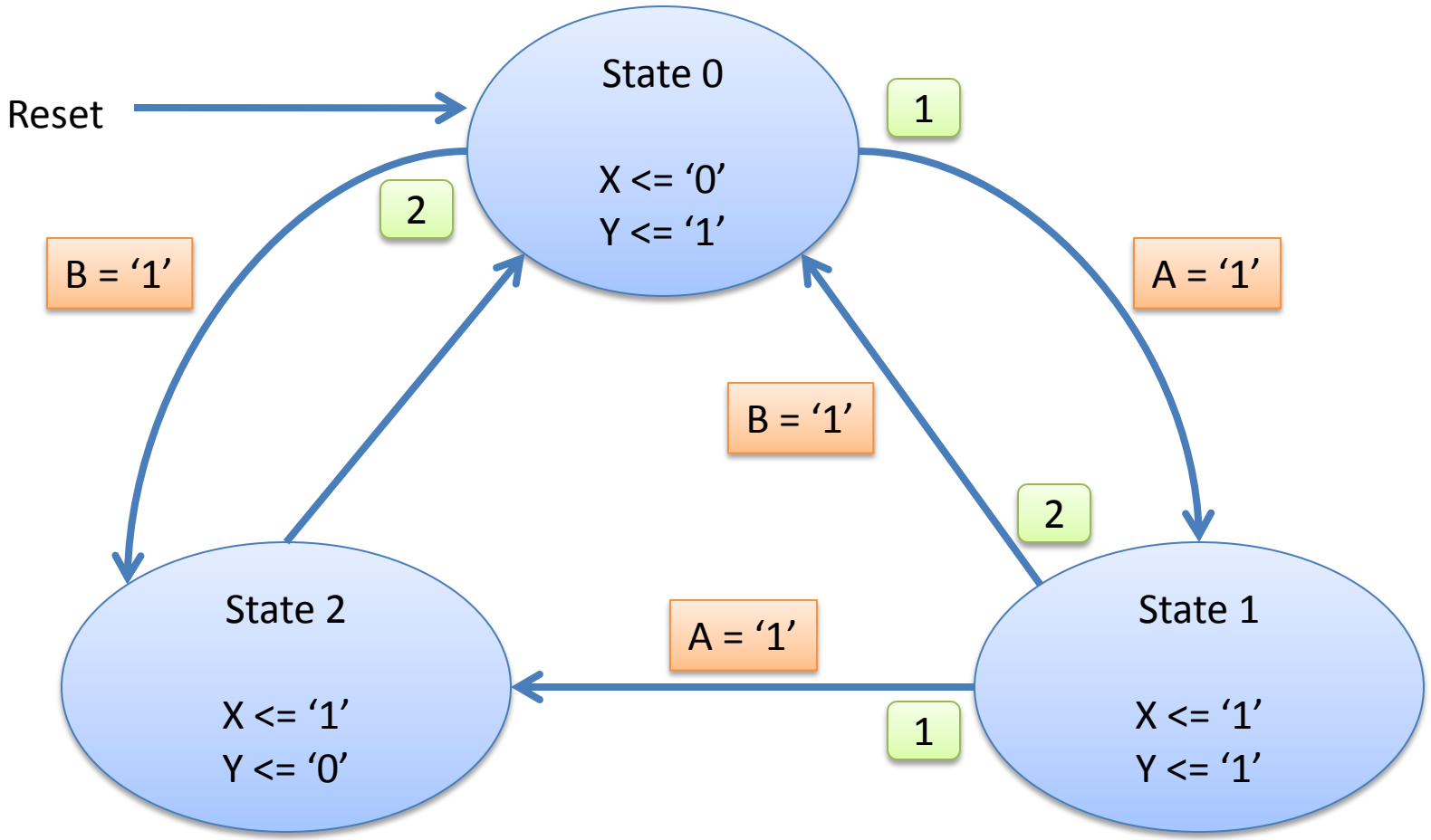
Въведение в VHDL

Проектиране на Крайни Автомати с VHDL

Диаграма на Състоянията на Краен Автомат



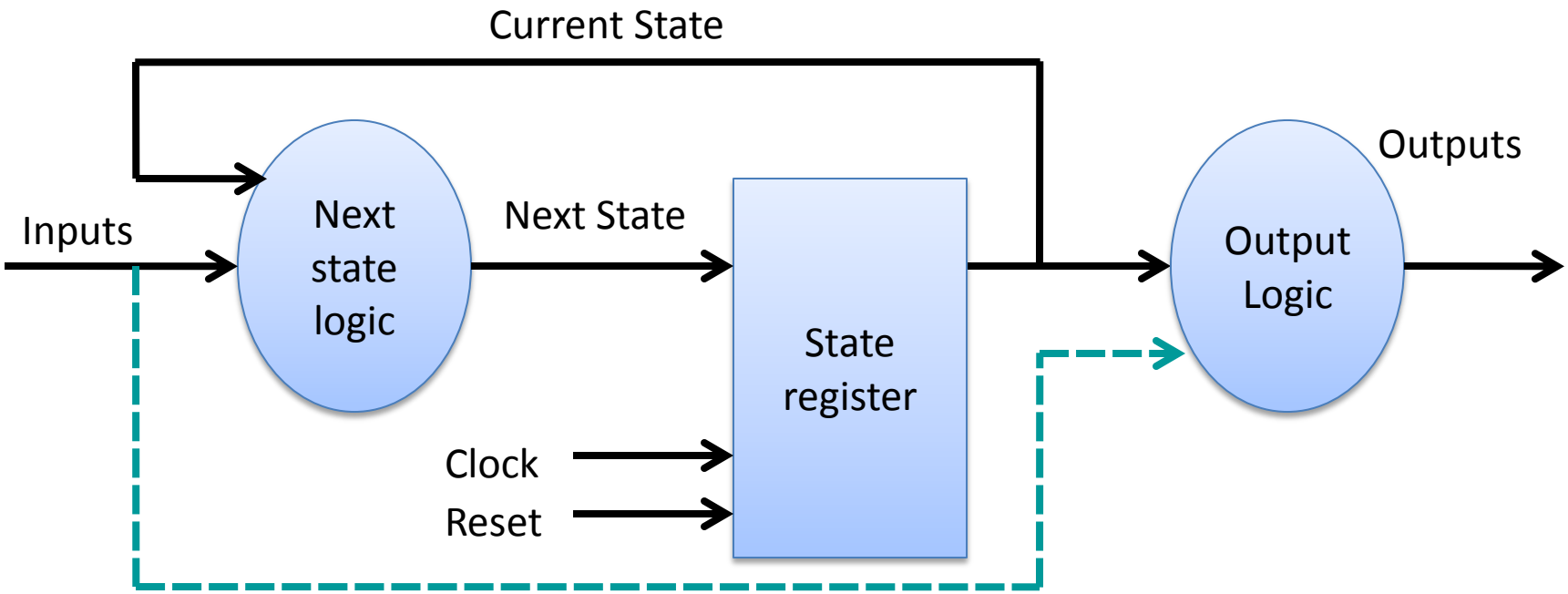
Състояние

State0
X <= '0'
Y <= '1'

Преход

A = '0'

Видове Крайни Автомати



Moore – Изхода зависи само от текущото състояние

Mealy – Изхода зависи от текущото състояние и от входа

Кодиране на Състоянията

Съответствие между символните имена на състоянията и конкретните двоични стойности.

Критерии за избор на кодиране:

–Минимален брой тригери

–Сложност на декодиращата логика

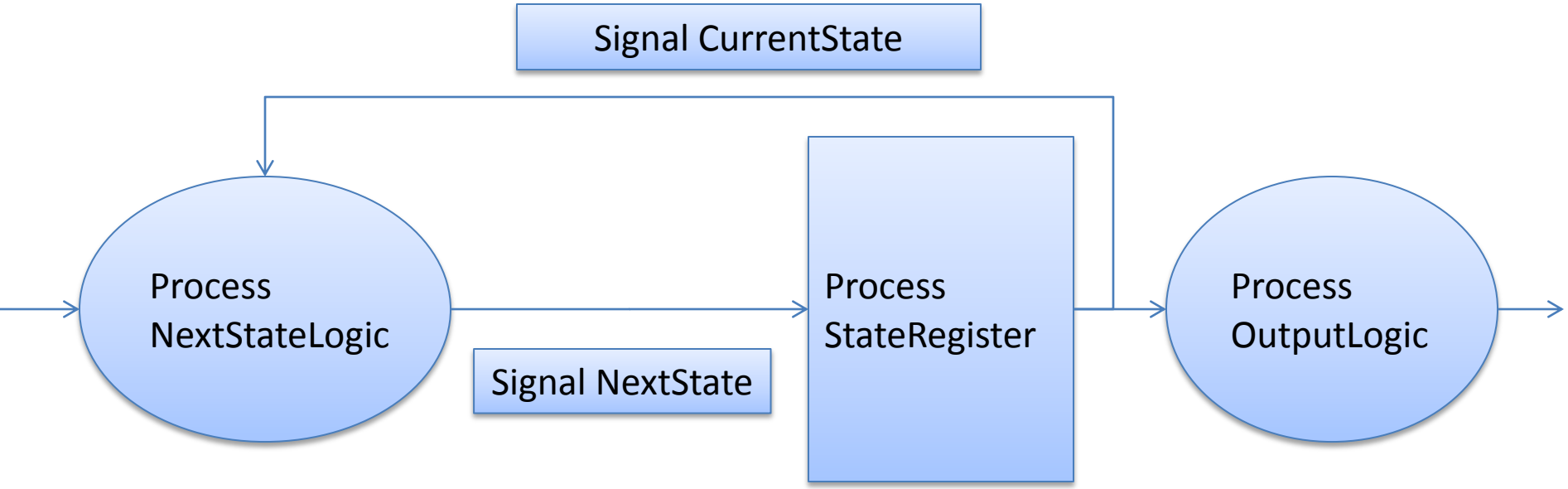
–Минимален брой променящи се битове при преход от състояние в състояние

Binary	Gray	One-Hot	Zero-Hot
000	000	00001	11110
001	001	00010	11101
010	011	00100	11011
011	010	01000	10111
100	110	10000	01111

Методология на Проектиране на Крайни Автомати с VHDL

1. Проектиране на алгоритъма на работа и представянето му като диаграма на състоянията.
2. Определяне на входовете и изходите. Съставяне на интерфейсната част на VHDL модела.
3. Описание на състоянията посредством изброими типове. Дефиниане на типове и сигнали.
4. Регистър на състоянието
5. Логика за следващо състояние
6. Логика за изходите

Структура на VHDL Модела



VHDL Код

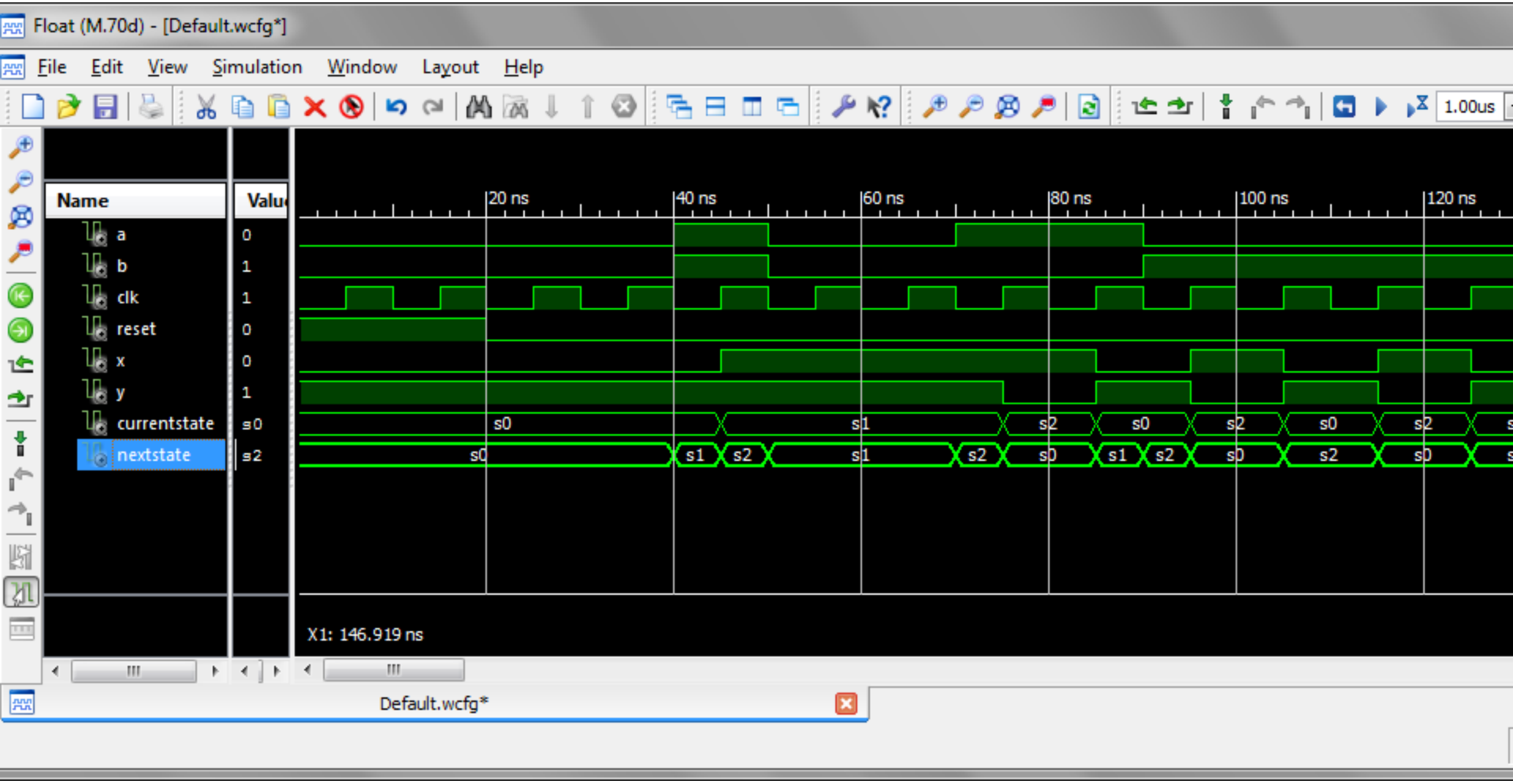
```
architecture FSM of DEMO is
    type StatesEnumeration is ( S1, S2, … );
    signal CurrentState : StatesEnumeration;
    signal NextState : StatesEnumeration;
begin
    StateRegister: process(clock, reset)
      ...
    end process;

    NextStateLogic: process(CurrentState, входове)
      ...
    end process;

    OutputLogic: process(CurrentState)
      ...
    end process;
end FSM;
```

Пример

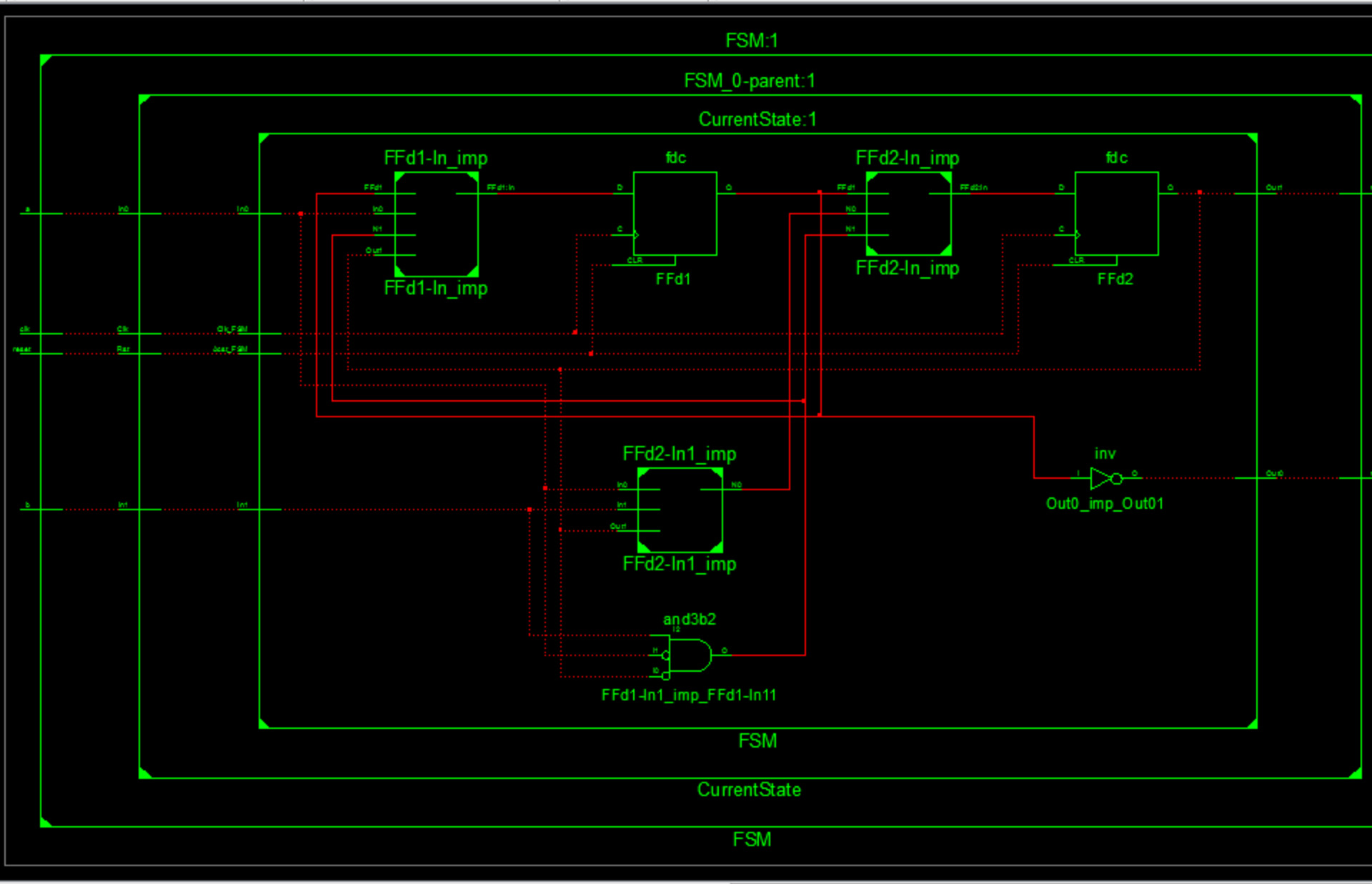
```
architecture Behavioral of FSM is
        type StateType is (S0, S1, S2);
        signal CurrentState, NextState : StateType;
begin
        StateRegister: process(clk, reset)
        begin
                if(reset = '1')then
                        CurrentState <= S0;
                elsif(rising_edge(clk))then
                        CurrentState <= NextState;
                end if;
        end process;
```

```
NextStateLogic: process (CurrentState, a, b)
        begin
                case CurrentState is
                when S0 =>
                        if(a = '1')then
                                NextState <= S1;
                        elsif (b = '1')then
                                NextState <= S2;
                        else
                                NextState <= S0;
                        end if;
                when S1 =>
                        if(a = '1')then
                                NextState <= S2;
                        elsif (b = '1')then
                                NextState <= S0;
                        else
                                NextState <= S1;
                        end if;
                when S2 => NextState <= S0;
                when others => NextState <= S0;
                end case;
        end process;
```

```
        OutputLogic: process (CurrentState) begin
                case CurrentState is
                when S0 =>
                        x <= '0';
                        y <= '1';
                when S1 =>
                        x <= '1';
                        y <= '1';
                when S2 =>
                        x <= '1';
                        y <= '0';
                when others => null;
                end case;
        end process;
end Behavioral;
```

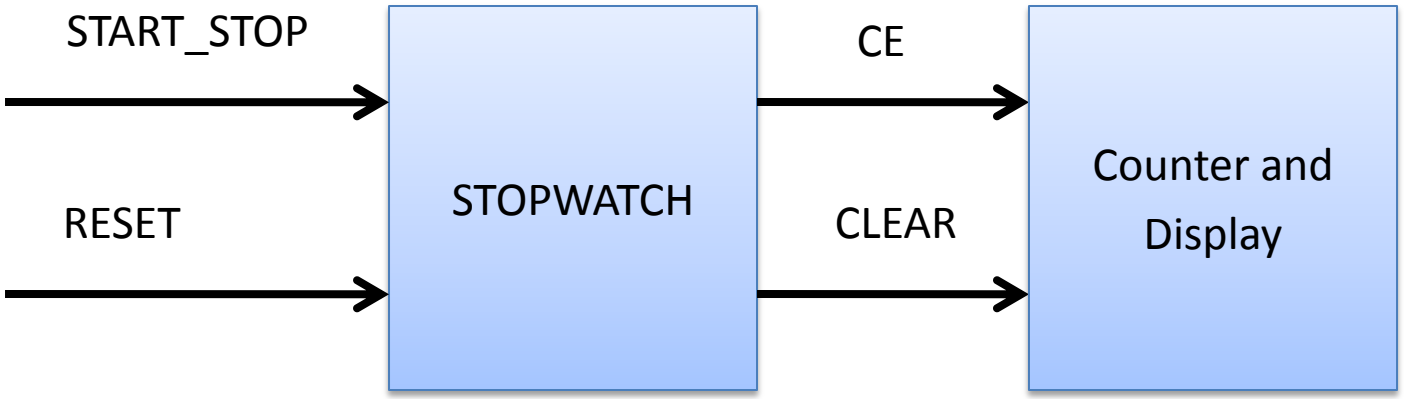

Float (M.70d) - [Default.wcfg*]
File
Edit
View
Simulation
Window
Layout
Help
1.00us
Name
Valu
a
b
clk
reset
x
y
currentstate
nextstate
0
1
1
0
0
1
s0
s2
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
X1: 146.919 ns
Default.wcfg*

Window Layout Help

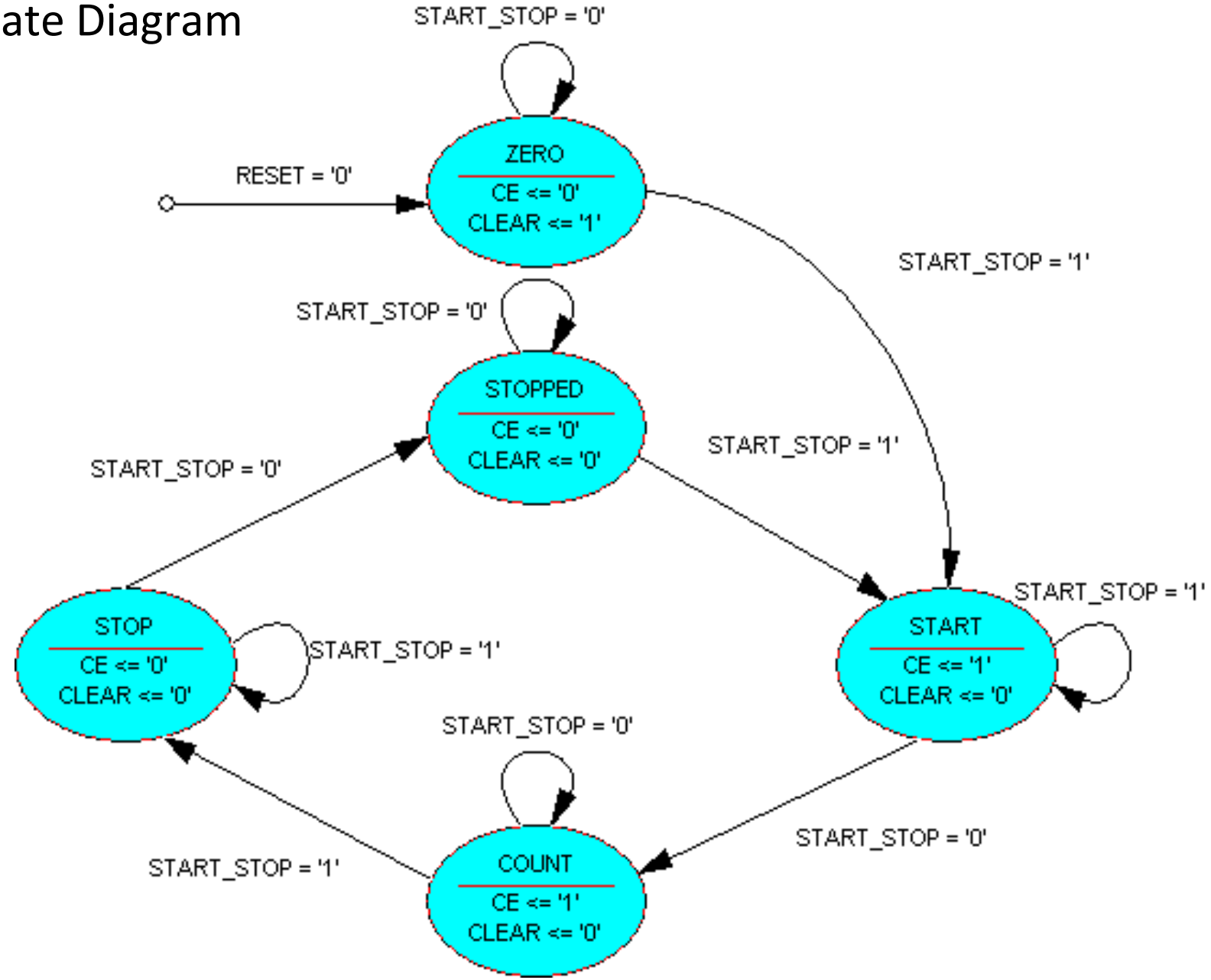


Пример – Хронометър

Проектиране на управляваща логика за хронометър



1 State Diagram

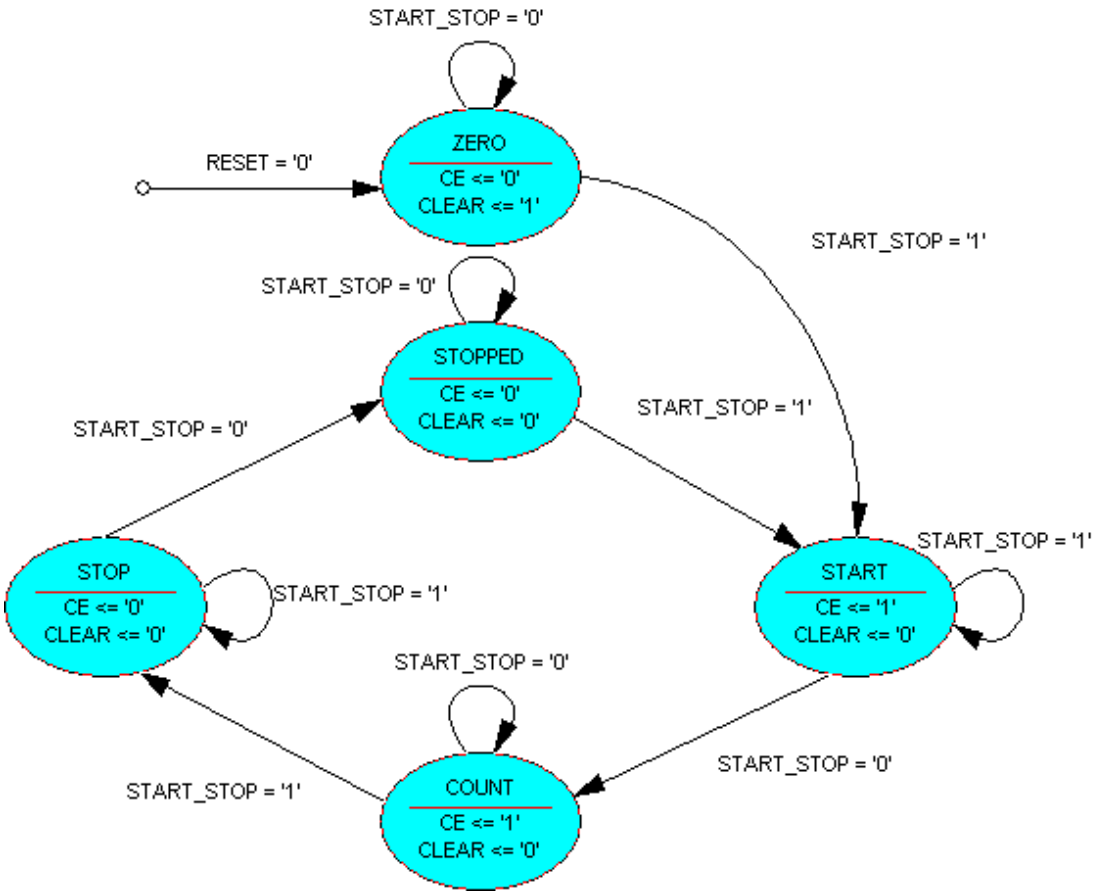


2 Entity

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

entity STOPWATCH is
     port( CLK, RESET, START_STOP: in std_logic;
        CE, CLEAR: out std_logic);
end STOPWATCH;
```

3 States Enumeration

```
architecture FSM of STOPWATCH is
    type STATE_TYPE is (ZERO, START, COUNT, STOP, STOPPED);
    ...
begin
    ...
end FSM;
```



4 State Register

```
architecture FSM of STOPWATCH_CTRL is
    type STATE_TYPE is (ZERO, START, COUNT, STOP, STOPPED);
    signal CURRENT_STATE, NEXT_STATE : STATE_TYPE;
begin
    STATE_REG: process(CLK, RESET)
    begin
      if(RESET = '0')then
            CURRENT_STATE <= ZERO;
      elsif(rising_edge(CLK))then
            CURRENT_STATE <= NEXT_STATE;
      end if;
    end process;
...
end FSM;
```

5 Next State Logic

```
NS_LOGIC: process (CURRENT_STATE, START_STOP) begin
        case CURRENT_STATE is
        when ZERO =>
                if(START_STOP = '1')then
                        NEXT_STATE <= START;
                else
                        NEXT_STATE <= ZERO;
                end if;
        when START =>
                if(START_STOP = '1')then
                        NEXT_STATE <= START;
                else
                        NEXT_STATE <= COUNT;
                end if;
        when COUNT =>
                if(START_STOP = '1')then
                        NEXT_STATE <= STOP;
                else
                        NEXT_STATE <= COUNT;
                end if;
```

5 Next State Logic

```
        when STOP =>
                if(START_STOP = '1')then
                        NEXT_STATE <= STOP;
                else
                        NEXT_STATE <= STOPPED;
                end if;
        when STOPPED =>
                if(START_STOP = '1')then
                        NEXT_STATE <= START;
                else
                        NEXT_STATE <= STOPPED;
                end if;
        when others =>
                NEXT_STATE <= ZERO;
        end case;
end process;
```

6 Output Logic

```
OUTPUT_LOGIC: process(CURRENT_STATE) begin
        case CURRENT_STATE is
        when ZERO =>
                CE <= '0'; CLEAR <= '1';
        when START =>
                CE <= '1'; CLEAR <= '0';
        when COUNT =>
                CE <= '1'; CLEAR <= '0';
        when STOP =>
                CE <= '0'; CLEAR <= '0';
        when STOPPED =>
                CE <= '0'; CLEAR <= '0';
        when others =>
                CE <= '0'; CLEAR <= '1';
        end case;
end process;
```