Laborator 3

O introducere în limbajul VHDL

Introducere

- cel mai cunoscut şi puternic limbaj de descriere hardware a circuitelor. Permite modelarea şi simularea sistemelor numerice la orice nivel de complexitate.

- posibilităţi ale programului:
  - descompunere ierarhică a sistemelor
  - descriere structurală a sistemelor
  - descriere comportamentală a sistemelor
  - permite verificarea funcţională a sistemului prin simulare
  - permite verificare temporală a sistemului
  - permite dezvoltarea unui proiect în echipă
  - independent de tehnologie şi proces de fabricaţie
  - portabilitate
  - suport CAD-CAE
  - disponibilitate publică

- o componentă hardware (poartă logică sau microprocesor, de exemplu) este reprezentată de un cuplu format dintr-o **entitate** şi o **arhitectură** asociată. Entitatea defineşte conexiunile cu exteriorul, denumite porturi, prin listarea numelor, a direcţiei datelor şi a tipului de date pentru fiecare port, iar arhitectura conţine o descriere a structurii componentei, sau a comportamentului acesteia.

Un exemplu simplu

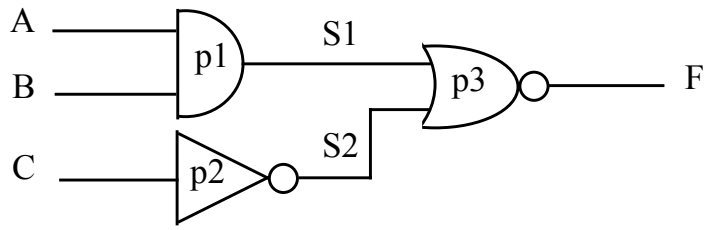

Fig. 12.1 *Schema logică a unei structuri combinaționale*

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

entity FUNC is
end;

architecture VAR1 of FUNC is

        component GATES
                port (A, B, C: in STD_LOGIC;
                        F: out STD_LOGIC);
        end component;

        signal A, B, C, F: STD_LOGIC;

        begin

                A <= '0', '1' after 100 NS, '0' after 300 NS;
                B <= '0', '1' after 200 NS, '0' after 400 NS;
                C <= '1', '0' after 350 NS;
                F <= not((A and B) or (not C));

                M: GATES port map (A, B, C, F);

        end VAR1;
```

Fig. 12.2 *Codul VHDL pentru descrierea comportamentală şi simularea circuitului din figura 12.1*

- simularea se poate face în regim continuu sau în paşi, dar în acest model nu s-au luat în considerare timpii de propagare prin porți.

- codul din figura 12.3 oferă o **descriere structurală** a circuitului, la nivel de componente şi interconexiuni. Fiecare poartă logică este introdusă prin directiva *component*, se declară semnalele de intrare/ieşire A, B, C, F, dar şi semnalele interne S1 şi S2 (vezi figura 12.1). Se precizează apoi variația semnalelor de intrare, iar semnalele de intrare/ieşire pentru fiecare componentă de circuit sunt *mapate* pe intrările/ieşirile fiecărei componente, conform schemei electrice a circuitului.

```
architecture VAR2 of FUNC is

        component and2gate
                port (A, B: in STD_LOGIC;
                        F: out STD_LOGIC);
        end component;

        component invgate
                port (A: in STD_LOGIC;
                        F: out STD_LOGIC);
        end component;

        component nor2gate
                port (A, B: in STD_LOGIC;
                        F: out STD_LOGIC);
        end component;

        signal A, B, C, F: STD_LOGIC;
        signal S1, S2: STD_LOGIC;

        begin

                A <= '0', '1' after 100 NS, '0' after 300 NS;
                B <= '0', '1' after 200 NS, '0' after 400 NS;
                C <= '1', '0' after 350 NS;

                p1: and2gate port map(A, B, S1);
                p2: invgate port map(C, S2);
                p3: nor2gate port map(S1, S2, F);

        end VAR2;
```

Fig. 12.3 *O altă descriere VHDL pentru circuitul din figura 12.1*

12.3 Descrierea VHDL a unui automat finit

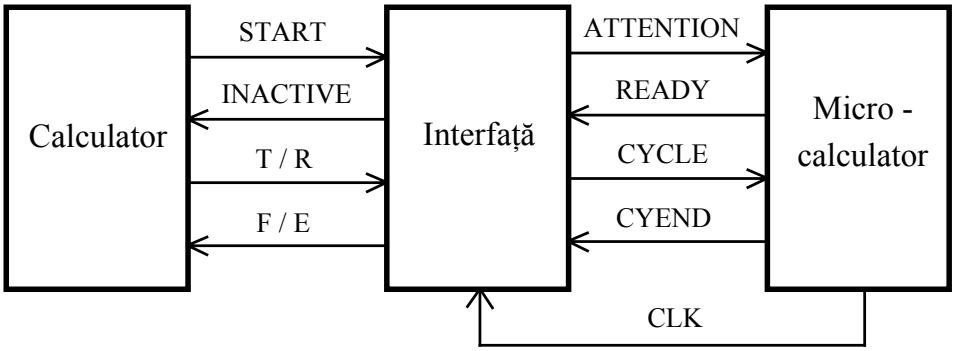

Fig. 12.4 *Semnalele interfeței calculator-microcalculator*

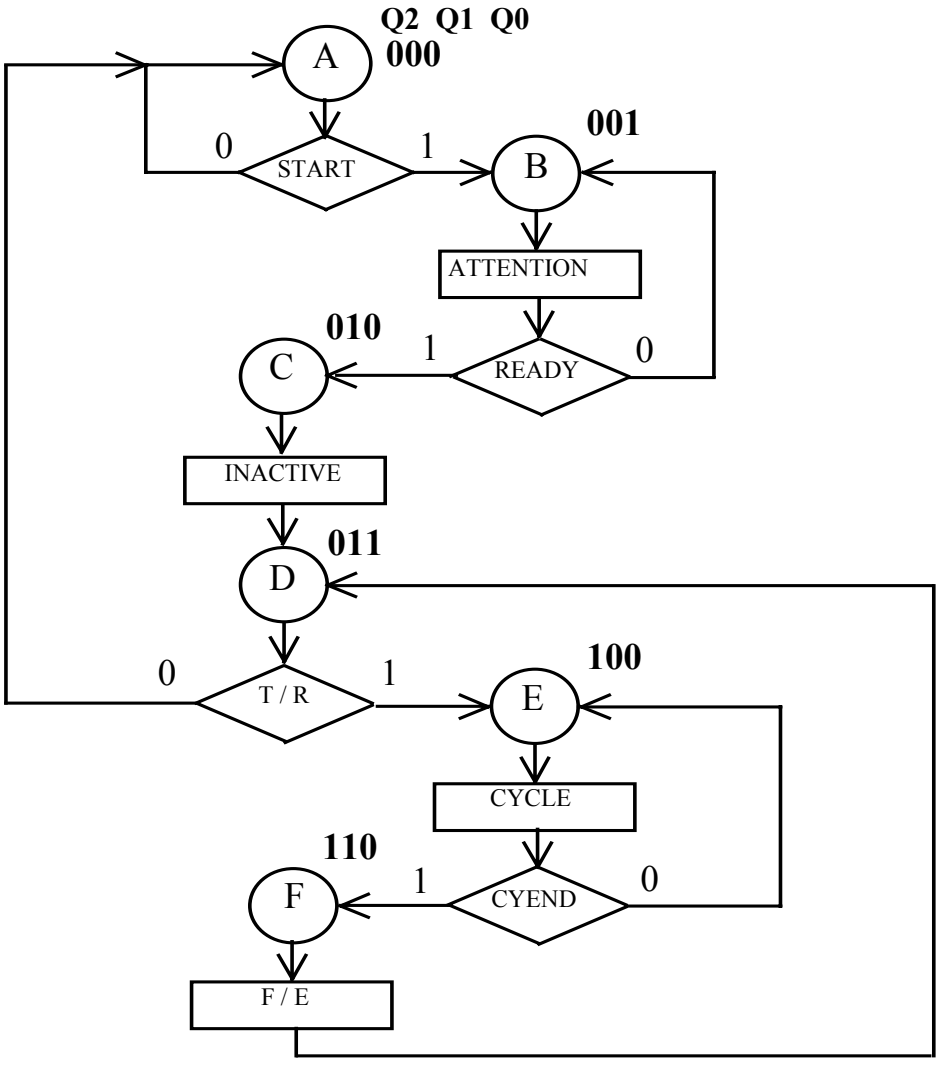

Fig. 12.5 *Organigrama interfeței*

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

entity interfata is
        port(CLOCK, START, READY, TR, CYEND: in std_logic;
                ATTENTION, INACTIVE, CYCLE, FE: out std_logic);
end interfata;

architecture automat_finit of interfata is
        type state_type is (A, B, C, D, E, F);
        signal state: state_type;
        begin
                process(CLOCK)
                begin
                        if CLOCK'event and CLOCK = '1' then
                                case state is
                                        when A =>
                                                if (START = '1') then
                                                        state <= B;
                                                else
                                                        state <= A;
                                                end if;
                                        when B =>
                                                if ( READY = '1') then
                                                        state <= C;
                                                else
                                                        state <= B;
                                                end if;
                                        when C =>
                                                state <= D;
                                        when D =>
                                                if (TR = '1') then
                                                        state <= E;
                                                else
                                                        state <= A;
                                                end if;
                                        when E =>
                                                if ( CYEND = '1') then
                                                        state <= F;
                                                else
                                                        state <= E;
                                                end if;
                                        when F =>
                                                state <= D;
                                end case;
                        end if;
                end process;

                ATTENTION <= '1' when (state = B) else '0';
                INACTIVE <= '1' when (state = C) else '       0';
                CYCLE <= '1' when (state = E) else '0';
                FE <= '1' when (state = F) else '0';
        end automat_finit;
```