------------------

**LIBRARY** IEEE;

**USE** IEEE.STD\_LOGIC\_1164.ALL;

**LIBRARY** UNISIM;

**USE** UNISIM.VComponents.all;

--------------------------------------------------------------------------------------

**ENTITY** fdivider is

**GENERIC**(MODULE:positive:=16);

**PORT**(reset,clk: **IN** STD\_LOGIC;

CE\_OUT: **OUT** STD\_LOGIC);

**END** fdivider;

--------------------------------------------------------------------------------------

**ARCHITECTURE** behavioral **OF** fdivider **IS**

**BEGIN**

**PROCESS** (reset, clk)

**SUBTYPE** count\_range **IS** integer **RANGE** 0 **TO** MODULE-1;

**VARIABLE** count: count\_range;

**BEGIN**

**IF** (reset='1') **THEN**

count:=count\_range'high;

CE\_OUT<='0';

**ELSIF** rising\_edge(clk) **THEN**

CE\_OUT <='0';

**IF** count /=0 **THEN**

count:=count-1;

**ELSE**

CE\_OUT<='1';

count:=count\_range'high;

**END IF;**

**END IF;**

**END PROCESS;**

**END** behavioral;

-------------------------------------------------------------------------------------------

**Código Testbench:**

----------------------------------------------------------------------------------

**LIBRARY** IEEE;

**USE** IEEE.STD\_LOGIC\_1164.ALL;

-----------------------------------------------------------------------------------

**ENTITY** state\_machine\_tb **IS**

-- Port ( );

**END** state\_machine\_tb;

-------------------------------------------------------------------------------------

**ARCHITECTURE** Behavioral **OF** state\_machine\_tb **IS**

**COMPONENT** parity\_machine **IS**

**PORT** (reset, x, clk: IN STD\_LOGIC;

parity: OUT STD\_LOGIC);

**END COMPONENT**;

**COMPONENT** fdivider **IS**

**GENERIC**(MODULE:positive:=16);

**PORT**(reset,clk: **IN** STD\_LOGIC;

CE\_OUT: **OUT** STD\_LOGIC);

**END COMPONENT**;

**SIGNAL** reset\_in, x\_in, clk\_in, clk\_intermedio, parity\_out: STD\_LOGIC;

**CONSTANT** k: TIME := 5 ns;

**BEGIN**

divisor1: fdivider **PORT MAP**(

reset=>reset\_in,

clk=>clk\_intermedio,

CE\_OUT=>clk\_in);

comp1: parity\_machine **PORT MAP**(

reset => reset\_in,

x => x\_in,

clk => clk\_in,

parity => parity\_out);

p1: **PROCESS**

**BEGIN**

clk\_in <= '0';

**WAIT FOR** 2\*k;

clk\_in <= '1';

**WAIT FOR** 2\*k;

**END PROCESS**;

p2: **PROCESS**

**BEGIN**

x\_in <= '1';

**WAIT FOR** 1.5\*k;

x\_in <= '0';

**WAIT FOR** 1.5\*k;

**END PROCESS**;

p3: **PROCESS**

**BEGIN**

**WAIT UNTIL** x\_in <= '0';

reset\_in <= '1';

**WAIT FOR** 20\*k;

reset\_in<='0';

**WAIT FOR** 20\*k;

**END PROCESS**;

**END** Behavioral;

-------------------------------------------------------------------------------------