# PROJETO: SNAKE

PSI3441 - PROJETO DE CIRCUITOS LÓGICOS INTEGRADOS.

FILIPE GABRIEL SANTOS VALENTIM
9837761
PROFESSOR: DR. MARIUS STRUM.

SÃO PAULO, 2019

Este relatório está dividido em 3 etapas que comprovam cada parte do funcionamento do snake. (Reorganizei as duas primeiras partes do relatório sugerido para facilitar a divisão de assuntos).

**1 - Reapresentação da LFSR Galois e sua integração ao SNAKE.**

- **A) O polinômio requerido**
- **B) A Máquina requerida**
- **C) Descrição VHDL**
- **D) Análise**
- **E) Integração ao SNAKE**
  - **E.1)O trecho correspondente à entity e à arquitetura do módulo lfsr.**
  - **E.2) Instanciação do componente lfsr dentro do módulo num_gen.**
  - **E.3) Imagem do ModelSim, onde fique evidente o valor da semente e os primeiros 5 valores.**
  - **E.4) Comparação entre os números aleatórios com aqueles obtidos por software on-line.**
  - **E.5) Apresentação dos 5 endereços de memória correspondentes aos 5 números acima.**
  - **E.6) Trecho do arquivo run_sim_1.do correspondente aos sinais acima.**

**2) Resultados das simulações do VHDL final (snake completo) e do seu testbench no ModelSim (para a condição de "game over")**

**3) Resultados das simulações do VHDL final (snake completo) e do seu testbench no ModelSim (situação de jogo longo)**

# 1 - Reapresentação da LFSR Galois e sua integração ao SNAKE.

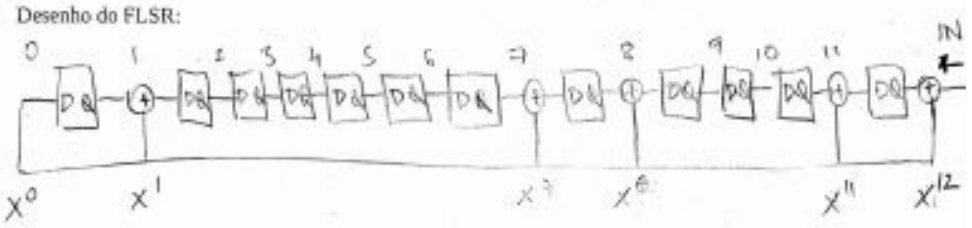
OBJETIVOS DA ETAPA: Descrever em VHDL uma máquina geradora de números pseudoaleatórios de Galois com grau 12. Adicionalmente, implementei um sistema de semente e polinômio gerador variáveis, permitindo que a aleatoriedade seja aumentada.

**A) O polinômio requerido:**

O polinômio é baseado no número USP do aluno, sendo:
9837761 mod 2048 = 1217
Após isso, deve-se transformar em binário, ou seja:
1217 = 0b10011000001
Por fim, adiciona-se os dois números obrigatórios de começo e fim:
Com 13 dígitos:  1100110000011
Portanto, o polinômio final é:
X12 + X11 + X8 + X7 + X1 + 1

**B) A Máquina requerida:**

Seguindo a descrição da apostila, temos a seguinte máquina:

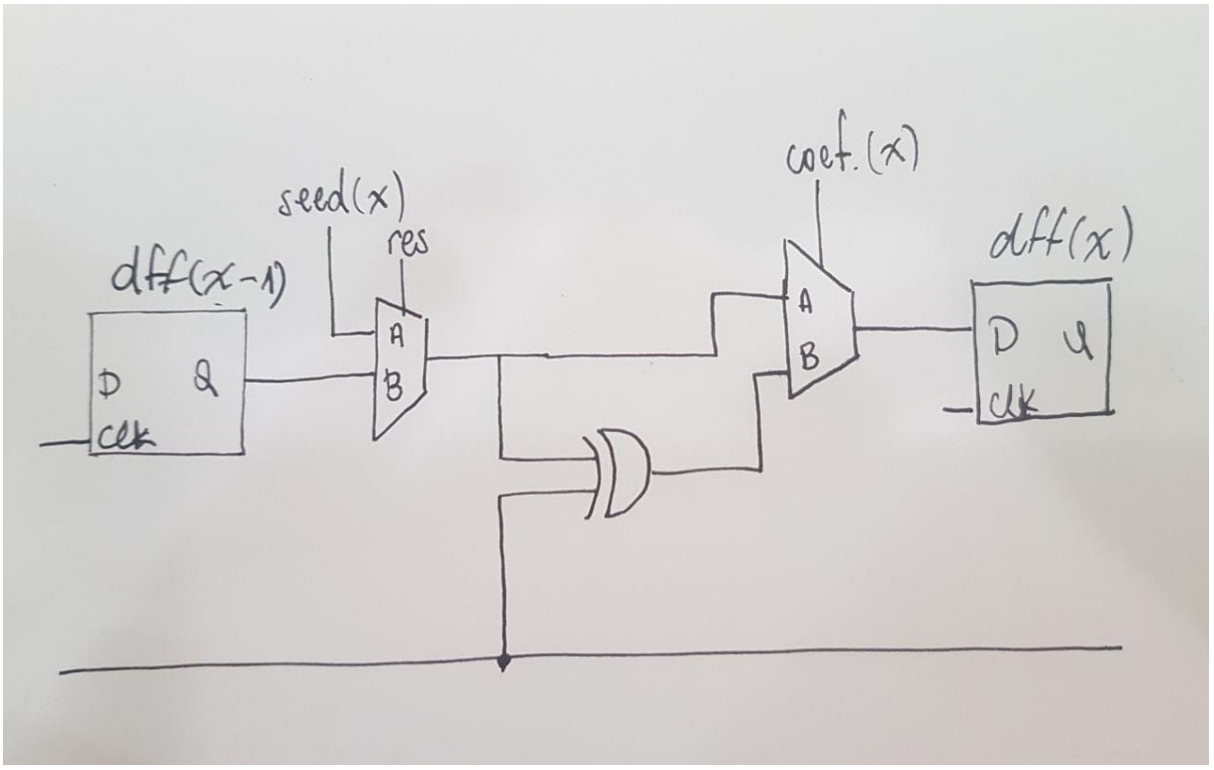



Resultados das simulações (para semente 1111111...)

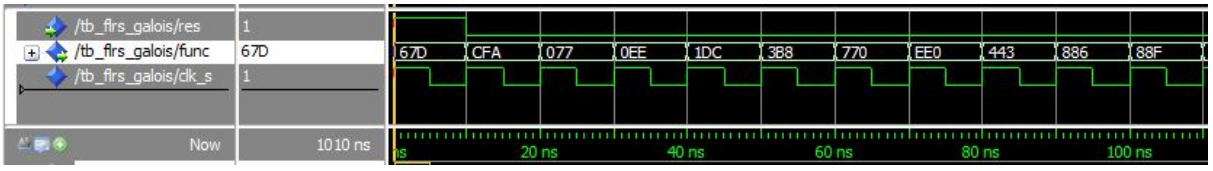
| | | | | |
|---|---|---|---|---|
| 1 | 67D | 0110 | 0111 | 1101 |
| 2 | CFA | 1100 | 1111 | 1010 |
| 3 | 77 | 0000 | 0111 | 0111 |
| 4 | EE | 0000 | 1110 | 1110 |
| 5 | 1DC | 0001 | 1101 | 1100 |
| 6 | 388 | 0011 | 1000 | 1000 |
| 7 | 770 | 0111 | 0111 | 0000 |
| 8 | EE0 | 1110 | 1110 | 0000 |
| 9 | 443 | 0100 | 0100 | 0011 |
| 10 | 886 | 1000 | 1000 | 0110 |

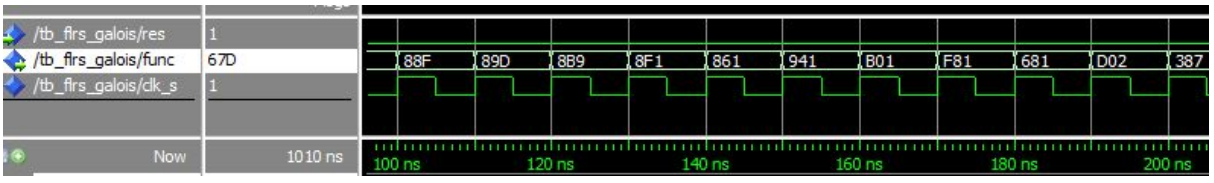
**C) Descrição VHDL**

Usando generate, repete-se o bloco abaixo 12 vezes. A 13ª, última, trata-se apenas de uma realimentação (feedback).




O código está na página abaixo. Esta descrição foi testada com um testbanch e stimuli generator, que levam a seguinte carta de tempos.

1. 

2. 


**D) Análise:**

A sequência seguida foi, sim, igual a do site recomendado. Sendo os primeiros 10 termos: 67D, CFA, 077, 0EE, 1DC, 388, 770, EE0, 443, 886. Quando a saída do circuito é a mesma do padrão que se usa para testá-lo, é chamado de *aliasing*.

OBS: na descrição VHDL, apesar de *seed* e *coeficients* serem descritos como *generic,* também é possível descrevê-los como sinais para que possam ser mudados em tempo de execução.

**E) Integração ao SNAKE**

**E.1)O trecho correspondente à entity e à arquitetura do módulo lfsr.**

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity FLSR is
	generic
	(
		SIZE			: NATURAL	:= 12;
		COEFICIENTS		: UNSIGNED	:= b"100000110011"; -- x^0 -----> x^12
		SEED			: UNSIGNED	:= b"111111111111"
	);

	--****************************************
	--**                ATENCAO             *
	--** APESAR DE X0 ESTAR NA FUNCAO, SEU  *
	--** COEFICIENTE PRECISA SER DESCONSID  *
	--** ERADO. ELE JA FOI IMPLEMENTADO FI  *
	--** SICAMENTE NO CIRCUITO, NO CODIGO.  *
	--****************************************

	port
	(
		clk		: in STD_LOGIC;
		res		: in STD_LOGIC;
		function_out	: out STD_LOGIC_VECTOR (SIZE-1 downto 0)
	);
end FLSR;

architecture arch of FLSR is

	component my_d_ff is
	port
	(
		clk	:	in std_logic;
		d	:	in std_logic;
		q	:	out std_logic
	);
	end component;


	component my_mux is
	port
	(
		SEL : in  STD_LOGIC;
		A   : in  STD_LOGIC;
		B   : in  STD_LOGIC;
```

```
            X   : out STD_LOGIC
        );
        end component;


        signal ff_d             : std_logic_vector (SIZE-1 downto 0);
        signal ff_q              : std_logic_vector (SIZE-1 downto 0);
        signal ff_xor           : std_logic_vector (SIZE-1 downto 0);
        signal ff_seed          : std_logic_vector (SIZE-1 downto 0);

begin

        G1: for i in 0 to SIZE-1 generate

                --*********************************************************************
                --***                                         COMO FUNCIONA:          ***
                --***    DIVIDE-SE O PROBLEMA EM 2: SHIFT-REGISTER E FEEDBACK  ***
                --***    SHIFT-REGISTER: ORGANIZA OS FF EM CASCATA
***
                --***    FEEDBACK: ESCOLHE SE O FF(i) DEVE RECEBER O XOR         ***
                --*********************************************************************

                --*************************************************************
               --***                                         SHIFT-REGISTER      ***
                --*************************************************************

                        ff: my_d_ff port map
                        (
                                clk     => clk,
                                d       => ff_d(i),
                                q       => ff_q(i)
                        );

                --*************************************************************
                --***             LINEAR FEEDBACK                             ***
                --*************************************************************
                ff_xor(i) <= ff_seed(i) xor ff_seed(SIZE-1);

                --SE FOR PARTE DO POLINOMIO, SERA REALIMENTADO
                --SE NAO FOR PARTE DO POLINOMIO, NAO SERA REALIMENTADO

                G2: if (i < SIZE-1) generate
                        mux_xor_or_q: my_mux port map
                        (
                                SEL             => COEFICIENTS(i),
                                A               => ff_xor(i),      --selecionado se 1
                                B               => ff_seed(i),   --selecionado se 0
                                X               => ff_d(i+1)
                        );
                end generate G2;
```

```
                ff_d(0)             <= ff_seed(SIZE-1); --FEEDBACK
                function_out(i)  <= ff_seed(i);

                --PARA IMPLEMENTAR UMA SEMENTE DIFERENTE DE 0xFFF:

                mux_seed_or_q: my_mux port map
                (
                        SEL             => res,
                        A               => SEED(SIZE-1 - i),            --selecionado se 1
                        B               => ff_q(i),                     --selecionado se 0
                        X               => ff_seed(i)
                );
        end generate G1;
end arch;
```

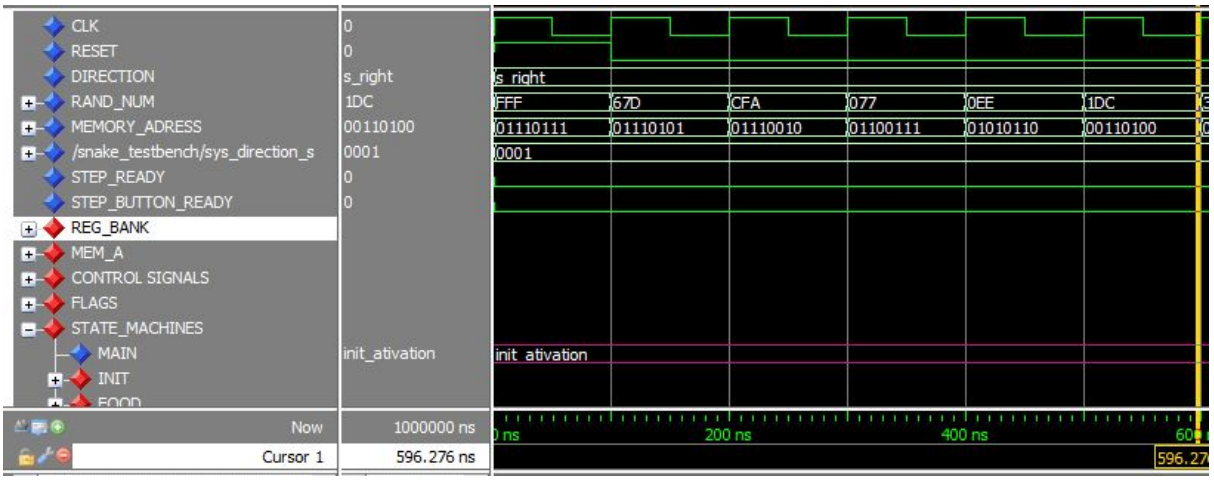
**E.2) Instanciação do componente lfsr dentro do módulo num_gen.**

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;


entity num_gen is
        generic
        (
                WIDTH: NATURAL    := 8; --tamanho dos vetores
                SIZE    : NATURAL    := 12
        );

        port
        (
                clk                     : in STD_LOGIC;
                res                     : in STD_LOGIC;
                pos_neg                 : in STD_LOGIC;
                one_num_gen             : in STD_LOGIC;
                number                  : out STD_LOGIC_VECTOR (WIDTH-1 downto 0)
        );

end num_gen;


architecture arch of num_gen is

        component FLSR is
                generic
                (
                        SIZE            : NATURAL    := 12;
```

```
                        COEFICIENTS : UNSIGNED   := b"100000110011";
                        SEED        : UNSIGNED   := b"111111111111"
                );

                port
                (
                        clk         : in STD_LOGIC;
                        res         : in STD_LOGIC;
                        function_out    : out STD_LOGIC_VECTOR (SIZE-1 downto 0)
                );
        end component;

signal pos_neg_s        : STD_LOGIC_VECTOR (WIDTH-1 downto 0);
signal one_gen_s        : STD_LOGIC_VECTOR (WIDTH-1 downto 0);
signal rand_num         : STD_LOGIC_VECTOR (WIDTH-1 downto 0);
signal pseudo_num       : STD_LOGIC_VECTOR (SIZE-1 downto 0);


begin

        rand: FLSR
                generic map
                (
                        SIZE            => SIZE,
                        COEFICIENTS     => b"100000110011",
                        SEED            => b"111111111111"

                )
                port map
                (
                        clk             => clk,
                        res             => res,
                        function_out    => pseudo_num
                );

        rand_num(7) <= '0';
        rand_num(6) <= pseudo_num(5);
        rand_num(5) <= pseudo_num(4);
        rand_num(4) <= pseudo_num(3);
        rand_num(3) <= '0';
        rand_num(2) <= pseudo_num(2);
        rand_num(1) <= pseudo_num(1);
        rand_num(0) <= pseudo_num(0);


        pos_neg_s       <=      std_logic_vector (to_unsigned(1, pos_neg_s'length))
                                                        when (pos_neg = '0') else
                                std_logic_vector (to_signed(-1, pos_neg_s'length))
                                                        when (pos_neg = '1') else
                                (others => 'X' );
```

```
        one_gen_s       <=      pos_neg_s    when (one_num_gen = '0') else
                                rand_num     when (one_num_gen = '1') else
                                             (others => 'X' );

        number          <=      one_gen_s ;

end arch;
```

**E.3) Imagem do ModelSim, onde fique evidente o valor da semente e os primeiros 5 valores aleatórios, junto aos correspondentes números para o endereço de memória.**




Na imagem acima o sinal RAND_NUM é a variável aleatória gerada pela LFSR, e o MEMORY_ADRESS é o sinal que é levado para a memória. Veja que enquanto o reset está ligado, o sinal RAND_NUM está com a seed FFF ou 0b1111 1111 1111.

**E.4) Comparação entre os números aleatórios com aqueles obtidos por software on-line**

A sequência seguida foi, sim, igual a do site recomendado. Sendo os primeiros 5 termos: 67D, CFA, 077, 0EE, 1DC.

**E.5) Apresentação dos 5 endereços de memória correspondentes aos 5 números acima.**

Os 5 primeiros endereços de memória são:
01110111
01110101
01110010
01100111
01010110

**E.6) Trecho do arquivo run_sim_1.do correspondente aos sinais acima:**

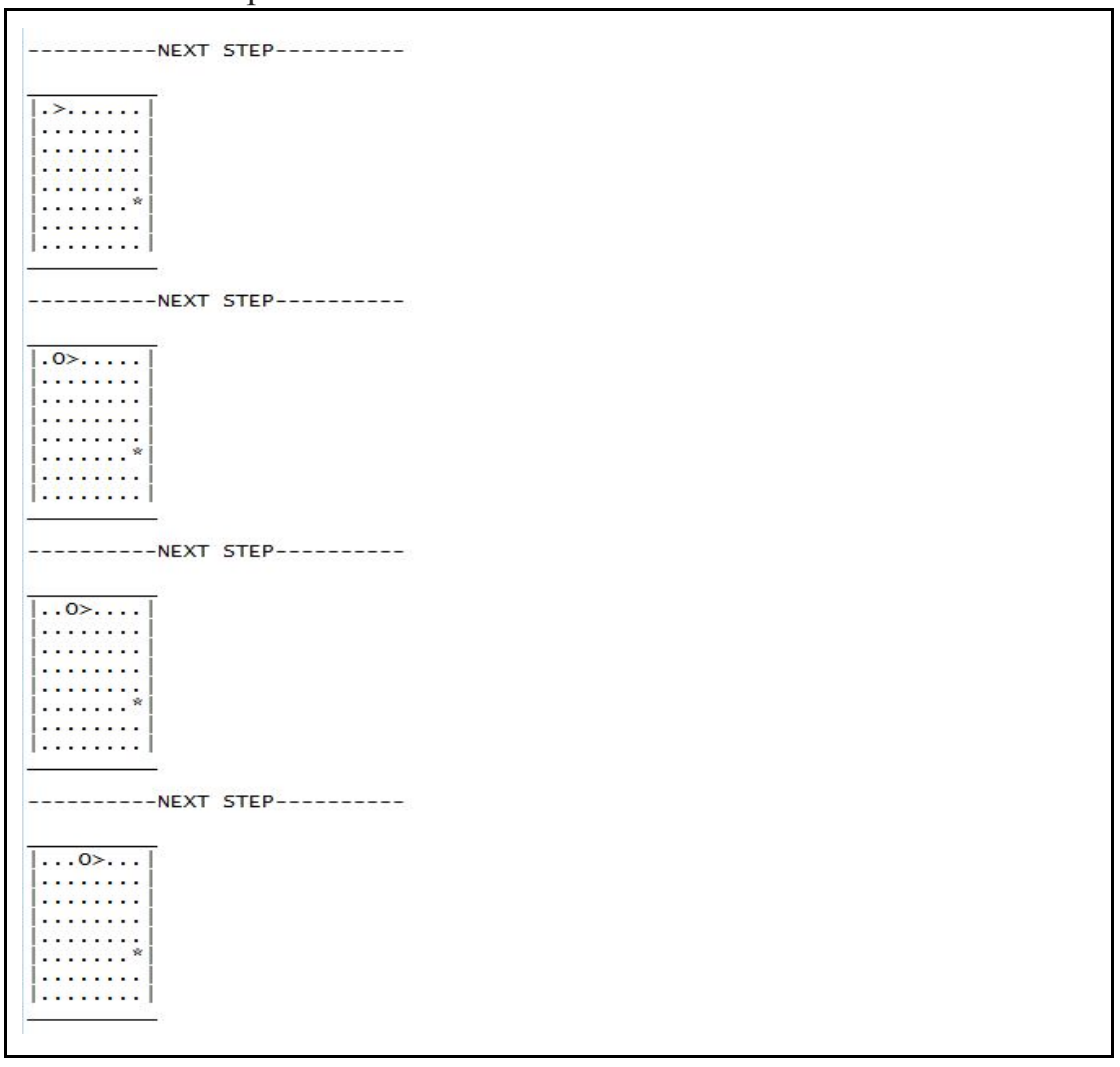
```
add wave -noupdate -label RAND_NUM /snake_testbench/dut/dp_dummy/n_g/rand/function_out
add wave -noupdate -label MEMORY_ADRESS /snake_testbench/dut/dp_dummy/n_g/rand_num
```

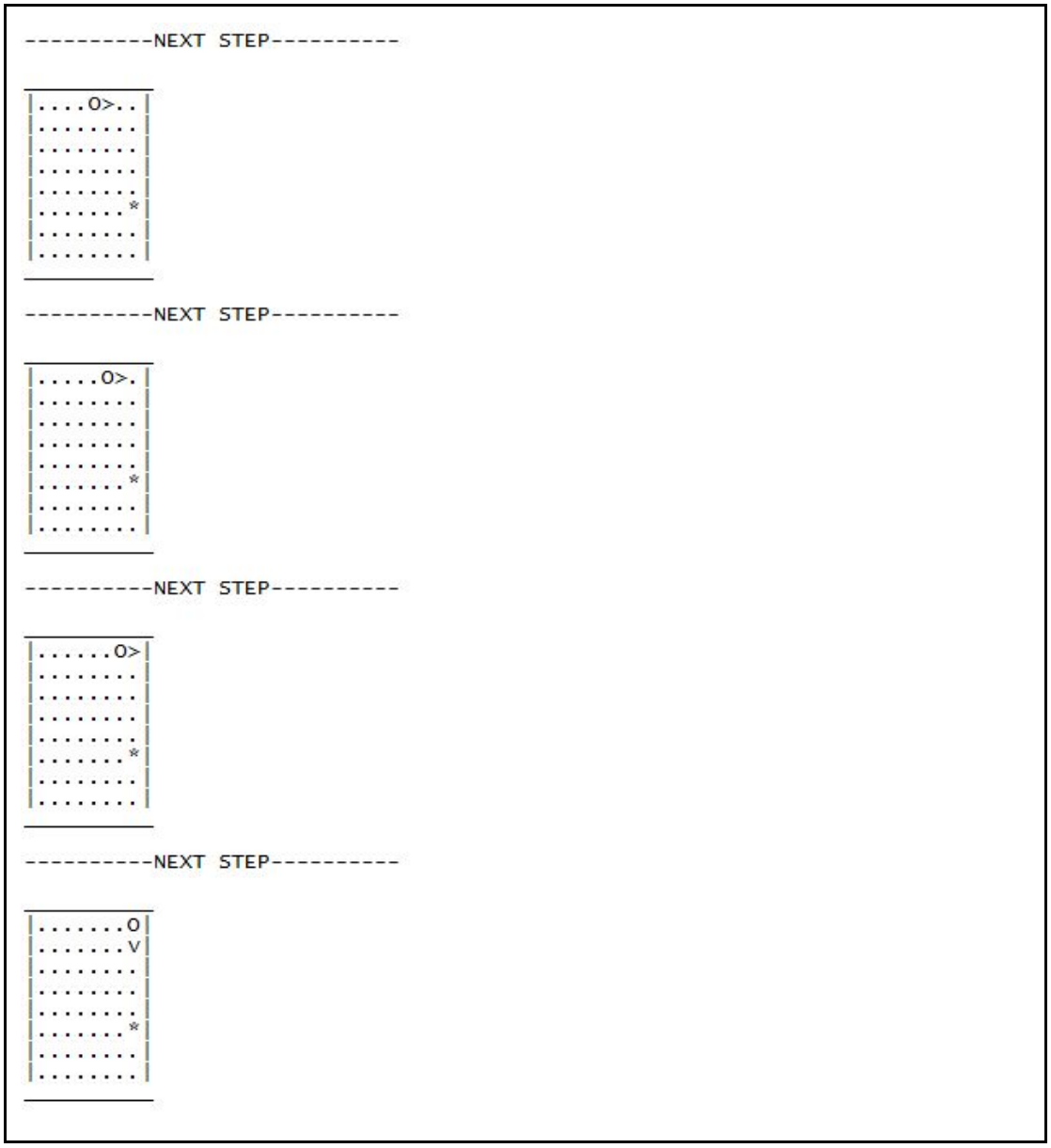
**2) Resultados das simulações do VHDL final (snake completo) e do seu testbench no ModelSim (para a condição de "game over")**

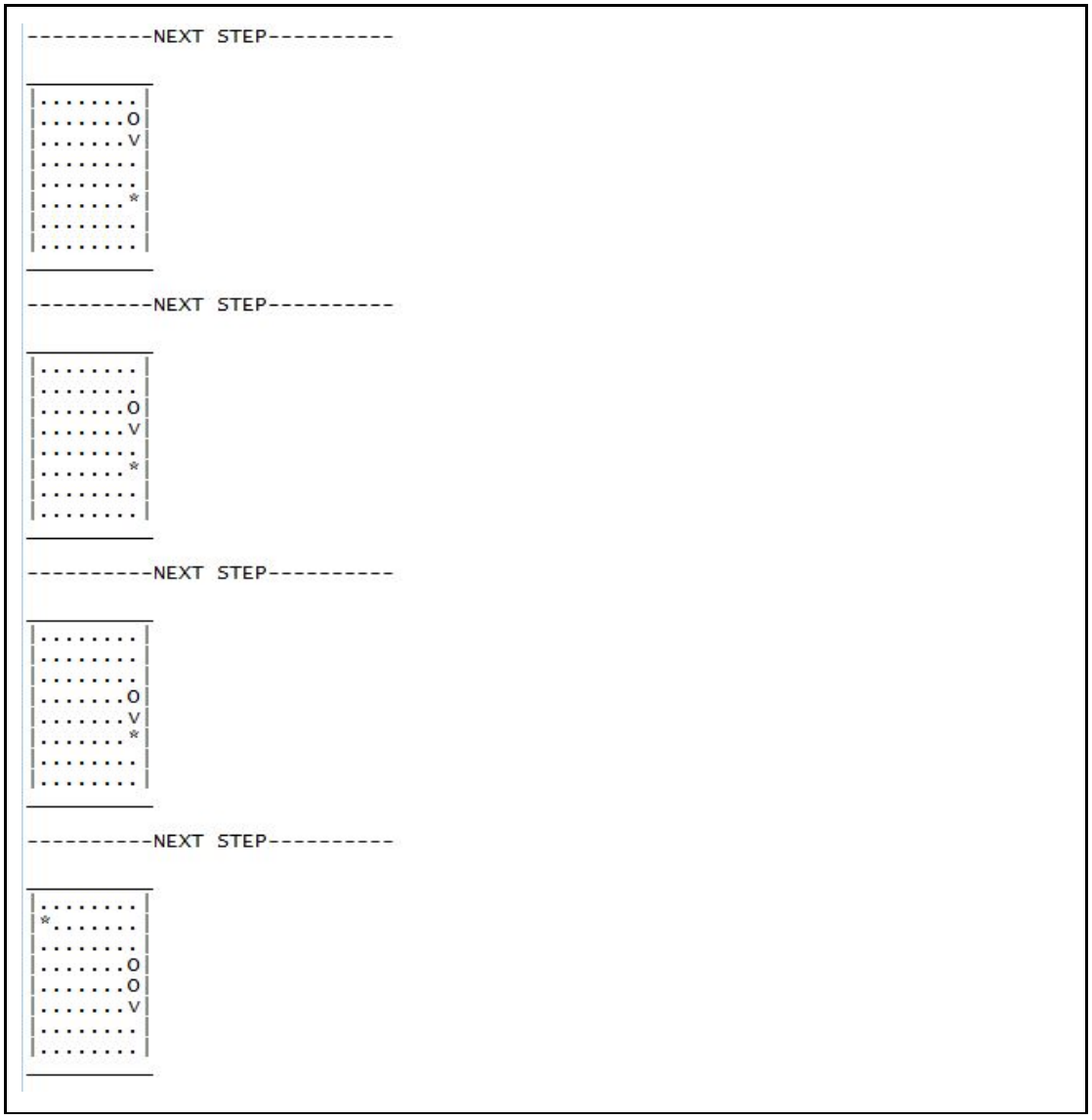
**NOTA: PERDI 2H30MIN TENTANDO DESCOBRIR UM ERRO CAUSADO PELAS CONSTANTES MUDANÇAS DE CONVENÇÕES ENTRE AS VARIADAS VERSÕES DO PROJETO (providas no moodle). SUGIRO ESTABELECER UMA SÓ, OU AVISAR OS ALUNOS DA POSSIBILIDADE DE MUDANÇA ENTRE AS VERSÕES. FALO DA MUDANÇA DAS DIREÇÕES DE NEW_BUTTON_HANDLER COM O STIMULI DADO.**

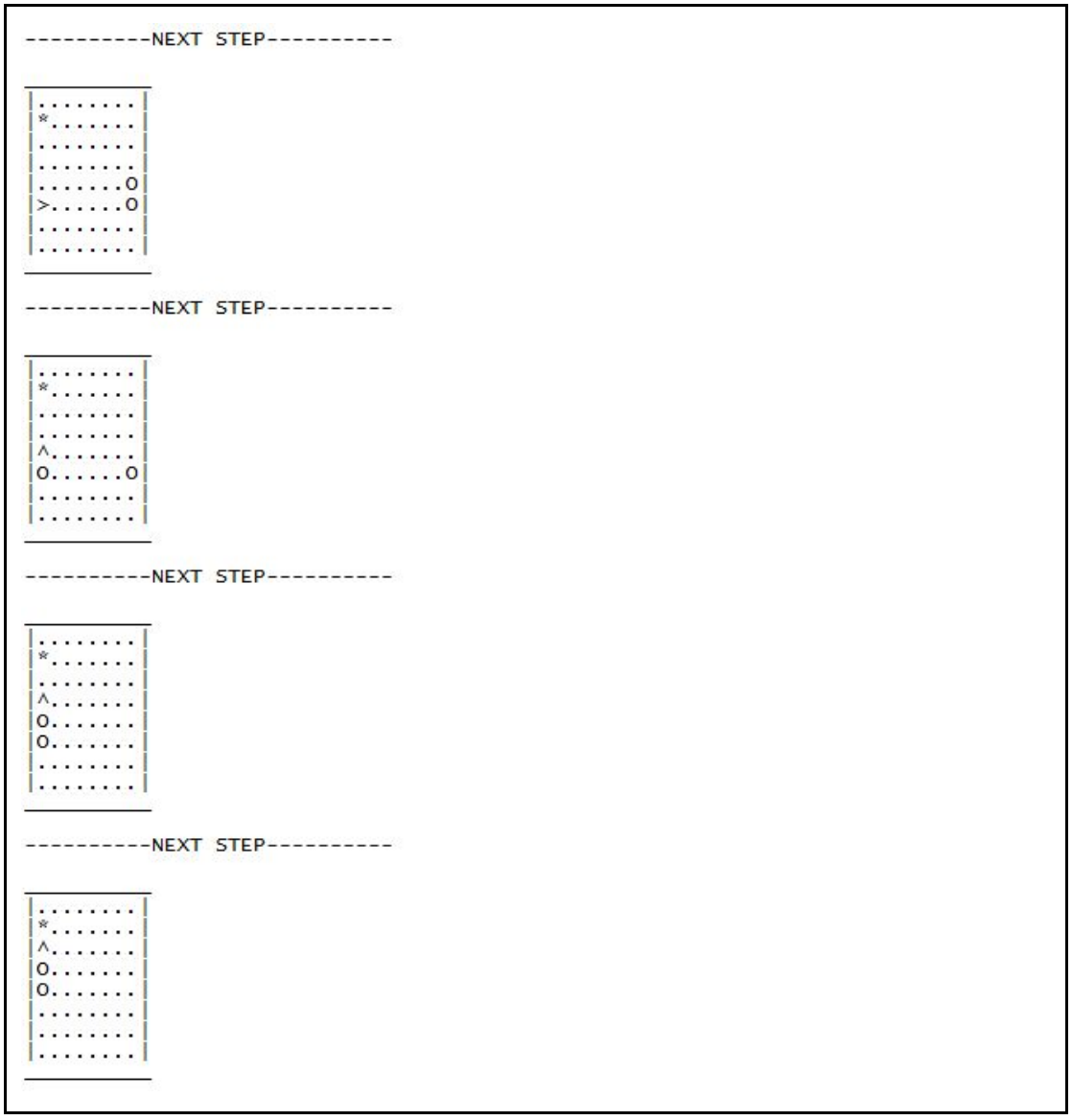
### A) Anexo do tabuleiro para game over

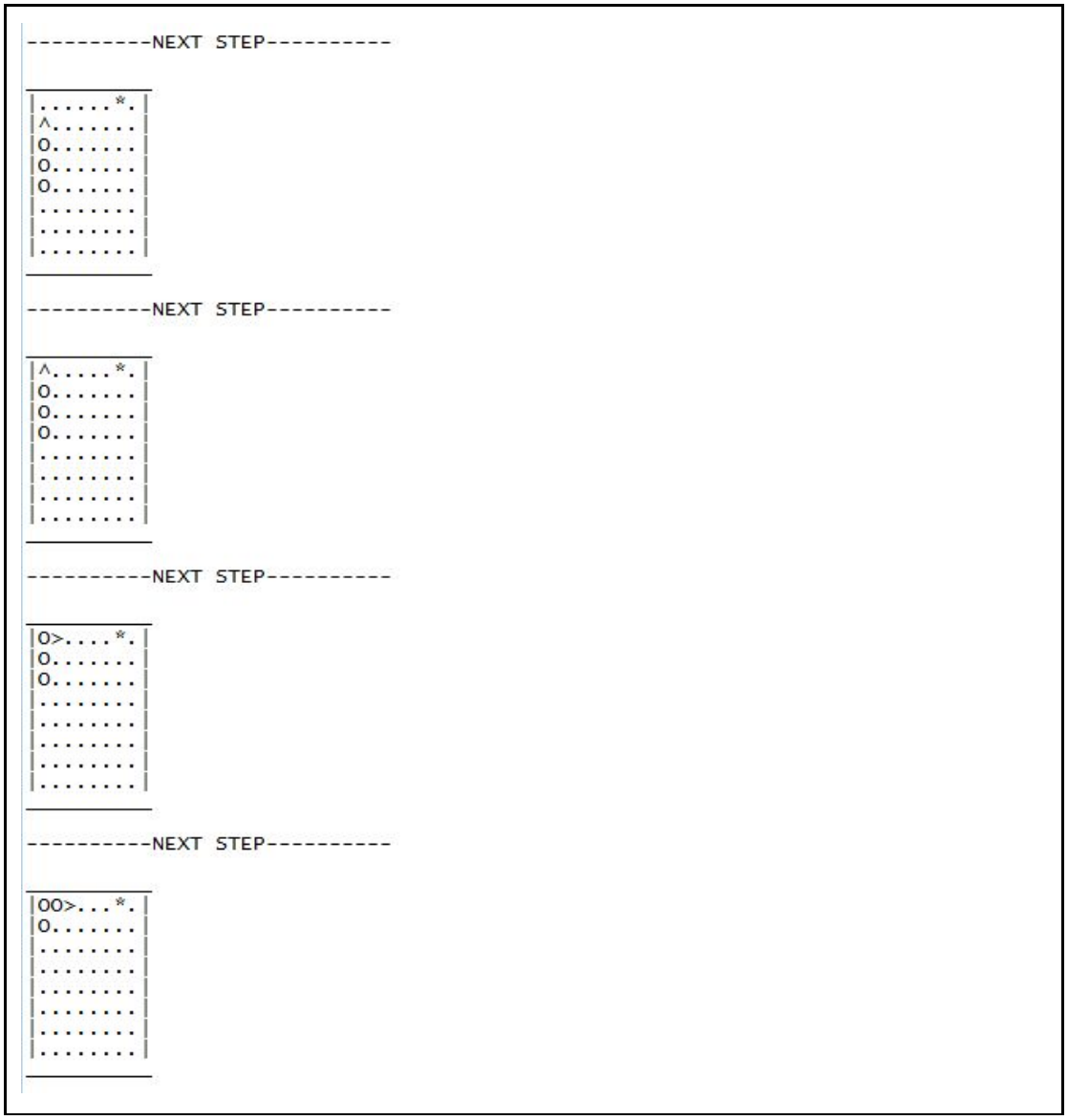
Abaixo estão as imagens do snake andando pelo tabuleiro, comendo, atravessando as paredes e morrendo.

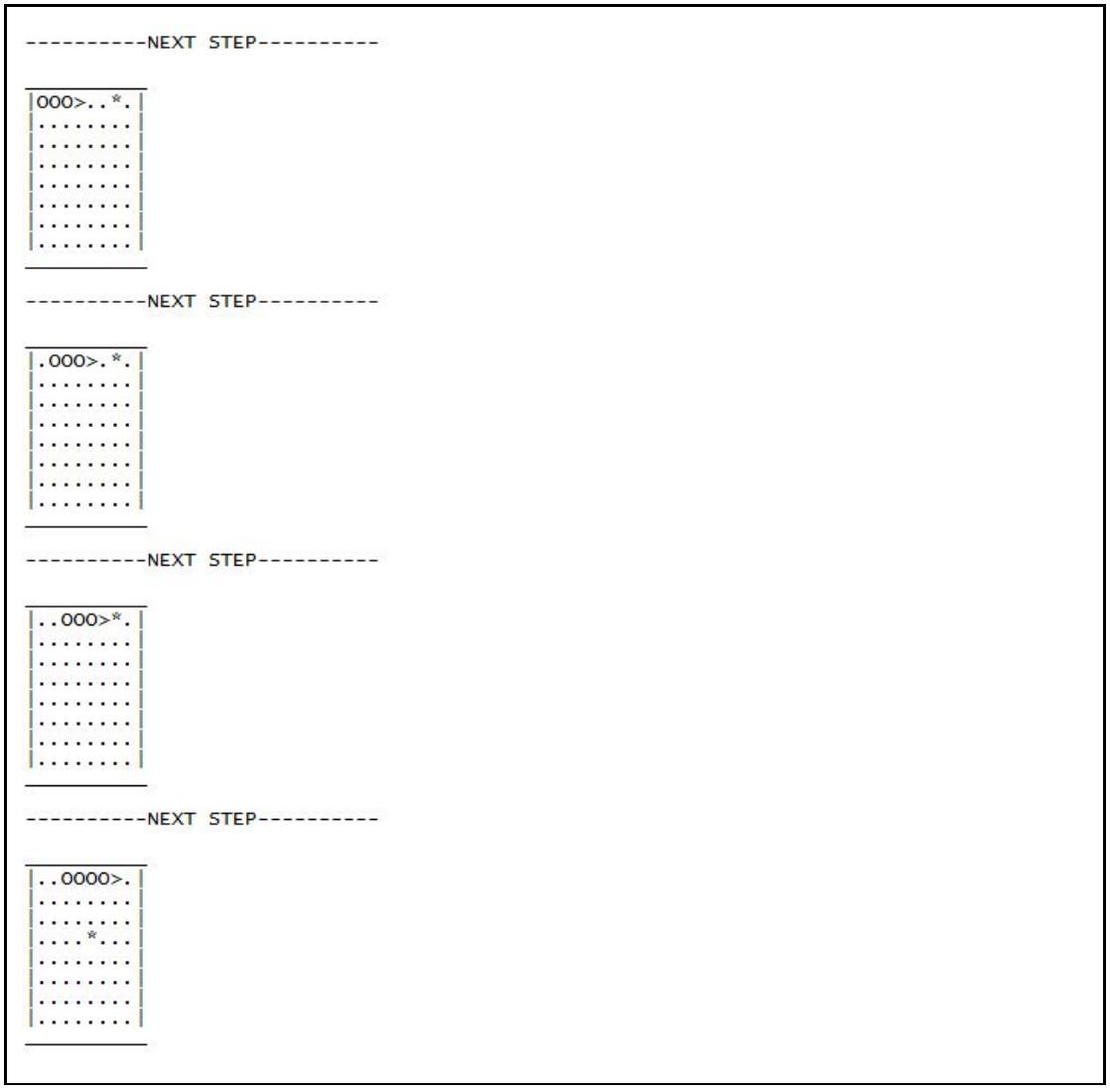



----------NEXT STEP----------
|....O>..|
|........|
|........|
|........|
|........|
|.......*|
|........|
|........|
----------NEXT STEP----------
|.....O>.|
|........|
|........|
|........|
|........|
|.......*|
|........|
|........|
----------NEXT STEP----------
|......O>|
|........|
|........|
|........|
|........|
|.......*|
|........|
|........|
----------NEXT STEP----------
|.......O|
|.......V|
|........|
|........|
|........|
|.......*|
|........|
|........|

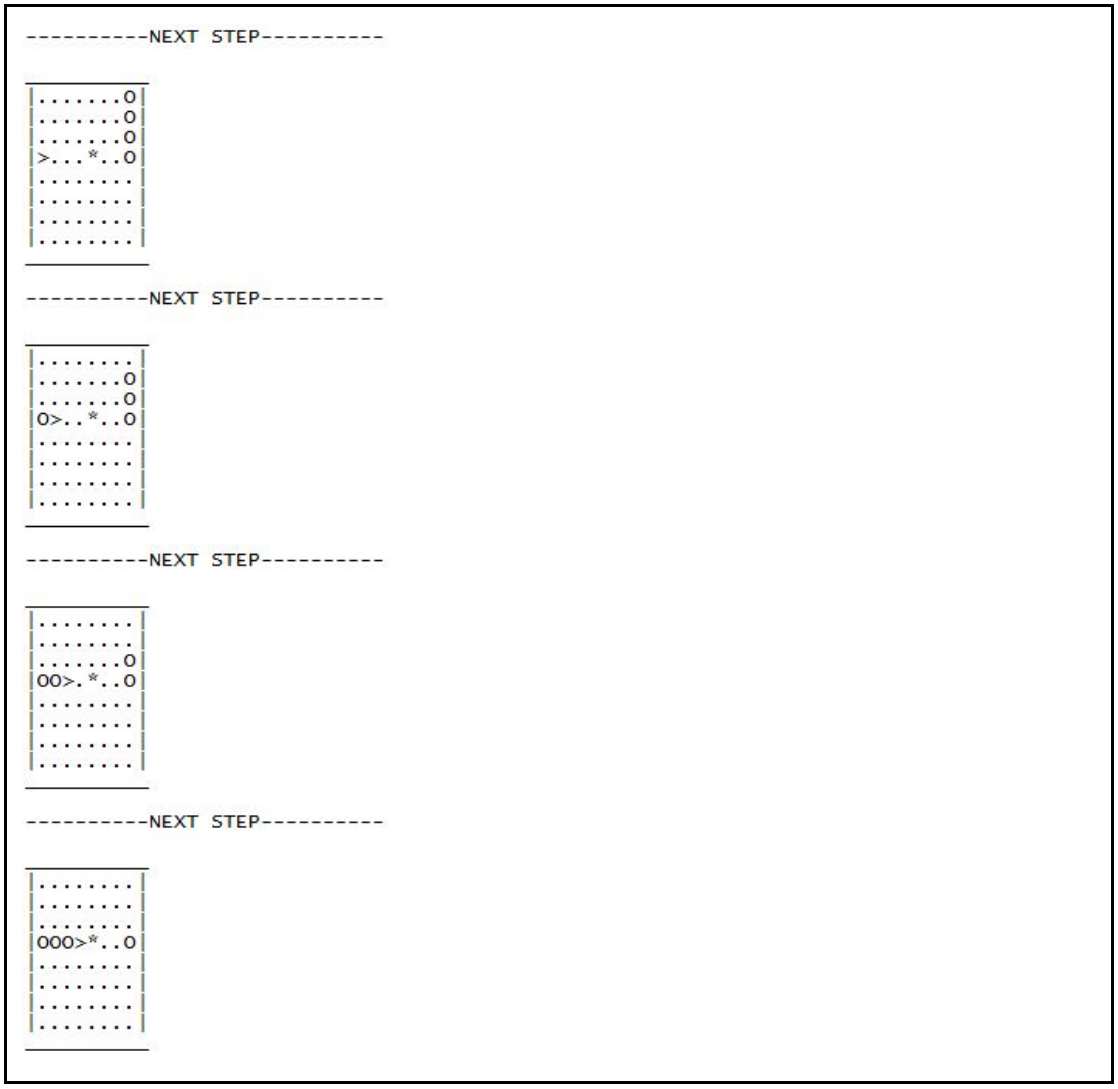

----------NEXT STEP----------
----------NEXT STEP----------
----------NEXT STEP----------
----------NEXT STEP----------


----------NEXT STEP----------
|........|
|*.......|
|........|
|........|
|.......O|
|>......O|
|........|
|........|
----------NEXT STEP----------
|........|
|*.......|
|........|
|........|
|^.......|
|O......O|
|........|
|........|
----------NEXT STEP----------
|........|
|*.......|
|........|
|^.......|
|O.......|
|O.......|
|........|
|........|
----------NEXT STEP----------
|........|
|*.......|
|^.......|
|O.......|
|O.......|
|........|
|........|
|........|


----------NEXT STEP----------
|.....*.|
|^.....|
|O.....|
|O.....|
|O.....|
|.....|
|.....|
|.....|
----------NEXT STEP----------
|^....*.|
|O.....|
|O.....|
|O.....|
|.....|
|.....|
|.....|
|.....|
----------NEXT STEP----------
|O>...*.|
|O.....|
|O.....|
|.....|
|.....|
|.....|
|.....|
|.....|
----------NEXT STEP----------
|OO>...*.|
|O.....|
|.....|
|.....|
|.....|
|.....|
|.....|
|.....|


----------NEXT STEP----------
----------NEXT STEP----------
----------NEXT STEP----------
----------NEXT STEP----------

```
----------NEXT STEP----------

 ________
|...OOOO>|
|........|
|........|
|....*...|
|........|
|........|
|........|
|........|
 ________

----------NEXT STEP----------

 ________
|....OOOO|
|.......v|
|........|
|....*...|
|........|
|........|
|........|
|........|
 ________

----------NEXT STEP----------

 ________
|.....OOO|
|.......O|
|.......v|
|....*...|
|........|
|........|
|........|
|........|
 ________

----------NEXT STEP----------

 ________
|......OO|
|.......O|
|.......O|
|....*..v|
|........|
|........|
|........|
|........|
 ________
```


----------NEXT STEP----------
----------NEXT STEP----------
----------NEXT STEP----------
----------NEXT STEP----------

```
----------NEXT STEP----------

 _________
|......*. |
|........ |
|........ |
|OOOOO>.. |
|........ |
|........ |
|........ |
|........ |
 _________

----------NEXT STEP----------

 _________
|......*. |
|........ |
|........ |
|.OOOOO.. |
|.....v.. |
|........ |
|........ |
|........ |
 _________

----------NEXT STEP----------

 _________
|......*. |
|........ |
|........ |
|..OOOO.. |
|.....O>. |
|........ |
|........ |
|........ |
 _________

----------NEXT STEP----------

 _________
|......*. |
|........ |
|........ |
|...OOO^. |
|.....OO. |
|........ |
|........ |
|........ |
 _________
**********************
*      GAME OVER     *
**********************
```

```
----------NEXT STEP----------

 __________
|......*.  |
|........  |
|........  |
|....O<O.  |
|.....OO.  |
|........  |
|........  |
|........  |
 __________
```

## B)Anexar o trecho da arquitetura do módulo snake_stimuli modificado com os comandos para se chegar ao game over

```
library IEEE;
use IEEE.std_logic_1164.all;


entity snake_stimuli is
	port
	(
	clk			: out STD_LOGIC;
	res			: out STD_LOGIC;
	sys_direction		: out STD_LOGIC_VECTOR (3 downto 0);
	sys_step_jumper	: out STD_LOGIC
	--PINS TO VGA
	);
end snake_stimuli;


architecture arch of snake_stimuli is

-- define some constants to help in the snake path test
constant NO : std_logic_vector(3 downto 0):= "0000"; -- no direction selected
constant RT : std_logic_vector(3 downto 0):= "0001"; -- right
constant LF : std_logic_vector(3 downto 0):= "1000"; -- left
constant UP : std_logic_vector(3 downto 0):= "0010"; -- up
constant DW : std_logic_vector(3 downto 0):= "0100"; -- down

signal clk_s		: STD_LOGIC	:= '0';


begin

	clk		<= clk_s;

	--basic process
	process
	begin
		clk_s <= not clk_s;
		wait for 50 ns;
	end process;

	process
	begin

		sys_direction <= RT;

		res <= '1';
		wait for 100 ns;
		res <= '0';

		sys_direction <= RT;

		wait for 11200 ns;
```

```
sys_direction <= NO;
--o jogo começou
--12100
wait for 12000 ns;
wait for 12000 ns;
wait for 12000 ns;
wait for 12000 ns;
wait for 12000 ns;
wait for 12000 ns;
wait for 12000 ns;
sys_direction <= DW;
wait for 12000 ns;
wait for 12000 ns;
wait for 12000 ns;
wait for 12000 ns;
wait for 12000 ns;

sys_direction <= RT;
wait for 12000 ns;

sys_direction <= UP;

wait for 12000 ns;
wait for 12000 ns;
wait for 12000 ns;
wait for 12000 ns;
wait for 12000 ns;

sys_direction <= RT;
wait for 12000 ns;
wait for 12000 ns;
wait for 12000 ns;
wait for 12000 ns;
wait for 12000 ns;
wait for 12000 ns;
wait for 12000 ns;

sys_direction <= DW;

wait for 12000 ns;
wait for 12000 ns;
wait for 12000 ns;


sys_direction <= RT;


wait for 12000 ns;
wait for 12000 ns;
wait for 12000 ns;
wait for 12000 ns;

wait for 12000 ns;
wait for 12000 ns;

--mata a cobra
sys_direction <= DW;
```

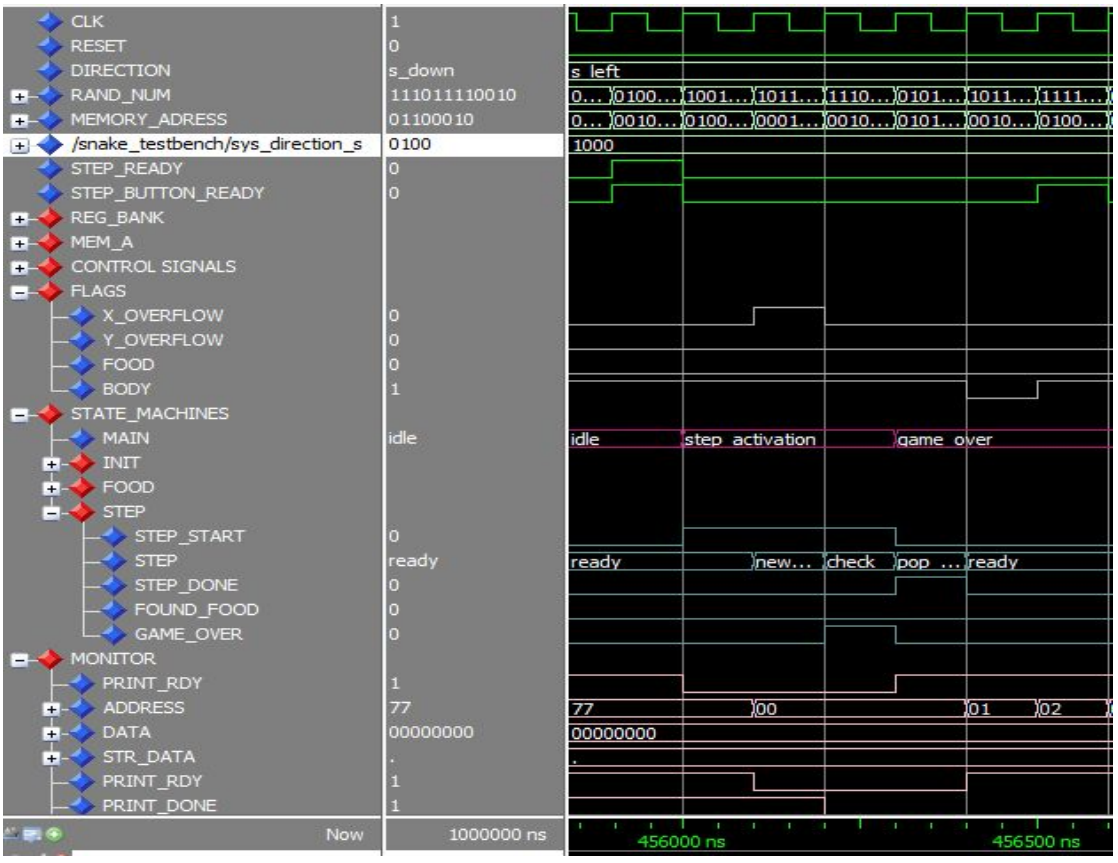
```
                    wait for 12000 ns;
                    sys_direction <= RT;
                    wait for 12000 ns;
                    sys_direction <= UP;
                    wait for 12000 ns;
                    sys_direction <= LF;
                    wait for 12000 ns;

                    wait;

        end process;

end arch;
```

## C)Descreva abaixo como os comandos do snake_stimuli correlacionam-se com os resultados do mapa do item a).

Os comando sys_direction <= X indica a direção que o Snake deve seguir no jogo. O wait 12000ns é o tempo que demora para que ele ande um quadrado. Desta maneira, fica fácil visualizar a relação entre os comandos, o tempo de espera, e o jogo.

**D) ) Copie a seguir imagem do ModelSim, onde fique evidente o estado de game_over sendo atingido, com a ativação dos sinais que levam a este estado (do conflito cabeça x corpo).**

Na imagem abaixo é possível ver os estados das máquinas e a flag BODY ativa quando a cabeça encontra o corpo.



## e) Faça uma breve descrição de como interpretar a curva do item d).

A flag BODY é ativa quando tenta-se colocar a cabeça em uma posição de memória que tem corpo da cobra. Ao término da operação da máquina STEP, verifica-se se ela está ou não ativa. Se estiver, é atribuido GAME OVER.

**f) Copie a seguir, o trecho do arquivo run_sim_1.do, correspondente às alterações citadas acima**

```
add wave -noupdate -group FLAGS -color Gray70 -label BODY
/snake_testbench/dut/cntrl_unit/dp_flags.cmp_body_flag
add wave -noupdate -expand -group STATE_MACHINES -color {Medium Violet
Red} -label MAIN /snake_testbench/dut/cntrl_unit/main/STATE
add wave -noupdate -expand -group STATE_MACHINES -group INIT -color
{Cadet Blue} -label INIT_START
/snake_testbench/dut/cntrl_unit/main/fsm_i_start
add wave -noupdate -expand -group STATE_MACHINES -group INIT -color
{Cadet Blue} -label INIT /snake_testbench/dut/cntrl_unit/init/STATE
add wave -noupdate -expand -group STATE_MACHINES -group INIT -color
{Cadet Blue} -label INIT_DONE /snake_testbench/dut/cntrl_unit/main/fsm_i_done
add wave -noupdate -expand -group STATE_MACHINES -group FOOD -color
{Cadet Blue} -label FOOD_START
/snake_testbench/dut/cntrl_unit/main/fsm_f_start
add wave -noupdate -expand -group STATE_MACHINES -group FOOD -color
{Cadet Blue} -label FOOD /snake_testbench/dut/cntrl_unit/food/STATE
add wave -noupdate -expand -group STATE_MACHINES -group FOOD -color
{Cadet Blue} -label FOOD_DONE
/snake_testbench/dut/cntrl_unit/main/fsm_f_done
add wave -noupdate -expand -group STATE_MACHINES -expand -group STEP
-color {Cadet Blue} -label STEP_START
/snake_testbench/dut/cntrl_unit/main/fsm_s_start
add wave -noupdate -expand -group STATE_MACHINES -expand -group STEP
-color {Cadet Blue} -label STEP /snake_testbench/dut/cntrl_unit/step/STATE
add wave -noupdate -expand -group STATE_MACHINES -expand -group STEP
-color {Cadet Blue} -label STEP_DONE
/snake_testbench/dut/cntrl_unit/main/fsm_s_done
add wave -noupdate -expand -group STATE_MACHINES -expand -group STEP
-color {Cadet Blue} -label FOUND_FOOD
/snake_testbench/dut/cntrl_unit/main/cmp_food_flag
add wave -noupdate -expand -group STATE_MACHINES -expand -group STEP
-color {Cadet Blue} -label GAME_OVER
/snake_testbench/dut/cntrl_unit/main/fsm_s_game_over
```

## 3) Resultados das simulações do VHDL final (snake completo) e do seu testbench no ModelSim (situação de jogo longo)

### A) Anexo de jogo longo

Em anexo (nome = JOGO-LONGO).

### B) Anexar o trecho da arquitetura do módulo snake_stimuli modificado com os comandos para se chegar à condição final

```
library IEEE;
use IEEE.std_logic_1164.all;

entity snake_stimuli is
	port
	(
	clk			: out STD_LOGIC;
	res			: out STD_LOGIC;
	sys_direction	: out STD_LOGIC_VECTOR (3 downto 0);
	sys_step_jumper		: out STD_LOGIC
	--PINS TO VGA
	);
end snake_stimuli;


architecture arch of snake_stimuli is

-- define some constants to help in the snake path test
constant NO : std_logic_vector(3 downto 0):= "0000"; -- no direction selected
constant RT : std_logic_vector(3 downto 0):= "0001"; -- right
constant LF : std_logic_vector(3 downto 0):= "1000"; -- left
constant UP : std_logic_vector(3 downto 0):= "0010"; -- up
constant DW : std_logic_vector(3 downto 0):= "0100"; -- down

signal clk_s	: STD_LOGIC  := '0';


begin

	clk	<= clk_s;
	--basic process
	process
	begin
		clk_s <= not clk_s;
		wait for 50 ns;
	end process;

	process
```

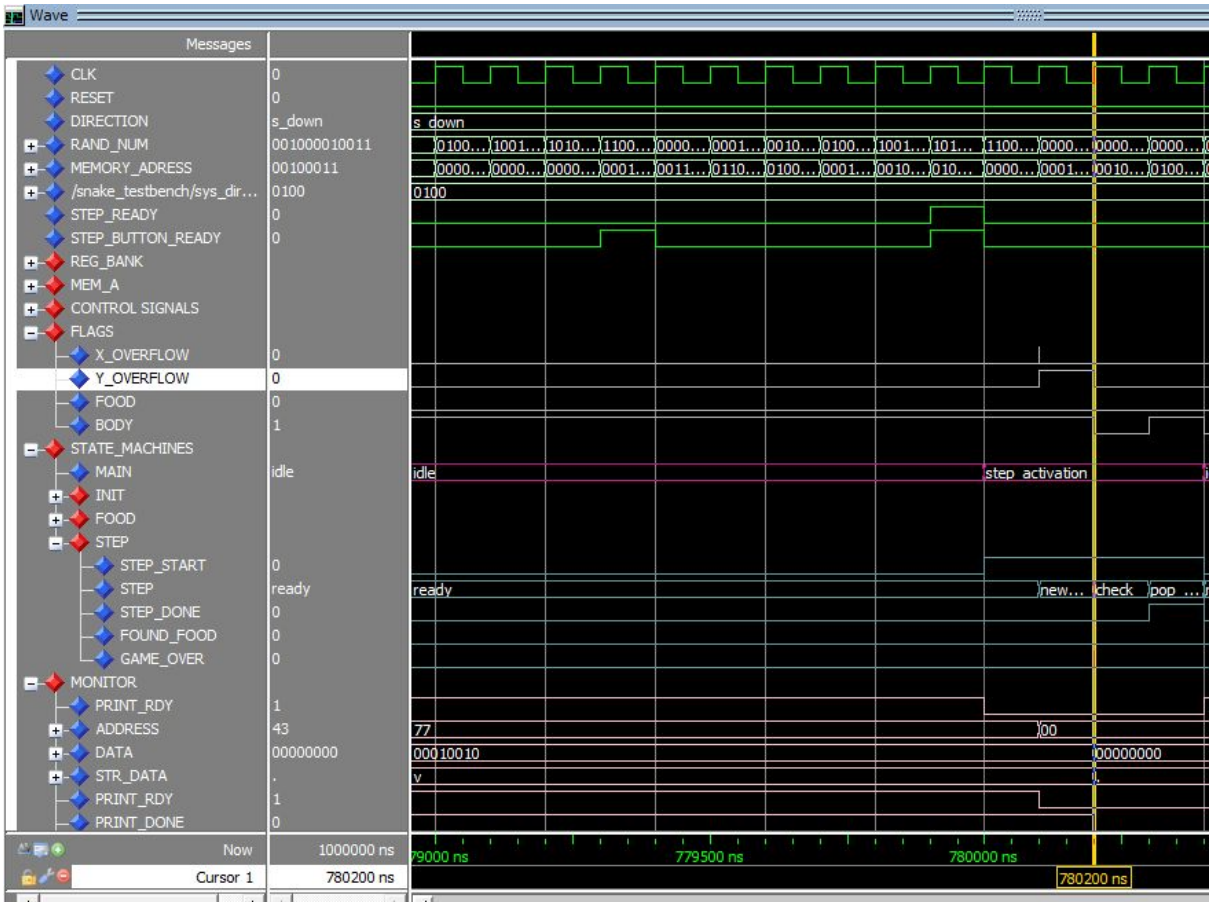
```
	begin

		sys_direction <= RT;

		res <= '1';
		wait for 100 ns;
		res <= '0';

		sys_direction <= RT;
		wait for 11200 ns;
		sys_direction <= NO;
		--o jogo começou
		--12100
		wait for 12000 ns;
		wait for 12000 ns;
		wait for 12000 ns;
		wait for 12000 ns;
		wait for 12000 ns;
		wait for 12000 ns;
		wait for 12000 ns;
		sys_direction <= DW;
		wait for 12000 ns;
		wait for 12000 ns;
		wait for 12000 ns;
		wait for 12000 ns;
		wait for 12000 ns;
		sys_direction <= RT;
		wait for 12000 ns;
		sys_direction <= UP;
		wait for 12000 ns;
		wait for 12000 ns;
		wait for 12000 ns;
		wait for 12000 ns;
		wait for 12000 ns;
		sys_direction <= RT;
		wait for 12000 ns;
		wait for 12000 ns;
		wait for 12000 ns;
		wait for 12000 ns;
		wait for 12000 ns;
		wait for 12000 ns;
		wait for 12000 ns;
		sys_direction <= DW;
		wait for 12000 ns;
		wait for 12000 ns;
		wait for 12000 ns;
		sys_direction <= RT;
		wait;
	end process;
end arch;
```

## c) Descreva abaixo como os comandos do snake_stimuli (em todas as direções) correlacionam-se com os resultados do mapa do item a)

Os comando sys_direction <= X indica a direção que o Snake deve seguir no jogo. O wait 12000ns é o tempo que demora para que ele ande um quadrado. Desta maneira, fica fácil visualizar a relação entre os comandos, o tempo de espera, e o jogo.

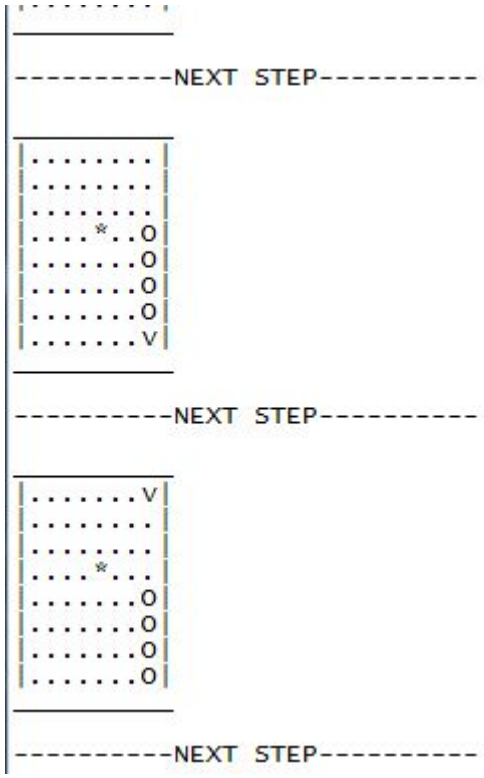
## 4.i. Observação de cruzamento da cobra por uma borda vertical

**A)**

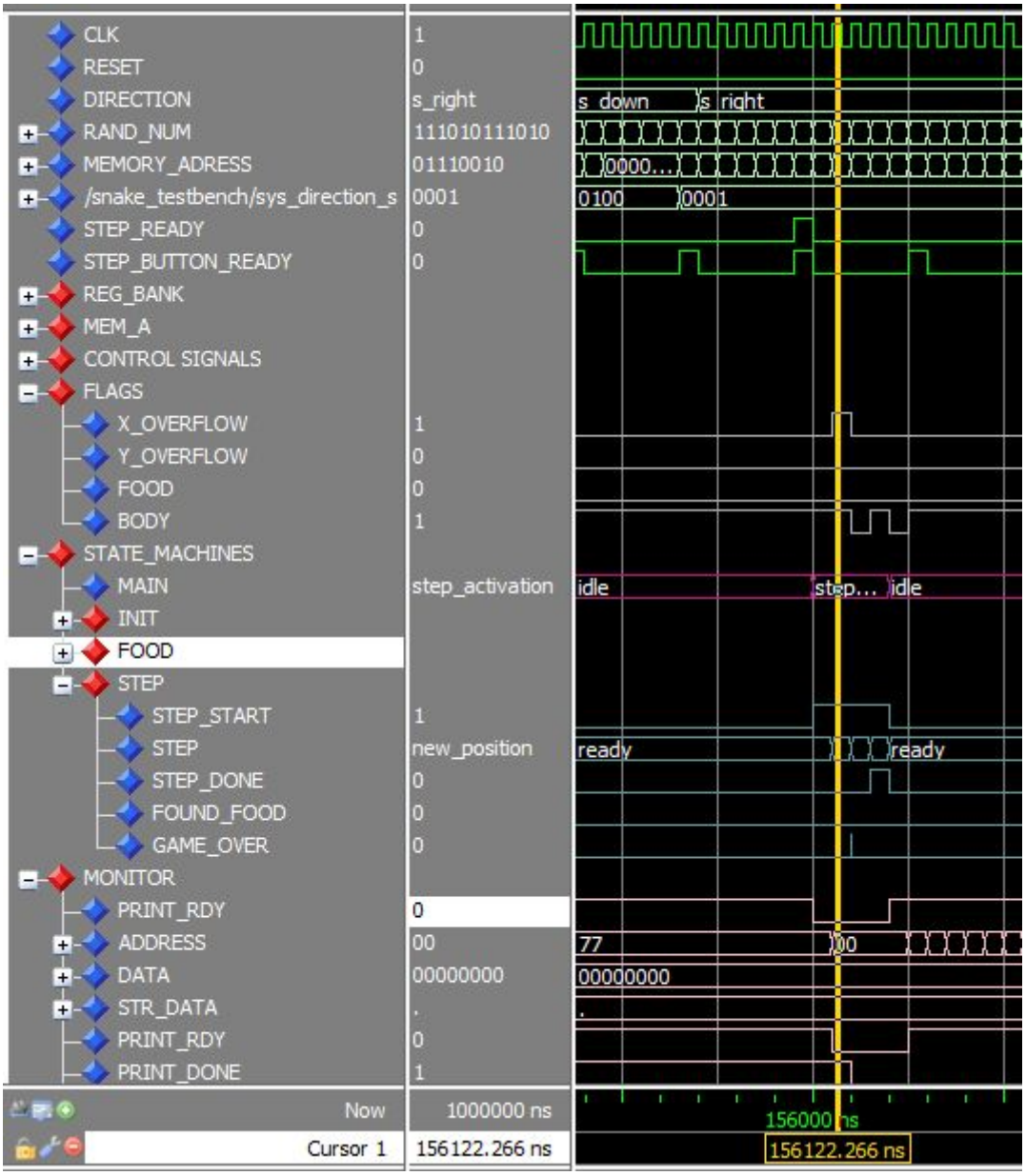



## B)Faça uma breve descrição de como interpretar a curva do item a) de acordo com observação desejada.

O snake passou da borda de baixo para a de cima. Isso gera o sinal overflow of y, que leva os endereços de memória para o overflow correction, que faz a operação módulo.


----------NEXT STEP----------
----------NEXT STEP----------
----------NEXT STEP----------

## 4.ii. Observação de cruzamento da cobra por uma borda horizontal

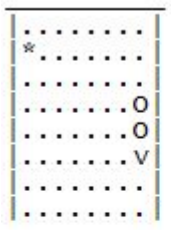



## 4.ii.b) Faça uma breve descrição de como interpretar a curva do item a) de acordo com observação desejada.

O snake passou da borda direita para a esquerda. Isso gera o sinal overflow of x, que leva os endereços de memória para o overflow correction, que faz a operação módulo.

## c) Associe a condição tratada com o passo do mapa de tabuleiro anexado (item 4.a).

----------NEXT STEP----------



----------NEXT STEP----------