

**NOMBRE Y APELLIDOS DE AMBOS ESTUDIANTES:**

* Ignacio Murube Crego
* Jerónimo Boza García

Escuela Politécnica Superior

Asignatura: Tecnología de Computadores

PROYECTO FINAL:

Documentación

# Índice/Tabla de contenidos

Tabla de contenido

[Índice/Tabla de contenidos 2](#_Toc154053810)

[1. Introducción Proyecto 3](#_Toc154053811)

[2. Procedimientos 4](#_Toc154053812)

[3. Explicación códigos VDHL 5](#_Toc154053813)

[4. Simulación 6](#_Toc154053814)

[5. Análisis 7](#_Toc154053815)

[6. Conclusión 8](#_Toc154053816)

# Introducción Proyecto

En este documento se va a comentar, describir y analizar el proyecto llevado a cabo en su totalidad, dicho proyecto constituido por las siguientes partes:

1. Enunciado
2. Procedimiento realizados y planificados para su elaboración.
3. Creación del código VDHL con su testbench incluido.
4. Ampliación.
5. Simulación del proyecto.
6. Análisis de los resultados obtenidos.
7. Conclusión final sobre el proyecto.

Primero empezaremos planteado el enunciado:

“El trabajo consiste en diseñar el sistema de control de una máquina expendedora de bebidas

que vende cada lata de refrescos a 2€. Se deben cumplir las siguientes características:

- La máquina puede recibir monedas de 1€ y de 2€ y billetes de 5€.

- La máquina devuelve cambio si se introduce más dinero del que cuesta la lata.

- La máquina siempre tiene latas disponibles, por lo que, para simplificar, no se considera

la posibilidad de quedarse sin stock.

Para diseñar la máquina expendedora se recomienda utilizar una máquina de estados síncrona.

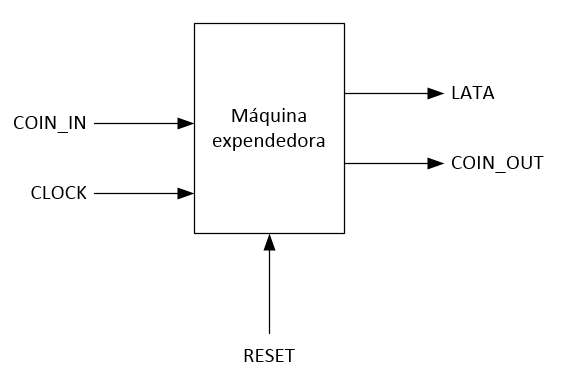
Asimismo, se recomienda que los alumnos realicen su diseño paso a paso empezando por un

análisis de posibilidades para entender las posibles combinaciones de monedas/billetes que se

pueden introducir en la máquina expendedora y su comportamiento esperado en cada caso.

Para mayor claridad y homogeneidad en los trabajos entregados por los alumnos, se recomienda

ceñirse al diagrama presentado a continuación y los nombres escogidos para las señales.”



I. Diagrama

# Procedimientos

Primer paso fue decidir el tipo de máquina de estado íbamos a realizar: una máquina de Moore o una máquina de Mealy.

Máquina de Moore 🡪 Las salidas dependen solamente del estado actual y no de las entradas.

Máquina de Mealy 🡪 Las salidas si dependen tanto de su estado actual como de la entrada.

Una vez planteado el ejercicio propuesto decidimos usar **una máquina de estados Mealy**.

Diagrama

Descripción generada automáticamente

II.Esquema Máquina de Mealy

En el esquema podemos apreciar que el estado ‘E0’ es el estado inicial, cuando la máquina espera la entrada de dinero derivando en varias posibilidades:

1. Entrada moneda de 1€ y paso a estado ‘ E1’.
2. Entrada moneda de 2€ y paso a estado ‘ E2’.
3. Entrada billete de 5€ y paso a estado ‘ E2’.
4. Resteo máquina en el caso de esperar entrada de dinero y no tener ninguna, vuelta a estado ‘E0’.

Como segundo estado tendríamos a ‘E1’ donde:

1. Entrada moneda de 1€ y paso a estado ‘ E2’.
2. Entrada moneda de 2€ y paso a estado ‘ E2’.
3. Entrada moneda de 5€ y paso a estado ‘ E2’.
4. Reseteo de máquina al no recibir más dinero y paso a estado ‘E0’.

Y por último tendremos el estado final que corresponde con ‘E2’.

Una vez hemos definido el esquema y el proceso de paso entre estados, pasamos a realizar el código VDHL donde definiremos dichos estados y su funcionamiento para manejar las entradas posibles.

# Explicación códigos VDHL

Ahora pasaremos a mostrar el código VDHL de cada uno de los archivos usados para el proyecto.

1. ***canMachineUnit.vhd***

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity canMachineUnit is

Port ( clk, rst : in std\_logic;

coin\_in : in std\_logic\_vector(2 downto 0);

empty : out std\_logic;

lata : out std\_logic;

coin\_out : out std\_logic\_vector(2 downto 0)

);

end canMachineUnit;

architecture Behavioral of canMachineUnit is

component canMachine is

Port ( clk, rst : in std\_logic;

coin\_in : in std\_logic\_vector(2 downto 0);

empty : in std\_logic;

lata : out std\_logic;

coin\_out : out std\_logic\_vector(2 downto 0)

);

end component;

component contador is

Port (clk, rst : in std\_logic;

lata : in std\_logic;

empty : out std\_logic

);

end component;

signal empty\_s : std\_logic;

signal lata\_s : std\_logic;

begin

empty <= empty\_s;

lata <= lata\_s;

-- Caso en el que el inventario estubiese vacio :

-- lata <= lata\_s when empty\_s = '0' else '0';

CAN\_MACHINE\_LOGIC : canMachine

port map ( clk => clk,

rst => rst,

coin\_in => coin\_in,

empty => empty\_s,

lata => lata\_s,

coin\_out => coin\_out

);

COUNT\_MODULE : contador

port map(clk => clk,

rst => rst,

lata => lata\_s,

empty => empty\_s

);

end Behavioral;

1. ***canMachine.vhd***

COUNT\_MODULE : contador

port map(clk => clk,

rst => rst,

lata => lata\_s,

empty => empty\_s

);

end Behavioral;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity canMachine is

Port ( clk, rst : in std\_logic;

coin\_in : in std\_logic\_vector(2 downto 0);

empty : in std\_logic;

lata : out std\_logic;

coin\_out : out std\_logic\_vector(2 downto 0)

);

end canMachine;

architecture Behavioral of canMachine is

type state is (E0,E1,E2);

signal CS,NS : state;

signal money\_in : std\_logic\_vector(2 downto 0);

constant precio\_lata : unsigned(2 downto 0) := "010";

begin

process(clk)

begin

if rising\_edge(clk) then

if rst = '0' then

CS <= NS;

else

CS <= E0;

end if;

end if;

end process;

process(CS,coin\_in,empty)

begin

case CS is

when E0 =>

money\_in <= "000";

lata <= '0';

coin\_out <= "000";

if coin\_in = "001" then

NS <= E1;

money\_in <= "001";

elsif coin\_in = "010" then

NS <= E2;

money\_in <= "010";

elsif coin\_in = "101" then

NS <= E2;

money\_in <= "101";

else

NS <= CS;

end if;

when E1 =>

lata <= '0';

if rst = '1' then

coin\_out <= "001";

else

coin\_out <= "000";

end if;

if coin\_in = "001" then

NS <= E2;

money\_in <= "010";

elsif coin\_in = "010" then

NS <= E2;

money\_in <= "011";

elsif coin\_in = "101" then

NS <= E2;

money\_in <= "110";

else

NS <= CS;

end if;

when E2 =>

--lata <= '1';

--coin\_out <= std\_logic\_vector(unsigned(money\_in) - unsigned(precio\_lata));

--NS <= E0;

if empty = '0' then

lata <= '1';

coin\_out <= std\_logic\_vector(unsigned(money\_in) - unsigned(precio\_lata));

else

lata <= '0';

coin\_out <= money\_in;

end if;

NS <= E0;

-- Caso en el que el inventario estubiese vacio :

-- if empty = '0' then

-- lata <= '1';

-- else

-- coin\_out <= money\_in;

-- NS <= E0;

end case;

end process;

end Behavioral; lata => lata\_s,

coin\_out => coin\_out

);

COUNT\_MODULE : contador

port map(clk => clk,

rst => rst,

lata => lata\_s,

empty => empty\_s

);

end Behavioral;

elsif coin\_in = "010" then

NS <= E2;

money\_in <= "010";

elsif coin\_in = "101" then

NS <= E2;

money\_in <= "101";

else

NS <= CS;

end if;

when E1 =>

lata <= '0';

if rst = '1' then

coin\_out <= "001";

else

coin\_out <= "000";

end if;

if coin\_in = "001" then

NS <= E2;

money\_in <= "010";

elsif coin\_in = "010" then

NS <= E2;

money\_in <= "011";

elsif coin\_in = "101" then

NS <= E2;

money\_in <= "110";

else

NS <= CS;

end if;

when E2 =>

--lata <= '1';

--coin\_out <= std\_logic\_vector(unsigned(money\_in) - unsigned(precio\_lata));

--NS <= E0;

if empty = '0' then

lata <= '1';

coin\_out <= std\_logic\_vector(unsigned(money\_in) - unsigned(precio\_lata));

else

lata <= '0';

coin\_out <= money\_in;

end if;

NS <= E0;

-- Caso en el que el inventario estubiese vacio :

-- if empty = '0' then

-- lata <= '1';

-- else

-- coin\_out <= money\_in;

-- NS <= E0;

end case;

end process;

end Behavioral;

1. ***contador.vhd***

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity contador is

Port (clk, rst : in std\_logic;

lata : in std\_logic;

empty : out std\_logic

);

end contador;

architecture Behavioral of contador is

signal empty\_s : std\_logic := '0';

signal latas\_restantes : integer := 3;

begin

empty <= empty\_s;

process(clk)

begin

if rising\_edge(clk) then

if rst = '1' then

empty\_s <= '0';

latas\_restantes <= 3;

elsif lata = '1' and empty\_s = '0' then

if latas\_restantes > 1 then

latas\_restantes <= latas\_restantes - 1 ;

else

empty\_s <= '1';

end if;

end if;

end if;

end process;

end Behavioral;

1. ***canMachine\_tb.vhd***

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity canMachineUnit\_tb is

end canMachineUnit\_tb;

architecture Behavioral of canMachineUnit\_tb is

component canMachineUnit is

Port ( clk, rst : in std\_logic;

coin\_in : in std\_logic\_vector(2 downto 0);

empty : out std\_logic;

lata : out std\_logic;

coin\_out : out std\_logic\_vector(2 downto 0)

);

end component;

signal clk, rst : std\_logic := '0';

signal coin\_in : std\_logic\_vector(2 downto 0);

signal empty : std\_logic;

signal lata : std\_logic;

signal coin\_out : std\_logic\_vector(2 downto 0);

signal clk\_period : time := 10 ns;

begin

UUT : canMachineUnit

port map(

clk => clk,

rst => rst,

coin\_in => coin\_in,

empty => empty,

lata => lata,

coin\_out => coin\_out

);

-- Ciclos de reloj

process

begin

clk <= '0';

wait for clk\_period/2;

clk <= '1';

wait for clk\_period/2;

end process;

-- Casos de Estimulo :

process

begin

-- Caso 1

Coin\_in <= "001";

wait for clk\_period;

Coin\_in <= "000";

wait for clk\_period;

Coin\_in <= "001";

wait for clk\_period;

assert (lata = '1' and coin\_out = "000")

report "Error Test 1" severity error;

Coin\_in <= "000";

wait for clk\_period;

-- Caso Rst 1

Coin\_in <= "001";

wait for clk\_period;

rst <= '1';

Coin\_in <= "000";

wait for 2 ns;

assert (lata = '0' and coin\_out = "001")

report "Error Test 1 RST" severity error;

wait for clk\_period;

rst <= '0';

wait for clk\_period;

---------- Caso 2

Coin\_in <= "001";

wait for clk\_period;

Coin\_in <= "000";

wait for clk\_period;

Coin\_in <= "010";

wait for clk\_period;

assert (lata = '1' and coin\_out = "001")

report "Error Test 2" severity error;

Coin\_in <= "000";

wait for clk\_period;

---------- Caso 3

Coin\_in <= "001";

wait for clk\_period;

Coin\_in <= "000";

wait for clk\_period;

Coin\_in <= "101";

wait for clk\_period;

assert (lata = '1' and coin\_out = "100")

report "Error Test 3" severity error;

Coin\_in <= "000";

wait for clk\_period;

---------- Caso 4

Coin\_in <= "010";

wait for clk\_period;

assert (lata = '1' and coin\_out = "000")

report "Error Test 4" severity error;

Coin\_in <= "000";

wait for clk\_period;

---------- Caso 5

Coin\_in <= "101";

wait for clk\_period;

assert (lata = '1' and coin\_out = "011")

report "Error Test 5" severity error;

Coin\_in <= "000";

wait for clk\_period;

end process;

end Behavioral;

assert (lata = '1' and coin\_out = "000")

report "Error Test 1" severity error;

Coin\_in <= "000";

wait for clk\_period;

-- Caso Rst 1

Coin\_in <= "001";

wait for clk\_period;

rst <= '1';

Coin\_in <= "000";

wait for 2 ns;

assert (lata = '0' and coin\_out = "001")

report "Error Test 1 RST" severity error;

wait for clk\_period;

rst <= '0';

wait for clk\_period;

---------- Caso 2

Coin\_in <= "001";

wait for clk\_period;

Coin\_in <= "000";

wait for clk\_period;

Coin\_in <= "010";

wait for clk\_period;

assert (lata = '1' and coin\_out = "001")

report "Error Test 2" severity error;

Coin\_in <= "000";

wait for clk\_period;

---------- Caso 3

Coin\_in <= "001";

wait for clk\_period;

Coin\_in <= "000";

wait for clk\_period;

Coin\_in <= "101";

wait for clk\_period;

assert (lata = '1' and coin\_out = "100")

report "Error Test 3" severity error;

Coin\_in <= "000";

wait for clk\_period;

---------- Caso 4

Coin\_in <= "010";

wait for clk\_period;

assert (lata = '1' and coin\_out = "000")

report "Error Test 4" severity error;

Coin\_in <= "000";

wait for clk\_period;

---------- Caso 5

Coin\_in <= "101";

wait for clk\_period;

assert (lata = '1' and coin\_out = "011")

report "Error Test 5" severity error;

Coin\_in <= "000";

wait for clk\_period;

end process;

end Behavioral;

1. ***constrains.xdc***

---------- Caso 5

Coin\_in <= "101";

wait for clk\_period;

assert (lata = '1' and coin\_out = "011")

report "Error Test 5" severity error;

Coin\_in <= "000";

wait for clk\_period;

end process;

end Behavioral;

# Map Input Clock

set\_property PACKAGE\_PIN E3 [get\_ports CLK]

set\_property IOSTANDARD LVCMOS33 [get\_ports CLK]

# Map Output LED

set\_property PACKAGE\_PIN H17 [get\_ports LED]

set\_property IOSTANDARD LVCMOS33 [get\_ports LED]

create\_clock -period 10.000 -name clk -waveform {0.000 5.000} [get\_ports clk]

Imagen que contiene Gráfico

Descripción generada automáticamente

III.Elaborate Desing

# Simulación

Diagrama, Escala de tiempo

Descripción generada automáticamente🡪

🡪Escala de tiempo

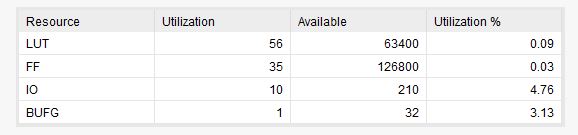
Descripción generada automáticamente

# Análisis

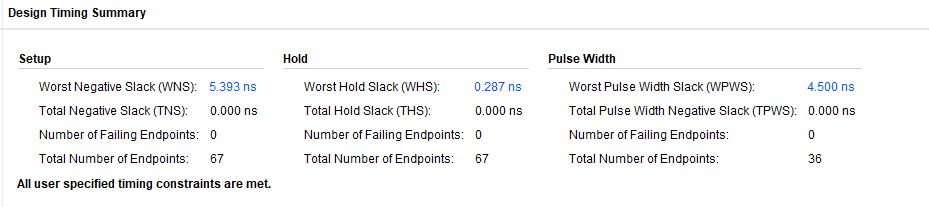
1. Análisis de área

Imagen que contiene Gráfico

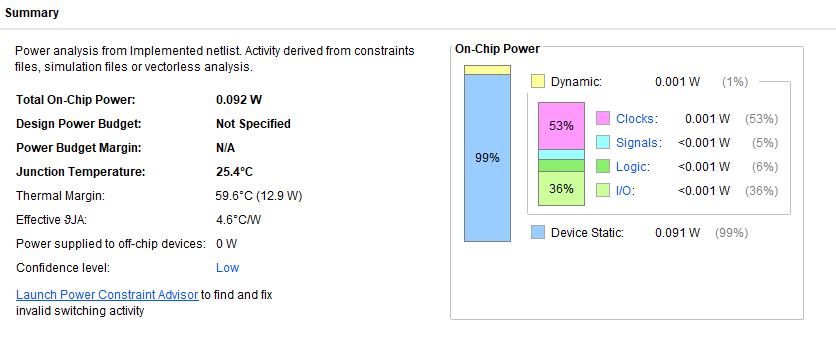
Descripción generada automáticamente



1. Análisis de tiempo



1. Análisis de consumo



# Conclusión