Université du Québec à Montréal

Cours : EMB7020

Codesign

Devoir 01:

Conception de FSM en VHDL avec la carte Altera DE2

Présenté par :

Luis Fabiano Batista : BATL24077305

Trimestre : Hiver 2013

1. Introduction

Ce projet consiste à développer un contrôleur basé en FSM pour une machine distributrice à boissons, que devra être implémenté sur la carte Altera DE2. Chaque boisson coute 25 cents, et la machine doit accepter monnaies de 25, 10 et 5 cents. Lorsque le montant total égal ou supérieur à 25 cents est inséré, la machine doit libérer la boisson et retourner le change approprie, s'il y a lieu. Les sorties de la FSM sont :

Sortie	Description
Distribuer	Signal pour libérer la boisson
Retourner5	Signal pour libérer une pièce de 5 cents
Retourner10	Signal pour libérer une pièce de 10 cents
RetournerDeux10	Signal pour libérer deux pièces de 10 cents

2. Description Logique de la FSM

2.1 Diagramme de FSM de Mealy

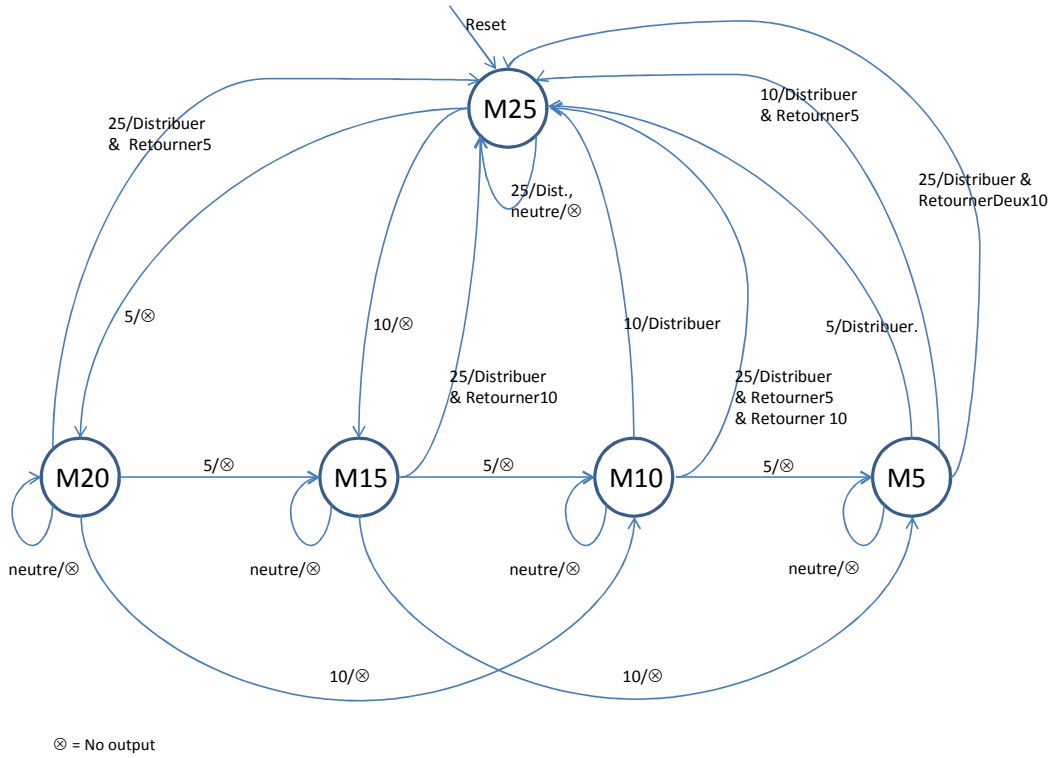

Figure 1 - Diagramme de transition d'état

Pour faciliter la compréhension du diagramme d'état, il faut considérer que chaque nom d'état a un chiffre que correspond au montant nécessaire qui manque pour que la machine puisse distribuer une boisson. Par exemple, M5 indique que 5 cents sont manquants pour que la boisson soit libéré (« M » de « manquant »).

2.2 Codage d'états, d'entrées et de sorties

État	Codage		
	S_2	S_1	S_0
M5	0	0	0
M10	0	0	1
M15	0	1	0
M20	0	1	1
M25	1	0	0

Entrée	Codage	
	I_1	I_0
5 cents	0	0
10 cents	0	1
25 cents	1	0
Neutre[1]	1	1

Sortie	Signal de sortie
Retourner5	O_0
Retouner10	O_1
RetournerDeux10	O_2
Distribuer	O_3

2.3 Table de Transition de État e de Sortie de FSM de Mealy

La table suivante correspond au diagramme de transition d'états et sorties présentés dans la page 02.

[1] L'entrée "neutre" ne causera de changement d'état, et pour question de simplification, il n'a pas été considéré dans le tableau 2.3. Voir la section 3.3 pour plus de renseignement sur son utilité.

État présent			Entrée		Prochain état			Sortie			
S_2	S_1	S_0	I_1	I_0	S'_2	S'_1	S'_0	O_3	O_2	O_1	O_0
0	0	0	0	0	1	0	0	1	0	0	0
0	0	0	0	1	1	0	0	1	0	1	0
0	0	0	1	0	1	0	0	1	1	0	0
0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	0	1	1	0	0	1	0	0	0
0	0	1	1	0	1	0	0	1	0	1	1
0	1	0	0	0	0	0	1	0	0	0	0
0	1	0	0	1	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0	1	0	1	0
0	1	1	0	0	0	1	0	0	0	0	0
0	1	1	0	1	0	0	1	0	0	0	0
0	1	1	1	0	1	0	0	1	0	0	1
1	0	0	0	0	0	1	1	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0
1	0	0	1	0	1	0	0	1	0	0	0

3. Implementation en VHDL

Le code VHDL correspondant au diagramme de la figure 2.1 a été implémenté en utilisant des « statetype », pour simplifier la description du circuit. De cette façon, le codage de chaque état est choisi par le compilateur du logiciel Quartus II.

Le circuit a été implémenté en trois bloc de code VHDL :

- LCD7Segment.vhd : code que sert à contrôler l'allumage d'un chiffre composé par 7 segments de LED.
- LCDDriver.vhd : code que informe aux blocs LCD7Segment.vhd le chiffre être présenté, correspondant au montant de change à être retourné.
- MachineDistributice : code principal de plus haut niveau du système que contient la logique de la machine d'état.

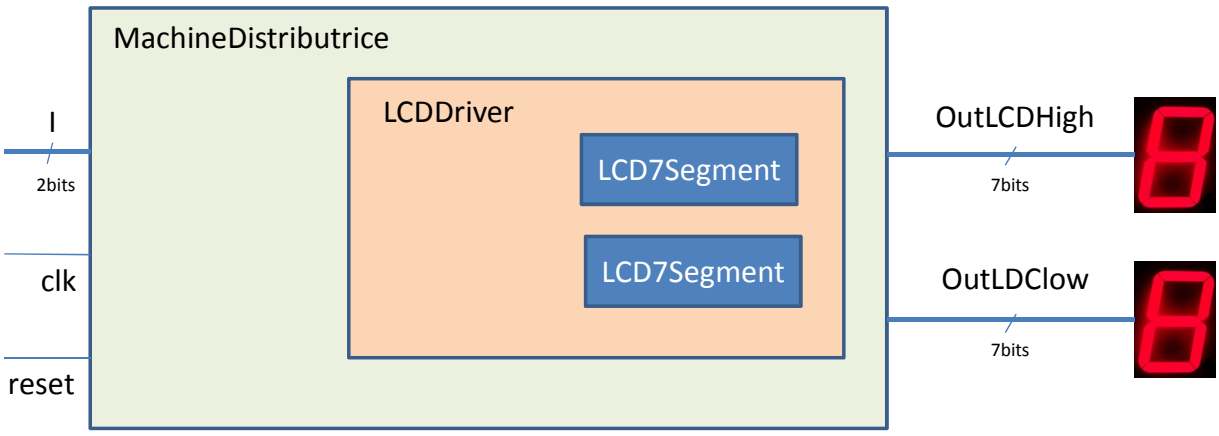

Figure 2 - Relation entre les composants et le code principal

3.1 Affichage sur le LCD 7 Segments

Les composants « LCD7Segment » iront convertir une entré binaire de 4 bits dans au chiffres correspondants, en allument les LEDs pertinents (abcdefg).

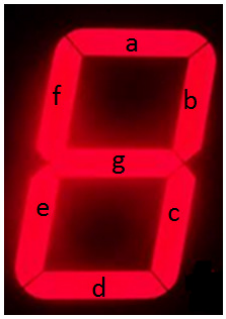

Figure 3 - Identification des signaux sur le segment de 7 LEDs

Pour la carte Altera utilisé dans cet expriment, les 7 segments de chaque chiffre sont allumés avec le niveau 0, donc chaque composent LCD7Segment doit convertir l'entrée à la sortie correspondent à chaque entrée I, conforme démontré dans le code ***LCD7Segment.vhd*** ci-dessous :

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity LCD7Segment is
port (I: in STD_LOGIC_VECTOR(3 downto 0);
segment7: out STD_LOGIC_VECTOR(6 downto 0));
end;

architecture codage of LCD7Segment is
begin
process (I) begin
        case I is
                                        --abcdefg
        when "0000"=> segment7 <="0000001";  -- '0'
        when "0001"=> segment7 <="1001111";  -- '1'
        when "0010"=> segment7 <="0010010";  -- '2'
        when "0011"=> segment7 <="0000110";  -- '3'
        when "0100"=> segment7 <="1001100";  -- '4'
        when "0101"=> segment7 <="0100100";  -- '5'
        when "0110"=> segment7 <="0100000";  -- '6'
        when "0111"=> segment7 <="0001111";  -- '7'
        when "1000"=> segment7 <="0000000";  -- '8'
        when "1001"=> segment7 <="0000100";  -- '9'
 --nothing is displayed when a number more than 9 is given as input.
        when others=> segment7 <="1111111";
        end case;
end process;
end architecture;
```

Notez bien que la logique des bits *abcdefg* est inversé (0 allume et 1 étain le LED du afficheur).

3.2 Affichage la sortie de la machine d'état

Le rôle du code LCDDriver est de passer au bloc LCD7Segment quel sont les chiffres que doivent être affichés pour chaque combinaison d'état de la machine FSM et de ses entrées. Par exemple, si la FSM arrive au résultat « RetournerDeux10 », le bloc composant LCDDriver recevra l'entré $O_2O_1O_0$ = 100 et enverra les signaux 0010b et 0000b (2 et 0) au composants Segment7. Ce bloc encapsule deux composants du type LCD7Segment.

Le code du composent LCDDriver.vhd est présenté ci-dessous.

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity LCDDriver is
port (I: in STD_LOGIC_VECTOR(3 downto 0);
		LCDOutLow, LCDOutHigh: out STD_LOGIC_VECTOR(6 downto 0));
end;


architecture codage of LCDDriver is
component LCD7Segment
port (I: in STD_LOGIC_VECTOR(3 downto 0);
		segment7: out STD_LOGIC_VECTOR(6 downto 0));
end component;
--Ilow and Ihigh are the signals that will drive the LCD7Segment component
signal Ilow, Ihigh: STD_LOGIC_VECTOR(3 downto 0);
begin
	--lowLCD: LCD7Segment port map (Ilow, Olow);
	--highLCD: LCD7Segment port map (Ihigh, Ohigh);
	lowLCD: LCD7Segment port map (Ilow, LCDOutLow);
	highLCD: LCD7Segment port map (Ihigh, LCDOutHigh);
process(I) begin
	if I = "1000" then -- beverage dispensed - no coin returned
		Ilow <= "0000";
		Ihigh <= "1111"; -- turn off all segments
	elsif I = "1001" then -- beverage dispensed - 5 cents returned
		Ilow <= "0101";
		Ihigh <= "1111";
	elsif I = "1010" then -- beverage dispensed - 10 cents returned
		Ilow <= "0000";
		Ihigh <= "0001";
	elsif I = "1011" then -- beverage dispensed - 10 cents returned
		Ilow <= "0101";
		Ihigh <= "0001";
	elsif I = "1100" then -- beverage dispensed - 2x 10 cents returned
		Ilow <= "0000";
		Ihigh <= "0010";
	else
		Ilow <= "1111";
		Ihigh <= "1111";  --nothing is displayed when a number more than 9 is given as input.
	end if;
end process;

end architecture;
```

3.3 Code principal de la machine distributrice

Le code MachineDistributrice.vhd contient la logique de transition d'états et aussi le composant qui encapsulent les autres fonctions auxiliaires (tel comme le LCDDriver).

La machine utilise le signal de clock (clk) pour effectuer les transitions d'états. La FSM utilisé l'stratégie de codage de Mealy, la sortie dépend du état actuel de la FSM et de ses entrées. Pour décider sur le prochain état, les entrées et l'état actuel doivent être stables. Pour ce motif, les transitions d'état (état actuel reçoit la valeur du état future) arrivent au front de descendant du signal d'horloge. De cette façon, la valeur du état future, qui dépend de l'stabilité du état actuel et du signal d'entrée, et calculé seulement au front de montée du signal d'horloge.

Un signal de reset asynchrone a été implémenté pour ce circuit. Si le signal reset est égal a « 1 », le état actuel de la machine passera à M25, et les LEDs seront tous fermés. La logique d'analyse d'états a été insérée dans la branche qui sera exécuté quand reset sera égal à 0.

Une entrée neutre « void » a été ajouté (I= « 1 1 »), parce que, dans une implémentation réelle, le signal de horloge pourra être fourni par un oscillateur. Sans l'existence d'une entrée neutre, chaque transition d'horloge causera une transition dans la machine FSM, parce que les entrées 00, 01 et 10 seront toujours interprétées comme pièces insérées dans la machine. De cette façon, un état neutre permettra que le signal de horloge oscille sans causer des transitions d'état non-désirées.

Le code principal de la machine distributrice est présenté ci-dessous.

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity MachineDistributrice is
	port (clk, reset: in STD_LOGIC;
			distr: out STD_LOGIC;
			I: in STD_LOGIC_VECTOR(1 downto 0);
			OutLCDlow, OutLCDHigh: out STD_LOGIC_VECTOR(6 downto 0));
end;

architecture synth of MachineDistributrice is
	component LCDDriver
		port (I: in STD_LOGIC_VECTOR(3 downto 0);
				LCDOutLow, LCDOutHigh: out STD_LOGIC_VECTOR(6 downto 0));
	end component;

	type statetype is (M25, M20, M15, M10, M5);
	signal state, nextstate: statetype := M25; --declaring states and giving initial state for power on
	signal O: STD_LOGIC_VECTOR(3 downto 0);
	begin
		LCDBar: LCDDriver port map (O,OutLCDlow,OutLCDHigh);
		process (clk,reset,state) begin
			if reset='0' then

				if clk'event and clk='0' then
					state <= nextstate;
				end if;
```

```
if clk'event and clk ='1' then
        case state is
                when M5 => if I = "00" then
                        nextstate <= M25;
                        O <= "1000";
                        elsif I = "01" then
                                nextstate <= M25;
                                O <= "1001";
                        elsif I = "10" then
                                nextstate <= M25;
                                O <= "1100";
                        elsif I = "11" then
                                O <= "0000";
                        end if;
                when M10 => if I = "00" then
                                nextstate <= M5;
                                O <= "0000";
                        elsif I = "01" then
                                nextstate <= M25;
                                O <= "1000";
                        elsif I = "10" then
                                nextstate <= M25;
                                O <= "1011";
                        elsif I = "11" then
                                O <= "0000";
                        end if;
                when M15 => if I = "00" then
                                nextstate <= M10;
                                O <= "0000";
                        elsif I = "01" then
                                nextstate <= M5;
                                O <= "0000";
                        elsif I = "10" then
                                nextstate <= M25;
                                O <= "1010";
                        elsif I = "11" then
                                O <= "0000";
                        end if;
                when M20 => if I = "00" then
                                nextstate <= M15;
                                O <= "0000";
                        elsif I = "01" then
                                nextstate <= M10;
                                O <= "0000";
                        elsif I = "10" then
                                nextstate <= M25;
                                O <= "1001";
                        elsif I = "11" then
                                O <= "0000";
                        end if;
```

```
                                        when M25 => if I = "00" then
                                                        nextstate <= M20;
                                                        O <= "0000";
                                                    elsif I = "01" then
                                                        nextstate <= M15;
                                                        O <= "0000";
                                                    elsif I = "10" then
                                                        nextstate <= M25;
                                                        O <= "1000";
                                                    elsif I = "11" then
                                                        O <= "0000";
                                                    end if;
                                    end case;
                                end if;
                            else
                                state <= M25;
                                nextstate <= M25;
                                O <= "0000";
                            end if;
                    end process;
            distr <= O(3);
end;
```

3.4 Testbench

Un code de testbench a été développé pour l'évaluation du bon fonctionnement de la machine. Le période du signal de horloge du testbench est de 20ns, et les signaux d'entrée ont été générés de façon qu'ils soient stables pendent le front de montée du signal d'horloge.

Pour chaque cycle d'horloge, les entrées et les sorties ont été évaluées. Le code du testbench est presenté ci-dessous.

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity MachineDistributrice_tb is
end;

architecture analysis of MachineDistributrice_tb is
        component MachineDistributrice
                port (clk, reset: in STD_LOGIC;
                                distr: out STD_LOGIC;
                                I: in STD_LOGIC_VECTOR(1 downto 0);
                                OutLCDlow, OutLCDHigh: out STD_LOGIC_VECTOR(6 downto 0));
        end component;

        signal clk, reset: STD_LOGIC :='0';
```

```
	signal distr: STD_LOGIC;
	signal I: STD_LOGIC_VECTOR(1 downto 0);
	signal OutLCDlow, OutLCDHigh: STD_LOGIC_VECTOR(6 downto 0);
	begin
		DUT: MachineDistributrice port map (clk, reset, distr, I, OutLCDlow, OutLCDHigh);

		-- ############ SIMULATION DESCRIPTION #################
		-- Input signal I will be synthetized to verify the machine state dynamic behavior and
outputs
		--  Input signal I will be generated in the falling edge of the system clock signal, because
		-- it is necessary to make sure the signals (next state and I) are stable in the flip-flop
inputs
		-- The ¨state¨ variable will receive the value of ¨nextstate¨ when the clock´s falling edge
		-- In the clock risign edge the ¨state¨ and ¨I¨ variables are evaluateed to calculate the
next state.
		-- At the same time, the output signals are determined.
		-- In this test, the output signals will be evaluated at 75% of the clock period, because the
signals
		-- will not be stable in the clock edge. Since the clock period is 20ns, the output signal
will be
		-- asserted 15ns after the clock rising edge.

		--simulation of a clock signal with a 20ns period
		process begin
			wait for 10ns; clk <= not clk;
		end process;

		process begin

		-- ######################################################################
		-- Initialization test: M25--> M25, no coin,  no change returned, no beverage distribution
		I <="11"; --> void signal (no coin inserted)
		wait for 15ns;
		assert OutLCDlow="1111111" report "Test 0: OutLCDlow failed for I=11, staying at
same state";
		assert OutLCDHigh="1111111" report "Test 0: OutLCDHigh failed for I=11, staying at
same state";
		assert distr='0' report "Test 2: distr failed for I=11, staying at same state";
		wait for 5ns;

		-- ######################################################################
		-- First test: M25--> M25, distribute,  no change returned
		-- State M25, inserting 25cents, going to M25

		I <= "10";
		wait for 15ns;
		assert OutLCDlow = "0000001" report "Test 1: OutLCDlow failed for I=10, from state
M25 to M25";
		assert OutLCDHigh = "1111111" report "Test 1: OutLCDHigh failed for I=10, from
state M25 to M25";
		assert distr='1' report "Test 1: distr failed for I=10, from state M25 to M25";
```

```
		wait for 5ns;


		-- ####################################################################
		-- Second test: M25--> M20 --> M25, distribute,  return 5 cents
		-- State M25, inserting 5cents, going to M20
		I <= "00";
		wait for 15ns;
		assert OutLCDlow="1111111" report "Test 2: OutLCDlow failed for I=00, from state
M25 to M20";
		assert OutLCDHigh="1111111" report "Test 2: OutLCDHigh failed for I=00, from state
M25 to M20";
		assert distr='0' report "Test 2: distr failed for I=00, from state M25 to M20";
		wait for 5ns;

		-- State M20, staying at same state for once clock cycle
		I <="11"; --> void signal (no coin inserted)
		wait for 15ns;
		assert OutLCDlow="1111111" report "Test 0: OutLCDlow failed for I=11, staying at
same state";
		assert OutLCDHigh="1111111" report "Test 0: OutLCDHigh failed for I=11, staying at
same state";
		assert distr='0' report "Test 2: distr failed for I=11, staying at same state";
		wait for 5ns;


		-- State M20, inserting 25cents, going to M25
		I <= "10";
		wait for 15ns;
		assert OutLCDlow="0100100" report "Test 2: OutLCDlow failed for I=10, from state
M20 to M25";
		assert OutLCDHigh="1111111" report "Test 2: OutLCDHigh failed for I=10, from state
M20 to M25";
		assert distr='1' report "Test 2: distr failed for I=10, from state M20 to M25";
		wait for 5ns;

		-- ####################################################################
		-- Third test: M25--> M15 --> M25, distribute,  return 10 cents
		-- State M25, inserting 10cents, going to M15
		I <= "01";
		wait for 15ns;
		assert OutLCDlow="1111111" report "Test 3: OutLCDlow failed for I=01, from state
M25 to M15";
		assert OutLCDHigh="1111111" report "Test 3: OutLCDHigh failed for I=01, from state
M25 to M15";
		assert distr='0' report "Test 3: distr failed for I=01, from state M25 to M15";
		wait for 5ns;

		-- State M15, staying at same state for once clock cycle
		I <="11"; --> void signal (no coin inserted)
		wait for 15ns;
```

```
                assert OutLCDlow="1111111" report "Test 0: OutLCDlow failed for I=11, staying at
same state";
                assert OutLCDHigh="1111111" report "Test 0: OutLCDHigh failed for I=11, staying at
same state";
                assert distr='0' report "Test 2: distr failed for I=11, staying at same state";
                wait for 5ns;

                -- State M15, inserting 25cents, distribute, going to M25
                I <= "10";
                wait for 15ns;
                assert OutLCDlow="0000001" report "Test 3: OutLCDlow failed for I=10, from state
M15 to M25";
                assert OutLCDHigh="1001111" report "Test 3: OutLCDHigh failed for I=10, from state
M15 to M25";
                assert distr='1' report "Test 3: distr failed for I=10, from state M15 to M25";
                wait for 5ns;

                -- ######################################################################
                -- Fourth test: M25--> M20 --> M10 --> M25, distribute,  no change returned
                -- State M25, inserting 5cents, going to M20
                I <= "00";
                wait for 15ns;
                assert OutLCDlow="1111111" report "Test 4: OutLCDlow failed for I=00, from state
M25 to M20";
                assert OutLCDHigh="1111111" report "Test 4: OutLCDHigh failed for I=00, from state
M5 to M20";
                assert distr='0' report "Test 4: distr failed for I=00, from state M25 to M20";
                wait for 5ns;

                -- State M20, inserting 10cents, going to M10
                I <= "01";
                wait for 15ns;
                assert OutLCDlow="1111111" report "Test 4: OutLCDlow failed for I=01, from state
M20 to M10";
                assert OutLCDHigh="1111111" report "Test 4: OutLCDHigh failed for I=01, from state
M20 to M10";
                assert distr='0' report "Test 4: distr failed for I=01, from state M20 to M10";
                wait for 5ns;

                -- State M10, staying at same state for once clock cycle
                I <="11"; --> void signal (no coin inserted)
                wait for 15ns;
                assert OutLCDlow="1111111" report "Test 0: OutLCDlow failed for I=11, staying at
same state";
                assert OutLCDHigh="1111111" report "Test 0: OutLCDHigh failed for I=11, staying at
same state";
                assert distr='0' report "Test 2: distr failed for I=11, staying at same state";
                wait for 5ns;

                -- State M10, inserting 10cents, going to M25
                I <= "01";
```

```
		wait for 15ns;
		assert OutLCDlow="0000001" report "Test 4: OutLCDlow failed for I=01, from state
M10 to M25";
		assert OutLCDHigh="1111111" report "Test 4: OutLCDHigh failed for I=01, from state
M10 to M25";
		assert distr='1' report "Test 4: distr failed for I=01, from state M10 to M25";
		wait for 5ns;

		-- ######################################################################
		-- Fifth test: M25--> M15 --> M10 --> M25, distribute,  return 5 and 10
		-- State M25, inserting 10cents, going to M15
		I <= "01";
		wait for 15ns;
		assert OutLCDlow="1111111" report "Test 5: OutLCDlow failed for I=01, from state
M25 to M15";
		assert OutLCDHigh="1111111" report "Test 5: OutLCDHigh failed for I=01, from state
M25 to M15";
		assert distr='0' report "Test 5: distr failed for I=01, from state M25 to M15";
		wait for 5ns;

		-- State M15, inserting 5cents, going to M10
		I <= "00";
		wait for 15ns;
		assert OutLCDlow="1111111" report "Test 5: OutLCDlow failed for I=00, from state
M15 to M10";
		assert OutLCDHigh="1111111" report "Test 5: OutLCDHigh failed for I=00, from state
M15 to M10";
		assert distr='0' report "Test 5: distr failed for I=00, from state M15 to M10";
		wait for 5ns;

		-- State M10, inserting 25cents, going to M25
		I <= "10";
		wait for 15ns;
		assert OutLCDlow="0100100" report "Test 5: OutLCDlow failed for I=10, from state
M10 to M25";
		assert OutLCDHigh="1001111" report "Test 5: OutLCDHigh failed for I=10, from state
M10 to M25";
		assert distr='1' report "Test 5: distr failed for I=00, from state M10 to M25";
		wait for 5ns;

		-- ######################################################################
		-- Sixth test: M25--> M15 --> M5 --> M25, distribute,  no return
		-- State M25, inserting 10cents, going to M15
		I <= "01";
		wait for 15ns;
		assert OutLCDlow="1111111" report "Test 6: OutLCDlow failed for I=01, from state
M25 to M15";
		assert OutLCDHigh="1111111" report "Test 6: OutLCDHigh failed for I=01, from state
M25 to M15";
		assert distr='0' report "Test 6: distr failed for I=01, from state M25 to M15";
		wait for 5ns;
```

```
                -- State M15, inserting 10cents, going to M5
                I <= "01";
                wait for 15ns;
                assert OutLCDlow="1111111" report "Test 6: OutLCDlow failed for I=01, from state
M15 to M5";
                assert OutLCDHigh="1111111" report "Test 6: OutLCDHigh failed for I=01, from state
M15 to M5";
                assert distr='0' report "Test 6: distr failed for I=01, from state M15 to M5";
                wait for 5ns;

                -- State M5, staying at same state for once clock cycle
                I <="11"; --> void signal (no coin inserted)
                wait for 15ns;
                assert OutLCDlow="1111111" report "Test 0: OutLCDlow failed for I=11, staying at
same state";
                assert OutLCDHigh="1111111" report "Test 0: OutLCDHigh failed for I=11, staying at
same state";
                assert distr='0' report "Test 2: distr failed for I=11, staying at same state";
                wait for 5ns;

                -- State M5, inserting 5cents, going to M25
                I <= "00";
                wait for 15ns;
                assert OutLCDlow="0000001" report "Test 6: OutLCDlow failed for I=00, from state
M5 to M25";
                assert OutLCDHigh="1111111" report "Test 6: OutLCDHigh failed for I=00, from state
M5 to M25";
                assert distr='1' report "Test 6: distr failed for I=00, from state M5 to M25";
                wait for 5ns;

                -- ######################################################################
                -- Seventh test: M25--> M15 --> M5 --> M25, distribute,  return 5 cents
                -- State M25, inserting 10cents, going to M15
                I <= "01";
                wait for 15ns;
                assert OutLCDlow="1111111" report "Test 7: OutLCDlow failed for I=01, from state
M25 to M15";
                assert OutLCDHigh="1111111" report "Test 7: OutLCDHigh failed for I=01, from state
M25 to M15";
                assert distr='0' report "Test 7: distr failed for I=01, from state M25 to M15";
                wait for 5ns;

                -- State M15, inserting 10cents, going to M5
                I <= "01";
                wait for 15ns;
                assert OutLCDlow="1111111" report "Test 7: OutLCDlow failed for I=01, from state
M15 to M5";
                assert OutLCDHigh="1111111" report "Test 7: OutLCDHigh failed for I=01, from state
M15 to M5";
                assert distr='0' report "Test 7: distr failed for I=01, from state M15 to M5";
```

```
                wait for 5ns;

                -- State M5, inserting 10 cents, going to M25
                I <= "01";
                wait for 15ns;
                assert OutLCDlow="0100100" report "Test 7: OutLCDlow failed for I=01, from state
M5 to M25";
                assert OutLCDHigh="1111111" report "Test 7: OutLCDHigh failed for I=01, from state
M5 to M25";
                assert distr='1' report "Test 7: distr failed for I=01, from state M5 to M25";
                wait for 5ns;

                -- ######################################################################
                -- Eighth test: M25--> M20 --> M15 --> M10 --> M5 --> M25, distribute,  return 2X 10
cents
                -- State M25, inserting 5cents, going to M20
                I <= "00";
                wait for 15ns;
                assert OutLCDlow="1111111" report "Test 8: OutLCDlow failed for I=00, from state
M25 to M20";
                assert OutLCDHigh="1111111" report "Test 8: OutLCDHigh failed for I=00, from state
M5 to M20";
                assert distr='0' report "Test 8: distr failed for I=00, from state M25 to M20";
                wait for 5ns;

                -- State M20, inserting 5cents, going to M15
                I <= "00";
                wait for 15ns;
                assert OutLCDlow="1111111" report "Test 8: OutLCDlow failed for I=00, from state
M20 to M15";
                assert OutLCDHigh="1111111" report "Test 8: OutLCDHigh failed for I=00, from state
M20 to M15";
                assert distr='0' report "Test 8: distr failed for I=00, from state M20 to M15";
                wait for 5ns;

                -- State M15, inserting 5 cents, going to M10
                I <= "00";
                wait for 15ns;
                assert OutLCDlow="1111111" report "Test 8: OutLCDlow failed for I=00, from state
M15 to M10";
                assert OutLCDHigh="1111111" report "Test 8: OutLCDHigh failed for I=00, from state
M15 to M10";
                assert distr='0' report "Test 8: distr failed for I=00, from state M15 to M10";
                wait for 5ns;

                -- State M10, inserting 5 cents, going to M5
                I <= "00";
                wait for 15ns;
                assert OutLCDlow="1111111" report "Test 8: OutLCDlow failed for I=00, from state
M10 to M5";
```

```
		assert OutLCDHigh="1111111" report "Test 8: OutLCDHigh failed for I=01, from state
M10 to M5";
		assert distr='0' report "Test 8: distr failed for I=00, from state M10 to M5";
		wait for 5ns;

		-- State M5, inserting 25 cents, going to M25
		I <= "10";
		wait for 15ns;
		assert OutLCDlow="0000001" report "Test 8: OutLCDlow failed for I=10, from state
M5 to M25";
		assert OutLCDHigh="0010010" report "Test 8: OutLCDHigh failed for I=01, from state
M5 to M25";
		assert distr='1' report "Test 8: distr failed for I=10, from state M5 to M25";
		wait for 5ns;

		--#####################################################
		-- This final test will evaluate if the RESET function works properly
		-- Nineth test: M25 -> M20 -> RESET -> M25 --> M25, distribute, no return

		-- State M25, inserting 5cents, going to M20
		I <= "00";
		wait for 15ns;
		assert OutLCDlow="1111111" report "Test 9: OutLCDlow failed for I=00, from state
M25 to M20";
		assert OutLCDHigh="1111111" report "Test 9: OutLCDHigh failed for I=00, from state
M5 to M20";
		assert distr='0' report "Test 9: distr failed for I=00, from state M25 to M20";
		wait for 5ns;

		-- reset pulse (arbitrary duration, in this case 37ns)
		reset <= '1';
		assert OutLCDlow="1111111" report "Test 9: OutLCDlow failed for reset";
		assert OutLCDHigh="1111111" report "Test 9: OutLCDHigh failed for reset";
		assert distr='0' report "Test 9: distr failed for reset";
		wait for 37ns;
		reset <= '0';


		-- State M25, inserting 25cents, going to M25

		I <= "10";
		wait for 15ns;
		assert OutLCDlow = "0000001" report "Test 1: OutLCDlow failed for I=10, from state
M25 to M25";
		assert OutLCDHigh = "1111111" report "Test 1: OutLCDHigh failed for I=10, from
state M25 to M25";
		assert distr='1' report "Test 1: distr failed for I=10, from state M25 to M25";
		wait for 5ns;
		--

		wait; -- wait forever
```

```
                end process;
end;
```

Avec le code testbench montré ci-dessus, les conditions suivantes ont été évaluées :

Test	Transition d'état et sortie	Résultat attendu	Verdict
0	M25 →M25, pas de monnaie inséré	Pas de retour de monnaie, pas de distribution de boisson	OK
1	M25 → M25, 25 cents inséré	Pas de retour de monnaie, boisson distribué	OK
2	M25→ M20→M25, 5 e 25 cents insérés, avec entrée neutre au milieu	Retourne 5 cents, boisson distribué	OK
3	M25 → M15 → M25, 10 e 25 cents insérés, avec entrée neutre au milieu	Retourne 10 cents, boisson distribué	OK
4	M25→ M20→ M10→ M25, 5, 10 et 10 cents insérés, avec entrée neutre au milieu	Pas de retour de monnaie, boisson distribué	OK
5	M25→M15→M10→M25, 10, 5 et 25 insérés	Retourne 5 et 10 cents, boisson distribué	OK
6	M25→M15→M5→M25, 10, 10 et 5 cents insérés, avec entrée neutre au milieu	Pas de retour de monnaie, boisson distribué	OK
7	M25→M15→M5→M25, 3 pièces de 10 cents insérés	Retourne 5 cents, boisson distribué	OK
8	M25→M20→M15→M10→M5→M25, 4 pièces de 10 cents et une pièce de 25 cents insérés	Retourne 2 pièces de 10 cents, boisson distribué	OK
9	M25→M20→(Reset) → M25→M25, 25 cents inséré	Pas de retour de monnaie, boisson distribué	OK

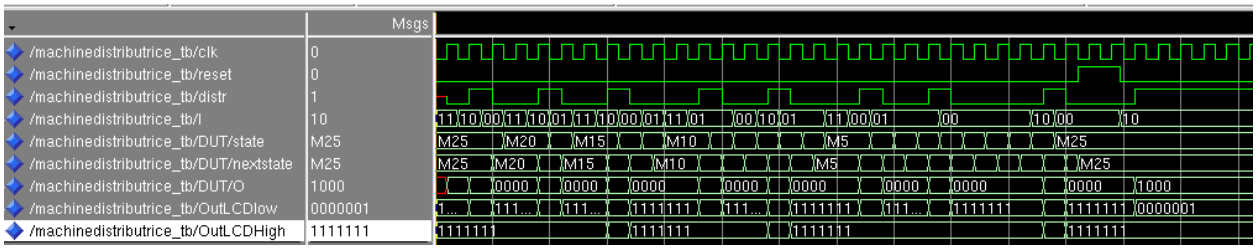

Figure 4 - Testbench - Signaux d'entrée et sortie

4. Affectations de brochage dans la carte Altera

Le brochage de la carte a été choisi conforme indiqué ci-dessous :

Signal d'entrée et sortie du code principal VHDL	Variable correspondant	Type E=Entrée S =Sortie	Composant dans la carte Altera
Horloge	clk	E	KEY3
Pièce inseré (bit 0)	I(0)	E	SW0
Pièce inseré (bit 1)	I(1)	E	SW1
Reset	reset	E	SW3
Distribuer boisson	distr	E	LEDG7
Segment LCD 7 – (chiffe moins significatif) – bit a	OutLCDlow(6)	S	HEX0[0]
Segment LCD 7 – (chiffe moins significatif) – bit b	OutLCDlow(5)	S	HEX0[1]
Segment LCD 7 – (chiffe moins significatif) – bit c	OutLCDlow(4)	S	HEX0[2]
Segment LCD 7 – (chiffe moins significatif) – bit d	OutLCDlow(3)	S	HEX0[3]
Segment LCD 7 – (chiffe moins significatif) – bit e	OutLCDlow(2)	S	HEX0[4]
Segment LCD 7 – (chiffe moins significatif) – bit f	OutLCDlow(1)	S	HEX0[5]
Segment LCD 7 – (chiffe moins significatif) – bit g	OutLCDlow(0)	S	HEX0[6]
Segment LCD 7 – (chiffe plus significatif) – bit a	OutLCDHigh(6)	S	HEX1[0]
Segment LCD 7 – (chiffe plus significatif) – bit b	OutLCDHigh(5)	S	HEX1[1]
Segment LCD 7 – (chiffe plus significatif) – bit c	OutLCDHigh(4)	S	HEX1[2]
Segment LCD 7 – (chiffe plus significatif) – bit d	OutLCDHigh(3)	S	HEX1[3]
Segment LCD 7 – (chiffe plus significatif) – bit e	OutLCDHigh(2)	S	HEX1[4]
Segment LCD 7 – (chiffe plus significatif) – bit f	OutLCDHigh(1)	S	HEX1[5]
Segment LCD 7 – (chiffe plus significatif) – bit g	OutLCDHigh(0)	S	HEX1[6]

Notez bien que le KEY3 correspond au « push button » sur la carte Altera DE2. Quand le bouton est poussé, le signal de l'horloge passe à 0, et quand relâché, le signal remonte à 1.

5. Fichiers VHDL

Les fichiers VHDL correspondant à ce rapport sont inclus dans le fichier VHDL.zip fourni avec ce document.