Universidade do Vale do Itajaí - UNIVALI
Engenharia de Computação - 2º Período
Disciplina: Projetos de Sistemas Digitais
Alunos: Taryck, Pedro Kons, Cauã Domingos

Avaliação 3 – Projeto de Circuitos Sequenciais - FSM

Circuito 1: Máquina de estados sem entradas e três saídas

Enunciado: Implemente a máquina de estados especificada no Exercício 3.24 do livro texto (Sistemas Digitais, de Frank Vahid).

3.24 *Desenhe o diagrama de estados de uma FSM sem entradas e três saídas, x, y e z. Os valores de xyz devem seguir sempre a sequência: 000, 001,010,100, repetir. A saída deverá mudar apenas na borda de subida do relógio. Torne 000 o estado inicial.*

Circuito 1 - Testbench:

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

entity tb_maquina_design is
end tb_maquina_design;

architecture arch_1 of tb_maquina_design is

component maquina_design is
  port (i_CLR : in std_logic;
        i_CLK : in std_logic;
        o_QX  : out std_logic;
        o_QY  : out std_logic;
        o_QZ  : out std_logic);
end component;

signal w_CLR, w_CLK, w_QX, w_QY, w_QZ : std_logic;
begin

u_DUT: maquina_design port map (i_CLR => w_CLR,
                                i_CLK => w_CLK,
```

```
                                    o_QX  => w_QX,
                                    o_QY  => w_QY,
                                    o_QZ  => w_QZ);

process
     begin

 -- RESET

 w_CLR <= '0';
 w_CLK <= '0';
 wait for 1 ns;
 assert(w_CLR <='0' and w_CLK <='0') report "Fail @ 00"
severity error;

 w_CLR <= '1';
 wait for 1 ns;

 w_CLR <= '0';
 wait for 1 ns;
 --CLOCK

 w_CLK <= '1';
 wait for 1 ns;
 assert(w_CLK <='0') report "Fail @ 1" severity error;

 w_CLK <= '0';
 wait for 1 ns;
 assert(w_CLK <='0') report "Fail @ 0" severity error;

 w_CLK <= '1';
 wait for 1 ns;
 assert(w_CLK <='0') report "Fail @ 1" severity error;

 w_CLK <= '0';
 wait for 1 ns;
 assert(w_CLK <='0') report "Fail @ 0" severity error;

 w_CLK <= '1';
 wait for 1 ns;
 assert(w_CLK <='0') report "Fail @ 1" severity error;
 w_CLK <= '0';
 wait for 1 ns;
```

```
 assert(w_CLK <='0') report "Fail @ 0" severity error;

 w_CLK <= '1';
 wait for 1 ns;
 assert(w_CLK <='0') report "Fail @ 1" severity error;

 w_CLK <= '0';
 wait for 1 ns;
 assert(w_CLK <='0') report "Fail @ 0" severity error;

 w_CLR <= '1';
 wait;
end process;
end arch_1;
```

Circuito 1 - Design

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

entity maquina_design is
port ( i_CLR : in std_logic;
       i_CLK   : in std_logic;
       o_QX     : out std_logic;
       o_QY     : out std_logic;
       o_QZ     : out std_logic);
end maquina_design;

architecture arch_1 of maquina_design is
  type t_STATE is (q_0, q_1, q_2, q_3);
  signal r_STATE : t_STATE;
  signal w_NEXT : t_STATE;

begin

    p_STATE: process(i_CLR, i_CLK)
    begin
      if (i_CLR = '1') then
        r_STATE <= q_0;
      elsif (rising_edge(i_CLK)) then
        r_STATE <= w_NEXT;
      end if;
    end process;
```

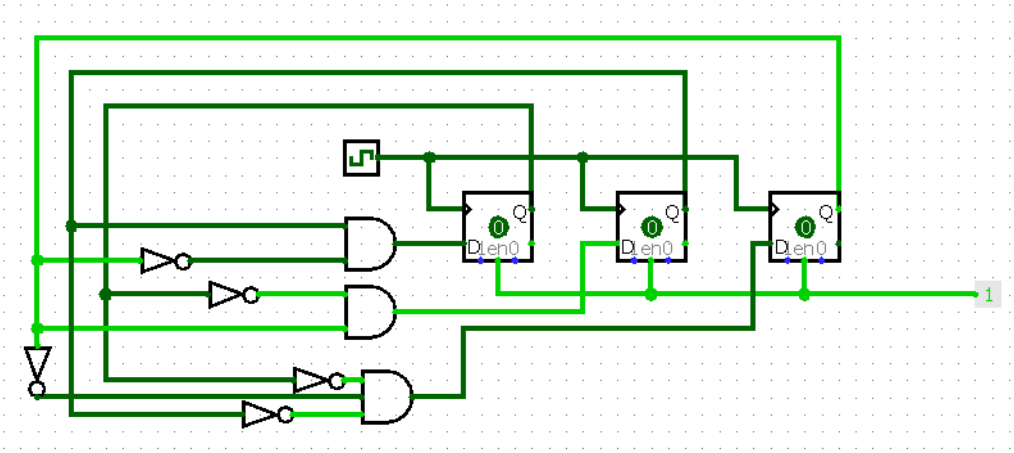
```
  p_NEXT: process(r_STATE)
  begin
    case (r_STATE) is
      when q_0 => w_NEXT <= q_1;

      when q_1 => w_NEXT <= q_2;

      when q_2 => w_NEXT <= q_3;

      when others => w_NEXT <= q_0;
    end case;

      if(r_STATE = q_3) then
          o_QX <= '1';
      else
          o_QX <= '0';
      end if;

      if(r_STATE = q_2) then
          o_QY <= '1';
      else
          o_QY <= '0';
      end if;

      if(r_STATE = q_1) then
          o_QZ <= '1';
      else
          o_QZ <= '0';
      end if;

 end process;
end arch_1;
```

Circuito 1: Tabela verdade

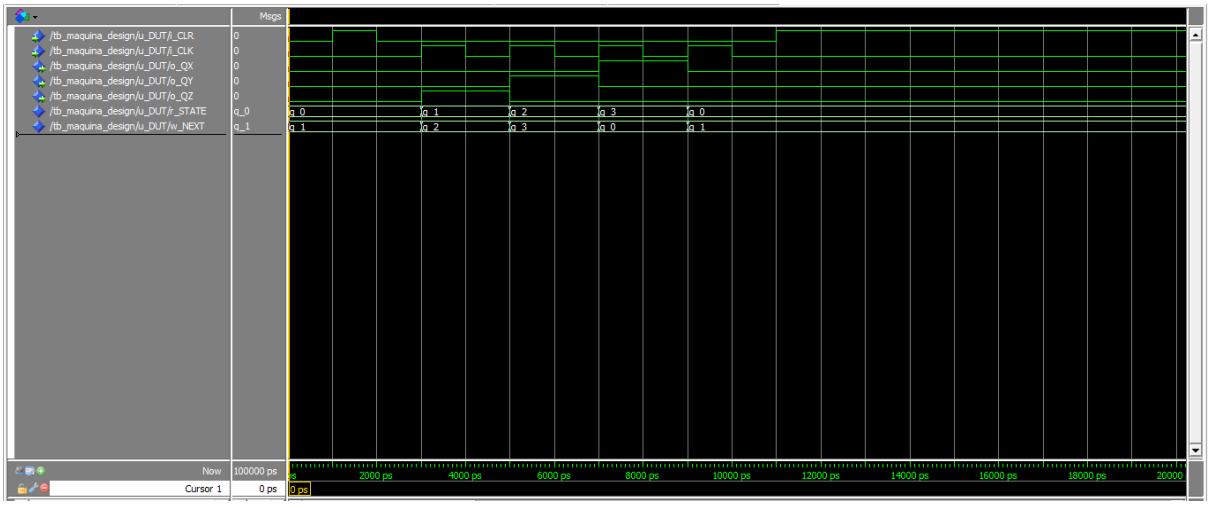
Tabela verdade - 1° FSM

CLOCK	ENTRADA D2	ENTRADA D1	ENTRADA D0	SAÍDA Q2	SAÍDA Q1	SAÍDA Q0
0	X	X	X	X	X	X
1	0	0	1	0	0	0
1	0	1	0	0	0	1
1	1	0	0	0	1	0
1	0	0	0	1	0	0

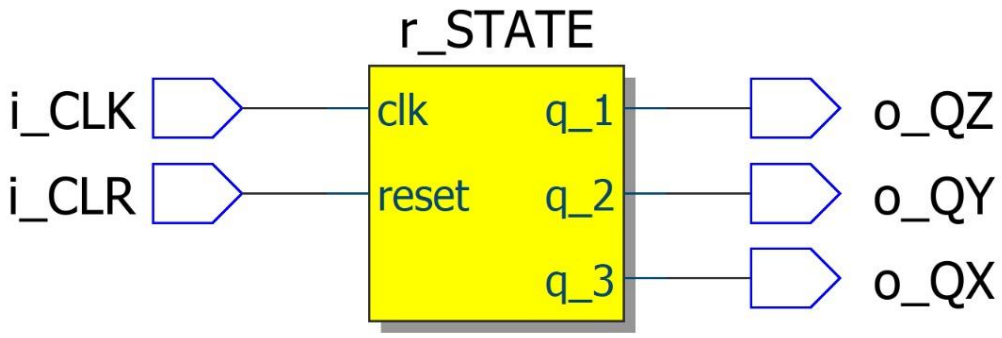
Projeto de funcionamento esperado (logisim):



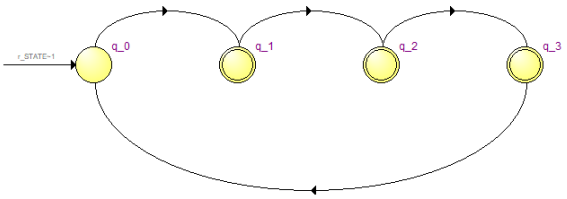
Resultado: Diagrama de ondas



Resultado: QUARTUS



FSM da arquitetura no QUARTUS



Discussão dos resultados:

O circuito apresentado anteriormente neste exercício tem uma peculiaridade: o seu Flip-Flop “não contém entrada”. Obviamente, para o funcionamento de um circuito deste tipo é necessário entradas, mas neste caso, as entradas são combinações das próprias saídas dos flip-flops, fazendo que ele dependa unicamente de si mesmo para funcionar. Além disso, foi adicionado um sinal de ***clock*** para que houvesse o funcionamento do flip-flop, além do **clear** para limpar as entradas quando selecionado.

Para que o circuito não necessitasse de entradas e seu funcionamento ser infinito (mas com estados finitos), foi atribuído um bloco de controle que armazena as coordenadas do presente e realiza lógicas combinacionais para as saídas do futuro.

Após ser realizada a lógica de controle/saída e a lógica de implementação do flip-flop, foi feita a lógica para a mudança de estados da máquina, como sugerido pelo enunciado (one-hot porém com **000** como estado inicial). Após todas as saídas

possíveis serem apresentadas, o circuito volta ao seu estado inicial fazendo com que reinicie todo o esquema.

O circuito funciona da seguinte maneira: a cada período de clock (ida e volta), através das lógicas combinacionais adotadas por portas **AND**, as saídas se movem de acordo com o que apresenta após todo o processo de transformação. O armazenamento e transformação em um período de clock no presente resultarão nas saídas no período de clock futuro a essa.

Circuito 2: Máquina de estados com uma entrada e três saídas

Enunciado: Implemente a máquina de estados especificada no Exercício 3.29 do livro texto (Sistemas Digitais, de Frank Vahid).

3.29 *Desenhe o diagrama de estados de uma FSM que tem uma entrada gent e três saídas, x, y e z. As saídas xyz geram uma sequência chamada "código gray" em que exatamente uma das três saídas muda de 0 para 1 ou de 1 para 0. A sequência em código Gray que a FSM deve produzir é 000, 010, 011, 001, 101, 111, 110, 100, voltando a se repetir. A saída deve se mudar apenas na borda de subida do relógio quando gent = 1. Faça 000 ser o estado inicial.*

Circuito 2 - Testbench:

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

entity tb_fsm_gray is
--empty
end tb_fsm_gray;

architecture arch_1 of tb_fsm_gray is

component fsm_gray is
port(     i_CLR : in std_logic; -- clear/reset
          i_CLK : in std_logic; -- clock
        i_GCNT : in std_logic;
        o_X : out std_logic;
        o_Y : out std_logic;
        o_Z : out std_logic);
```

```
end component;

signal w_CLR, w_CLK, w_GCNT, w_X, w_Y, w_Z : std_logic;
begin

     --connect DUT
    u_DUT : fsm_gray port map(i_CLR => w_CLR,
                              i_CLK => w_CLK,
                              i_GCNT => w_GCNT,
                              o_X => w_X,
                              o_Y => w_Y,
                              o_Z => w_Z);

process
     begin
    --reset
    w_CLR <= '0';
    w_CLK <= '0';
    w_GCNT <= '0';
    wait for 1ns;
    assert(w_CLR <='0' and w_CLK <='0') report "Fail @ 00"
severity error;

    w_CLR <= '1';
    w_GCNT <= '0';
    wait for 1ns;

    w_CLR <= '0';
    w_GCNT <= '0';
    wait for 1ns;

    --clock and gcnt
    w_CLK <= '1';
    w_GCNT <= '0';
    wait for 1ns;

    w_CLK <= '0';
    w_GCNT <= '0';
    wait for 1ns;

    w_CLK <= '1';
    w_GCNT <= '0';
    wait for 1ns;
```

```
w_CLK <= '0';
w_GCNT <= '0';
wait for 1ns;

w_CLK <= '1';
w_GCNT <= '0';
wait for 1ns;

w_CLK <= '0';
w_GCNT <= '0';
wait for 1ns;

w_CLK <= '1';
w_GCNT <= '0';
wait for 1ns;

w_CLK <= '0';
w_GCNT <= '0';
wait for 1ns;

w_CLK <= '1';
w_GCNT <= '0';
wait for 1ns;

w_CLK <= '0';
w_GCNT <= '0';
wait for 1ns;

w_CLK <= '1';
w_GCNT <= '0';
wait for 1ns;

w_CLK <= '0';
w_GCNT <= '0';
wait for 1ns;

w_CLK <= '1';
w_GCNT <= '0';
wait for 1ns;

w_CLK <= '0';
w_GCNT <= '0';
```

```
    wait for 1ns;

    w_CLK <= '1';
    w_GCNT <= '0';
    wait for 1ns;

    w_CLK <= '0';
    w_GCNT <= '0';
    wait for 1ns;

    w_CLK <= '1';
    w_GCNT <= '1';
    wait for 1ns;

    w_CLK <= '0';
    w_GCNT <= '1';
    wait for 1ns;

    w_CLK <= '1';
    w_GCNT <= '1';
    wait for 1ns;

    w_CLK <= '0';
    w_GCNT <= '1';
    wait for 1ns;

    w_CLK <= '1';
    w_GCNT <= '1';
    wait for 1ns;

    w_CLK <= '0';
    w_GCNT <= '1';
    wait for 1ns;

    w_CLK <= '1';
    w_GCNT <= '1';
    wait for 1ns;

    w_CLK <= '0';
    w_GCNT <= '1';
    wait for 1ns;

    w_CLK <= '1';
```

```
    w_GCNT <= '1';
    wait for 1ns;

    w_CLK <= '0';
    w_GCNT <= '1';
    wait for 1ns;

    w_CLK <= '1';
    w_GCNT <= '1';
    wait for 1ns;

    w_CLK <= '0';
    w_GCNT <= '1';
    wait for 1ns;

    w_CLK <= '1';
    w_GCNT <= '1';
    wait for 1ns;

    w_CLK <= '0';
    w_GCNT <= '1';
    wait for 1ns;

    w_CLK <= '1';
    w_GCNT <= '1';
    wait for 1ns;

    w_CLK <= '0';
    w_GCNT <= '1';
    wait for 1ns;

    --clear inputs
    w_CLR <= '1';
    wait;
     end process;
end arch_1;
```

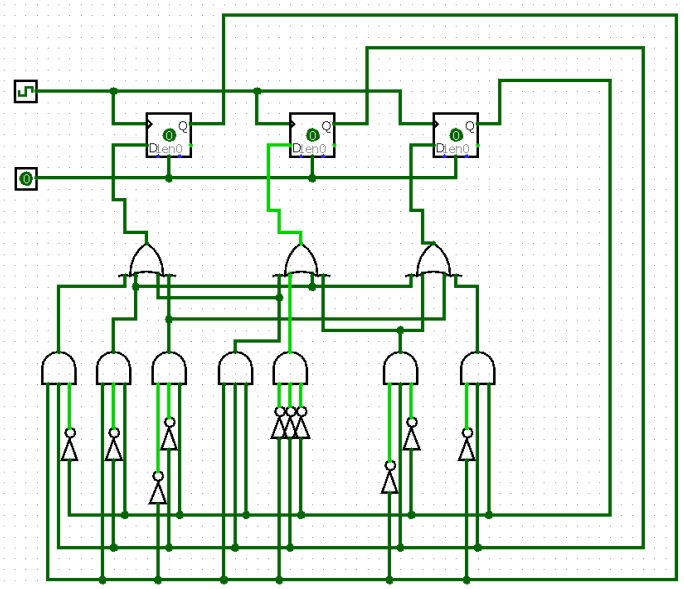
Circuito 2 - Design

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

```
entity fsm_gray is
port(     i_CLR : in std_logic; -- clear/reset
          i_CLK : in std_logic; -- clock
        i_GCNT : in std_logic;
        o_X : out std_logic;
        o_Y : out std_logic;
        o_Z : out std_logic);
end fsm_gray;

architecture arch_1 of fsm_gray is
     type t_STATE is (s_0, s_1,s_2,s_3,s_4,s_5,s_6,s_7); --
new FSM type
     signal r_STATE : t_STATE; -- state register
     signal w_NEXT : t_STATE; -- next state

begin

     -- state register
     p_STATE: process (i_CLR, i_CLK)
    begin
     if (i_CLR = '1') then
          r_STATE <= s_0; --inicial state
            elsif (rising_edge(i_CLK)) then
               r_STATE <= w_NEXT; -- next state
            end if;
    end process;

      --next state
    p_NEXT: process (r_STATE, i_GCNT)
    begin
     case (r_STATE) is
          when s_0 => if (i_GCNT = '1') then
                              w_NEXT <= s_1;
                        else
                         w_NEXT <= s_0;
                        end if;
          when s_1 => if (i_GCNT = '1') then
                              w_NEXT <= s_2;
                        else
                         w_NEXT <= s_1;
                        end if;
          when s_2 => if (i_GCNT = '1') then
                              w_NEXT <= s_3;
```

```
                        else
                         w_NEXT <= s_2;
                        end if;
          when s_3 => if (i_GCNT = '1') then
                              w_NEXT <= s_4;
                        else
                         w_NEXT <= s_3;
                        end if;
          when s_4 => if (i_GCNT = '1') then
                              w_NEXT <= s_5;
                        else
                         w_NEXT <= s_4;
                        end if;
          when s_5 => if (i_GCNT = '1') then
                              w_NEXT <= s_6;
                        else
                         w_NEXT <= s_5;
                        end if;
          when s_6 => if (i_GCNT = '1') then
                              w_NEXT <= s_7;
                        else
                         w_NEXT <= s_6;
                        end if;
               when others => w_NEXT <= s_0;
          end case;
     end process;
      -- output
    o_X <= '1' when (r_STATE = s_4 or r_STATE = s_5 or
r_STATE = s_6 or r_STATE = s_7) else '0';
      o_Y <= '1' when (r_STATE = s_1 or r_STATE = s_2 or
r_STATE = s_5 or r_STATE = s_6) else '0';
      o_Z <= '1' when (r_STATE = s_2 or r_STATE = s_3 or
r_STATE = s_4 or r_STATE = s_5) else '0';

end arch_1;
```

Projeto de funcionamento esperado (logisim):

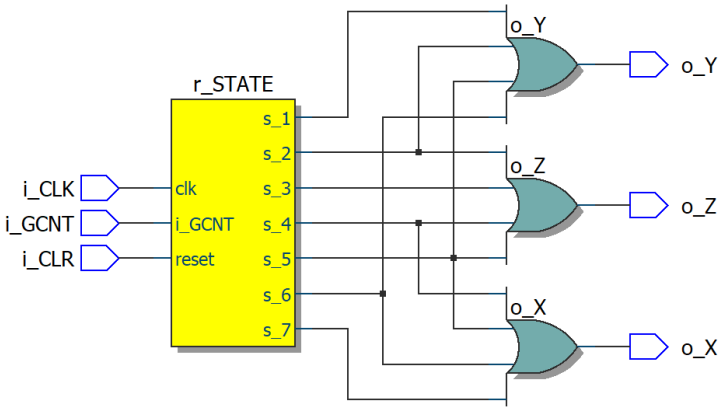


Circuito 2: Tabela verdade

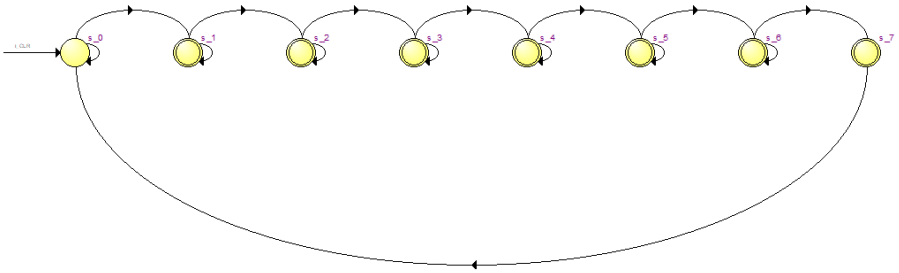
Tabela verdade - 2° FSM

CLOCK	GENT	ENTRADA D2	ENTRADA D1	ENTRADA D0	SAÍDA Q2	SAÍDA Q1	SAÍDA Q0
0	X	X	X	X	X	X	X
1	0	X	X	X	X	X	X
	1	0	1	0	0	0	0
		0	1	1	0	1	0
		0	0	1	0	1	1
		1	0	1	0	0	1
		1	1	1	1	0	1
		1	1	0	1	1	1
		1	0	0	1	1	0
		0	0	0	1	0	0

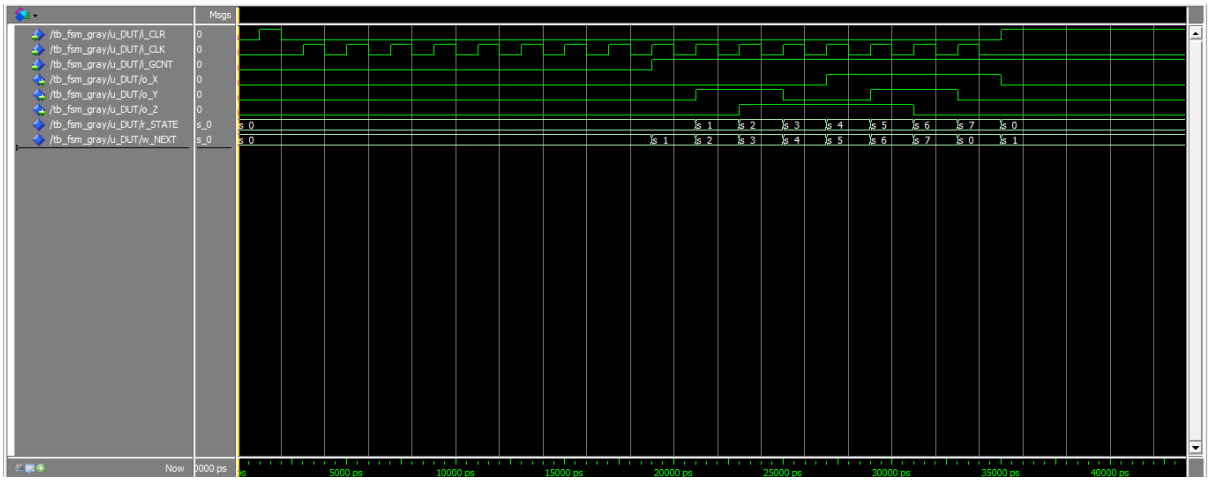
Resultado quartus:



FSM da arquitetura no QUARTUS



Resultado: diagrama de ondas



Discussão dos resultados:

O circuito 2 tem uma pequena diferença para o primeiro, onde suas saídas dependem diretamente da implementação do bloco de controle, mas além disso, há uma combinação das saídas do bloco para as saídas do circuito. Os sinais **o_X, o_Y** e **o_Z**, com perfil direcional de saídas (**OUT**) são apenas resultado da entrada (depende do estado anterior) e da única entrada deste circuito: a **i_gcnt**.

Neste circuito foi adicionada uma entrada denominada ***gcnt*** que integra no circuito fazendo um papel do *enable*: o funcionamento do flip-flop se dá apenas quando o ***gcnt*** está ativado. Caso desativado, o sinal do clock não mudará a saída do circuito até que ***gcnt*** seja positivo novamente.

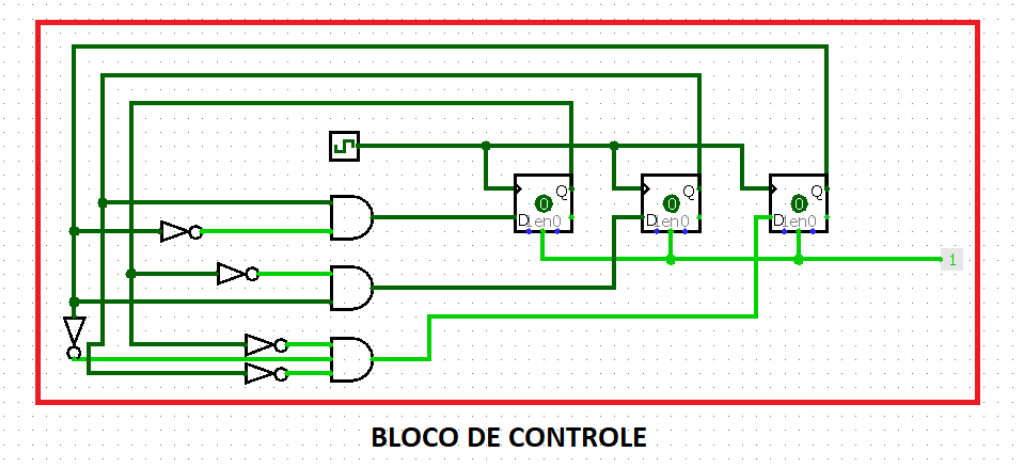
O segundo circuito tem 8 estados (000, 010, 011, 001, 101, 111, 110, 100), então foram adicionadas 7 portas AND para que cada estado tenha sua saída executada sem erros (000 é saída, mas é o estado inicial). O circuito também funciona na borda de subida do clock e com o ***gcnt*** ativado.

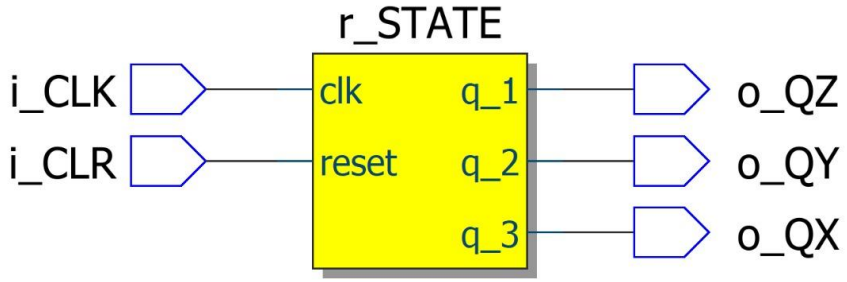
Foi observado que as saídas dependem unicamente das lógicas dentro do bloco de controle e suas lógicas estabelecidas. As saídas mostradas no presente são espelhos das combinações apresentadas em um clock passado. Quando o circuito estabelece seu último estado, 000 volta a ser ejetado novamente. Após a implementação do circuito, foi dado o veredito do funcionamento correto.

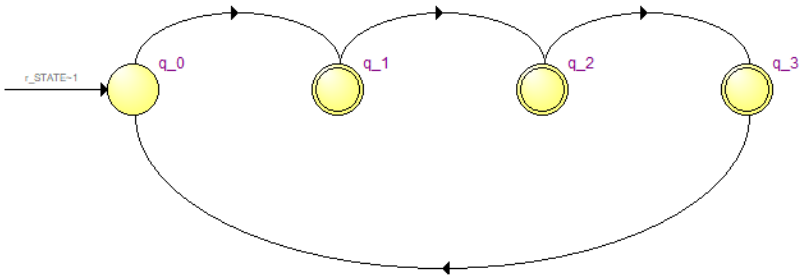
Circuito 3:

Enunciado: Implemente o bloco de controle especificado no Exercício 3.40 do livro texto (Sistemas Digitais, de Frank Vahid). Note que, neste projeto, as funções de transição de estado e de saída devem ser obtidas com o uso do processo de projeto de bloco de controle em cinco passos apresentado na p. 136 (Seção 3.4 do livro texto) e ilustrado no Exemplo 3.7.

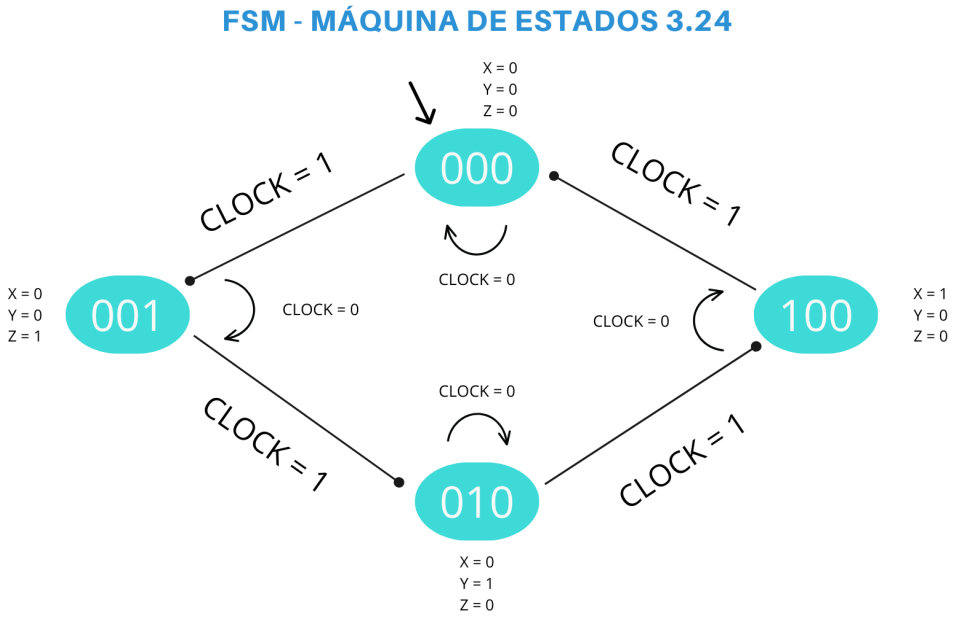
3.40 *Usando o processo de cinco passos para se projetar um bloco de controle, converta a FSM que você criou no exercício 3.24 em um bloco de controle. Implemente-o usando um registrador de estado e portas lógicas.*



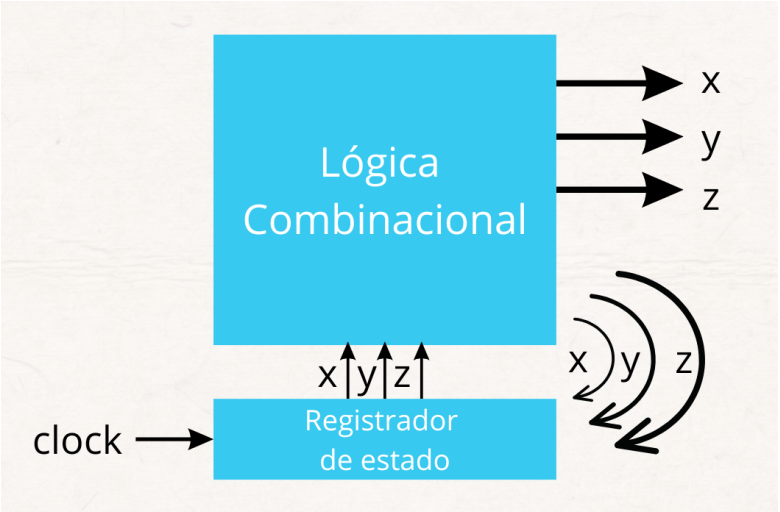




Passo 1 - CRIAR O FSM



Passo 2 - ARQUITETURA



Passo 3 - ATRIBUIÇÃO DE BINÁRIOS

Estado 1 = 00;
Estado 2 = 01;
Estado 3 = 10;
Estado 4 = 11.

Passo 4 - TABELA VERDADE

Tabela verdade - 1° FSM

CLOCK	ENTRADA D2	ENTRADA D1	ENTRADA D0	SAÍDA Q2	SAÍDA Q1	SAÍDA Q0
0	X	X	X	X	X	X
1	0	0	1	0	0	0
1	0	1	0	0	0	1
1	1	0	0	0	1	0
1	0	0	0	1	0	0

Passo 5 - LÓGICA COMBINACIONAL

X = YZ’
Y = X’Z

$Z = X'Y'Z'$
