Experimento 8:
MÁQUINAS DE ESTADO EM VHDL
(opcional, não vale nota)

1 DESCRIÇÃO DO PROBLEMA

Considere uma fechadura digital. Para destrancar a fechadura, deve-se entrar com o código correto na ordem correta. O código que destranca a fechadura é o código 011 – 110 – 101. A qualquer momento, um código errado leva a fechadura para o estado inicial.

2 FLIP-FLOPS EM VHDL

2.1 Flip-Flop D

Considere um simples flip-flop D, conforme visto abaixo:

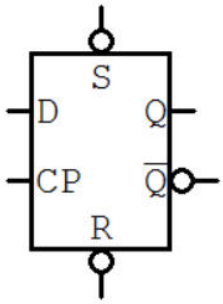

Figura 1 - Flip-Flop D

O funcionamento desse flip-flop é simples: na subida do clock (transição de 0 para 1), o conteúdo da entrada D é copiado para a saída Q. A saída $\overline{Q}$ é sempre o inverso de Q. As entradas RESET (ou CLEAR) e SET (ou PRESET) são assíncronas (isto é, independem do clock), e levam o flip-flop para o estado Q = 1 (com PRESET = 0) ou Q = 0 (com CLEAR = 0). As entradas PRESET e CLEAR não devem ser ativas ao mesmo tempo. Podemos implementar esse flip-flop com o seguinte circuito:

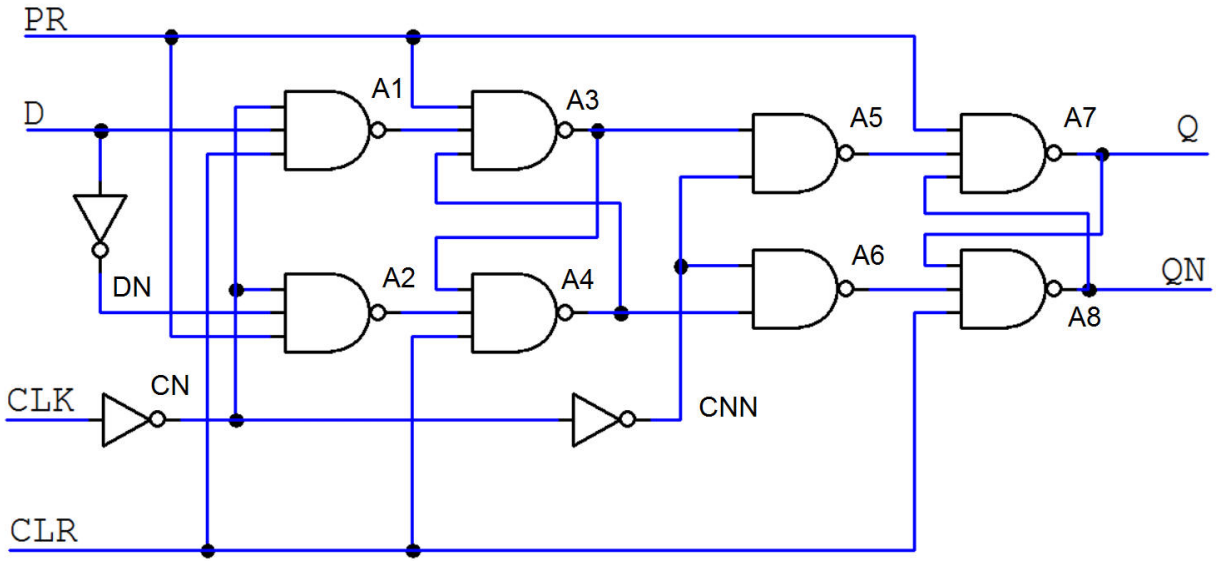

Figura 2 - Circuito do Flip-Flop D gatilhado

A descrição da entidade desse flip-flop é simples:

```
entity flipflop_d is
  port (D, CLK, CLR, PRESET : in  STD_LOGIC;
                      Q, QN : out STD_LOGIC);
end flipflop_d;
```

Porém, a implementação da arquitetura dessa entidade pode ser feita de formas diferentes. A primeira, mais óbvia, é chamada de *dataflow* (fluxo de dados), e consiste em realmente implementar o fluxo de dados que ocorre no circuito. Para isso, podemos usar sinais intermediários para representar os dados em cada etapa (no exemplo, em cada porta lógica). Uma implementação dessa arquitetura usando essa abordagem é vista abaixo:

```
architecture flipflop_d_dataflow of flipflop_d is

signal DN,CN,CNN,A1,A2,A3,A4,A5,A6,A7,A8 : STD_LOGIC;

begin

DN <= not(D);
CN <= not(CLK);

A1 <= not(CN and D and CLR);
A2 <= not(CN and DN and PRESET);

A3 <= not(PRESET and A1 and A4);
A4 <= not(CLR and A2 and A3);

CNN <= not(CN);

A5 <= not(A3 and CNN);
A6 <= not(A4 and CNN);

A7 <= not(PRESET and A5 and A8);
A8 <= not(CLR and A7 and A6);

Q  <= A7;
QN <= A8;

end flipflop_d_dataflow;
```

As vantagens da abordagem de dataflow é que ela me diz exatamente o que acontece em cada ponto do circuito, e um engenheiro eletrônico consegue facilmente reproduzir o circuito da Figura 2 a partir do código da arquitetura (isto é, o código da arquitetura está realmente descrevendo o hardware). Porém, o VHDL nos oferece uma abordagem mais simples para resolver o circuito: a abordagem comportamental (*behavioral*). Nessa abordagem, tentamos descrever *o que* o elemento faz, ao invés de descrever *como* ele atinge esse objetivo. Para isso, usamos um elemento do VHDL chamado *processo*. A sintaxe do processo é:

```
label : process (sensitivity_list)
        begin
           (sequential statements)
        end process label;
```

Um processo é concorrente (isto é, ocorre ao mesmo tempo) de outras declarações (inclusive outros processos). Porém, dentro de um processo, as declarações são executadas *sequencialmente*. Porém, um processo é executado apenas quando ocorre uma mudança em um dos elementos de sua lista de sensibilidade (*sensitivity list*). Veja abaixo a implementação do Flip-Flop D utilizando a abordagem comportamental.

```
architecture flipflop_d_behavioral of flipflop_d is
begin
dff: process (CLK,CLR,PRESET)
     begin
         if (CLR = '0') then
             Q  <= '0';
            QN <= '1';
        elsif (PRESET = '0') then
             Q  <= '1';
            QN <= '0';
         elsif (rising_edge(CLK)) then
             Q  <= D;
             QN <= not(D);
         end if;
    end process dff;
end flipflop_d_behavioral;
```

No código acima, a lista de sensibilidade do processo tem três elementos: CLK, CLR e PRESET. Qualquer mudança em qualquer um desses elementos causará a execução do processo – uma vez em execução, o processo é executado sempre sequencialmente. Depois disso, ele primeiro verifica se a entrada CLR está ativa (ou seja, em nível 0) – caso positivo, ele muda as saídas de acordo. Caso contrário, ele verifica se a entrada PRESET está ativa (em nível 0) – caso positivo, ele muda as saídas de acordo. Finalmente, caso a mudança ocorrida foi uma mudança de 0 para 1 na entrada CLK (descrita pela função rising_edge()), ele copia o valor de D para a saída Q.

Note que ambas as arquiteturas, *flipflop_d_behavioral* e *flipflop_d_dataflow*, descrevem o mesmo elemento! Porém, não sabemos como ficará o circuito descrito pela arquitetura comportamental – isso é trabalho do sintetizador que utilizaremos (no nosso caso, o Xilinx ISE). Ao contrário, a arquitetura comportamental descreve exatamente o comportamento do circuito do flip-flop D, sem se preocupar com a implementação em portas lógicas.

2.2 Projeto de um Contador de 2 bits

Considere um contador de 2 bits, com o diagrama de estados dado pela figura.

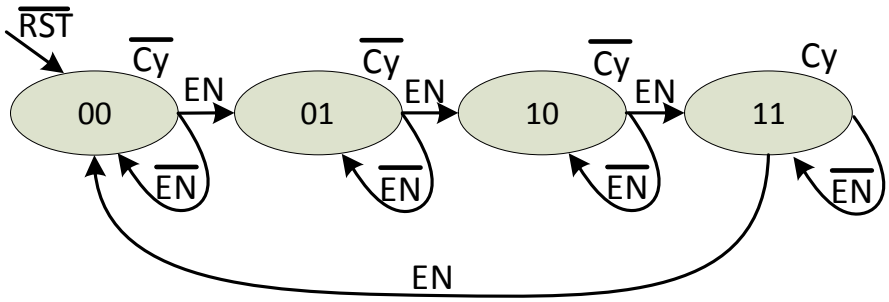

Figura 3 - Diagrama de Estados do Contador de 2 bits

Podemos definir:

ENTRADAS:
C – Clock. – Sincroniza a mudança de estados.
EN – Enable – Habilita a mudança de estados.
RST – RESET – Leva a máquina para o estado 00.

SAÍDAS:
Cy – Indica o carry do contador.

Com base no diagrama de estados, podemos montar a tabela de estados e saída do circuito.

Tabela 1- Tabela de Estados e Saída

Estado	Estado	Próximo Estado se EN = 0	Próximo Estado se EN = 1	Cy
E_0	00	00	01	0
E_1	01	01	10	0
E_2	10	10	11	0
E_3	11	11	00	1

Supondo agora o uso de flip-flops tipo D, podemos agora montar os mapas de Karnaugh para as variáveis de entrada desses flip-flops, D_1 e D_0, onde D_1 é a entrada do flip-flop que gera Q_1 e D_0 a entrada do flip-flop que gera Q_0.

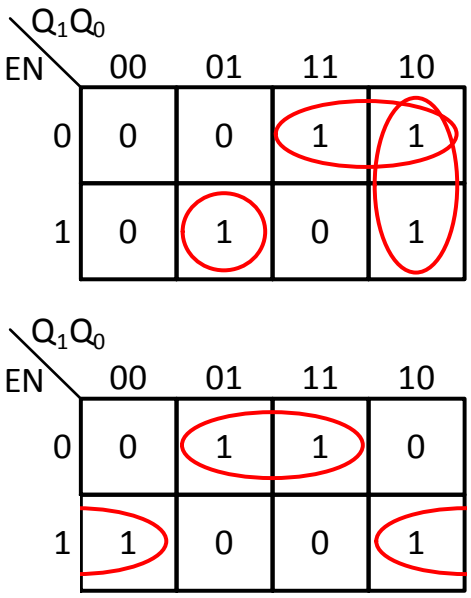

Figura 4 - Mapas de Karnaugh para o flip-flop D

Minimizando os termos, temos:

$$D_1 = Q_1\overline{Q_0} + Q_1\overline{EN} + EN\,\overline{Q_1}Q_0$$

$$D_0 = \overline{EN}Q_0 + EN\overline{Q_0}$$

Equação 1 - Equações para o Flip-Flop D

É fácil observar que as saídas são função apenas dos estado da máquina, não das entradas (isto é, a máquina é uma máquina do tipo Moore). Portanto:

$$Cy = Q_1Q_0$$

Equação 2 - Equações de Saída

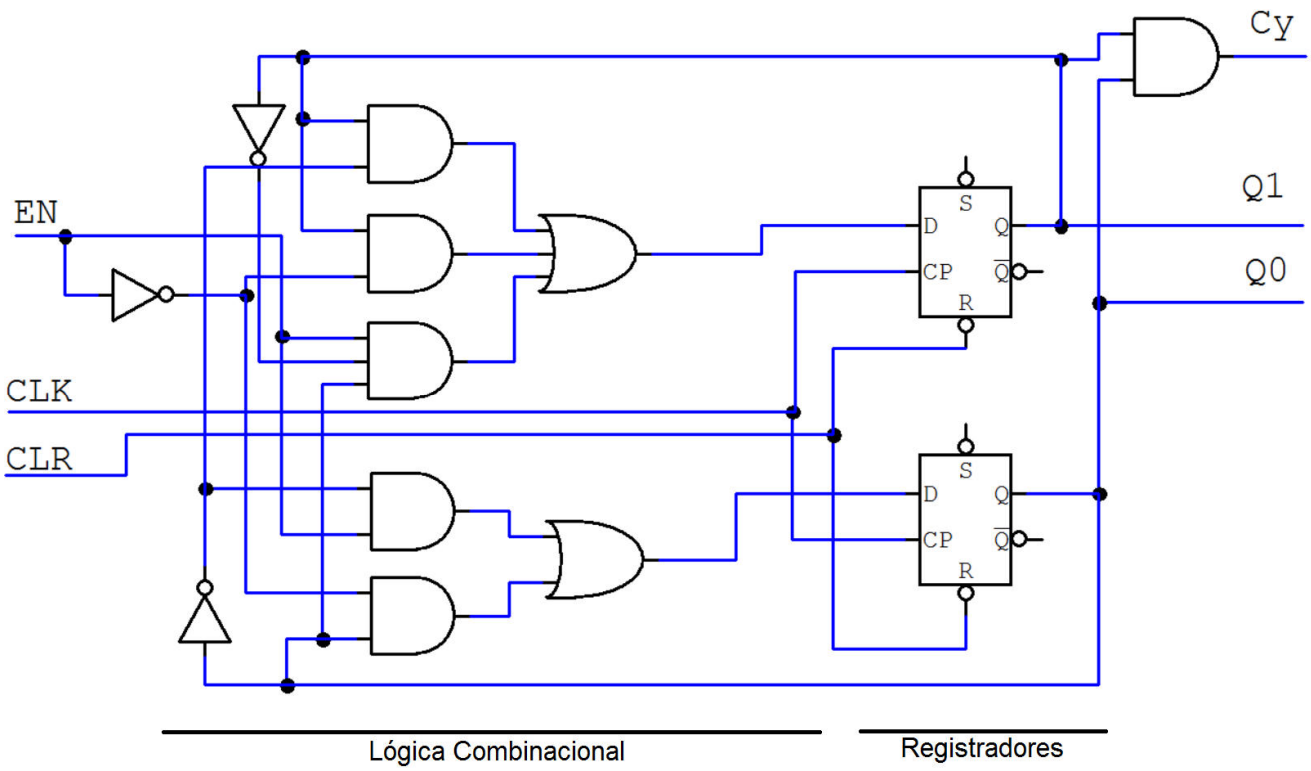

Figura 5 - Contador de 2 bits

2.3 Implementação em VHDL

Para implementar esse contador em VHDL, primeiro criamos a entidade, que descreve as entradas e saídas:

```
entity contador_2bits is
  port (EN, CLK, CLR : in STD_LOGIC;
                   Q : out STD_LOGIC_VECTOR(1 downto 0);
                  Cy : out STD_LOGIC);
end contador_2bits;
```

Em seguida, tempos três opções para descrever a arquitetura. A primeira é a arquitetura dataflow, que segue o fluxo de dados. A segunda é a arquitetura RTL (*register transfer logic*) e a última a arquitetura comportamental.

2.4 Arquitetura Dataflow

Novamente, um engenheiro consegue "enxergar" como esse circuito será montado. A descrição segue exatamente as equações da máquina de estado, vemos que o estado é guardado em 2 bits, cada um representado por um flip-flop D (inclusive com a representação escolhida, pelo port map). Note que esta é exatamente a descrição do circuito mostrado na Figura 5.

```
architecture contador_2bits_dataflow_op of contador_2bits is
component flipflop_d is
  port (D, CLK, CLR : in  STD_LOGIC;
              Q, QN : out STD_LOGIC);
end component;

signal D1,D0 : STD_LOGIC;
signal Q1,Q0,Q1N,Q0N : STD_LOGIC;

begin

D1 <= (Q1 and Q0N) or (Q1 and not(EN)) or (EN and Q1N and Q0);
D0 <= (not(EN) and Q0) or (EN and Q0N);

U1: flipflop_d port map(D0,CLK,CLR,Q0,Q0N);
U2: flipflop_d port map(D1,CLK,CLR,Q1,Q1N);

Q <= Q1 & Q0;

Cy <= (Q1 and Q0);

end contador_2bits_dataflow_op;
```

2.5 Arquitetura RTL

Porém, temos outras formas de descrever esse circuito. Observe que a implementação do circuito tem duas áreas bem definidas: lógica combinacional e registradores. Os registradores são responsáveis pela memória do circuito, e a lógica combinacional decide a transição da memória. Esse modelo é chamado de *register transfer logic* , ou RTL.

Vários novos conceitos são explorados. Primeiramente, criamos um tipo especial de variável chamada *estado* (através do comando *type*) para guardar os estados da nossa máquina. Essa variável tem apenas quatro valores possíveis: (ST0,ST1,ST2,ST3) . Note que, a princípio, não se sabe como esses valores serão representados pelo sintetizador – apenas sabemos que são quatro valores distintos.

Em seguida, criamos dois processos: *comb_proc* e *sync_proc* (respectivamente, combinacional e sequencial). O primeiro gera a lógica de próximo estado (NS, no código), enquanto o segundo trata de sincronizar esse estado com o estado atual (PS, no código). Note que a lógica combinacional dentro de *comb_proc* é realizada utilizando uma estrutura *switch-case* – dependendo do estado atual, assinalamos valores ao próximo estado e as saídas. Note que, embora não seja possível "enxergar" o circuito pelo exemplo do código RTL, conseguimos ver claramente a separação em camadas – isto é, a lógica combinacional e a lógica dos registradores.

```
architecture contador_2bits_rtl_op of contador_2bits is

type estado is (ST0,ST1,ST2,ST3);
signal PS,NS : estado; -- PS: estado atual, NS: próximo estado

begin
 sync_proc: process(CLK,CLR)   -- processo síncrono
 begin
   if (CLR = '0') then
     PS <= ST0;
   elsif (rising_edge(CLK)) then
     PS <= NS;
   end if;
 end process sync_proc;

 comb_proc: process(PS,EN)    -- processo combinacional
 begin
   case PS is
     when ST0 =>
       Cy <= '0';
       Q <= "00";
       if (EN = '1') then
         NS <= ST1;
       else
         NS <= ST0;
       end if;
     when ST1 =>
       Cy <= '0';
       Q <= "01";
       if (EN = '1') then
         NS <= ST2;
       else
         NS <= ST1;
       end if;
     when ST2 =>
       Cy <= '0';
       Q <= "10";
       if (EN = '1') then
         NS <= ST3;
       else
         NS <= ST2;
       end if;
     when ST3 =>
       Cy <= '1';
       Q <= "11";
       if (EN = '1') then
         NS <= ST0;
       else
         NS <= ST3;
       end if;
     when others =>
       Cy <= '0';
       Q <= "00";
       NS <= ST0;
   end case;
 end process comb_proc;

 end contador_2bits_rtl_op;
```

2.6 Arquitetura Comportamental

Nesse último exemplo, uma arquitetura totalmente comportamental foi realizada. Note que temos um único processo, *behavioral_proc*, que responde à mudanças no CLK e CLR. Como um contador é um circuito simples, o próximo estado é sempre o incremento do estado atual, condicionado à entrada EN. Note que, nesse exemplo, não apenas não conseguimos "enxergar" o circuito, como nem mesmo sabemos onde estão os registradores e a sua relação com a lógica combinacional. Apenas modelamos o comportamento do circuito que queremos, e deixamos o sintetizador trabalhar.

```
architecture contador_2bits_behavioral_op of contador_2bits is

signal S : STD_LOGIC_VECTOR(1 downto 0);

begin

 Cy <= '1' when (S = "11") else '0';
 Q <= S;

 behavioral_proc: process(CLK,CLR)
 begin
   if (CLR = '0') then
     S <= "00";
   elsif (rising_edge(CLK) and (EN = '1')) then
     S <= S + '1';
   end if;
 end process behavioral_proc;

 end contador_2bits_behavioral_op;
```

2.7 Implementando o CLK na placa Xilinx Basys 2

Na implementação dessa máquina de estados na placa FPGA, podemos colocar o CLK como sendo um dos botões de uma posição da placa (por exemplo, o botão 0, no pino G12). Porém, para isso, temos que lembrar de adicionar o comando para desabilitar uma rota dedicada para o clock, como mostrado abaixo.

```
NET "CLK" LOC = "G12";  #Botão 0
NET "CLK" CLOCK_DEDICATED_ROUTE = FALSE;
```

3 EXPERIMENTO

O experimento consiste em montar uma fechadura digital, descrita pelo seguinte diagrama de estados:

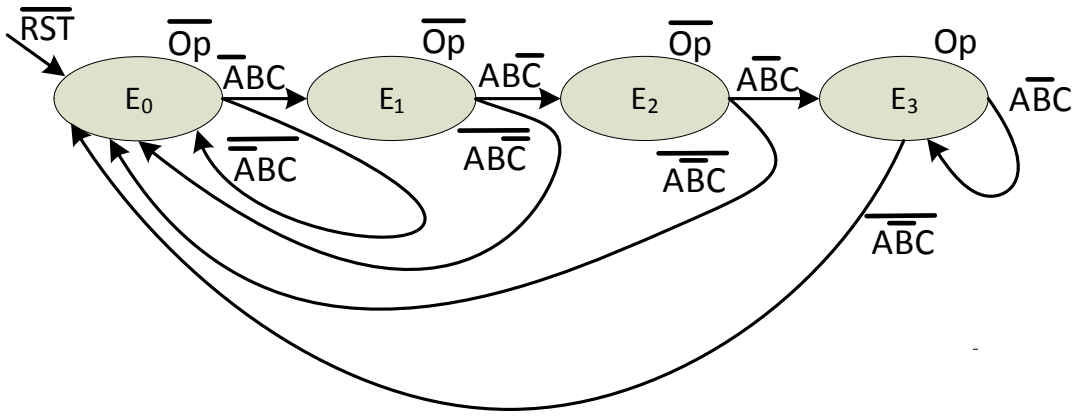

Figura 6 - Diagrama de Estados da Fechadura Digital

O circuito contém 5 entradas:

ENTRADAS:
C – Clock. – Sincroniza a mudança de estados.
ABC – Chaves – Esta é a combinação da fechadura
RST – RESET – Leva a máquina para o estado 00.

SAÍDAS:
Op – Indica se a fechadura está aberta ou fechada.

Em outras palavras, o usuário deve colocar a combinação da fechadura para aquele estado e apertar o clock. Se a combinação for correta, a fechadura vai para o próximo estado (ou fica no mesmo estado, se o último estado estiver sido atingido). Se a combinação for incorreta, a fechadura volta para o estado inicial. A combinação para abrir a fechadura é: 011 – 110 – 101 .

3.1 Pré-Relatório

O pré-relatório consiste no projeto da máquina de estados para implementação em circuito, utilizando flip-flop D. Apresente a tabela de transição de estados, mapas de Karnaugh, equações (em especial as equações de entrada para cada flip-flop) e o esquemático do circuito.

3.2 Vistos do Experimento

O experimento consiste em dois vistos:
1) Implementar e testar a fechadura no FPGA (placa Basys 2, da Xilinx) usando arquitetura de *dataflow*.
2) Implementar e testar a fechadura no FPGA usando arquitetura RTL.