PSI-3451 Projeto de CI Lógicos Integrados

Luiz Sol - 8586861

Experimento 6 - Implementação do LFSR

O aluno deve apresentar o polinômio, explicando como chegou à configuração

```
NUSP = 8586861
```

```
8586861 % 2048 = 1645 = 0b11001101101
```

O que resultará no polinômio:

$$x^{12} + x^{11} + x^{10} + x^7 + x^6 + x^4 + x^3 + x^1 + 1$$

Figure 1: Polinômio resultante

Apresentar um esboço (em forma digital ou manuscrito) do esquema do circuito LFSR desenvolvido.

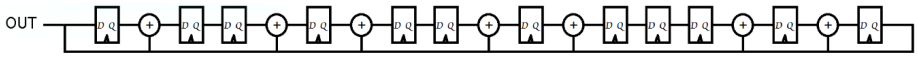
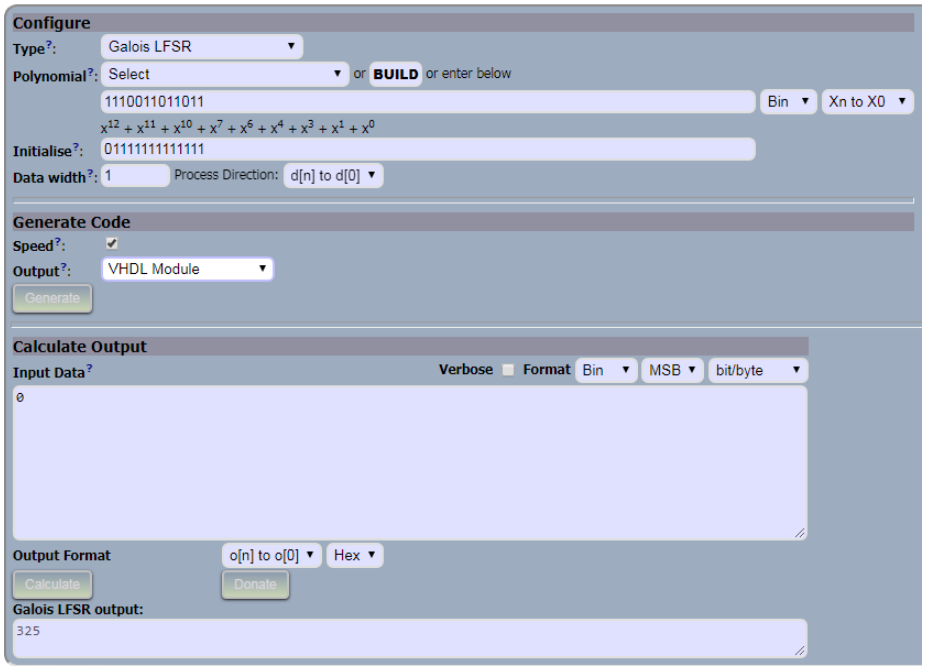
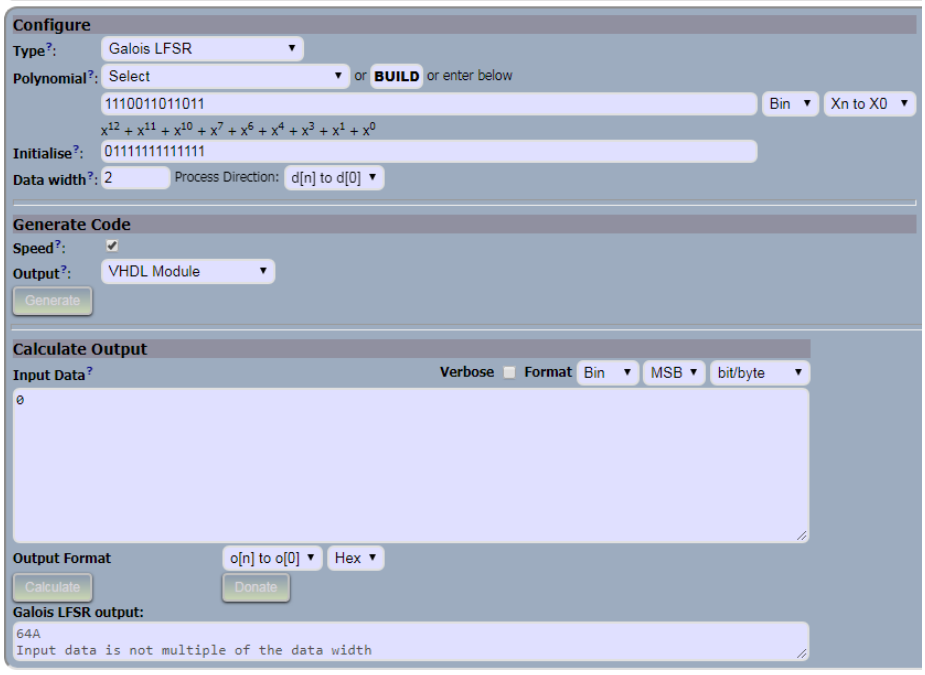
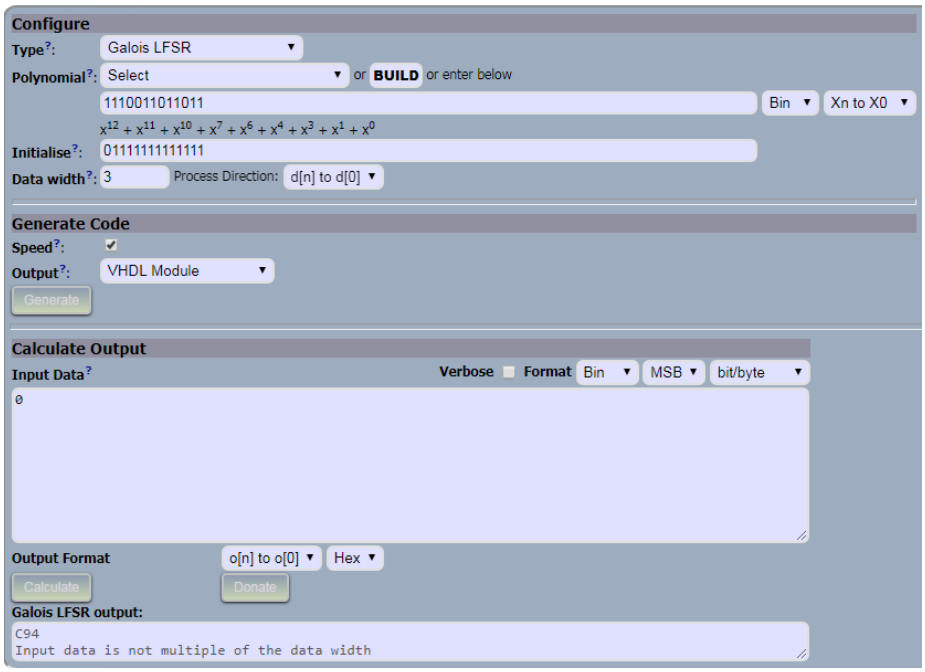


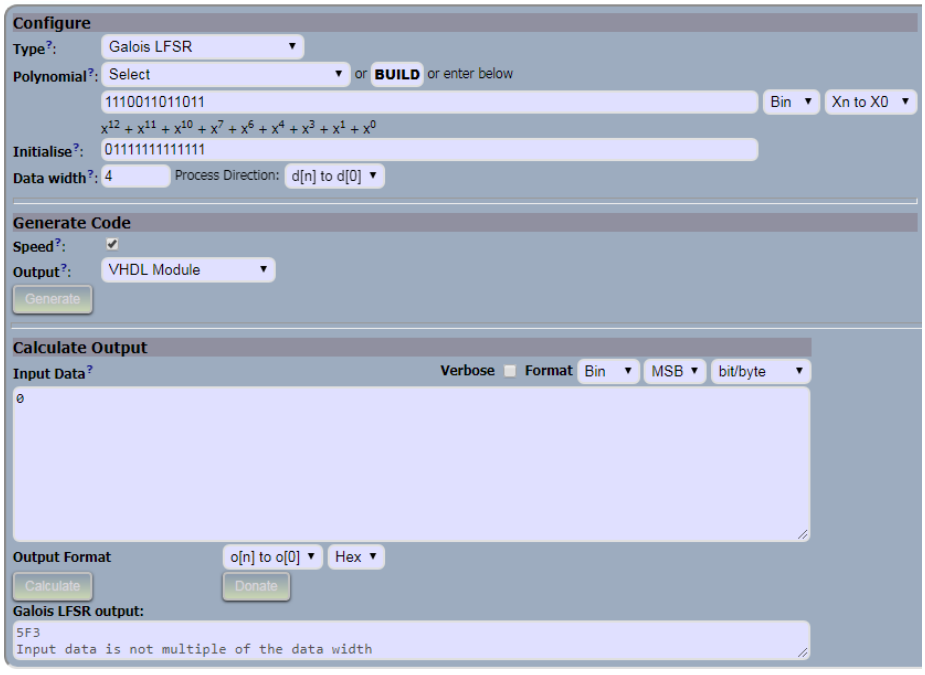
Figure 2: Estrutura resultante (Figura 1.b)

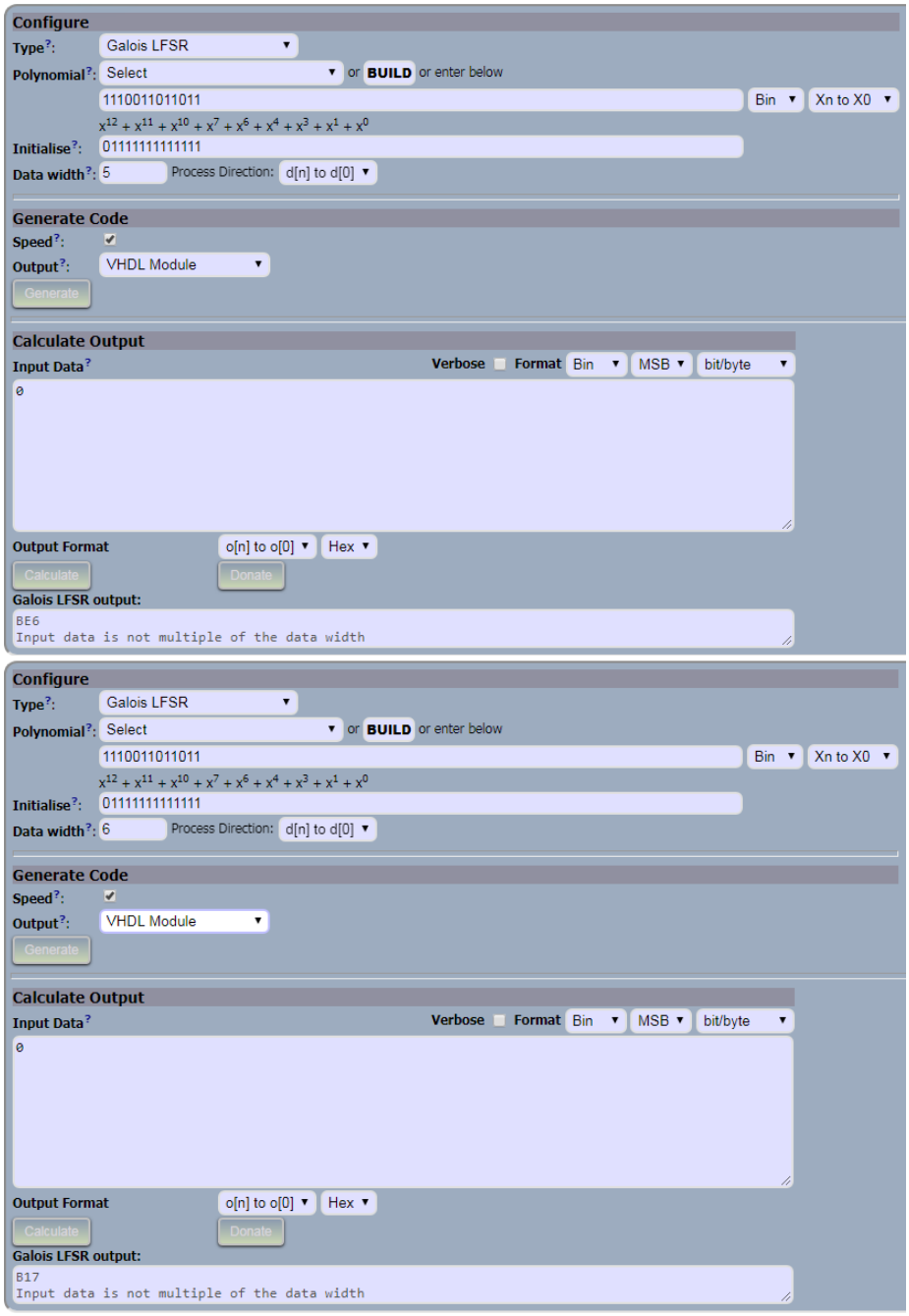
Impressão das imagens de tela com os resultados da simulação (10 ciclos).

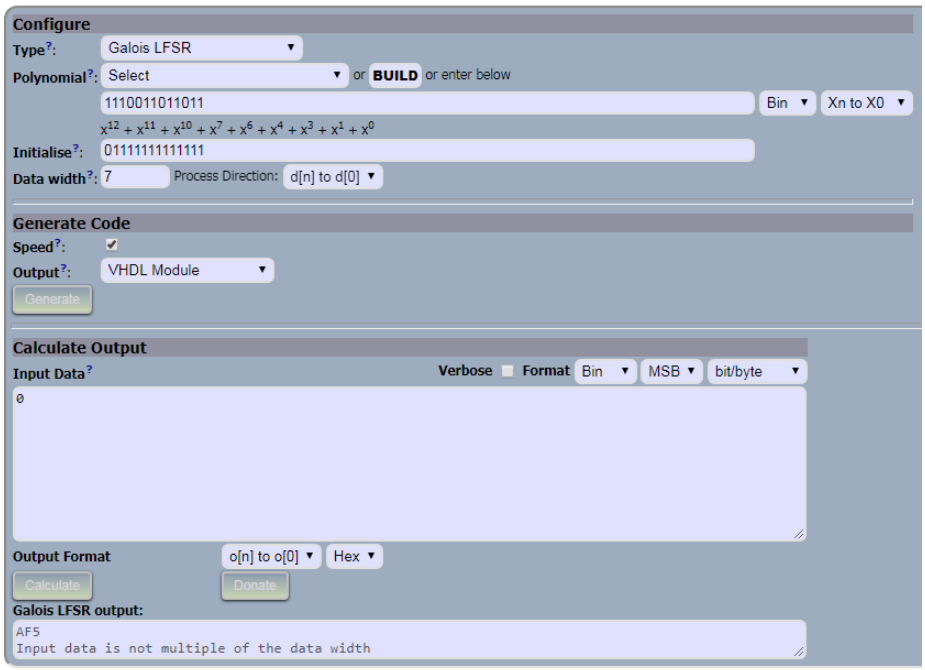

Configure
Type?: Galois LFSR
Polynomial?: Select or BUILD or enter below
1110011011011
Bin Xn to X0
x12 + x11 + x10 + x7 + x6 + x4 + x3 + x1 + x0
Initialise?: 01111111111111
Data width?: 1 Process Direction: d[n] to d[0]
Generate Code
Speed?:
Output?: VHDL Module
Generate
Calculate Output
Input Data? Verbose Format Bin MSB bit/byte
0
Output Format o[n] to o[0] Hex
Calculate Donate
Galois LFSR output:
325

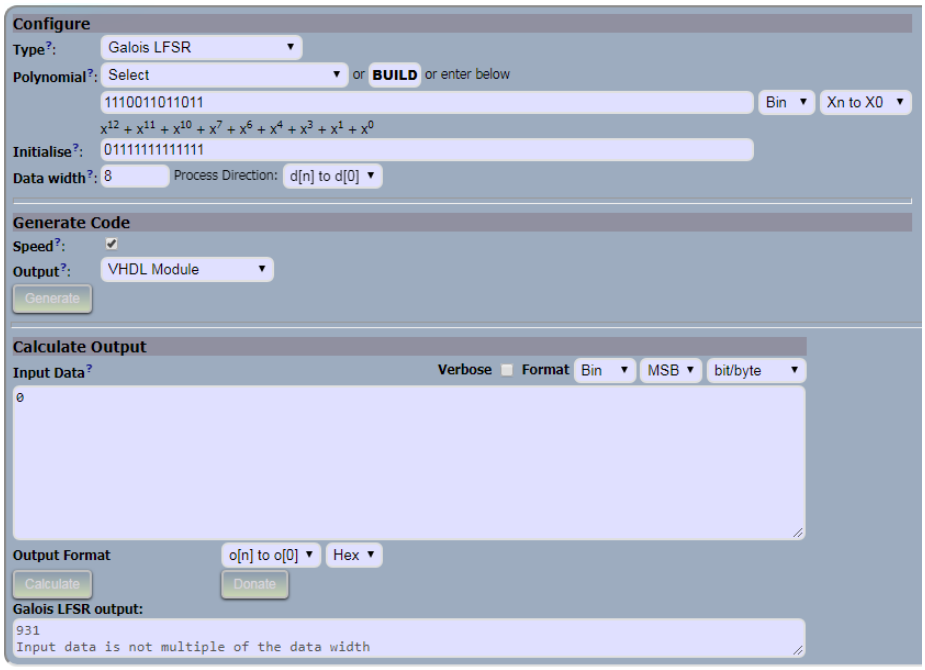

Configure
Type?: Galois LFSR
Polynomial?: Select or BUILD or enter below
1110011011011
Bin Xn to X0
x12 + x11 + x10 + x7 + x6 + x4 + x3 + x1 + x0
Initialise?: 01111111111111
Data width?: 2 Process Direction: d[n] to d[0]
Generate Code
Speed?:
Output?: VHDL Module
Generate
Calculate Output
Input Data? Verbose Format Bin MSB bit/byte
0
Output Format o[n] to o[0] Hex
Calculate Donate
Galois LFSR output:
64A
Input data is not multiple of the data width

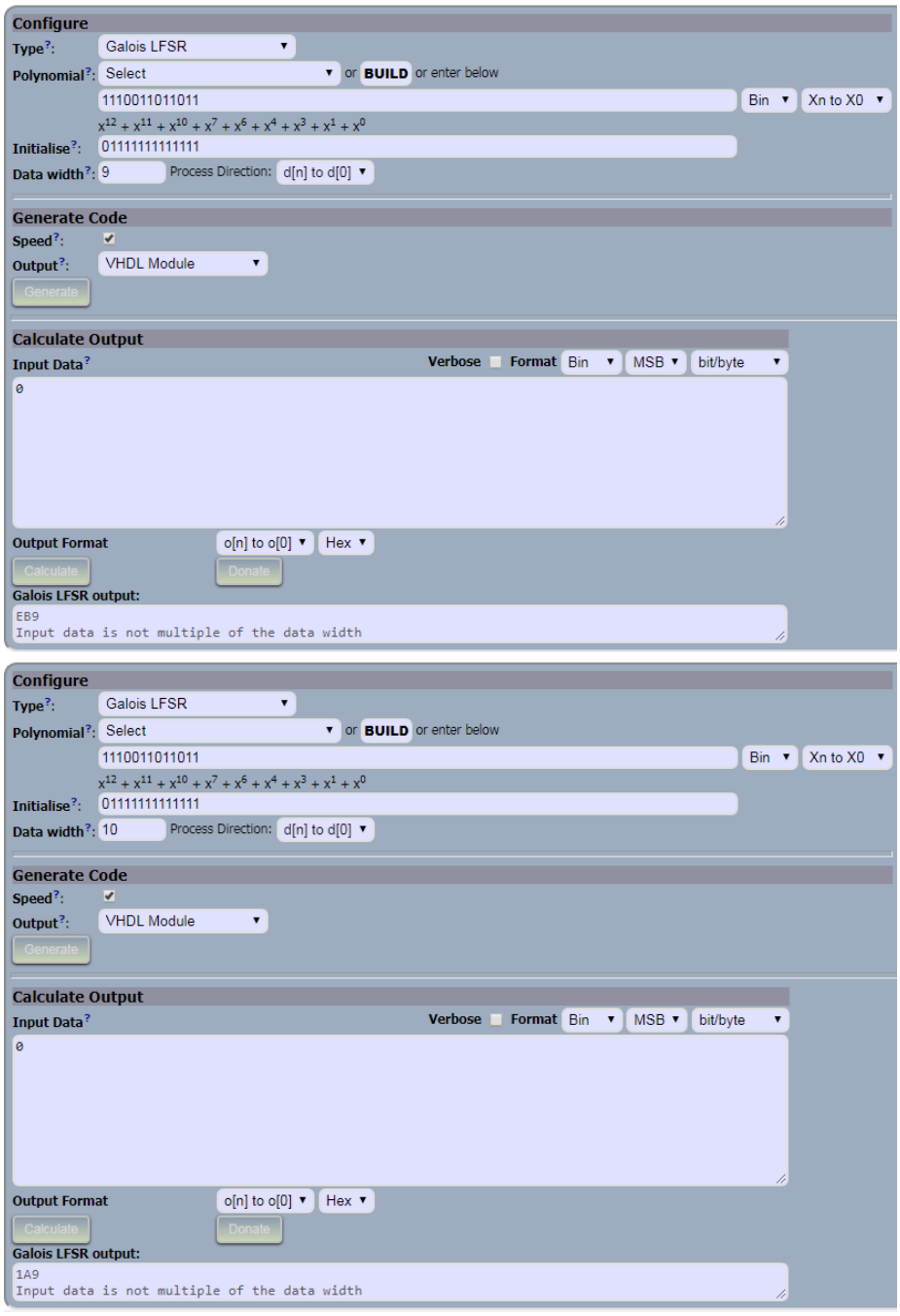

Configure
Type?: Galois LFSR
Polynomial?: Select or BUILD or enter below
1110011011011
Bin Xn to X0
x12 + x11 + x10 + x7 + x6 + x4 + x3 + x1 + x0
Initialise?: 01111111111111
Data width?: 3 Process Direction: d[n] to d[0]
Generate Code
Speed?:
Output?: VHDL Module
Generate
Calculate Output
Input Data? Verbose Format Bin MSB bit/byte
0
Output Format o[n] to o[0] Hex
Calculate Donate
Galois LFSR output:
C94
Input data is not multiple of the data width


Configure
Type?: Galois LFSR
Polynomial?: Select or BUILD or enter below
1110011011011
Bin Xn to X0
x12 + x11 + x10 + x7 + x6 + x4 + x3 + x1 + x0
Initialise?: 01111111111111
Data width?: 4 Process Direction: d[n] to d[0]
Generate Code
Speed?:
Output?: VHDL Module
Generate
Calculate Output
Input Data? Verbose Format Bin MSB bit/byte
0
Output Format o[n] to o[0] Hex
Calculate Donate
Galois LFSR output:
5F3
Input data is not multiple of the data width


Configure
Type?: Galois LFSR
Polynomial?: Select or BUILD or enter below
1110011011011
Bin Xn to X0
$x^{12} + x^{11} + x^{10} + x^7 + x^6 + x^4 + x^3 + x^1 + x^0$
Initialise?: 01111111111111
Data width?: 5 Process Direction: d[n] to d[0]
Generate Code
Speed?:
Output?: VHDL Module
Generate
Calculate Output
Input Data? Verbose Format Bin MSB bit/byte
0
Output Format o[n] to o[0] Hex
Calculate Donate
Galois LFSR output:
BE6
Input data is not multiple of the data width
Configure
Type?: Galois LFSR
Polynomial?: Select or BUILD or enter below
1110011011011
Bin Xn to X0
$x^{12} + x^{11} + x^{10} + x^7 + x^6 + x^4 + x^3 + x^1 + x^0$
Initialise?: 01111111111111
Data width?: 6 Process Direction: d[n] to d[0]
Generate Code
Speed?:
Output?: VHDL Module
Generate
Calculate Output
Input Data? Verbose Format Bin MSB bit/byte
0
Output Format o[n] to o[0] Hex
Calculate Donate
Galois LFSR output:
B17
Input data is not multiple of the data width


Configure
Type?: Galois LFSR
Polynomial?: Select or BUILD or enter below
1110011011011
Bin
Xn to X0
x12 + x11 + x10 + x7 + x6 + x4 + x3 + x1 + x0
Initialise?: 01111111111111
Data width?: 7
Process Direction: d[n] to d[0]
Generate Code
Speed?:
Output?: VHDL Module
Generate
Calculate Output
Input Data?
Verbose
Format
Bin
MSB
bit/byte
0
Output Format
o[n] to o[0]
Hex
Calculate
Donate
Galois LFSR output:
AF5
Input data is not multiple of the data width


Configure
Type?: Galois LFSR
Polynomial?: Select or BUILD or enter below
1110011011011
Bin
Xn to X0
x12 + x11 + x10 + x7 + x6 + x4 + x3 + x1 + x0
Initialise?: 01111111111111
Data width?: 8
Process Direction: d[n] to d[0]
Generate Code
Speed?:
Output?: VHDL Module
Generate
Calculate Output
Input Data?
Verbose
Format
Bin
MSB
bit/byte
0
Output Format
o[n] to o[0]
Hex
Calculate
Donate
Galois LFSR output:
931
Input data is not multiple of the data width



foi gerada a seguinte sequência:

Data width	Galois LFSR Output	Binário	Binário Modificado	Decimal Modificado
1	325	0011 0010 0101	0001 0101	21
2	64A	0110 0100 1010	0011 0010	50
3	C94	1100 1001 0100	0110 0100	100
4	5F3	0101 1111 0011	0010 0011	35
5	BE6	1011 1110 0110	0101 0110	86
6	B17	1011 0001 0111	0101 0111	87
7	AF5	1010 1111 0101	0101 0101	85
8	931	1001 0011 0001	0100 0001	65
9	EB9	1110 1011 1001	0111 0001	113
10	1A9	0001 1010 1001	0000 0001	1

Faça a descrição equivalente do circuito em VHDL (figura 1.b) seguindo o esquema dado no item c) (atenção: utilize os mesmos nomes para os sinais e portos).

- O DFF contendo um sinal de set é fornecido ao aluno (na área da disciplina no NEWSERVERLAB).
- Para o XOR, use o módulo utilizado para os somadores das aulas anteriores.
- Você deverá usar obrigatoriamente o comando GENERATE para construir o conjunto do LFSR. O comando deverá ser usado de forma "inteligente" de forma a otimizar a codificação.
- O vetor de estados conterá os bits de saída dos FFs (dos bits 12 a 0) e portos do módulo deverão ser de acordo com o especificado na figura 1.b.

incluir a descrição do projeto em VHDL.

A seguir está o VHDL que descreve o LFSR:

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

entity lfsr is
    generic(
        WIDTH: natural := 12;
        POL  : STD_LOGIC_VECTOR (12 downto 0) := "1110011011011"
    );
    port (
        clk: in  STD_LOGIC;
        res: in  STD_LOGIC;
        o  : out STD_LOGIC_VECTOR (7 downto 0)
    );
```

```
end lfsr;

architecture arch of lfsr is
    COMPONENT xor2
        PORT (
            x, y: IN STD_LOGIC;
            z:    OUT STD_LOGIC
        );
    END COMPONENT;

    signal prev_state: STD_LOGIC_VECTOR(WIDTH - 1 downto 0);
    signal next_state: STD_LOGIC_VECTOR(WIDTH - 1 downto 0);
    signal or_signals: STD_LOGIC_VECTOR(WIDTH - 1 downto 0);

begin
    seq : process(clk)
    begin
        if clk'EVENT AND clk = '1' then
            prev_state <= next_state;
        end if;
    end process;

    layout: for I in 1 to (WIDTH - 1) generate
        zeros: if POL(I) = '0' generate
            next_state(I) <= prev_state(I - 1) or res;
        end generate zeros;
        ones: if POL(I) = '1' generate
            generated_xor2: xor2 port map (
                prev_state(I - 1),
                prev_state(WIDTH - 1),
                or_signals(I)
            );

            next_state(I) <= or_signals(I) or res;
        end generate ones;
    end generate layout;
    next_state(0) <= prev_state(WIDTH - 1) or res;

    o <= '0' & prev_state(WIDTH - 1 downto WIDTH - 3) & '0' & prev_state(2 downto 0);
end arch;
```

Incluir a descrição VHDL do testbench com seus componentes.

`clock.vhd`:

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

```
use IEEE.NUMERIC_STD.all;

entity clock is
    port (
        clk: out  STD_LOGIC
    );
end clock;

architecture arch of clock is
    constant CLOCK_PERIOD: time := 10 ns;
begin
    clk_generation: process
    begin
        clk <= '1';
        wait FOR CLOCK_PERIOD / 2;
        clk <= '0';
        wait FOR CLOCK_PERIOD / 2;

    end process clk_generation;

end architecture arch;
```

reg.vhd:

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

entity reg is
    generic(
        WIDTH: natural := 8
    );

    port (
        clk : in  STD_LOGIC;
        load: in  STD_LOGIC;
        d   : in  STD_LOGIC_VECTOR(WIDTH - 1 downto 0);
        q   : out STD_LOGIC_VECTOR(WIDTH - 1 downto 0)
    );
end reg;

architecture arch of reg is
    signal q_s : STD_LOGIC_VECTOR(WIDTH - 1 downto 0) := (others => '0');

begin
    q <= q_s;
    -- Register with active-high clock & asynchronous clear
```

```
    process(clk)
    begin
        if clk'EVENT AND clk = '1' then
            if (load = '1') then
                q_s <= d;
            end if;
        end if;
    end process;
end arch;
```

stimuli_lfsr.vhd:

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

entity stimuli_module is
    port (
        c: out STD_LOGIC;
        res  : out STD_LOGIC
    );
end stimuli_module ;

architecture test of stimuli_module  is
-- "Time" that will elapse between test vectors we submit to the component.
constant TIME_DELTA : time := 10 ns;      -- choose any value

    component clock
        port (
            clk: out STD_LOGIC
        );
    end component ;

begin
    clock_component : clock
        port map(
            clk => c
        );

    simulation : process
        procedure assign_reset (filler: in STD_LOGIC) is
        begin
        -- Assign values to estimuli_module´s outputs.
            res <= '1';
            wait for 2 * TIME_DELTA;
            res <= '0';
        end procedure assign_reset;
```

```
        begin
            -- test vectors application
            wait for 15 * TIME_DELTA;
            assign_reset('1');
            wait for 20 * TIME_DELTA;
            assign_reset('1');
            wait for 25 * TIME_DELTA;
            -- wait;
    end process simulation;
end architecture test;
```

testbench_lfsr.vhd:

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

entity tb_lfsr is
    generic (
        WIDTH: natural := 12
    );
end tb_lfsr;

architecture test of tb_lfsr is

    component stimuli_module
        port (
            c: out STD_LOGIC;
            res  : out STD_LOGIC
        );
    end component ;

    component lfsr
        generic(
            WIDTH: natural := 12;
            POL  : STD_LOGIC_VECTOR (12 downto 0) := "1110011011011"
        );
        port (
            clk: in  STD_LOGIC;
            res: in  STD_LOGIC;
            o  : out STD_LOGIC_VECTOR (7 downto 0)
        );
    end component ;

    signal clk_s, res_s: STD_LOGIC;
    signal o_s: STD_LOGIC_VECTOR(7 downto 0);
```

```
begin
    -- Instantiate DUT
    dut : lfsr
        generic map(WIDTH => WIDTH)
        port map(
            clk => clk_s,
            res => res_s,
            o => o_s
        );

    -- Instantiate test module
    test : stimuli_module
        port map(
            c => clk_s,
            res => res_s
        );

end architecture test;
```

xor2.vhd:

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;

ENTITY xor2 IS
    GENERIC(t_xor : time := 4 ns);
    PORT( x, y: IN STD_LOGIC;
             z: OUT STD_LOGIC);
END xor2;

ARCHITECTURE dataflow OF xor2 IS
BEGIN
    z <= x XOR y AFTER t_xor;
END dataflow;
```

Impressão legível da carta de tempos com pelo menos 20 ciclos de relógio (deixe os sinais importantes evidentes). Apresente tanto os sinais dos estados(12 bits) como de saída (8 bits). Obs. Represente os números aleatórios gerados com a mesma base numérica utilizada em d).

Relatório do projeto: Discuta e demonstre que (se) os resultados da simulação do projeto e da execução do software rodado são os mesmos (compatíveis entre si).

A sequência de números obtida a partir da simulação é exatamente igual à esper-

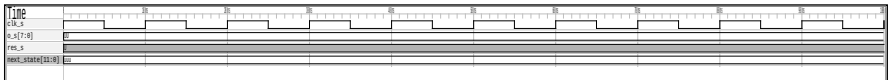


Figure 3: De 0[ns] a 100[ns]

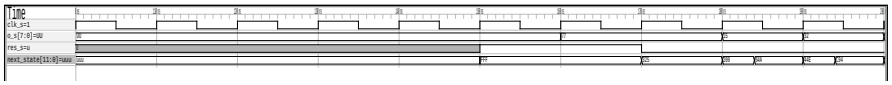


Figure 4: De 100[ns] a 200[ns]

Figure 5: De 200[ns] a 300[ns]

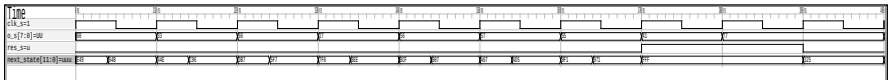


Figure 6: De 300[ns] a 400[ns]

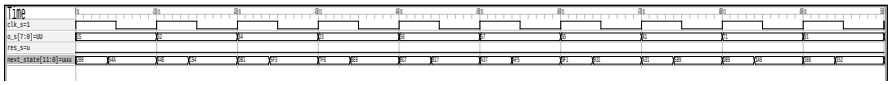


Figure 7: De 400[ns] a 500[ns]

Figure 8: De 500[ns] a 600[ns]

Figure 9: De 600[ns] a 700[ns]

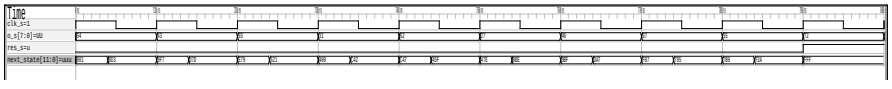


Figure 10: De 700[ns] a 800[ns]

ada na especificação (tanto a saída completa de 12 bits como a modificada de 8 bits), portanto é razoável concluir que a LFSR foi implementada corretamente.