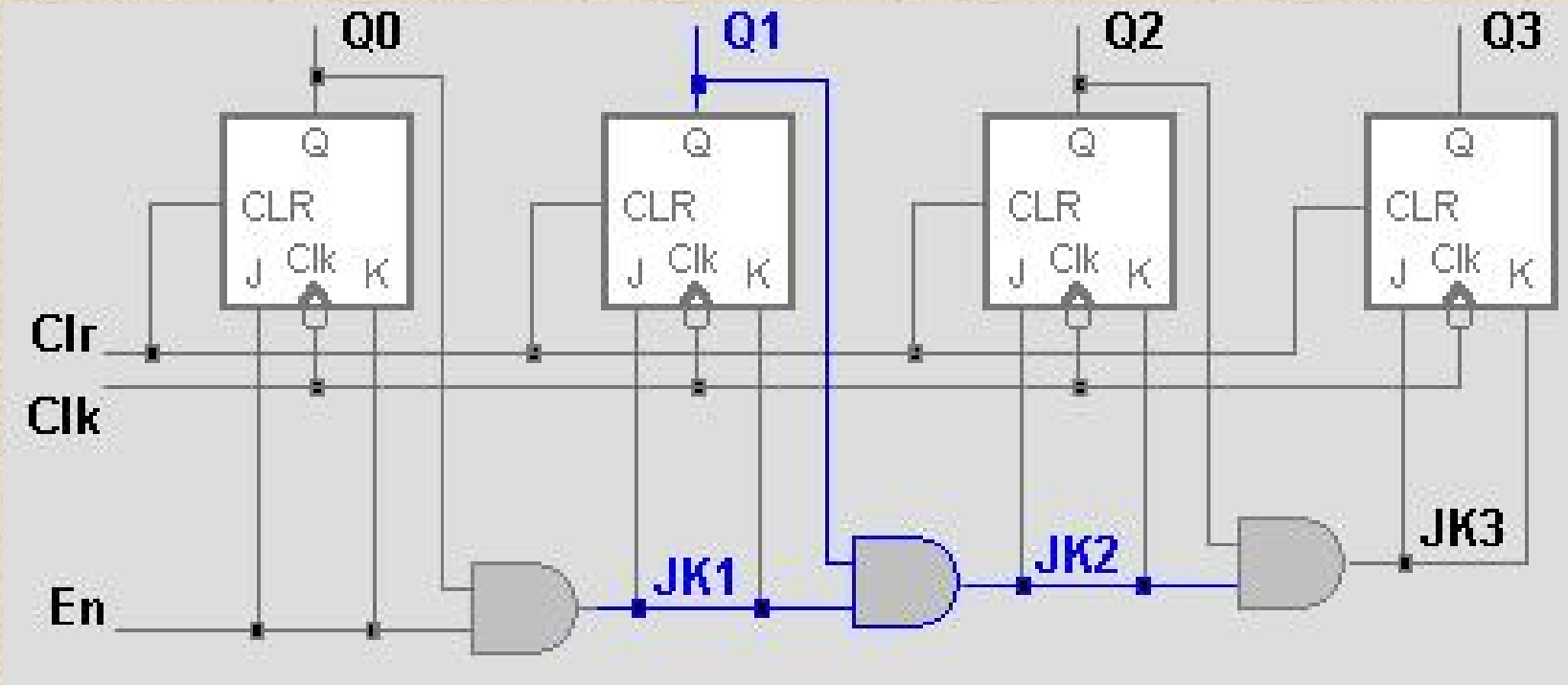
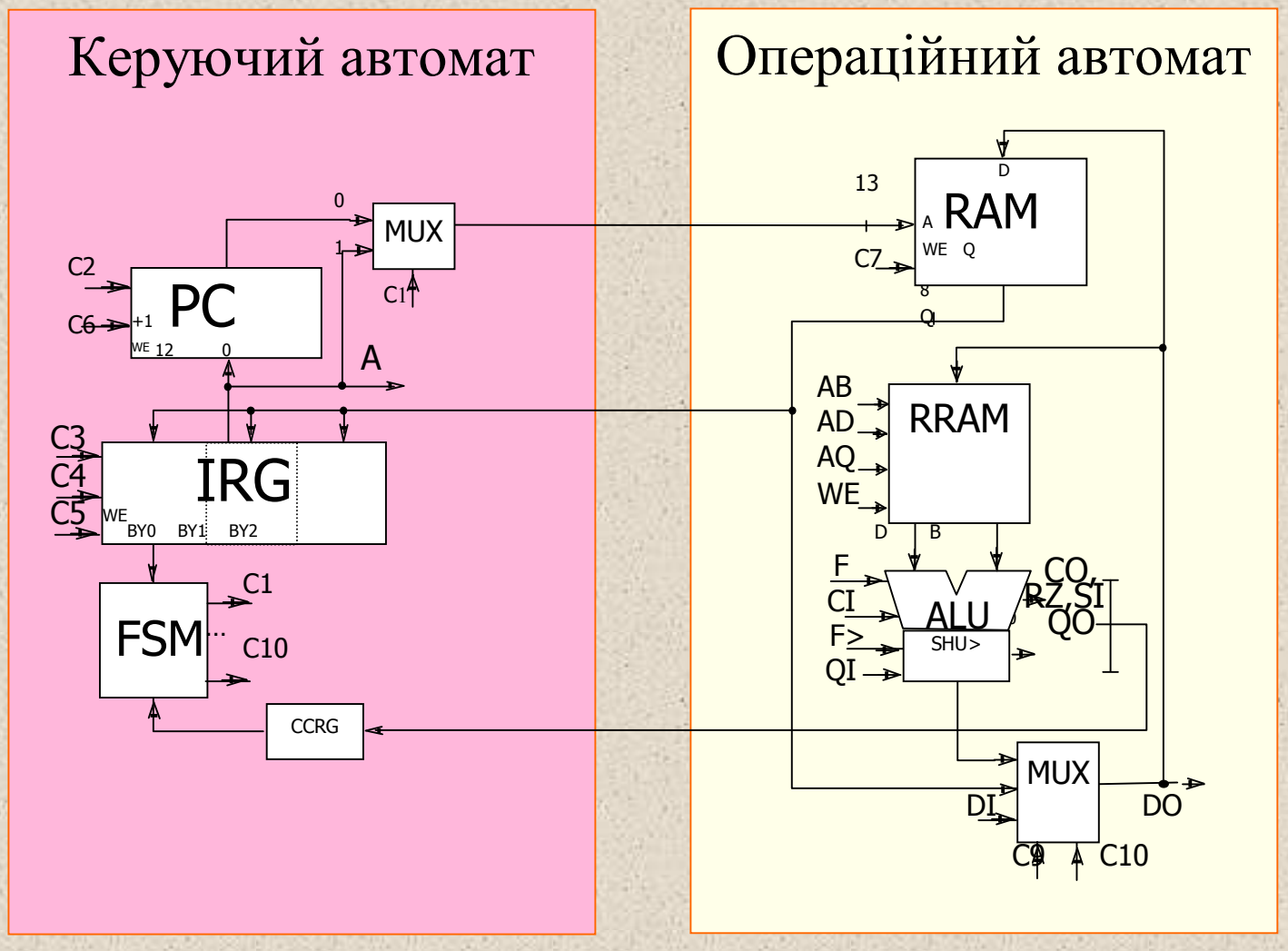
Проектування керування і керування проектуванням



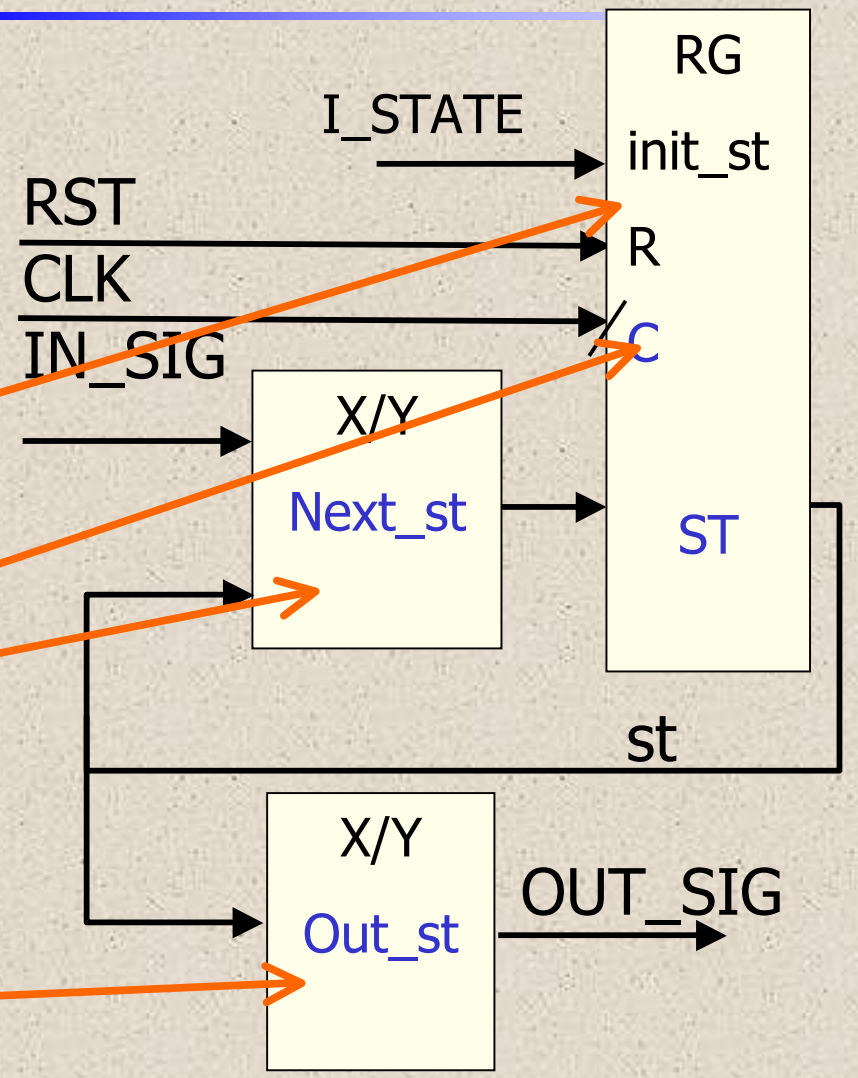
Концепція операційного і керуючого автоматів



Автомат Мура

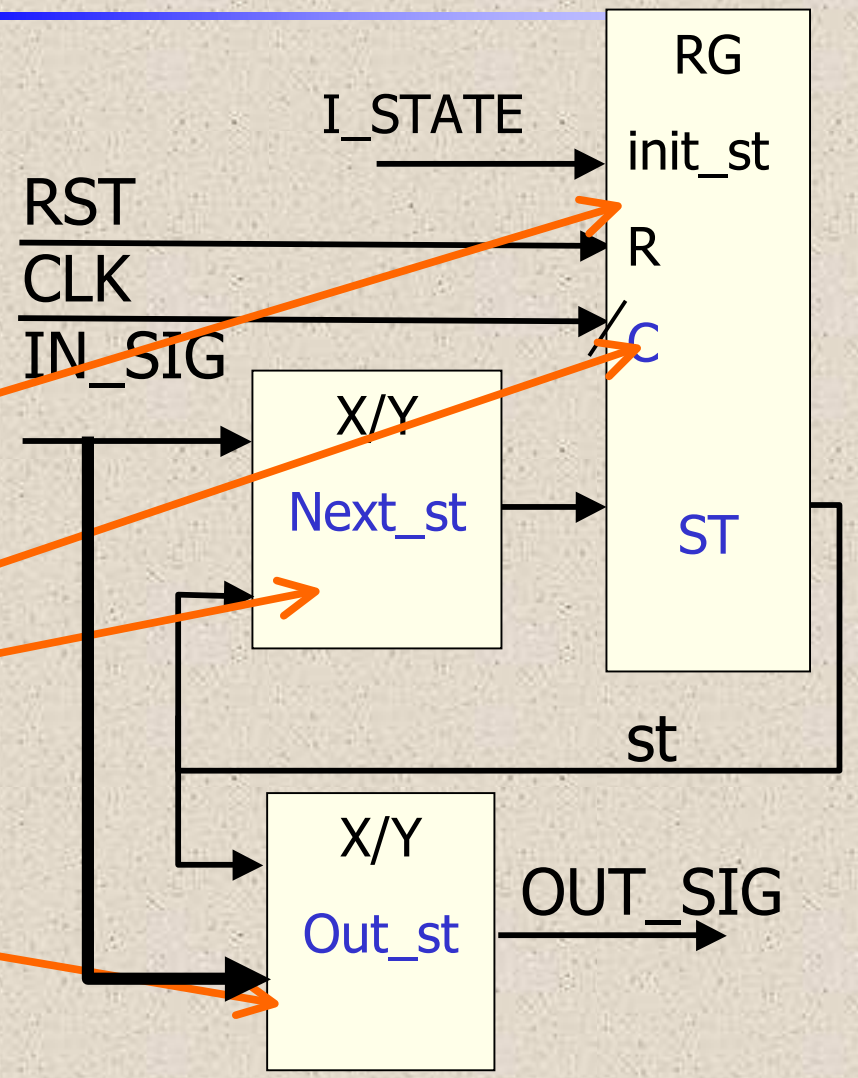
```
architecture moore of automata is
   signal st:STATE;
 constant init_st:STATE:=I_STATE;
begin
   STATE_RG:process(CLK, RST)
   begin
      if RST='1' then
         st<=init_st;
      elsif Rising_edge(CLK) then
         st<=Next_st(st, IN_SIG);
      end if;
   end process;

   OUT_S:process(st) begin
      OUT_SIG<=Out_st(st);
   end process;
end  ;
```



Автомат Мілі

```
architecture mealy of automata is
   signal st: STATE;
 constant init_st:STATE:=I_STATE ;
begin
   STATE_RG:process(CLK,RST)
   begin
      if RST='1' then
         st<=init_st;
      elsif Rising_edge(CLK) then
         st<=Next_st(st, IN_SIG);
      end if;
   end process;
   OUT_S:process(st,IN_SIG) begin
      OUT_SIG<=Out_st(st,IN_SIG);
   end process;
end mealy;
```



Приклад проектування автомата

Автомат продає воду, приймає монети по 5 і 10 копійок. Спроектувати керуючий автомат, який би спрацьовував при наборі 25 коп.

Вхідні сигнали:

V5 –кинуто 5 коп

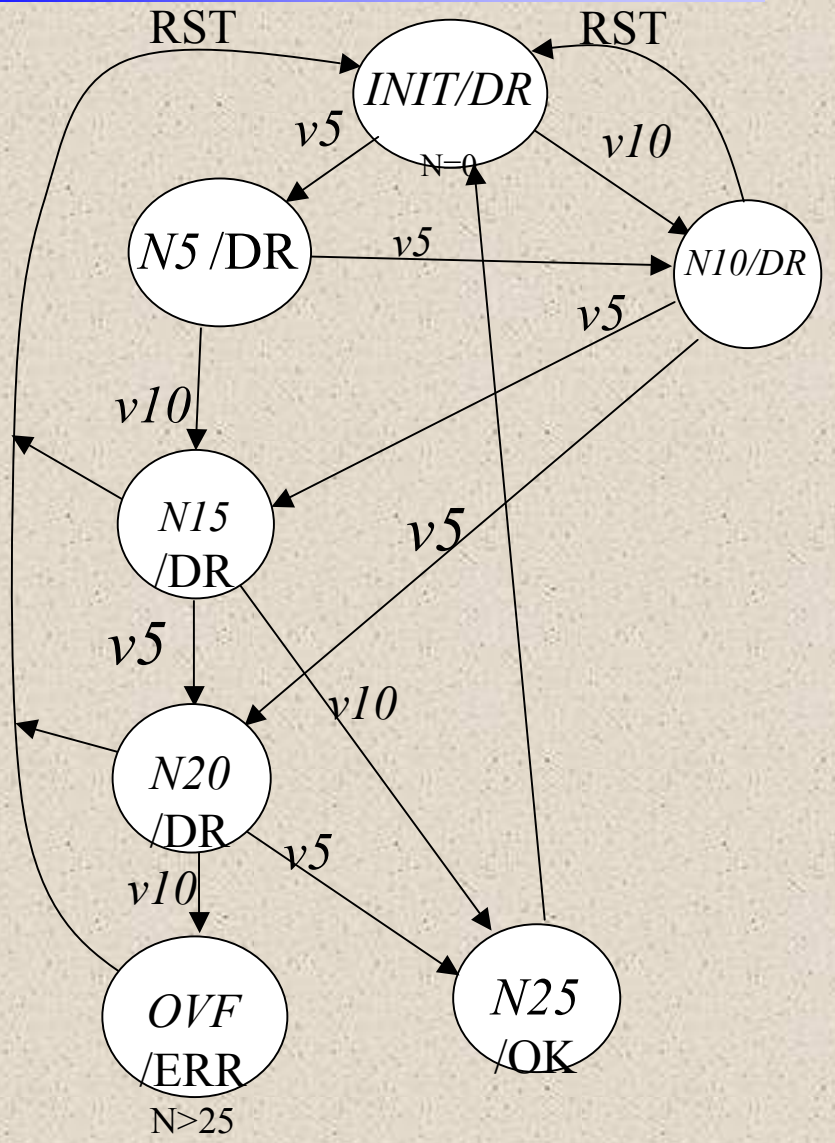
V10 -кинуто 10 коп

Вихідні сигнали:

DR – можна кидати

OK - готово

Fail - помилка

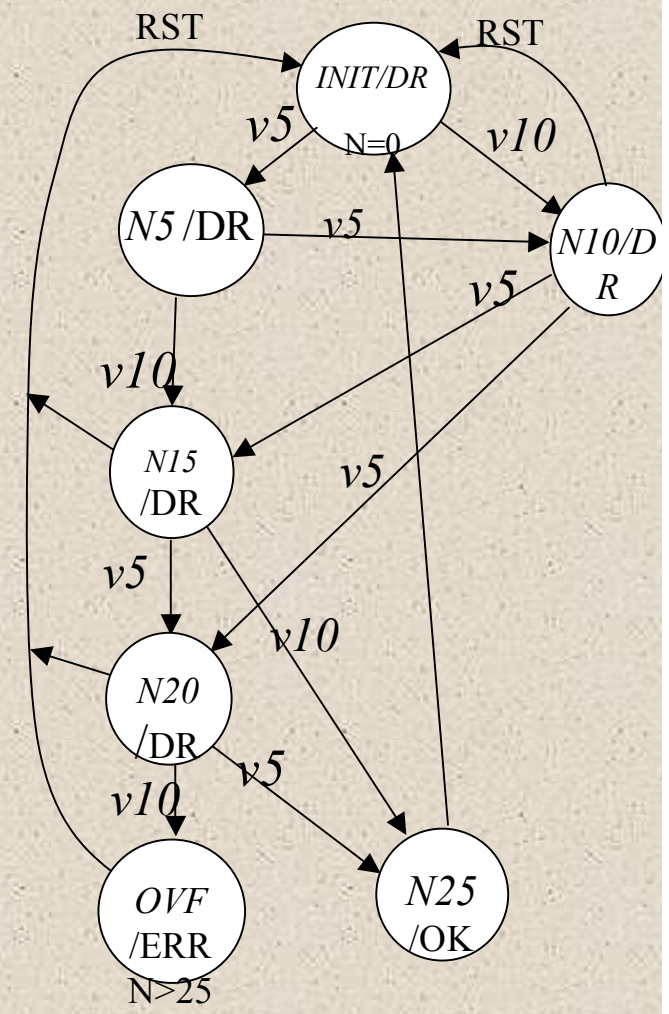


Приклад проектування автомата

```
entity FSM is
        port(CLK:in BIT; -- синхросигнал
       RST:in BIT;  -- начальне встановлення
        V5:in BIT;-- монета 5 коп вкинута
        V10:in BIT;-- монета 10 коп вкинута
        DR:out BIT;-- команда: "Drop a coin"
        OK:out BIT;-- команда: “Налити води"
        ERR: out BIT);-- набрана сума неправильна
End FSM;
architecture BEH of FSM is
                        --стани  автомата
Type STATE is (INIT, N5, N10, N15, N20, N25, OVF);
signal st : STATE;
Begin
STATES:  process(CLK,RST)      -- FSM process
        begin
                if RST='1' then
                        st<=INIT;   --начальне встановлення
                elsif CLK='1' and CLK'event then --регістр станів
```

Приклад проектування автомата

```
case st is     --наступний стан автомата
        when INIT => if V5='1' then st<=N5;
                elsif V10='1' then st<=N10;
                end if;
        when N5 => if V5='1' then st<=N10;
                elsif V10='1' then st<=N15;
                end if;
        when N10 => if V5='1' then st<=N15;
                elsif V10='1' then st<=N20;
                end if;
        when N15 => if V5='1' then st<=N20;
                elsif V10='1' then st<=N25;
                end if;
        when N20 => if V5='1' then st<=N25;
                elsif V10='1' then st<=OVF;
                end if;
        when N25 => st<=INIT;
        when others => null; -- нічого
end case;
end if;
end process;
```

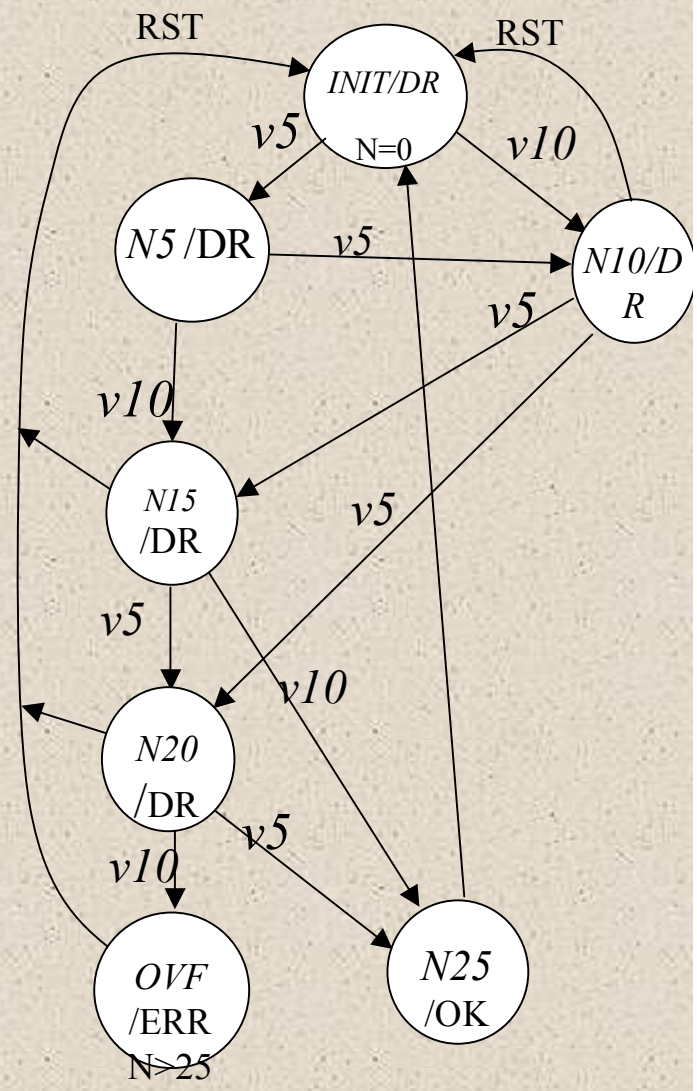


Приклад проектування автомата

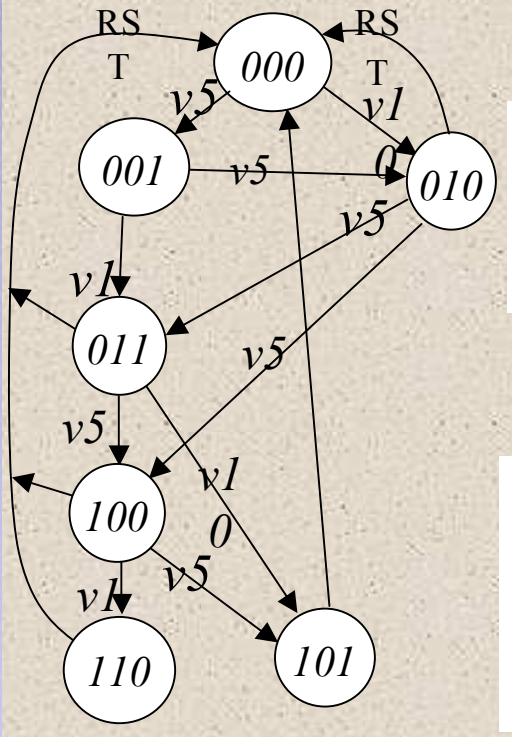
```
-- вихідні сигнали

        DR<='1' when st=INIT
            or  st=N5 or  st=N10
            or st=N15 or st=N20 else '0';
    -- належна сума досягнута
        OK<='1' when st = N25 else '0';
     -- досягнута сума неправильна
        ERR<='1' when st=OVF else '0';

end BEH;
```



Способи кодування станів автомата

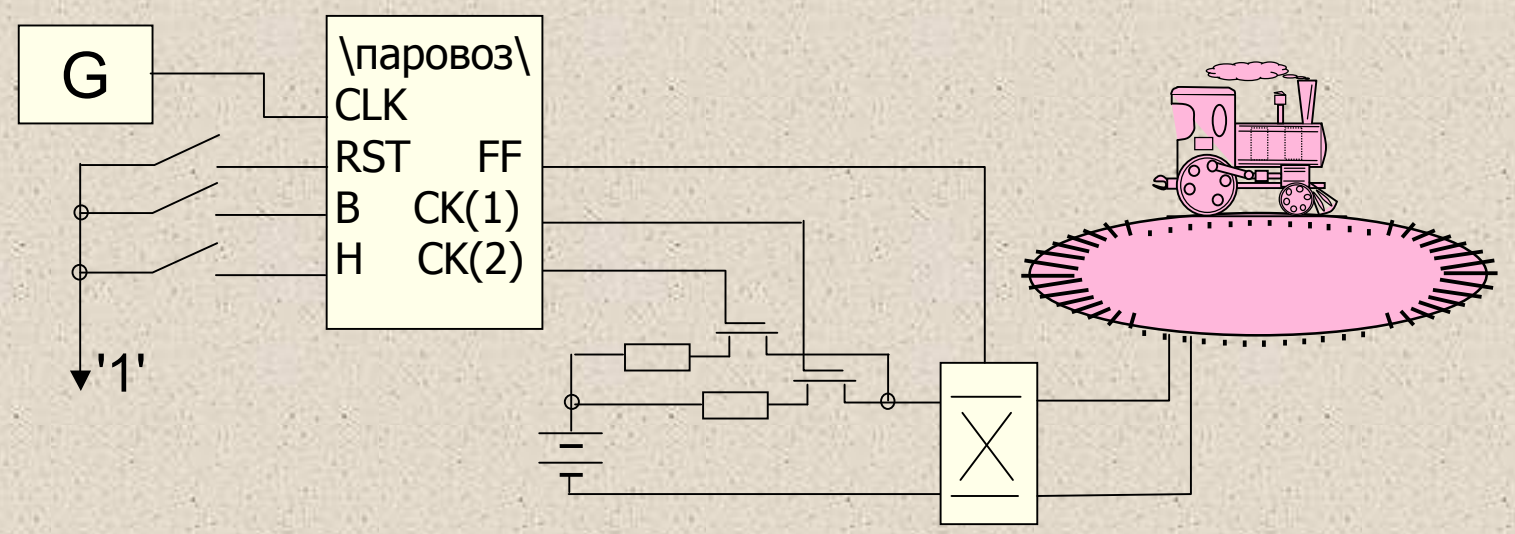



```
attribute enum_encoding : string;
attribute enum_encoding of STATE : type is
"000 001 010 011 100 101 110" ;
```

```
attribute fsm_encoding: string;
attribute fsm_encoding of STATE :signal is
        "{auto | one-hot | compact |gray |
            sequential | johnson | user}";
```

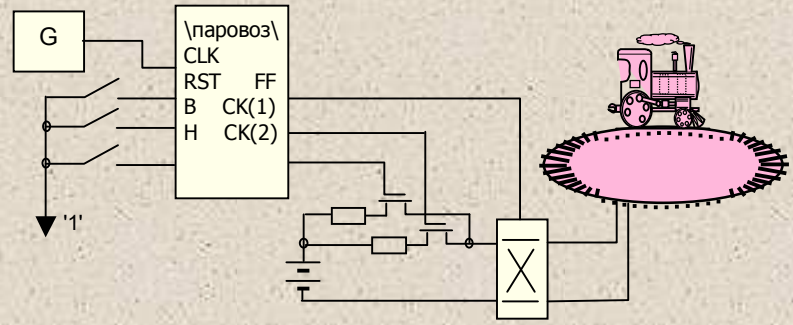
```
"0000001 0000010 0000100 0001000 0010000 0100000 1000000"; --one-hot
"000 001 011 010 110 111 101" ;          --gray
"0000 0001 0011 0111 1111 1110 1100"; -- johnson
```

Приклад проектування автомата керування моделлю залізниці



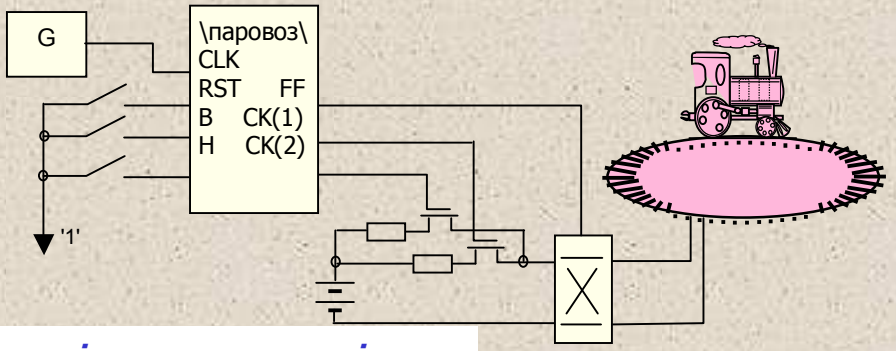

```
architecture synthesis of \паровоз\ is
  signal st: integer range -3  to 3;--стани автомата
begin
  STATE_RG: process(CLK,RST)  --процес керування станом автомата
  begin
    if RST = '1' then
       st<=0;                        -- стан при сигналі начального встановлення
    elsif CLK='1' and CLK'event then
```

Приклад проектування автомата керування моделлю залізниці



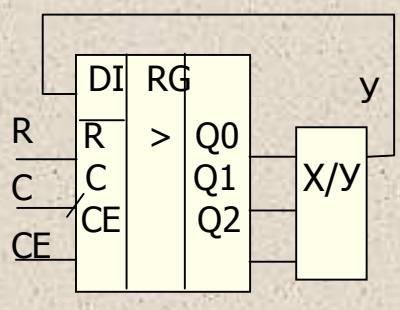

```
case st is                    -- функція наступного стану
        when 0 | 1| 2| -1| -2 => if B='1' then
                                    st<= st+1;
                                 elsif H='1' then
                                    st<= st-1;
                                 end if;
        when 3 =>                if H='1' then  --стан макс. Швидкості вперед
                                    st<= st-1;
                                 end if;
        when others  =>          if B='1' then --стан макс. швидкості назад
                                    st<= st+1;
                                 end if;
      end case;
    end if;
  end process;
```

Приклад проектування автомата керування моделлю залізниці

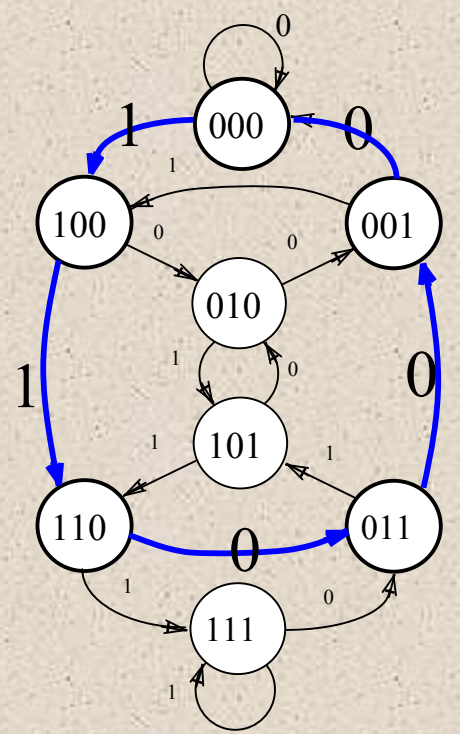



```
OUT_S:process(st)  -- процес логіки вихідних сигналів
  begin
      if st<0 then  --логіка сигнала напрямку руху
         FF<='1';
      else
         FF<='0';
      end if;
      case st is      --логіка сигналу швидкості руху
         when 0 =>  CK<="00";
         when 1|-1 =>  CK<="01";
         when 2|-2 =>  CK<="10";
         when others =>  CK<="11";
      end case;
  end process;
end synthesis;
```

Приклад лічильника по модулю 5 на регістрі зсуву

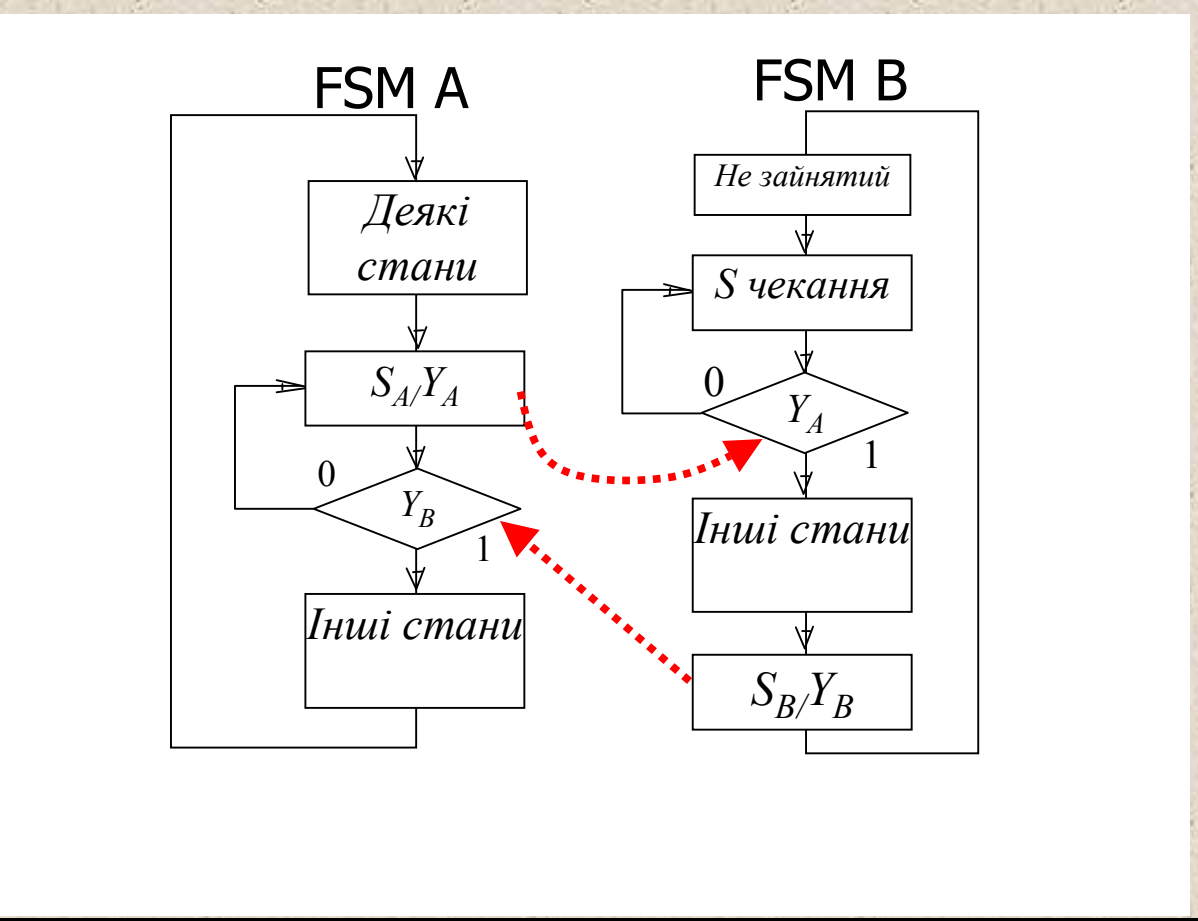



```
SHIFT_RG: process(C,R)  --регістр зсуву
  begin
    if R = '1' then
       Q<="000";  -- начальне встановлення
    elsif CLK='1' and CLK'event then
          if CE='1' then
                Q<=Q(1 downto 0)&Y;
          end if;
    end if;
  end process;
                    --функція Х/У - біт, що засувається
Y<= '1' when (Q="000" or Q="100") else '0';
```



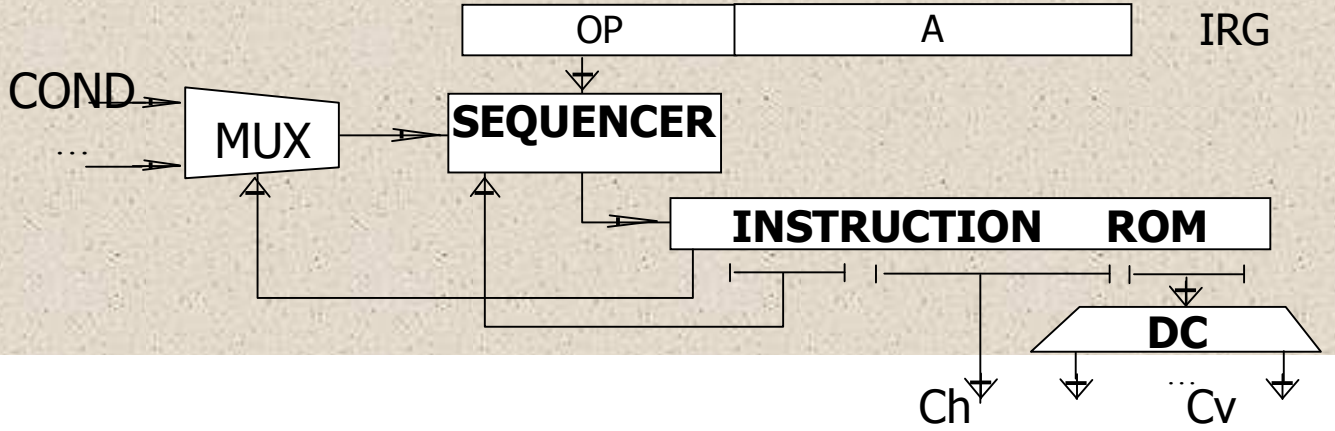
Проектування складних автоматів

Один ведучий автомат FSM A і керовані автомати FSM B



Проектування складних автоматів

Мікропрограмний автомат




```
. . .
constant BRA:   BIT_VECTOR(2 downto 0):="110";
constant LJMP:    BIT_VECTOR(2 downto 0):="111";
type MEM8KX16 is array(0 to 8191) of BIT_VECTOR(15 downto 0);
constant ROM_init: MEM8KX16:=  -- стан мікропрограмної пам"яті
-- адреса,КОП,      операнди,          коментарі
(0=>    SUB      &R0&R0&R0,              -- R0=0 –1а команда
1=>     LD       &R2&R1&R0,             -- дане в R2 з RAM[R1]
2=>     ADDINC &R1&R1&R0,     -- R1=R1+1
3=>     SUBDEC &R2&R0&R2,              -- R2=R2-1, тобто лічильник
4=>     BRA      &NEQ&"1111111101",    --перехід на -2, при/=0
5=>     LJMP    &"0000000110000",      --  перехід на адресу 48
6=>     CALL    &"0000000100000",      -- виклик ПП за адресою 32
```

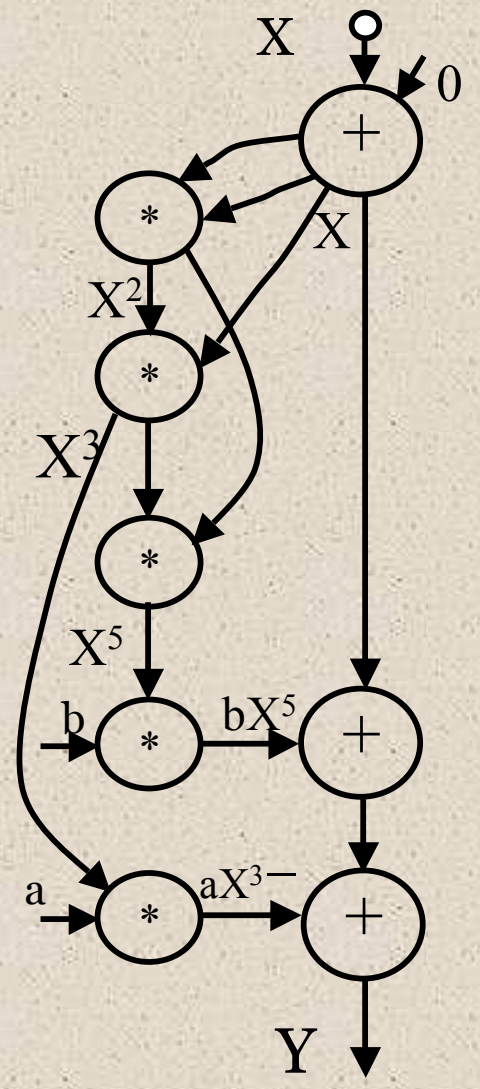
Блок обчислення спецфункції

Приклад проектування блоку, що виконує функцію:

$\sin(X) = X - X^3/6 + X^5/120 = X - a*X^3 + b*X^5$.

Граф алгоритму :

Алгоритм має
5 операцій множення і
2 операції додавання,
не приймаючи до уваги
додавання з 0.

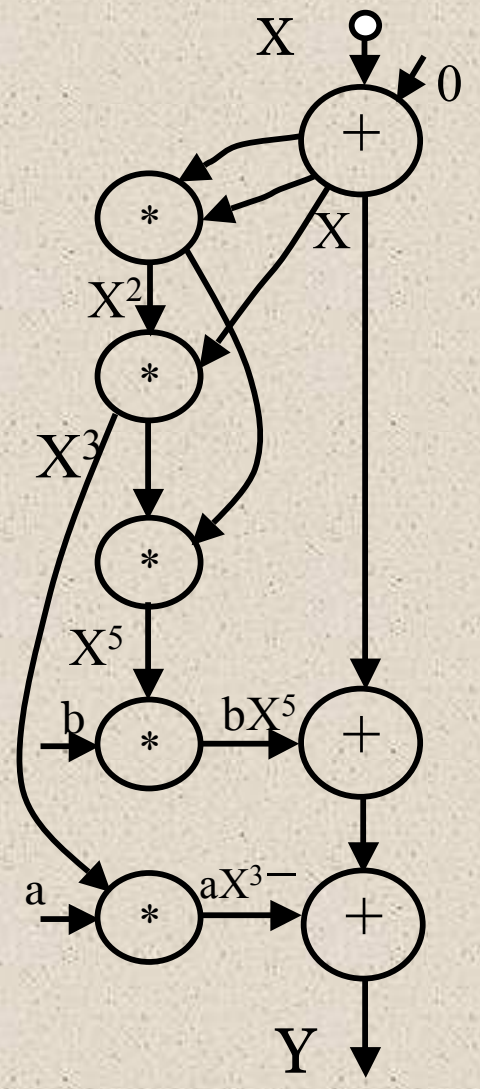


Блок обчислення спецфункції

Метод проектування:

1. **Вибір** обчислювальних **ресурсів**
2. **Складання розкладу** виконання операцій у ресурсах
3. **Призначення** операцій **на ресурси**.

Складання структурної схеми, синтез керуючого автомату

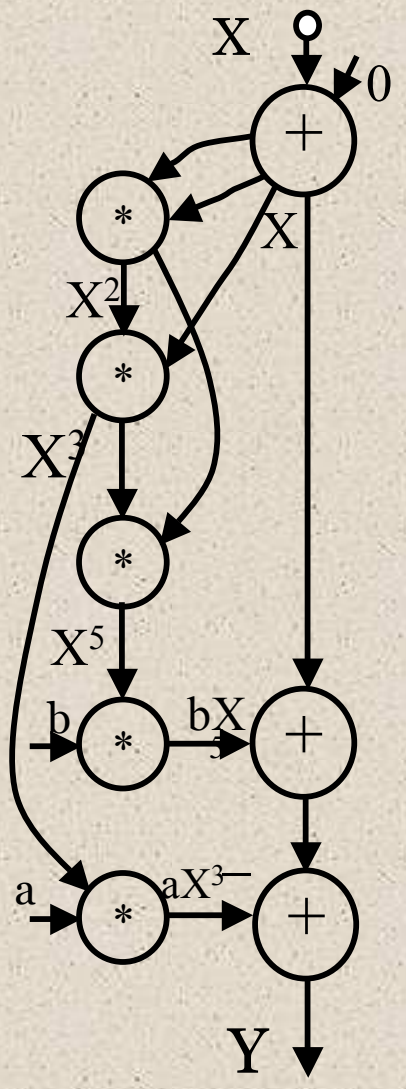


Блок обчислення спецфункції

1.Вибір ресурсів

Для обчислень треба **суматор** з регістром S ,
блок множення з регістром P,
регістри для квадрата X2, куба X3
і результату Y.

Вхідне дане X зразу завантажується в регістр суматора S, де зберігається протягом 5 тактів і бере участь в операціїї додавання в 5-му такті.

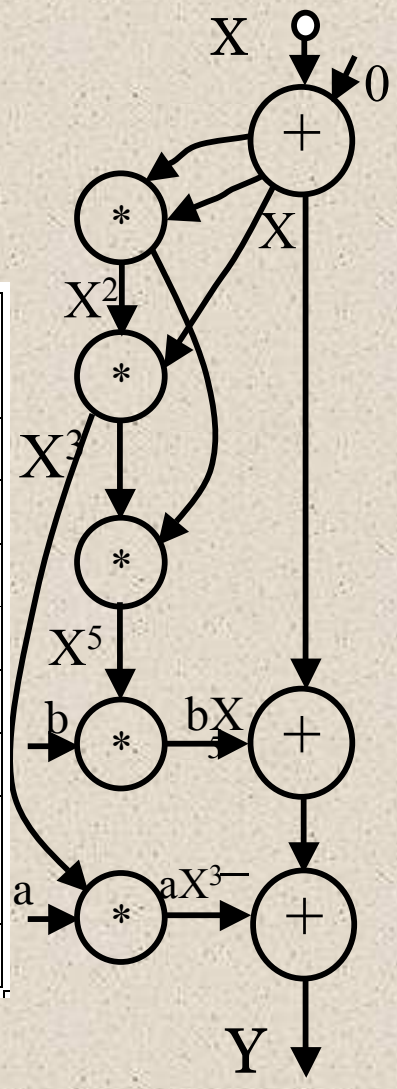


Блок обчислення спецфункції

2.Складання розкладу

Стан регістрів блоку при виконанні алгоритму

№ такту	Суматор S	Блок множ. P	Рег. X2	Рег. X3	Рег. Y
0	X				
1	X	$X^2 = X*X$			
2	X	$X^3 = X^2*X$	X^2		
3	X	$X^5 = X^2*X^3$		X^3	
4	X	$b*X^5$		X^3	
5	$X + b*X^5$	$a*X^3$			
6	$X + b*X^5 - a*X^3$				
7					Y



Блок обчислення спецфункції

3. Призначення операцій на ресурси

Об'єкт блоку

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_arith.all;
entity X_Y_SIN is port(CLK : in STD_LOGIC;      --синхросерія
        RST : in STD_LOGIC; --сигнал начального встановлення
        START : in STD_LOGIC;       --сигнал запуску обчислень
        X : in STD_LOGIC_VECTOR(15 downto 0); --вхідне дане
        Y : out STD_LOGIC_VECTOR(15 downto 0); --вихідне дане
        RDY : out STD_LOGIC);      --сигнал готовності результату
end X_Y_SIN;
```

Блок обчислення спецфункції

@ a і b задаються як 16-бітні вектори, які є цілими числами зі знаком і які представляють дробні числа 1/6 і 1/120, мають масштабний коефіцієнт 2^{-15}

@ добуток P - 32-розрядне -сума розрядності операнда і константи.

@ з P вибираються 16 старших розряди, крім найстаршого, так як добуток чисел зі знаком має 2 однакових знакових розряди.

@ Сума S має 1 додатковий розряд щоб запобігти переповненню.

Архітектура блоку описується як наступна

Декларативна частина

```
architecture X_Y_SIN of X_Y_SIN is
        constant a:SIGNED(15 downto 0):= --константа a
        SIGNED(CONV_STD_LOGIC_VECTOR(integer(1.0/6.0*2.0**15),16));
        constant b:SIGNED(15 downto 0):=  --константа b
SIGNED(CONV_STD_LOGIC_VECTOR(integer(1.0/120.0*2.0**15),16));
        signal  s:SIGNED(16 downto 0);--adder
        signal  p:SIGNED(31 downto 0);--multiplier
        signal  x2,x3:SIGNED(15 downto 0);--intermediate results
        signal ct2:natural range 0 to 7;      --лічильник станів
begin
```

Блок обчислення спецфункції

Автомат керування

```
FSM:process(CLK,RST) begin
                    if RST='1' then
                                ct2<=6; RDY<='0';
                    elsif CLK='1' and CLK'event then
                                if ct2=7 then
                                            RDY<='1';  -- calculation is ready
                                end if;
                                if START='1' then
                                            ct2<=0;            --start of calculations
                                            RDY<='0';
                                elsif ct2<7 then
                                            ct2<=ct2+1; -- лічильник тактів
                                end if;
                    end if;
          end process;
```

Блок обчислення спецфункції

```
RALU:process(CLK,RST)   begin
        if RST='1' then s<=(others=>'0'); p<=(others=>'0');
                x2<=(others=>'0');x3<=(others=>'0'); Y<=(others=>'0');
        elsif CLK='1' and CLK'event then
        case ct2 is
                when 0=> s<= signed(SXT((X),17));--input datum  X
                when 1=> p<= s(15 downto 0)*s(15 downto 0);--X^2
                when 2=> p<= p(30 downto 15)*s(15 downto 0);--X^3
                        x2<=p(30 downto 15);                    -- X^2
                when 3=> p<= p(30 downto 15)*x2;          --X^5
                        x3<=p(30 downto 15);                    -- X^3
```

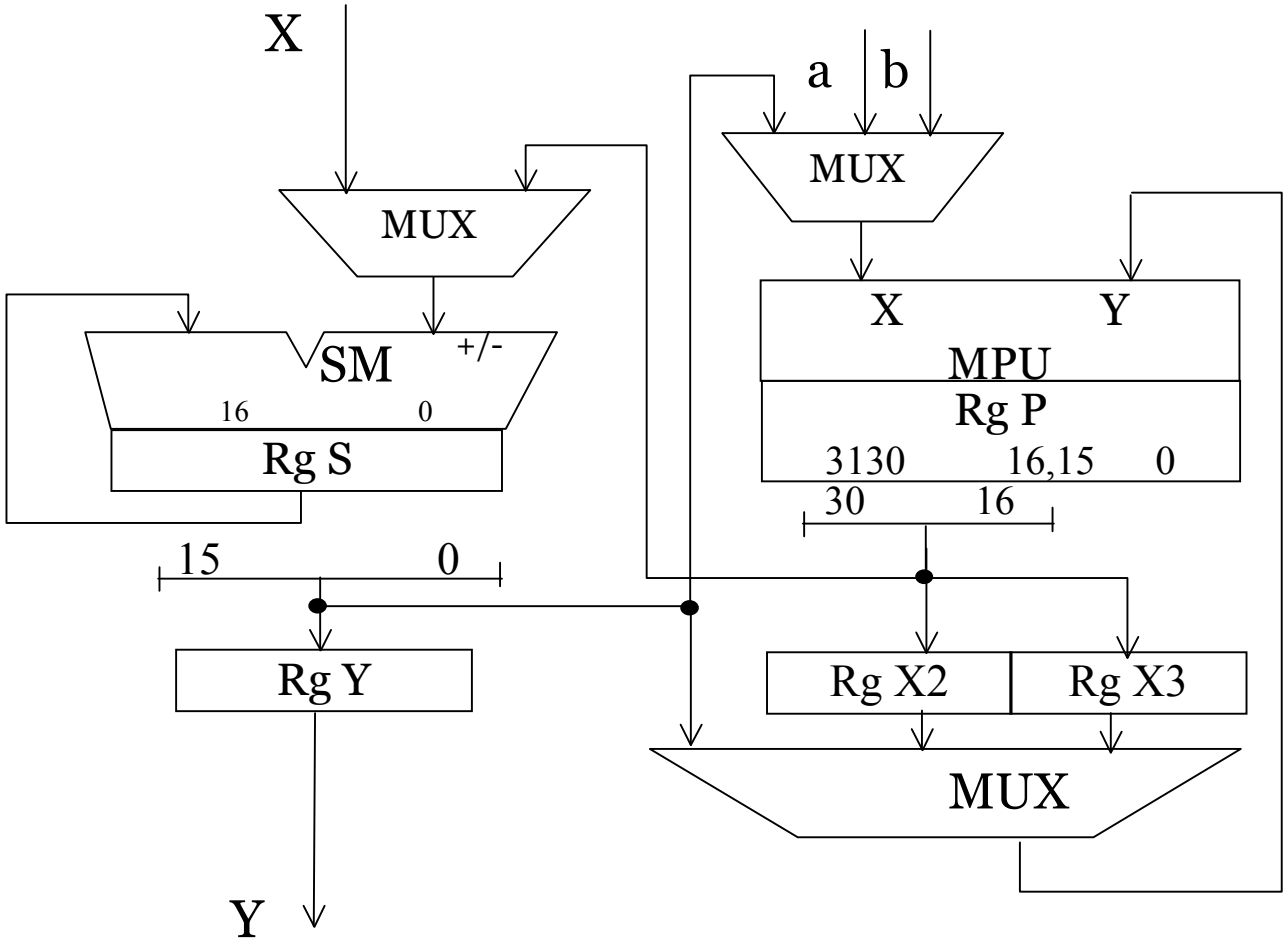
№ такту	Суматор S	Блок множ. P	Рег. X2	Рег. X3	Рег. Y
0	X				
1	X	$X^2 = X*X$			
2	X	$X^3 = X^2*X$	X^2		
3	X	$X^5 = X^2*X^3$		X^3	
4	X	$b*X^5$		X^3	
5	$X+ b*X^5$	$a*X^3$			

Блок обчислення спецфункції

```
                when 4=> p<= p(30 downto 15)*b;      --b*X^5
                when 5=> p<= x3*a;                   --a*X^3
                    s<= s + p(31 downto 15);          --X+b*X^5
                when 6=> s<= s - p(31 downto 15);--X+b*X^5-a*X^3
            when others=> Y<= STD_LOGIC_VECTOR(s(15 downto 0));-- Y
            end case;
            end if;
end process;
end X_Y_SIN;
```

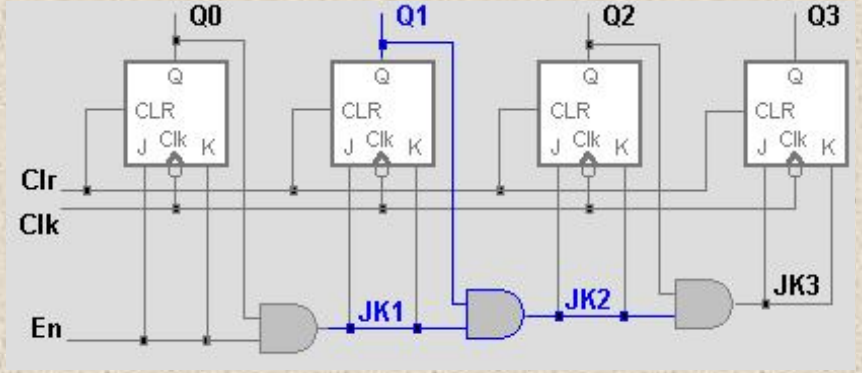
4	X	$b*X^5$		X^3	
5	$X+ b*X^5$	$a*X^3$			
6	$X + b*X^5 – a*X^3$				
7					Y

Блок обчислення спецфункції



Структура блока обчислення Y=F(X)

Керування проектуванням



- **Файл обмежень (constraint file)**

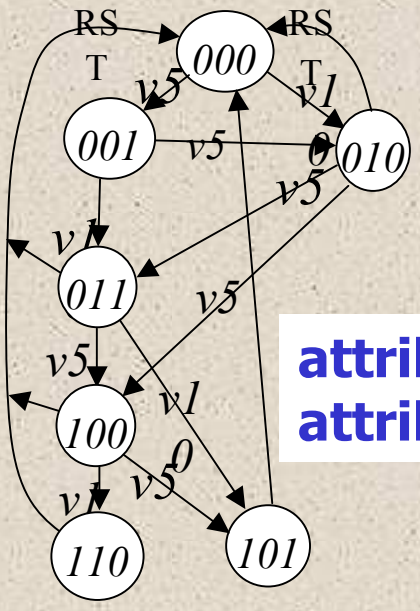
User Constraint File (UCF) – пише користувач,

Netlist Constraint File (NCF) – пише синтезатор

- **Вставлення вказівок компілятору (-- pragma …)**
- **Вставлення атрибутів користувача**
- **Встановлення режимів САПР**

Атрибути керування синтезом

Керування розкриттям ієрархії проекту
Чи розпізнавати DC, MUX, RAM, ROM, схеми зсуву, автомати
Стиль виконання RAM, ROM, блоків множення
Чи виконувати конвейеризацію і яким чином
Чи виконувати ресинхронізацію і дублювання регістрів
Оптимізувати за швидкістю і/або площі розміщення
Максимальний ступінь розгалуження (fanout)
Кодування станів автомата і т.і.




```
attribute enum_encoding : string;
attribute enum_encoding of STATE : type is
"000 001 010 011 100 101 110" ;
```

```
attribute fsm_encoding: string;
attribute fsm_encoding of STATE :signal is " one-hot";
```

Атрибути обмеження розміщення

Можна обмежувати/призначати розміщення:

-Виводам мікросхеми

-Основним компонентам – LUT, RG, RAM і т.і.

-Вставленим компонентам проекта

-Надавати відносні координати і можливий діапазон

-Призначати тип, характеристики (струм, швидкість, режим) елемента (буфера, лінії зв”язку) сигналу

Приклад призначення виводів мікросхеми

```
attribute loc: string;
attribute loc of CLK        :signal is "C13";
attribute loc of RST      :signal is "L3"; --SW3
attribute loc of LED      :signal is "AF3"; --LED
```

Атрибути ініціалізації

Можна ініціалізувати: LUT, RG, RAM, ROM.

```
architecture distrib_rom of rom16 is
 component  ROM16X1 is
 --pragma translate_off
  generic (INIT : bit_vector);
  --pragma translate_on
      port (A0,A1,A2,A3 : in std_ulogic;
      O : out std_ulogic );
      end component;
  attribute init:  string;
  attribute init of U_ROM: label is X"AB56";
begin
U_ROM:ROM16X1
--pragma translate_off
  generic map(INIT=>X"AB56");
  --pragma translate_on
  port map(A0,A1,A2,A3, DO);
 end distrib_rom;
```

Приклад постійної Пам’яті

Обмеження часу

Задається тільки в файлах обмежень

- **Обмеження періоду синхросерії, часу попереднього встановлення вхідного сигналу**
- **Обмеження затримки від виходу одного елементу пам”яті до входу іншого (регістр, RAM,ROM) для одного сигналу або групи сигналів**
- **Позначення сигналів як такі, через які походить багатотактовий маршрут (false path)**