Porcelli Gianluca

Prof. DE LEONARDIS Francesco

Digital Programmable Systems

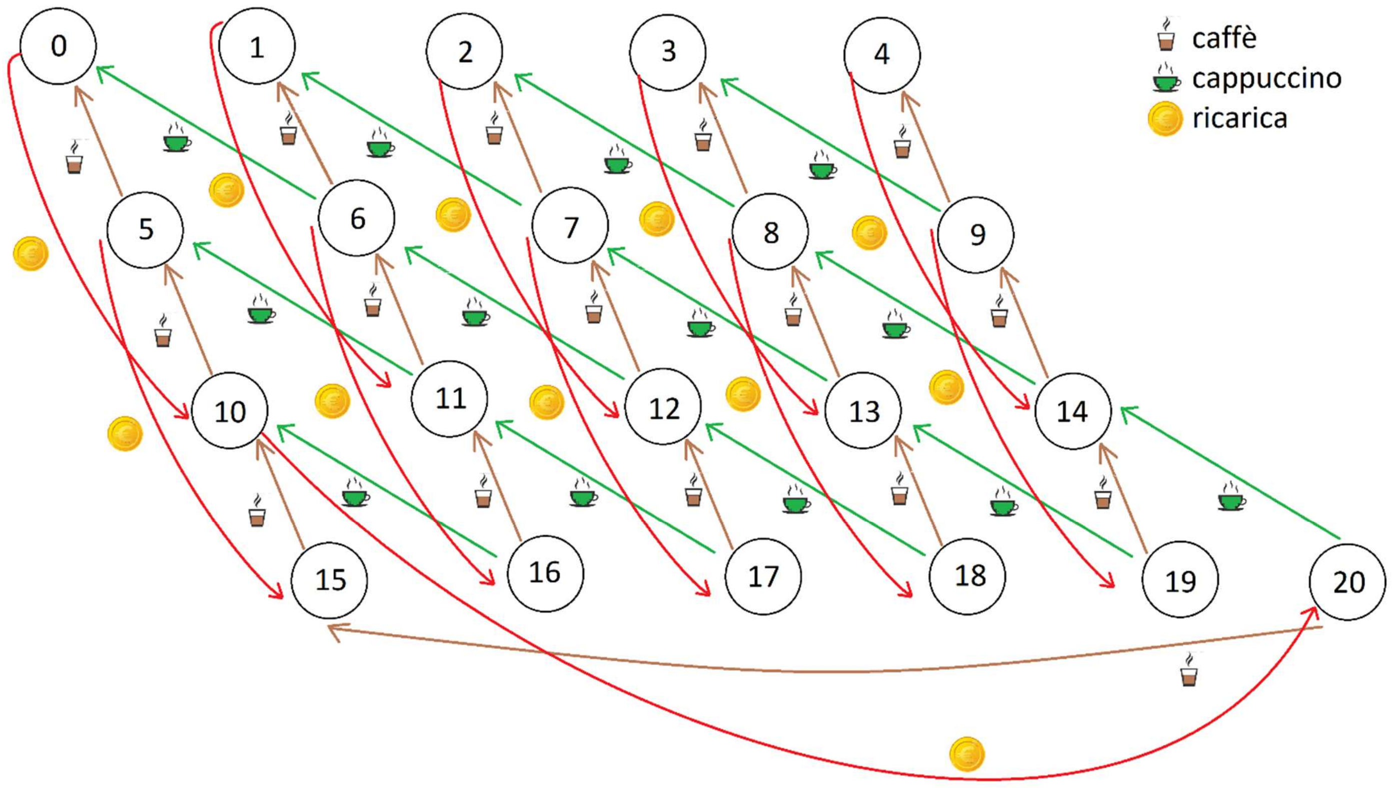
Automatic coffee machine

The work presented is designed to simulate the operation of an automatic coffee machine. It will manage the machine operation by exploiting the processing capabilities of the FPGA.

Indice

1.Design requirements

This project has the task of managing, starting from a variable initial situation, the behavior of a coffee machine. Analyzing the problem to manage, I decided to implement a Mealy's machine. **Mealy's machine** is a finite-state auotomaton whose output values are determined by the current state and the current input, unlike the Moore machine, which instead works only as a function of the current state. However, not all Mealy machines can be converted in an equivalent Moore machine. 21 states have been provided for which the remaining cash amount is linked. The default initial status is represented by the absence of residual credit (status "zero"), but it can be modified by entering the code of an eventual credit key (on which there is a cash credit) that will allow the reset of the current status. The key simulation has been implemented with 5 inputs (c1, c2, c3, c4, c5) with which it will be possible to represent all the possible 21 states of the machine (with a different combination of the inputs there would be a different state of the machine). With the machine in question it will be possible to manage a credit equal to € 2.00 but, increasing the number of states, any type of maximum amount that can be entered will be proportionally guaranteed. In relation to the number of current states, the top-up operation can be carried out up to the state "ten" (the last state reachable is "twenty" which corresponde € 2.00 of credit). The configuration of the states with the relative status changes is described below and will be understood through the description of the simulations. Each simulated process will describe a fundamental operation for the purpose but pieces of the entire project would be reused for other purposes.



The state from which the arrow starts is the current state stateMealy_reg, while the state pointed by the arrow is the state reached (stateMealy_next) starting from the current state when one of the allowed inputs is detected. The possible inputs are in the upper right corner and they are not evaluated for all states because in some situations, some of them will not be allowed. Given the impossibility of blocking an input signal that is not allowed, it will be ignored when assessing the future state. Since a priority of reading of the inputs has been foreseen, in the case of more than one active input, if the input with higher priority is not admissible in that specific state (and the other instead is lawful) no change of status and the ineligible input signal will be signaled.

Work instants:

I define the 3 work instants that will be better understand when the interval variable is treated:

T-	: instant just before the rising edge of the clock.
T	: rising edge of the clock.
T+	: moment that follows the ascent front.

WARNING: be careful when calculating the instants that must be integers.

Reset operation:

RESET = 1:

1. Instantly update the current status stateMealy_reg.
2. In correspondence with "**T-**" stateMealy_next follows the stateMealy_reg signal.
3. At the instant "**T+**" reset the value of all the variables associated to the inputs.

RESET different from 1:

1. In "**T-**" the current state stateMealy_reg follows the next state stateMealy_next.
2. In "**T**" the stateMealy_next next state is calculated based on stateMealy_reg and the input signals.
3. At the moment "**T+**" the variables representing the inputs will be reset.

The next status (if reset is different from 1) represents the residual credit after the input driven operation. The input operation is essentially a sum or subtraction of the amount actually present in the machine; after having calculated the remaining amount, the actual machine status is temporarily stored in stateMealy_next followed by stateMealy_reg in an instant later.

stateMealy_type:

The stateMealy_type will be a variable type of our project. It may contain one of the 21 states hypothesized for this project. Two stateMealy_type variables will be created: stateMealy_reg and stateMealy_next.

2.VHDL code

```
LIBRARY ieee;
LIBRARY STD;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_unsigned.ALL;
USE IEEE.numeric_std.ALL;

ENTITY coffe IS
        PORT (
                clk, reset : IN std_logic;
                caffe_b : IN std_logic;
                cappuccino_b : IN std_logic;
                ricarica_b : IN std_logic;
                c1 : IN std_logic;
                c2 : IN std_logic;
                c3 : IN std_logic;
                c4 : IN std_logic;
                c5 : IN std_logic;

                LED_out : OUT bit_vector (0 TO 6)
        );
END coffe;

ARCHITECTURE arch OF coffe IS

        TYPE stateMealy_type IS (zero, uno, due, tre, quattro, cinque, sei, sette, otto, nove, dieci, undici,
dodici, tredici, quattordici, quindici, sedici, diciassette, diciotto, diciannove, venti); -- 21 stati
        SIGNAL stateMealy_reg, stateMealy_next : stateMealy_type;
        SIGNAL count : INTEGER := 0;
        SIGNAL max : INTEGER := 3;
        SIGNAL interval : INTEGER := 100;
        SIGNAL caffe : std_logic := '0';
        SIGNAL cappuccino : std_logic := '0';
        SIGNAL ricarica : std_logic := '0';
        SIGNAL tmp : std_logic := '0';
        SIGNAL clock_out : std_logic := '0';

BEGIN
        PROCESS (clk)
        BEGIN
                IF (clk'EVENT AND clk = '1') THEN
                        count <= count + 1;
                        IF (count = max) THEN
                                tmp <= NOT tmp;
                                count <= 1;
                        END IF;
                END IF;
                clock_out <= tmp;
        END PROCESS;

        PROCESS (clock_out, reset, stateMealy_reg, count)
        BEGIN
                IF (reset = '1') THEN
                        IF ((c1 = '0') AND (c2 = '0') AND (c3 = '0') AND (c4 = '0') AND (c5 = '0')) THEN
                                stateMealy_reg <= zero;--stato iniziale
                        ELSIF ((c1 = '0') AND (c2 = '0') AND (c3 = '0') AND (c4 = '0') AND (c5 = '1')) THEN
                                stateMealy_reg <= uno;--stato iniziale
                        ELSIF ((c1 = '0') AND (c2 = '0') AND (c3 = '0') AND (c4 = '1') AND (c5 = '0')) THEN
                                stateMealy_reg <= due;--stato iniziale
                        ELSIF ((c1 = '0') AND (c2 = '0') AND (c3 = '0') AND (c4 = '1') AND (c5 = '1')) THEN
                                stateMealy_reg <= tre;--stato iniziale
                        ELSIF ((c1 = '0') AND (c2 = '0') AND (c3 = '1') AND (c4 = '0') AND (c5 = '0')) THEN
                                stateMealy_reg <= quattro;--stato iniziale
                        ELSIF ((c1 = '0') AND (c2 = '0') AND (c3 = '1') AND (c4 = '0') AND (c5 = '1')) THEN
                                stateMealy_reg <= cinque;--stato iniziale
                        ELSIF ((c1 = '0') AND (c2 = '0') AND (c3 = '1') AND (c4 = '1') AND (c5 = '0')) THEN
                                stateMealy_reg <= sei;--stato iniziale
                        ELSIF ((c1 = '0') AND (c2 = '0') AND (c3 = '1') AND (c4 = '1') AND (c5 = '1')) THEN
                                stateMealy_reg <= sette;--stato iniziale
                        ELSIF ((c1 = '0') AND (c2 = '1') AND (c3 = '0') AND (c4 = '0') AND (c5 = '0')) THEN
                                stateMealy_reg <= otto;--stato iniziale
                        ELSIF ((c1 = '0') AND (c2 = '1') AND (c3 = '0') AND (c4 = '0') AND (c5 = '1')) THEN
                                stateMealy_reg <= nove;--stato iniziale
                        ELSIF ((c1 = '0') AND (c2 = '1') AND (c3 = '0') AND (c4 = '1') AND (c5 = '0')) THEN
                                stateMealy_reg <= dieci;--stato iniziale
                        ELSIF ((c1 = '0') AND (c2 = '1') AND (c3 = '0') AND (c4 = '1') AND (c5 = '1')) THEN
                                stateMealy_reg <= undici;--stato iniziale
                        ELSIF ((c1 = '0') AND (c2 = '1') AND (c3 = '1') AND (c4 = '0') AND (c5 = '0')) THEN
                                stateMealy_reg <= dodici;--stato iniziale
                        ELSIF ((c1 = '0') AND (c2 = '1') AND (c3 = '1') AND (c4 = '0') AND (c5 = '1')) THEN
                                stateMealy_reg <= tredici;--stato iniziale
                        ELSIF ((c1 = '0') AND (c2 = '1') AND (c3 = '1') AND (c4 = '1') AND (c5 = '0')) THEN
                                stateMealy_reg <= quattordici;--stato iniziale
                        ELSIF ((c1 = '0') AND (c2 = '1') AND (c3 = '1') AND (c4 = '1') AND (c5 = '1')) THEN
                                stateMealy_reg <= quindici;--stato iniziale
                        ELSIF ((c1 = '1') AND (c2 = '0') AND (c3 = '0') AND (c4 = '0') AND (c5 = '0')) THEN
                                stateMealy_reg <= sedici;--stato iniziale
                        ELSIF ((c1 = '1') AND (c2 = '0') AND (c3 = '0') AND (c4 = '0') AND (c5 = '1')) THEN
                                stateMealy_reg <= diciassette;--stato iniziale
                        ELSIF ((c1 = '1') AND (c2 = '0') AND (c3 = '0') AND (c4 = '1') AND (c5 = '0')) THEN
                                stateMealy_reg <= diciotto;--stato iniziale
                        ELSIF ((c1 = '1') AND (c2 = '0') AND (c3 = '0') AND (c4 = '1') AND (c5 = '1')) THEN
                                stateMealy_reg <= diciannove;--stato iniziale
                        ELSIF ((c1 = '1') AND (c2 = '0') AND (c3 = '1') AND (c4 = '0') AND (c5 = '0')) THEN
                                stateMealy_reg <= venti;--stato iniziale

                        END IF;

                ELSIF (count = max * interval/100 AND clock_out = '0') THEN
                        stateMealy_reg <= stateMealy_next;
                ELSE
                        NULL;
                END IF;
        END PROCESS;
        PROCESS (caffe_b, count, clock_out)
        BEGIN
                IF (caffe_b'EVENT AND caffe_b = '1') THEN
                                caffe <= '1';
                END IF;
                IF (count = 2 * 100/interval AND clock_out = '1') THEN
                                caffe <= '0';
                END IF;

        END PROCESS;
        PROCESS (cappuccino_b, count, clock_out)
        BEGIN
                IF (cappuccino_b'EVENT AND cappuccino_b = '1') THEN
                                cappuccino <= '1';
                END IF;
                IF (count = 2 * 100/interval AND clock_out = '1') THEN
                                cappuccino <= '0';
                END IF;

        END PROCESS;
        PROCESS (ricarica_b, count, clock_out)
        BEGIN
                IF (ricarica_b'EVENT AND ricarica_b = '1') THEN
                                        ricarica <= '1';
                END IF;
                IF (count = 2 * 100/interval AND clock_out = '1') THEN
                                        ricarica <= '0';
                END IF;

        END PROCESS;
        --processo machine Mealy

        PROCESS (stateMealy_reg, caffe, cappuccino, ricarica, reset, clock_out, count)
        BEGIN
                IF (reset = '1') THEN
                        IF (count = max * interval/100 AND clock_out = '0') THEN
                                stateMealy_next <= stateMealy_reg;
                        END IF;

                ELSIF (clock_out'EVENT AND clock_out = '1') THEN

                        CASE stateMealy_reg IS
                                WHEN zero =>

                                --
                                        IF (caffe = '1') THEN
                                                stateMealy_next <= zero; --NON PERMESSO
                                                LED_out <= "0110000";
                                        ELSIF (cappuccino = '1') THEN
                                                stateMealy_next <= zero; --NON PERMESSO
                                                LED_out <= "0110000";
                                        ELSIF (ricarica = '1') THEN
                                                stateMealy_next <= dieci;
                                                LED_out <= "0111000";
                                        END IF;

                                WHEN uno =>
                                        --
                                        IF (caffe = '1') THEN
                                                stateMealy_next <= uno; --NON PERMESSO
                                                LED_out <= "0110000";
                                        ELSIF (cappuccino = '1') THEN
                        stateMealy_next <= uno; --NON PERMESSO
                        LED_out <= "0110000";
                ELSIF (ricarica = '1') THEN
                        stateMealy_next <= undici;
                        LED_out <= "0111000";
                END IF;

        WHEN due =>
        --
                IF caffe = '1' THEN
                        stateMealy_next <= due; --NON PERMESSO
                        LED_out <= "0110000";
                ELSIF cappuccino = '1' THEN
                        stateMealy_next <= due; --NON PERMESSO
                        LED_out <= "0110000";
                ELSIF ricarica = '1' THEN
                        stateMealy_next <= dodici;
                        LED_out <= "0111000";
                END IF;

        WHEN tre =>
        --
                IF caffe = '1' THEN
                        stateMealy_next <= tre; --NON PERMESSO
                        LED_out <= "0110000";
                ELSIF cappuccino = '1' THEN
                        stateMealy_next <= tre; --NON PERMESSO
                        LED_out <= "0110000";
                ELSIF ricarica = '1' THEN
                        stateMealy_next <= tredici;
                        LED_out <= "0111000";
                END IF;

        WHEN quattro =>
                --
                IF caffe = '1' THEN
                        stateMealy_next <= quattro; --NON PERMESSO
                        LED_out <= "0110000";
                ELSIF cappuccino = '1' THEN
                        stateMealy_next <= quattro; --NON PERMESSO
                        LED_out <= "0110000";
                ELSIF ricarica = '1' THEN
                        stateMealy_next <= quattordici;
                LED_out <= "0111000";
                END IF;

        WHEN cinque =>
        --
                IF caffe = '1' THEN
                        stateMealy_next <= zero;
                        LED_out <= "0110001";
                ELSIF cappuccino = '1' THEN
                        stateMealy_next <= cinque; --NON PERMESSO
                        LED_out <= "0110000";
                ELSIF ricarica = '1' THEN
                        stateMealy_next <= quindici;
                        LED_out <= "0111000";

                END IF;

        WHEN sei =>
                --
                IF caffe = '1' THEN
                        stateMealy_next <= uno;
                        LED_out <= "0110001";
                ELSIF cappuccino = '1' THEN
                        stateMealy_next <= zero;
                        LED_out <= "0000010";
                ELSIF ricarica = '1' THEN
                        stateMealy_next <= sedici;
                        LED_out <= "0111000";
                END IF;

        WHEN sette =>
        --
                IF caffe = '1' THEN
                        stateMealy_next <= due;
                        LED_out <= "0110001";
                ELSIF cappuccino = '1' THEN
                        stateMealy_next <= uno;
                        LED_out <= "0000010";
                ELSIF ricarica = '1' THEN
                        stateMealy_next <= diciassette;
                        LED_out <= "0111000";
                END IF;

        WHEN otto =>
        --
                IF caffe = '1' THEN
                        stateMealy_next <= tre;
                        LED_out <= "0110001";
                ELSIF cappuccino = '1' THEN
                        stateMealy_next <= due;
                        LED_out <= "0000010";
                ELSIF ricarica = '1' THEN
                        stateMealy_next <= diciotto;
                        LED_out <= "0111000";
                END IF;

        WHEN nove =>
                --
                IF caffe = '1' THEN
                        stateMealy_next <= quattro;
                        LED_out <= "0110001";
                ELSIF cappuccino = '1' THEN
                        stateMealy_next <= tre;
                        LED_out <= "0000010";
                ELSIF ricarica = '1' THEN
                        stateMealy_next <= diciannove;
                        LED_out <= "0111000";
                END IF;

        WHEN dieci =>
        --
                IF caffe = '1' THEN
                        stateMealy_next <= cinque;
                        LED_out <= "0110001";
                ELSIF cappuccino = '1' THEN
                        stateMealy_next <= quattro;
                        LED_out <= "0000010";
                ELSIF ricarica = '1' THEN
                        stateMealy_next <= venti;
                        LED_out <= "0111000";
                END IF;

        WHEN undici =>
        --
                IF caffe = '1' THEN
                        stateMealy_next <= sei;
                        LED_out <= "0110001";
                ELSIF cappuccino = '1' THEN
                        stateMealy_next <= cinque;
                        LED_out <= "0000010";
                ELSIF ricarica = '1' THEN
                        stateMealy_next <= undici;--NON PERMESSO
                        LED_out <= "0110000";

                END IF;

        WHEN dodici =>
                --
                IF caffe = '1' THEN
                        stateMealy_next <= sette;
                        LED_out <= "0110001";
                ELSIF cappuccino = '1' THEN
                        stateMealy_next <= sei;
                        LED_out <= "0000010";
                ELSIF ricarica = '1' THEN
                        stateMealy_next <= dodici;--NON PERMESSO
                        LED_out <= "0110000";
                END IF;

        WHEN tredici =>
        --
                IF caffe = '1' THEN
                        stateMealy_next <= otto;
                        LED_out <= "0110001";
                ELSIF cappuccino = '1' THEN
                        stateMealy_next <= sette;
                        LED_out <= "0000010";
                ELSIF ricarica = '1' THEN
                        stateMealy_next <= tredici;--NON PERMESSO
                        LED_out <= "0110000";
                END IF;

        WHEN quattordici =>
        --
                IF caffe = '1' THEN
```

```
                stateMealy_next <= nove;
                LED_out <= "0110001";
        ELSIF cappuccino = '1' THEN
                stateMealy_next <= otto;
                LED_out <= "0000010";
        ELSIF ricarica = '1' THEN
                stateMealy_next <= quattordici;--NON PERMESSO
                LED_out <= "0110000";
        END IF;

WHEN quindici =>
        --
        IF caffe = '1' THEN
                stateMealy_next <= dieci;
                LED_out <= "0110001";
        ELSIF cappuccino = '1' THEN
                stateMealy_next <= nove;
                LED_out <= "0000010";
        ELSIF ricarica = '1' THEN
                stateMealy_next <= quindici;--NON PERMESSO
                LED_out <= "0110000";
        END IF;

WHEN sedici =>
--
        IF caffe = '1' THEN
                stateMealy_next <= undici;
                LED_out <= "0110001";
        ELSIF cappuccino = '1' THEN
                stateMealy_next <= dieci;
                LED_out <= "0000010";
        ELSIF ricarica = '1' THEN
                stateMealy_next <= sedici;--NON PERMESSO
                LED_out <= "0110000";
        END IF;

WHEN diciassette =>
        --
        IF caffe = '1' THEN
                stateMealy_next <= dodici;
                LED_out <= "0110001";
        ELSIF cappuccino = '1' THEN
                stateMealy_next <= undici;
                LED_out <= "0000010";
        ELSIF ricarica = '1' THEN
                stateMealy_next <= diciassette;--NON PERMESSO
                LED_out <= "0110000";
        END IF;

WHEN diciotto =>
        --
        IF caffe = '1' THEN
                stateMealy_next <= tredici;
                LED_out <= "0110001";
        ELSIF cappuccino = '1' THEN
                stateMealy_next <= dodici;
                LED_out <= "0000010";
        ELSIF ricarica = '1' THEN
                stateMealy_next <= diciotto;--NON PERMESSO
                LED_out <= "0110000";
        END IF;

WHEN diciannove =>
        --
        IF caffe = '1' THEN
                stateMealy_next <= quattordici;
                LED_out <= "0110001";
        ELSIF cappuccino = '1' THEN
                stateMealy_next <= tredici;
                LED_out <= "0000010";
        ELSIF ricarica = '1' THEN
                stateMealy_next <= diciannove;--NON PERMESSO
                LED_out <= "0110000";
        END IF;

WHEN venti =>
--
        IF caffe = '1' THEN
                stateMealy_next <= quindici;
                LED_out <= "0110001";
        ELSIF cappuccino = '1' THEN
                stateMealy_next <= quattordici;
                LED_out <= "0000010";
        ELSIF ricarica = '1' THEN
```

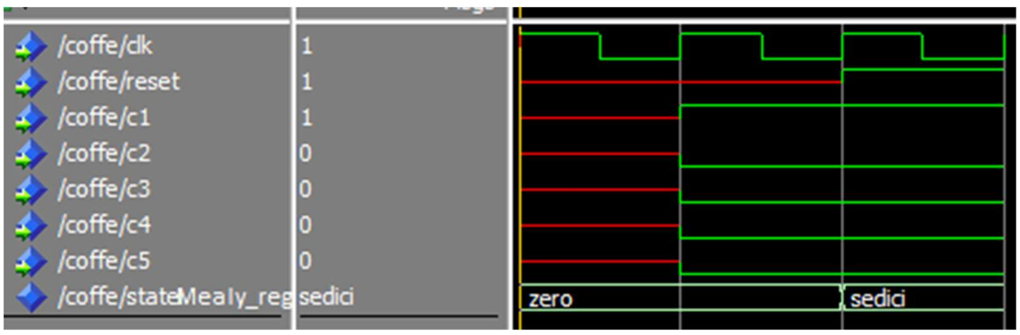
```
                                        stateMealy_next <= venti;--NON PERMESSO
                                        LED_out <= "0110000";
                                END IF;

                        END CASE;
                END IF;

        END PROCESS;
END arch;
```

3.Simulation of processes (modelsim)

Status reset:



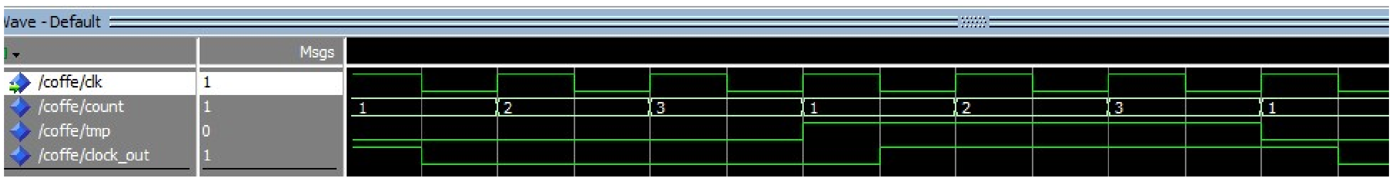
clk: clock of FPGA.

reset: input that sets the initial state based on the sequence of bits inserted in the input (c1,c2,c3,c4,c5).

c1,c2,c3,c4,c5: inputs representing the amount usable (in binary code) from the moment the reset button is pressed.

stateMealy_reg: current state of the machine which for the moment depends solely on the configuration of the inputs.

Clock frequency divider:



clk: clock of FPGA.

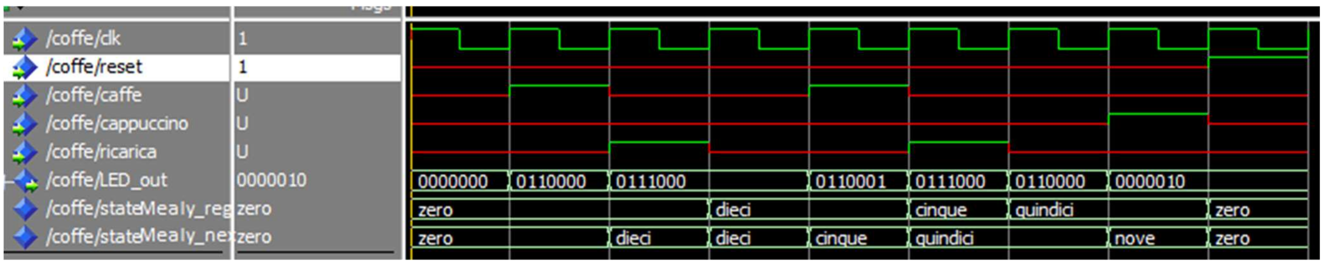
count: counts the rising edges of the input clock.

tmp: when the variable count reaches the desired count, tmp goes to alternate the trend of the generated virtual clock (alternating from 0 to 1 as the physical clock of the board).

clock_out: clock used for processes which allows the machine to operate correctly while respecting the storage times of the modified variables.

max: an entire limit has been added that sets the maximum count to reach before changing the clock_out value.

Mealy's machine simulation:



It should be noted that a precedence is foreseen on the inputs that will drive the status. If several inputs are selected at the same time (during the same clock cycle) the "coffee" input will have absolute precedence, followed by the "cappuccino" input, and finally the "recharge" input.

Knowing the current state of the machine, not all the entrances will be available as there may be a lack of residual credit or an excess of the credit out of the machine reach.

The top-up is scheduled for a maximum credit of € 2.00.

The possible states are 21 and to each one corresponds the residual credit of the machine coffee:

zero: 0.00€ one: 0.10€ two: 0.20€ … … … twenty: 2.00€

clk: clock of FPGA.

reset: returns to the "zero" state (in the absence of other directives).

coffee: input signal that selects the coffee drink (cost € 0.50).

cappuccino: input signal that selects the cappuccino drink (cost € 0.60).

recharge: input signal that tops up (€ 1.00).

LED_out: output signal that communicates if the requested operation has been carried out correctly or not.

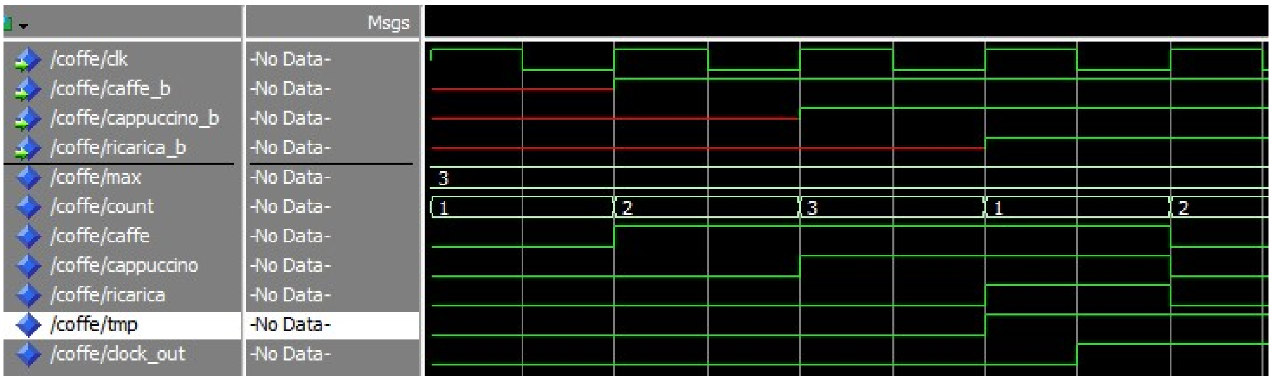
foreseen notices:

1. C: coffee selected correctly.
2. A: cappuccino selected correctly.
3. F: recharge selected correctly.
4. E: the selection made is not compatible with the current state of the machine.

stateMealy_reg: Actual state of machine.

stateMealy_next: Subsequent state of the machine which depends on the combination of current status and selected inputs.

Variables associated with the input signals:



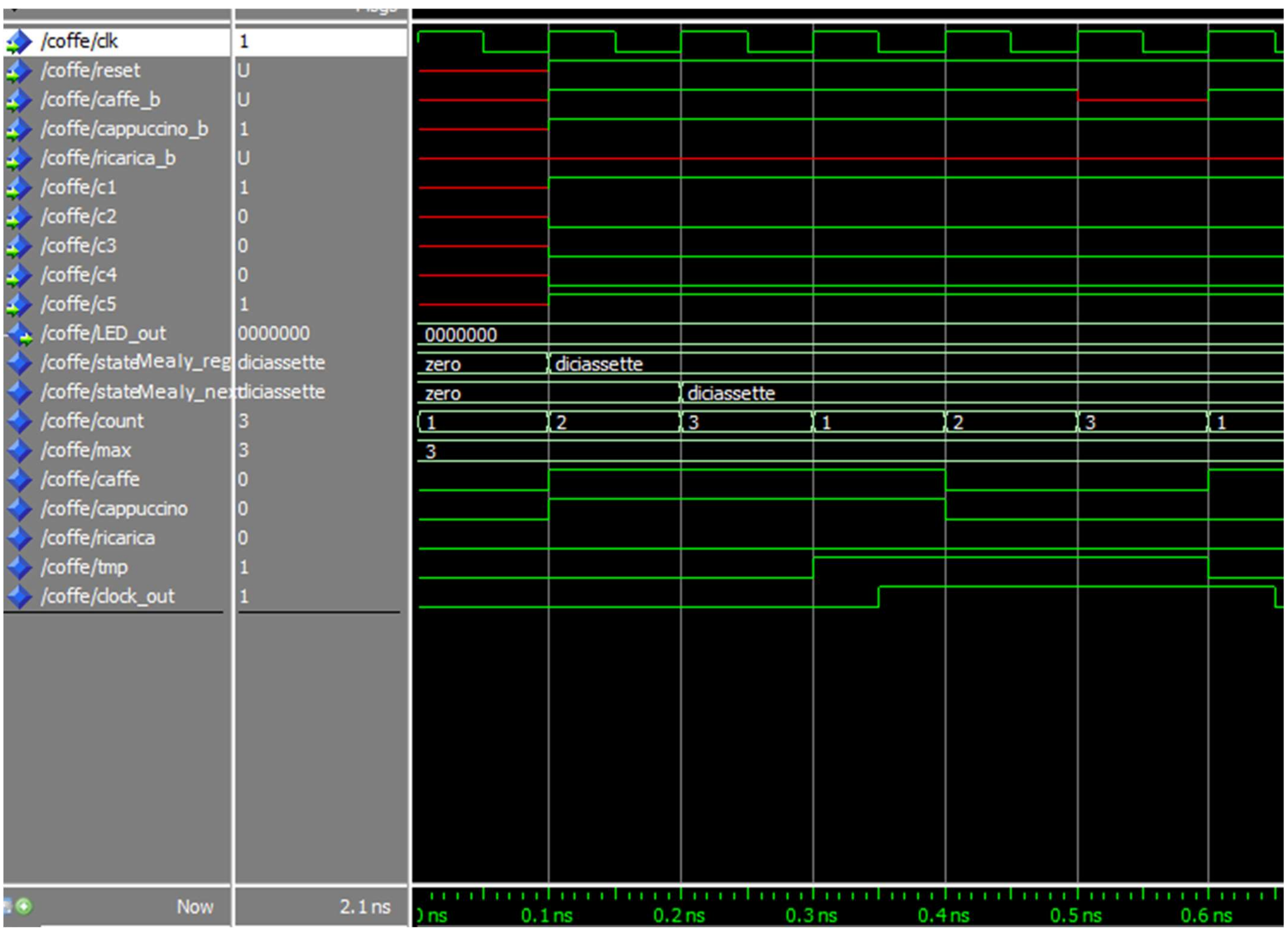
The processes defined previously are used to manage integer variables that will be able to drive the changes of the machine's state. The value of the aforementioned variables will be one if the rising edge of the input associated is detected and zero if an instant of the clock_out has elapsed (or if no rising edge of the inputs has been detected). The permanence for more than one instant of clock_out of any selected input (or more than one) is avoided since the evaluation of the state change is carried out precisely every rising edge of the clock. The permanence instead of the reset input is permissible because if it is positive no changes in the machine's state will be allowed. Previously coffee, cappuccino and refill were the input signals, while at the moment they are integers associated with the corresponding input signals (caffe_b, cappuccino_b, carica_b).

coffee: integer variable that detects the possible rising edge of the caffe_b input signal.

cappuccino: integer variable that detects any rising edge of the cappuccino_b input signal.

recharge: integer variable that detects any rising edge of the recharge input signal.

Complete simulation:



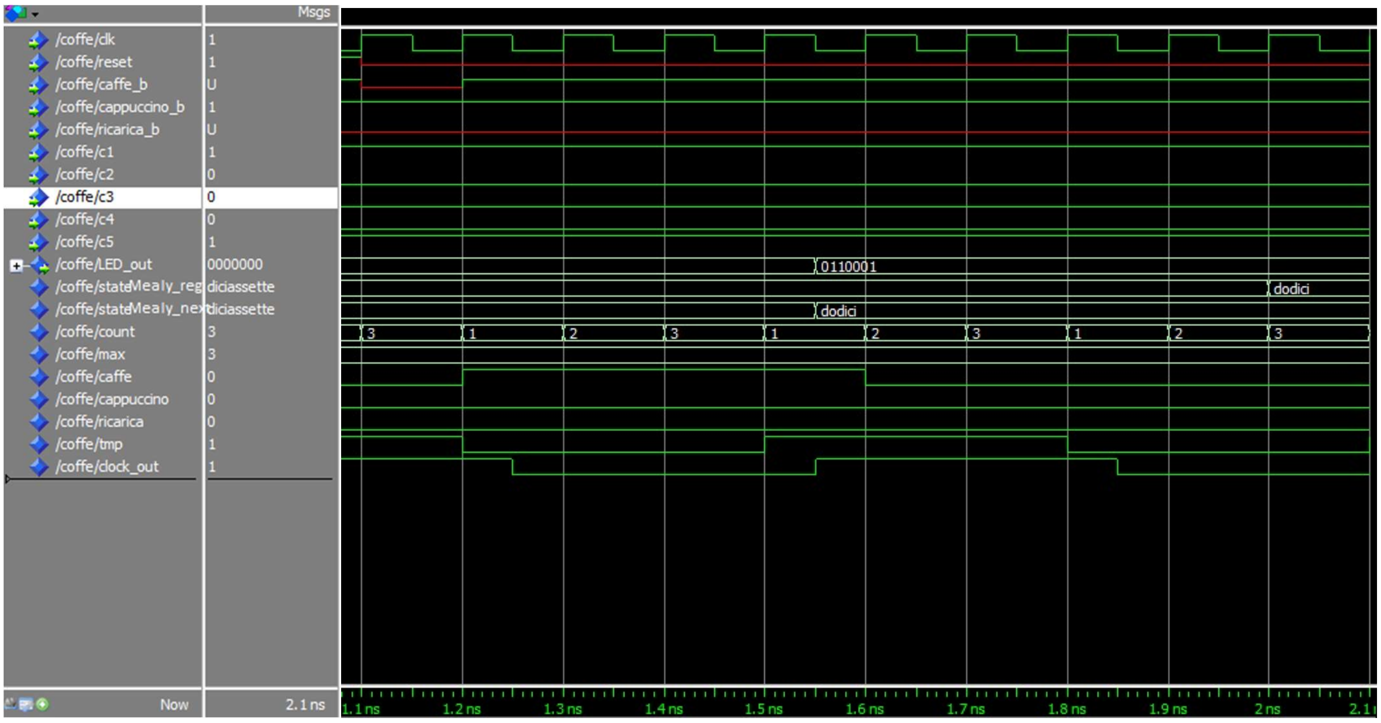
/coffe/clk	1
/coffe/reset	U
/coffe/caffe_b	U
/coffe/cappuccino_b	1
/coffe/ricarica_b	U
/coffe/c1	1
/coffe/c2	0
/coffe/c3	0
/coffe/c4	0
/coffe/c5	1
/coffe/LED_out	0000000
/coffe/stateMealy_reg	diciassette
/coffe/stateMealy_ne	diciassette
/coffe/count	3
/coffe/max	3
/coffe/caffe	0
/coffe/cappuccino	0
/coffe/ricarica	0
/coffe/tmp	1
/coffe/clock_out	1
Now	2.1 ns

The frequency divider is setting the **clock_out** every 3 rounds of the **clk** clock (**max** = 3). The **interval** variable was not included in this simulation, it will be useful only for the laboratory test, as it will allow us to correctly manage the three work moments defined at the beginning of the project.

Test description:

0,1 ns: At the same time a status **reset** is requested and the selection of two drinks. Having priority over the entire reset input, the current **stateMealy_reg** status will immediately be managed, to which the following **stateMealy_next** status will be adjusted to a subsequent instant (calculated starting from the rising edge **clock_out**). As mentioned above, two inputs are also selected at this moment: **caffe_b** and **cappuccino_b.**

0,6 ns: On the next rising edge of one of the entrances **caffe_b**, the value '1' is assigned to the associated variable **coffee**. It will not result in a change of status as the **reset** input is still active when evaluating the next state of the machine. The operation of the machine in the event of a reset equal to 1 is specified in the design hypotheses.



1,2 ns: The descent front of the **reset** input and of **caffe_b** is evident.

1,3 ns: A subsequent rising edge of the **caffe_b** entrance triggers the updating of the coffee variable again.

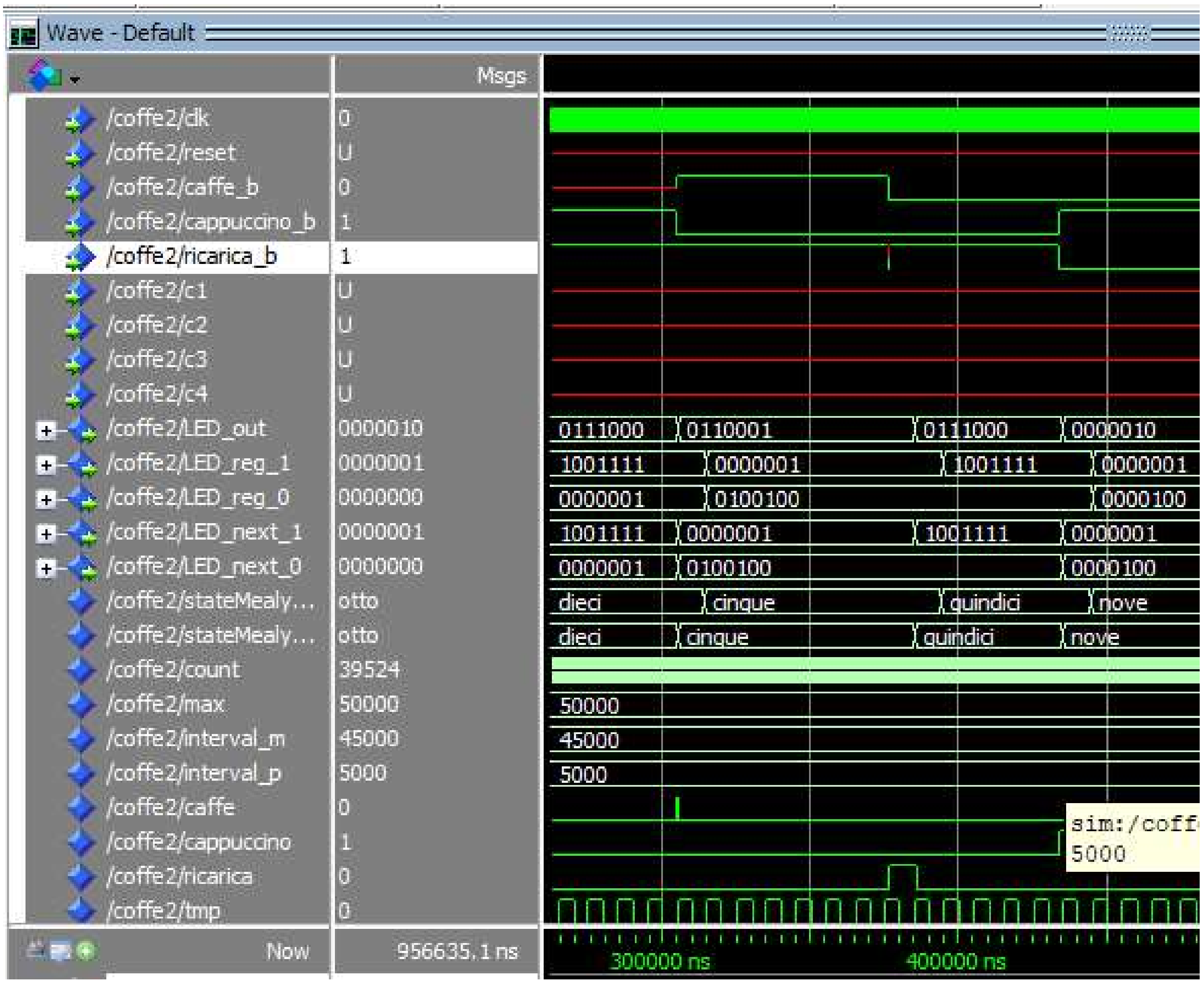
1,55 ns: There is yet another change of the next state which this time is due to a variation of one of the inputs (**caffe_b** input). Of course the **LED_out** will indicate the correct status change or an eventual illegal entry for the current machine's status.

1,6 ns: The reset of all input variables of the last clock instant is visible.

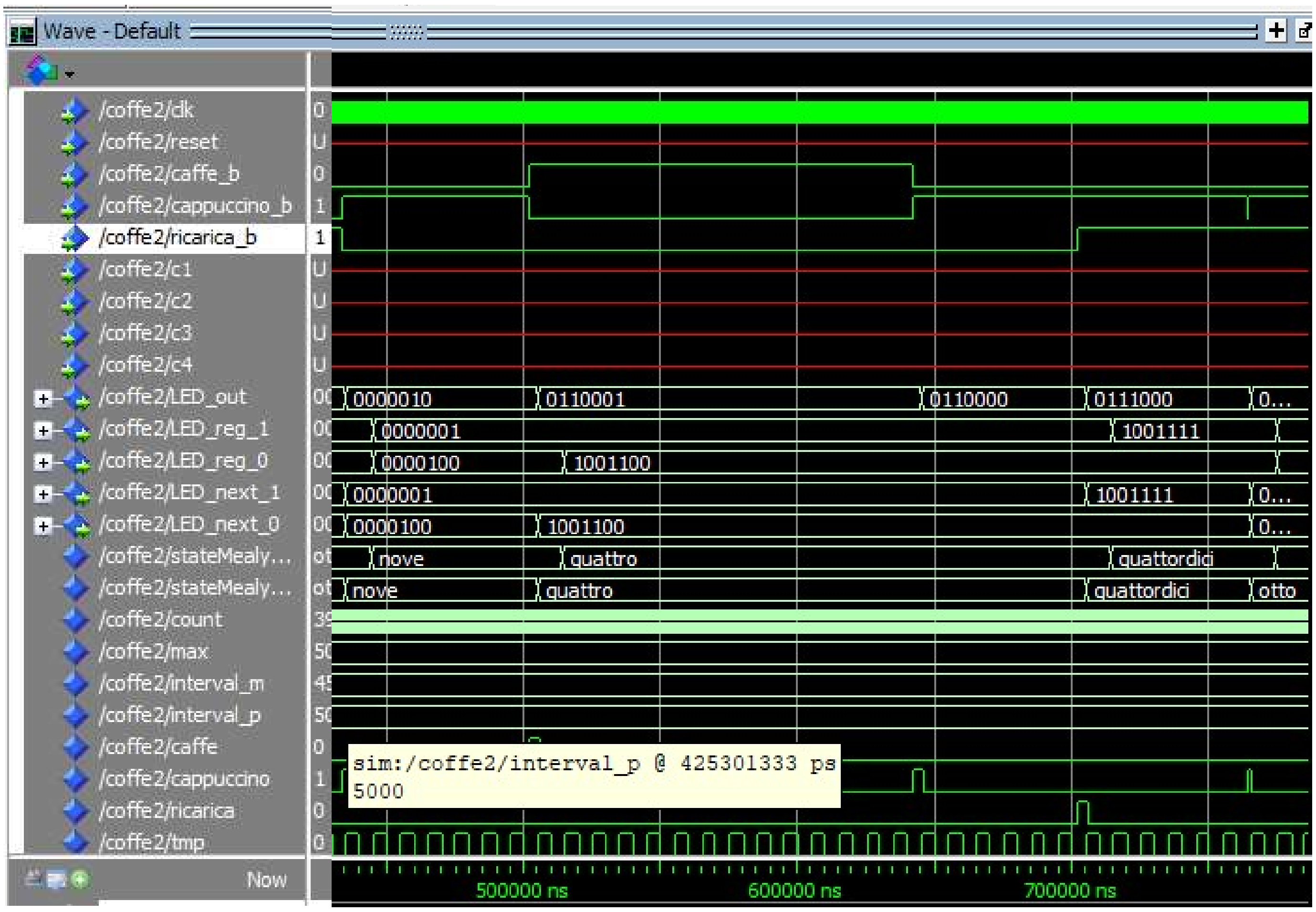
2,0 ns: The **stateMealy_reg** current state follows the **stateMealy_next** status (calculated last clock shot) to evaluate subsequent status changes. It takes place just before the rising edge of **clock_out** to allow the correct evaluation of the next state in which the machine will be found.

interval: The **interval** variable will allow you to manage 3 different clock times. One just before the rising edge of **clock_out**, one at the moment when the clock signal rises from the value 0 to the value 1 and finally a moment after the rising edge. It is important because in each of these instants a different operation will be execute that cannot temporally overlap with any of the other instructions performed before or after it.

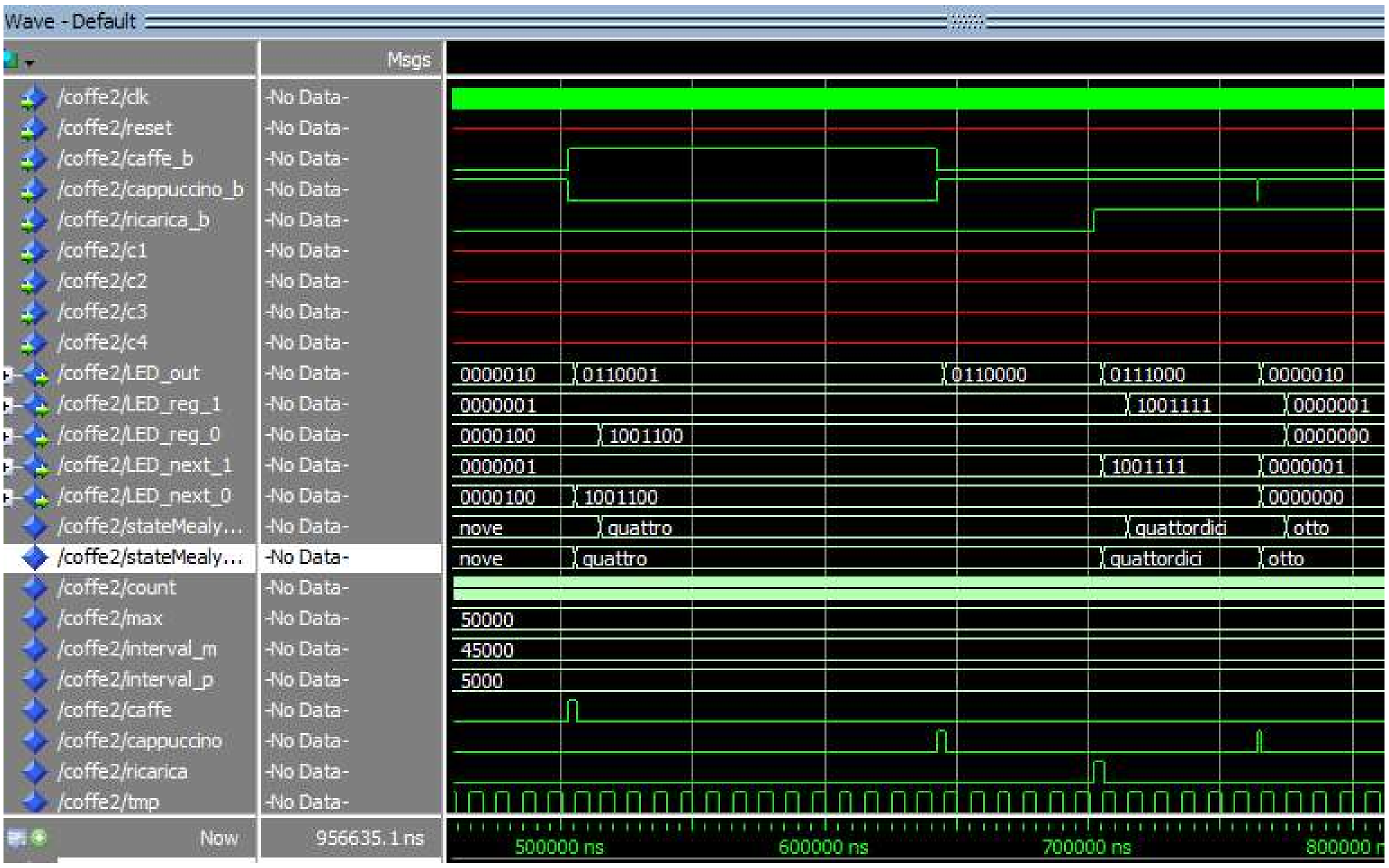
Simulazione con run -all:



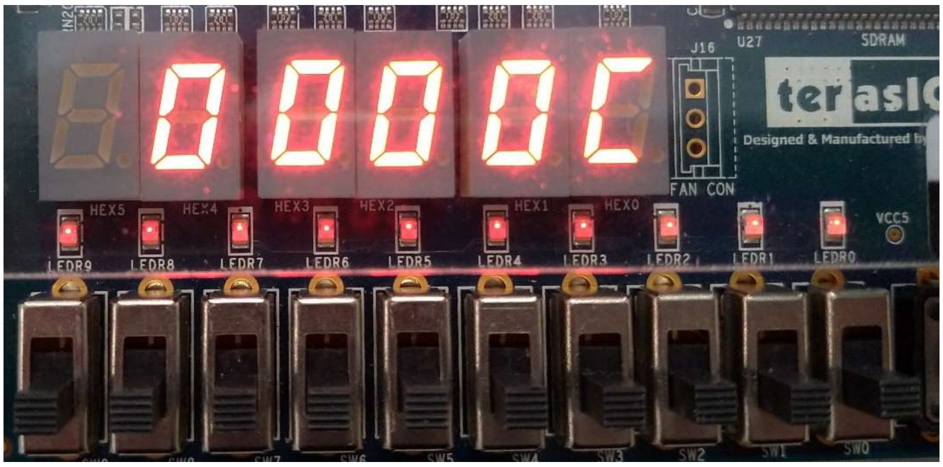
	Msgs
/coffe2/clk	0
/coffe2/reset	U
/coffe2/caffe_b	0
/coffe2/cappuccino_b	1
/coffe2/ricarica_b	1
/coffe2/c1	U
/coffe2/c2	U
/coffe2/c3	U
/coffe2/c4	U
/coffe2/LED_out	0000010
/coffe2/LED_reg_1	0000001
/coffe2/LED_reg_0	0000000
/coffe2/LED_next_1	0000001
/coffe2/LED_next_0	0000000
/coffe2/stateMealy...	otto
/coffe2/stateMealy...	otto
/coffe2/count	39524
/coffe2/max	50000
/coffe2/interval_m	45000
/coffe2/interval_p	5000
/coffe2/caffe	0
/coffe2/cappuccino	1
/coffe2/ricarica	0
/coffe2/tmp	0
Now	956635.1 ns



```
sim:/coffe2/interval_p @ 425301333 ps
5000
```


Wave - Default
Msgs
/coffe2/clk -No Data-
/coffe2/reset -No Data-
/coffe2/caffe_b -No Data-
/coffe2/cappuccino_b -No Data-
/coffe2/ricarica_b -No Data-
/coffe2/c1 -No Data-
/coffe2/c2 -No Data-
/coffe2/c3 -No Data-
/coffe2/c4 -No Data-
/coffe2/LED_out -No Data- 0000010 0110001 0110000 0111000 0000010
/coffe2/LED_reg_1 -No Data- 0000001 1001111 0000001
/coffe2/LED_reg_0 -No Data- 0000100 1001100 0000000
/coffe2/LED_next_1 -No Data- 0000001 1001111 0000001
/coffe2/LED_next_0 -No Data- 0000100 1001100 0000000
/coffe2/stateMealy... -No Data- nove quattro quattordici otto
/coffe2/stateMealy... -No Data- nove quattro quattordici otto
/coffe2/count -No Data-
/coffe2/max -No Data- 50000
/coffe2/interval_m -No Data- 45000
/coffe2/interval_p -No Data- 5000
/coffe2/caffe -No Data-
/coffe2/cappuccino -No Data-
/coffe2/ricarica -No Data-
/coffe2/tmp -No Data-
Now 956635.1 ns
500000 ns
600000 ns
700000 ns
800000

4.Laboratory test



0 1 2 3 4 5 6 7 8 9

Display

HEX[5]: Unused.

HEX[4]/[3]: Decimal value containing the current state stateMealy_reg.

HEX[2]/[1]: Decimal value containing the current state stateMealy_next.

HEX[0]: Last input acquired through one of the three input switches.

Switch configuration

0) Unused.

1) Unused.

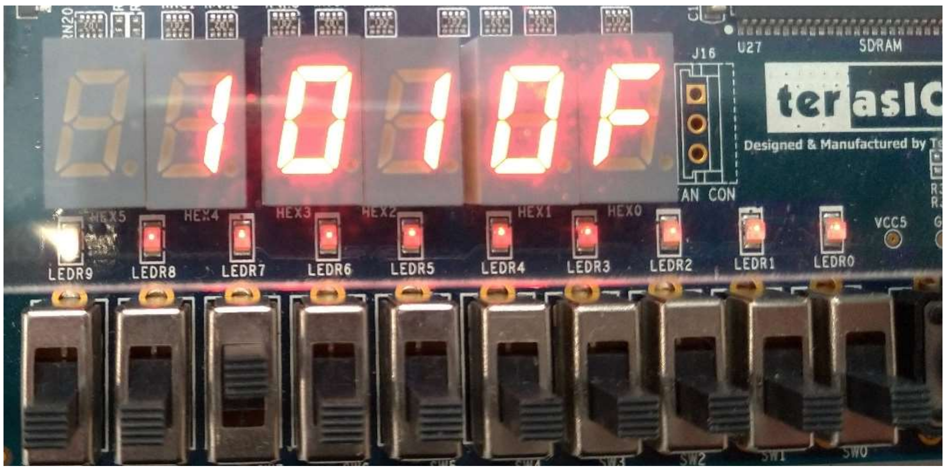
2) It’s used for a **recharge operation**.

3) It helps to take a **cappuccino**.

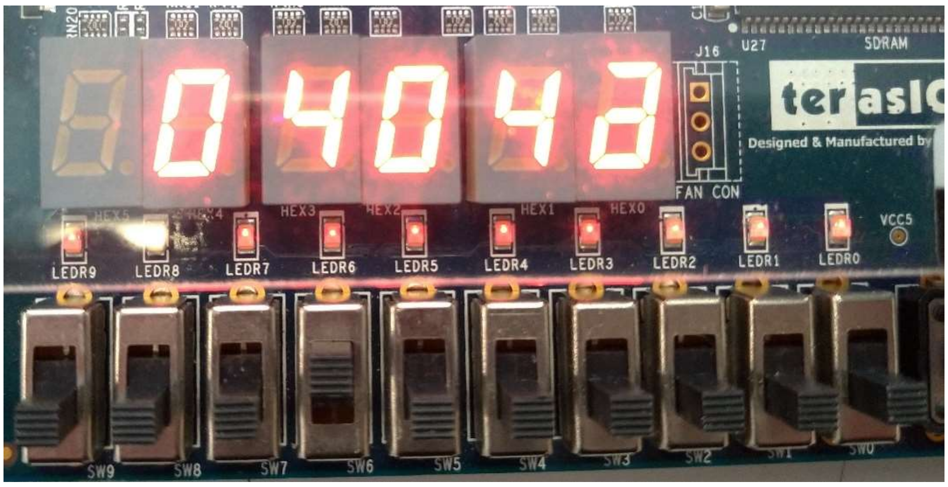
4) It is useful for selecting a **coffee**.

5/6/7/8) Reset state selection switch used with a simultaneous selection of the reset button. The status request is done by converting the selected number from binary to decimal. The order of the bits is inverted from the least significant to the most significant. Example: state 10= 0101.

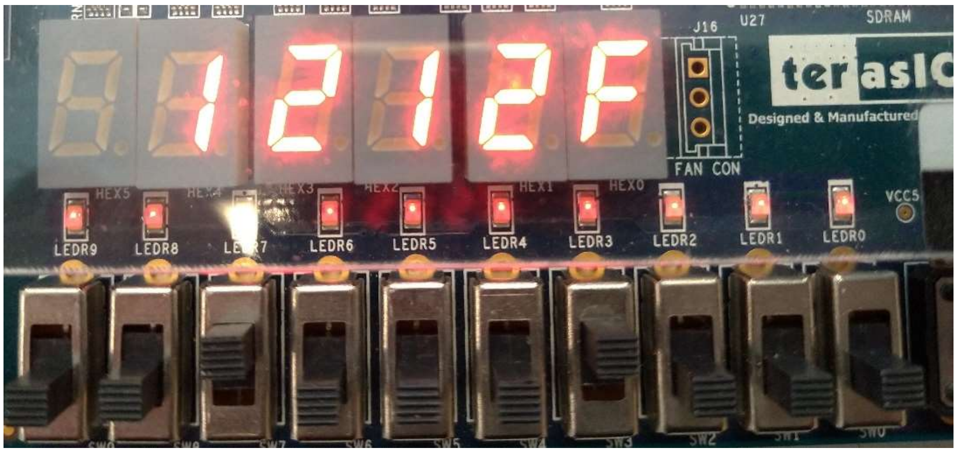
9) Reset switch.



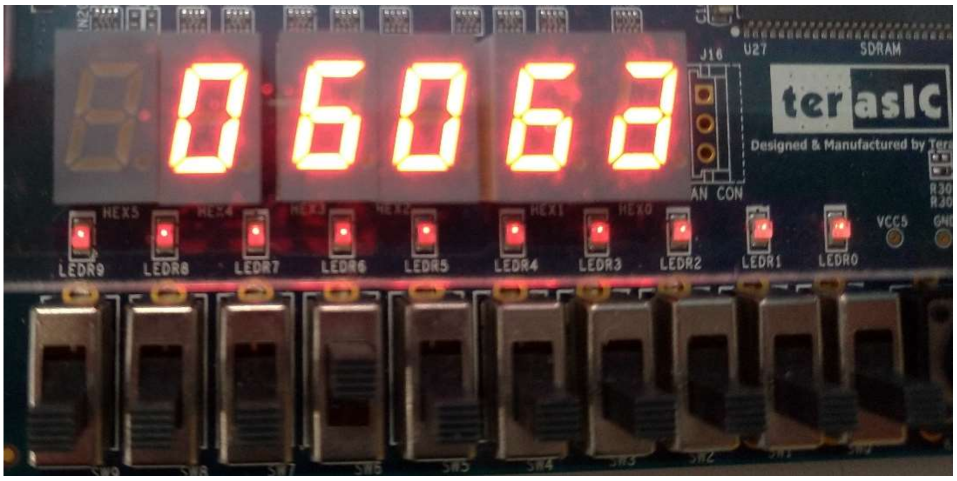
0 1 2 3 4 5 6 7 8 9



0 1 2 3 4 5 6 7 8 9

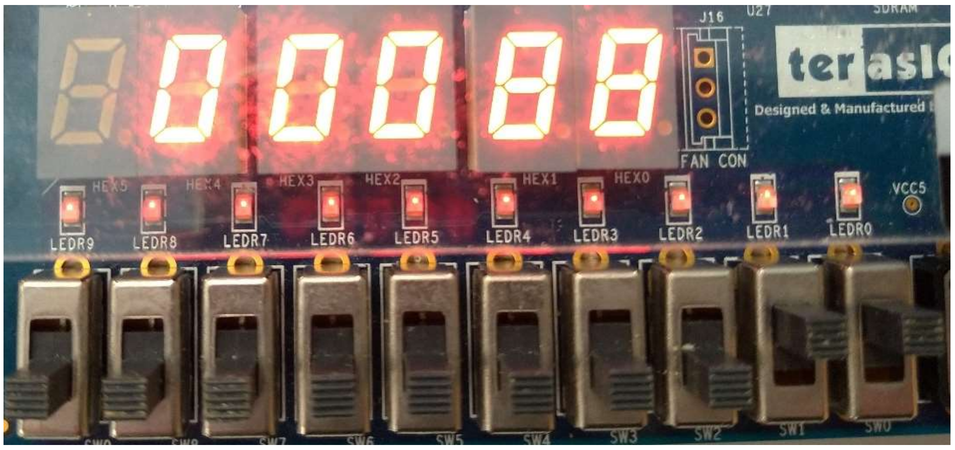


0 1 2 3 4 5 6 7 8 9

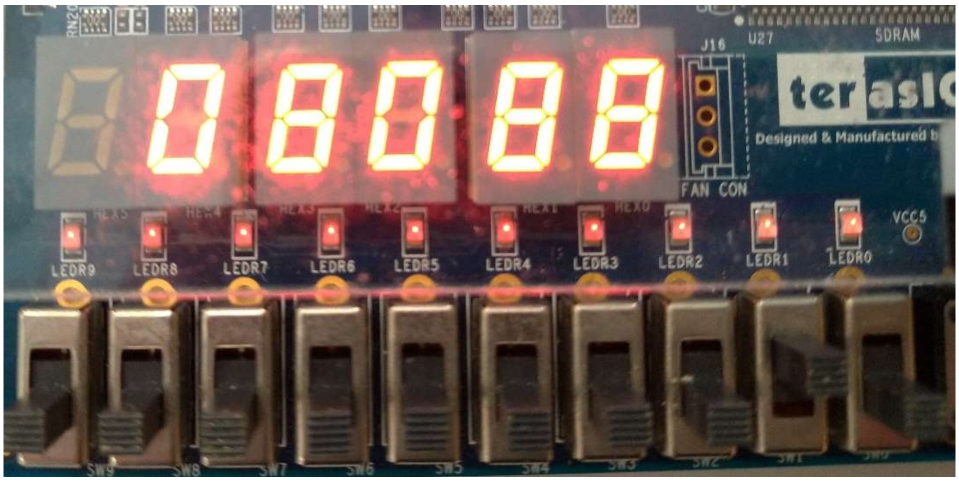


0 1 2 3 4 5 6 7 8 9

RESET



0 1 2 3 4 5 6 7 8 9



0 1 2 3 4 5 6 7 8 9

Additive implementation

Display

HEX[5]: Signals the selection of the reset process when it takes value 1.

HEX[4]/[3]: Decimal value containing the current state stateMealy_reg.

HEX[2]/[1]: Decimal value containing the current state stateMealy_next.

HEX[0]: Last input acquired through one of the three input switches.

Switch configuration

0) Unused.

1) Unused.

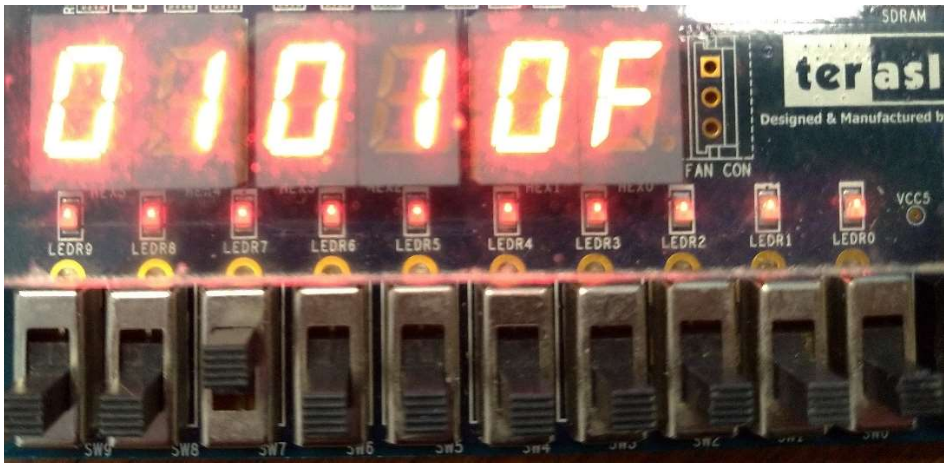
2) It's used for a **recharge operation**.

3) It helps to take a **cappuccino**.

4) It is useful for selecting a **coffee**.

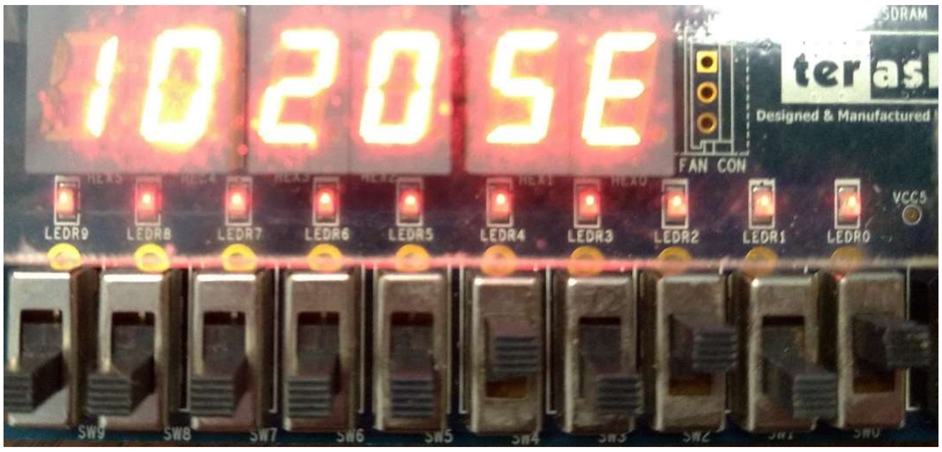
5/6/7/8) Reset state selection switch used with a simultaneous selection of the reset button. The status request is done by converting the selected number from binary to decimal. The order of the bits is inverted from the least significant to the most significant. Example: state 10= 0101.

9) Reset switch.



0 1 0 1 0 F
SDRAM
terasic
Designed & Manufactured by
FAN CON
LEDR9 LEDR8 LEDR7 LEDR6 LEDR5 LEDR4 LEDR3 LEDR2 LEDR1 LEDR0
VCC5
SW9 SW8 SW7 SW6 SW5 SW4 SW3 SW2 SW1 SW0

0 1 2 3 4 5 6 7 8 9



1 0 2 0 5 E
SDRAM
terasic
Designed & Manufactured
FAN CON
LEDR9 LEDR8 LEDR7 LEDR6 LEDR5 LEDR4 LEDR3 LEDR2 LEDR1 LEDR0
VCC5
SW9 SW8 SW7 SW6 SW5 SW4 SW3 SW2 SW1 SW0

0 1 2 3 4 5 6 7 8 9