```
ENTITY fibonacci IS
       PORT (SIGNAL write_enable_i      : IN STD_LOGIC;
             SIGNAL data_in             : IN NATURAL;
             SIGNAL read_enable_i  : IN STD_LOGIC;
             SIGNAL data_out            : OUT NATURAL;
             SIGNAL status_o            : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
             SIGNAL irq_o        : OUT STD_LOGIC);
END fibonacci;

       PROCESS

             VARIABLE     n_anterior1: NATURAL :=0;
             VARIABLE     n_anterior2: NATURAL :=0;
             VARIABLE     n_fibonacci: NATURAL :=0;
             VARIABLE     n_max        : NATURAL :=0;

       BEGIN

                           status_o <= "00";
                           irq_o <= '0';
                           data_out <= 0;

                           WAIT FOR 1*PERIOD;

                           WHILE(write_enable_i/='1') LOOP
                             WAIT FOR 1*PERIOD;
                           END LOOP;

                           n_anterior1 :=1;
                           n_anterior2 :=0;
                           n_max :=data_in;
                           status_o <= "01";
                           WAIT FOR 1*PERIOD;

                           IF (n_max =0) or (n_max=1) THEN
                                  n_fibonacci :=n_max;
                           ELSE
                                  WHILE (n_max /= 1) LOOP
                                  n_fibonacci :=n_anterior1+n_anterior2;
                                  n_anterior2 :=n_anterior1;
                                  n_anterior1 :=n_fibonacci;
                                  n_max :=n_max-1;
                                  END LOOP;
                           END IF;

                           data_out <= n_fibonacci;
                           irq_o  <= '1';
                           status_o <= "10";
                           WAIT FOR 1*PERIOD;

                           WHILE (read_enable_i/='1') LOOP
                              WAIT FOR 1*PERIOD;
                           END LOOP;

                           WAIT FOR 1*PERIOD;

       END PROCESS;

END behavior_w;
```

1) Separando (identificando) no código original:

- Linhas de código irrelevantes para a FSMD (identificar como XXX)
- Portos do *datapath*
- Portos da parte de controle (FSM)
- Linhas referentes à operações no DP
- Linhas relativas à operações na PC

```
XXX     ENTITY Fibonacci IS
XXX_PC - write_enable_i              : IN STD_LOGIC;
XXX_DP - SIGNAL data_in              : IN NATURAL;
XXX_PC - SIGNAL read_enable_i        : IN STD_LOGIC;
XXX_DP - SIGNAL data_out             : OUT NATURAL;
XXX_PC - SIGNAL status_o             : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
XXX_PC - SIGNAL irq_o         : OUT STD_LOGIC);
XXX     END fibonacci;

XXX     PROCESS
DP -           VARIABLE       n_anterior1    : NATURAL :=0;
DP -           VARIABLE       n_anterior2    : NATURAL :=0;
DP -           VARIABLE       n_fibonacci    : NATURAL :=0;
DP -           VARIABLE       n_max          : NATURAL :=0;

XXX     BEGIN
PC -                   status_o <= "00";
PC -                   irq_o <= '0';
DP -                   data_out <= 0;

XXX                            WAIT FOR 1*PERIOD;
PC -                           WHILE(write_enable_i/='1') LOOP

XXX                            WAIT FOR 1*PERIOD;
PC -                           END LOOP;

DP -                           n_anterior1 :=1;
DP -                           n_anterior2 :=0;
DP -                           n_max :=data_in;
PC -                           status_o <= "01";
XXX                            WAIT FOR 1*PERIOD;

DP -                           IF (n_max =0) or (n_max=1) THEN
DP -                                  n_fibonacci :=n_max;
DP -                           ELSE
DP -                                  WHILE (n_max /= 1) LOOP
DP -                                  n_fibonacci :=n_anterior1+n_anterior2;
DP -                                  n_anterior2 :=n_anterior1;
DP -                                  n_anterior1 :=n_fibonacci;
DP -                                  n_max :=n_max-1;
DP -                                  END LOOP;
DP -                           END IF;

DP -                           data_out <= n_fibonacci;
PC -                           irq_o  <= '1';
PC -                           status_o <= "10";
XXX                            WAIT FOR 1*PERIOD;

PC -                           WHILE (read_enable_i/='1') LOOP
XXX                               WAIT FOR 1*PERIOD;
PC -                           END LOOP;

XXX                            WAIT FOR 1*PERIOD;
XXX     END PROCESS;
XXX     END behavior_w;
```

2) Eliminando linhas XXX

```
DP -         VARIABLE     n_anterior1   : NATURAL :=0;
DP -         VARIABLE     n_anterior2   : NATURAL :=0;
DP -         VARIABLE     n_fibonacci   : NATURAL :=0;
DP -         VARIABLE     n_max         : NATURAL :=0;


PC -               status_o <= "00";
PC -               irq_o <= '0';
DP -               data_out <= 0;


PC -                     WHILE(write_enable_i/='1') LOOP


PC -                     END LOOP;

DP -                     n_anterior1 :=1;
DP -                     n_anterior2 :=0;
DP -                     n_max :=data_in;
PC -                     status_o <= "01";


DP -                     IF (n_max =0) or (n_max=1) THEN
DP -                           n_fibonacci :=n_max;
DP -                     ELSE
DP -                           WHILE (n_max /= 1) LOOP
DP -                           n_fibonacci :=n_anterior1+n_anterior2;
DP -                           n_anterior2 :=n_anterior1;
DP -                           n_anterior1 :=n_fibonacci;
DP -                           n_max :=n_max-1;
DP -                           END LOOP;
DP -                     END IF;

DP -                     data_out <= n_fibonacci;
PC -                     irq_o  <= '1';
PC -                     status_o <= "10";


PC -                     WHILE (read_enable_i/='1') LOOP

PC -                     END LOOP;
```

3) Identificando e eliminando linhas relativas a valores iniciais nas entradas (RESET) e saídas da PC

```
**RESET DP -           VARIABLE       n_anterior1    : NATURAL :=0;
**RESET        DP -           VARIABLE       n_anterior2    : NATURAL :=0;
**RESET        DP -           VARIABLE       n_fibonacci    : NATURAL :=0;
**RESET        DP -           VARIABLE       n_max          : NATURAL :=0;

**INICIAL PC -                status_o <= "00";
**INICIAL PC -                irq_o <= '0';

DP -                          data_out <= 0;

PC -                          WHILE(write_enable_i/='1') LOOP
PC -                          END LOOP;

DP -                          n_anterior1 :=1;
DP -                          n_anterior2 :=0;
DP -                          n_max :=data_in;

PC -                  status_o <= "01";

DP -                          IF (n_max =0) or (n_max=1) THEN
DP -                                 n_fibonacci :=n_max;
DP -                          ELSE
DP -                                 WHILE (n_max /= 1) LOOP
DP -                                 n_fibonacci :=n_anterior1+n_anterior2;
DP -                                 n_anterior2 :=n_anterior1;
DP -                                 n_anterior1 :=n_fibonacci;
DP -                                 n_max :=n_max-1;
DP -                                 END LOOP;
DP -                          END IF;
DP -                          data_out <= n_fibonacci;

PC -                  irq_o   <= '1';
PC -                  status_o <= "10";
PC -                  WHILE (read_enable_i/='1') LOOP
PC -                  END LOOP;
```

4) Identificando os estados da FSMD:
Separando e enumerando as linhas do DP que contém operações.
Adotar os padrões do livro texto (enumerar apenas o IF e o WHILE, não numerando o final destas condições)

```
DP -   1                    data_out <= 0;

PC -                  WHILE(write_enable_i/='1') LOOP
PC -                  END LOOP;

DP -   2                    n_anterior1 :=1;
DP -   3                    n_anterior2 :=0;
DP -   4                    n_max :=data_in;

PC -                  status_o <= "01";

DP -   5                    IF (n_max =0) or (n_max=1) THEN
DP -   6                    n_fibonacci :=n_max;
DP -                        ELSE
DP -   7                    WHILE (n_max /= 1) LOOP
DP -   8                    n_fibonacci :=n_anterior1+n_anterior2;
DP -   9                    n_anterior2 :=n_anterior1;
DP -   10                   n_anterior1 :=n_fibonacci;
DP -   11                   n_max :=n_max-1;
DP -                        END LOOP;
DP -                        END IF;
DP -   12                   data_out <= n_fibonacci;

PC -                  irq_o   <= '1';
PC -                  status_o <= "10";
PC -                  WHILE (read_enable_i/='1') LOOP
PC -                  END LOOP;
```

5) Expandindo a linha 5 (que apresenta uma operação complexa)

A linha 5 apresenta 2 comparações, (n_max com 0) e o mesmo (n_max com 1) e uma decisão que depende da operação OU entre os resultados das duas comparações.

```
DP -   5      IF (n_max =0) or (n_max=1) THEN
```

Esta operação complexa pode ser expandida da seguinte forma a fim de se tornar equivalente aos padrões do livro:

```
DP -   5      IF (n_max = 0)       -----  comparação de n_max com 0
DP -   6      n_fibonacci :=n_max;
DP -   6.1    ELSIF (n_max = 1)    -----  comparação de n_max com 1
DP -   6.2    n_fibonacci :=n_max;
              ELSIF OTHERS
DP            ENDIF
DP -   7      WHILE (n_max /= 1) LOOP
.............
```

Ou seja, são necessários mais um par de estados para acomodar a segunda comparação.

6) Substituindo e renumerando os estados da FSMD e associando as operações da PC aos estados do DP

```
PC -   1              status_o <= "00";
PC -   1              irq_o <= '0';

DP -   1                    data_out <= 0;

PC -   1              WHILE(write_enable_i/='1') LOOP
PC -   1              END LOOP;

DP -   2                    n_anterior1 :=1;
DP -   3                    n_anterior2 :=0;
DP -   4                    n_max :=data_in;

PC -   4 (ou 5)             status_o <= "01";

DP -   5      IF (n_max = 0)
DP -   6      n_fibonacci :=n_max;
DP -   7      ELSIF (n_max = 1)
DP -   8      n_fibonacci :=n_max;
              ELSIF OTHERS
DP            ENDIF
DP -   9      WHILE (n_max /= 1) LOOP

DP -   10                   n_fibonacci :=n_anterior1+n_anterior2;
DP -   11                   n_anterior2 :=n_anterior1;
DP -   12                   n_anterior1 :=n_fibonacci;
DP -   13                   n_max :=n_max-1;
DP -                        END LOOP;
DP -                        END IF;
DP -   14                   data_out <= n_fibonacci;
```

```
PC -    14              irq_o   <= '1';
PC -    14              status_o <= "10";
PC -    14              WHILE (read_enable_i/='1') LOOP
PC -                    END LOOP;
```

Temos, portanto uma FSMD com 14 estados.

Construindo o *datapath* (DP)

Atribuindo módulos funcionais ao DP:

Percorrendo o código e associando 1 MF a cada operação, 1 REG a cada variável (e constante) e 1 MUX a cada múltipla origem de dados:

3 comparadores

- Comp1 - (n_max=0)
- Comp2 - (n_max=1)
- Comp3 - (n_max /=1)

1 subtrator n_max := n_max-1
1 somador n_fibonacci := n_anterior1 + n_anterior2

7 Registradores

- Reg1 - n_anterior1
- Reg2 - n_anterior2
- Reg3 - n_max
- Reg4 - n_fibonacci
- Reg5 - data_out

Constantes

- Reg6 - 0
- Reg7 - 1

5 Multiplexadores (2x1)

- Mux1 - data_out
- Mux2 - n_anterior1
- Mux3 - n_anterior2
- Mux4 - n_max
- Mux5 - n_fibonacci

Entradas externas:

- data_in

Saídas externas:

- data_out

Construindo a máquina de estados (PC)

Entradas externas:

- write_enable_i
- read_enable_i

Saídas externas:

- status_o (1..0)
- irq_o

Entradas (flags do DP):

- S(n_max=0)
- S(n_max=1)
- S(n_max=/=1)

Saídas (flags para o DP)

- Ld(Reg1)
- Ld(Reg2)
- Ld(Reg3)
- Ld(Reg4)
- Ld(Reg5)
- Ld(Reg6)
- Ld(Reg7)

- Sel(Mux1)
- Sel(Mux2)
- Sel(Mux3)
- Sel(Mux4)
- Sel(Mux5)

Número de estados da FSM = 14

- **Codificar os estados**
- **Montar a tabela de transição de estados.**