Špecifikácia konfigurácie (Configuration Specifications)

Špecifikácia konfigurácie umožňuje špecifikovať výber deklarácie entity a tela architektúry pre danú inštanciu komponentu.

Všeobecný opis syntaxe *špecifikácie konfigurácie* má tvar:
configuration_specification ::=
  for instantiation_list : *component*_name binding_indication;

instantiation_list ::= *instantiation*_label {, *instantiation*_label}
  | **others** | **all**
binding_indication ::=
  [**use** entity_aspect]
  [**generic map** (*generic*_association-list)]
  [**port map** (*port*_association-list)]
entity_aspect ::=
  entity *entity*_name [(*architecture*_identifier)]
  | **configuration** *configuration*_name | **open**

```
entity JKFF is
    port(CLK, PRESET, CLEAR, J, K: Bit; Q, Q_bar: out Bit);
end JKFF;
package Global_signals is
    signal CLK: Bit;
    signal PRESET: Bit;
    signal CLEAR: Bit;
end Global_signals;
entity Design is
end Design;
use Work.Global_signals;
architecture Design of Design is
    signal S1, S2, S3, S4: Bit;
    component FF
        port (J, K: Bit; Q, Q_bar: out Bit);
    end component;
```

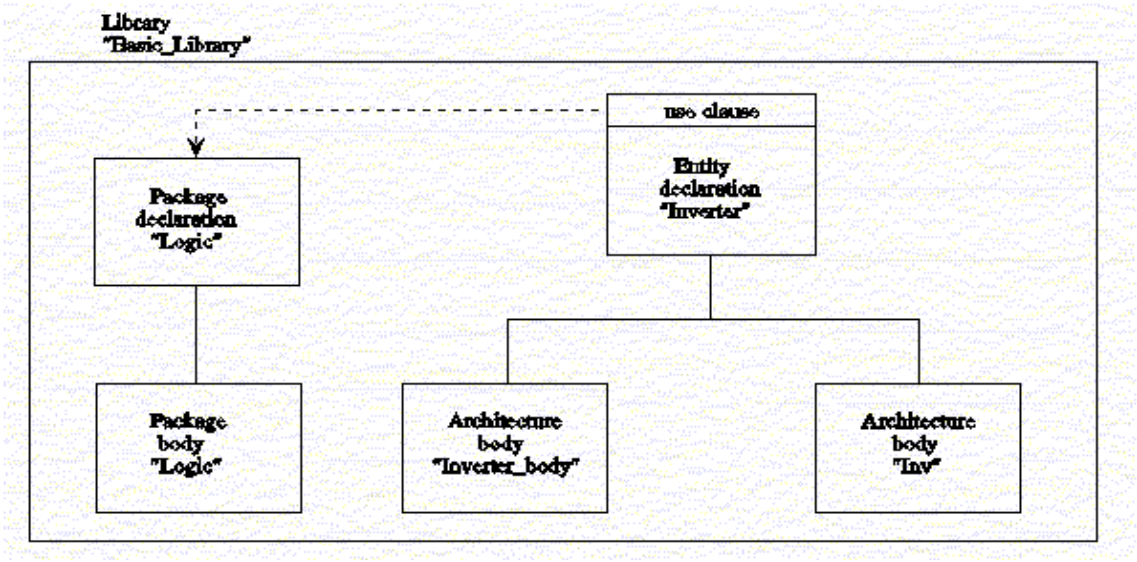
```
    for U0: FF use entity Work.JKFF
        port map(CLK => Global_signals.CLK,
                 PRESET => Global_signals.PRESET,
                 CLEAR => Global_signals.CLEAR,
                 J=>J, K=>K, Q=>Q, Q_bar=>Q_bar);
begin
    U0: FF port map (S1, S2, S3, S4);
end Design;



entity Inverter is
    port (I1: in Bit; O1: out Bit);
end Inverter;
entity Inverter_user is
end Inverter_user;
architecture Inverter_user of Inverter_user is
    signal S: Bit;
    signal S_bar: Bit;
    component Inv
        port(In1: in Bit; Out1: out Bit);
    end component;
    for U1: Inv use entity Work.Inverter(Inverter_body)
        port map (I1 => In1, O1 => Out1);
begin
    U1: Inv port map (S, S_bar);
end Inverter_user;
architecture Inverter_body of Inverter is
begin
    O1 <= not I1 after 5 ns;
end Inverter_body;
```

Deklarácia konfigurácie (Configuration Declaration)

```
configuration Parts of Inverter_user is
    for Inverter_user
        for U1:     Inv
                    use entity Work.Inverter(Inverter_body)
                    port map (I1 => In1, O1 => Out1);
        end for;
    end for;
end Parts;

configuration FULL_SLOT of COMM_BOARD is
    attribute KIND of COMM_BOARD: entity is FULL_SIZE;
    for LOW_COST -- an architecture of COMM_BOARD
        for CPU: PROCESSOR
            use entity STD_PARTS.SPARC (Fujitsu)
                generic map (Clock => 40ns);
        end for; -- for PROCESSOR
        for BUS_CONTROLLER
            for Protocol_Chip: IEEE_488
                use entity STD_PARTS.NEC;
            end for;
        end for; -- for BUS_CONTROLLER
        ...
    end for; -- for LOW_COST
end FULL_SLOT;

for CPU: PROCESSOR use entity STD_PARTS.SPARC (Fujitsu)
   generic map (Clock => 40ns);
      for Fujitsu
         -- component configurations for instantiations,
         -- or block configurations for blocks in Fujitsu
      end for;
end for; -- for PROCESSOR
```

```
for B1: BUS_CONTROLLER
   use entity COMPANY_LIBRARY.STD_CONTROLLER (Std_body);
      for Std_body
         for Protocol_Chip: IEEE_488
            use entity STD_PARTS.NEC;
         end for;
      end for;
end for;


for B:COMM_BOARD
         use configuration HALF_SLOT;
     end for;
```

KNIŽNICE



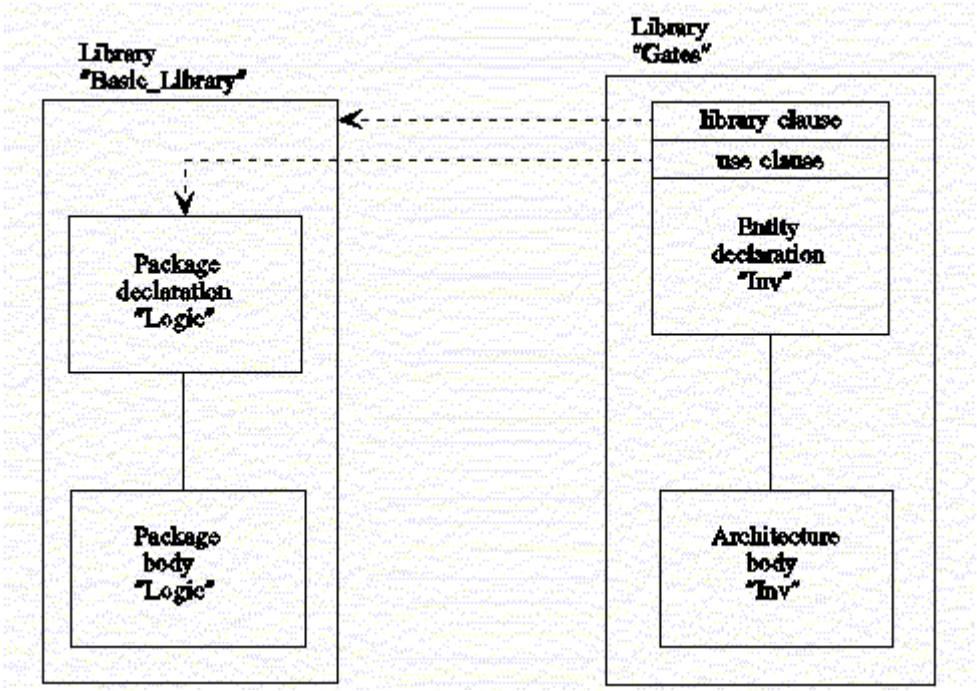
use Basic_library.Logic;

use Logic.**all;**

entity Inv **is**

port(X:**in** Three_level_logic;Y:**out** Three_level_logic);

end Inv;




```
library Basic_library;
use Basic_library.Logic;
use Logic.all;
entity Inverter is
port(X:in Three_level_logic;Y:out Three_level_logic);
end Inverter;
```

Konštrukcie, ktoré vytvárajú nový *deklaračný región:*

- Deklarácia entity a k nej priradená architektúra.
- Príkaz blok.
- Príkaz proces.
- Balík.
- Podprogram.
- Konfigurácia.
- Typy záznam.
- Príkaz cyklu.
- Deklarácia komponentu.

```
-- a block declaration
B1: block
    -- containing a signal declaration
    signal SUM, I1, I2: Bit;
```

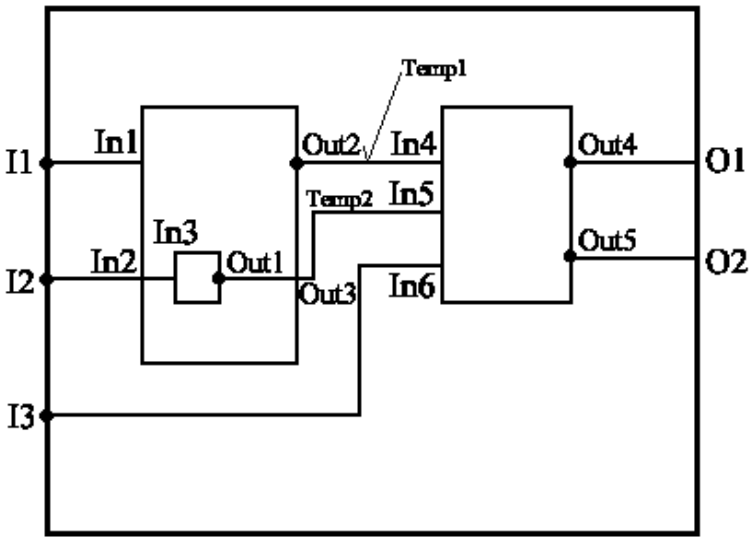
```
begin
    -- a nested block declaration
    B2: block
        -- This SUM hides the one in block B1,
        signal SUM: Bit;
    begin
        -- so this assigns to the SUM in B2,
        SUM <= I1 and I2 after 5.5 ns;
        -- whereas this would assign to the SUM in B1.
        -- B1.SUM <= I1 and I2 after 5.5 ns;
    end block;
    -- And this assigns to the SUM in B1.
    SUM <= I1 and I2 after 5.5 ns;
end block;
```

Ilustračný príklad:

```
package SIG is
    signal X: INTEGER:= 1;
end;
use WORK.SIG;
entity Y is
    signal X: INTEGER:= 2;
end Y;
architecture Z of Y is
    signal Z1, Z2, Z3, Z4, Z5: INTEGER:= 0;
    function R return INTEGER is
        variable X: INTEGER:= 3;
    begin
        return X;
    end R;
begin
    B: block
        signal X: INTEGER:= 4;
        signal Z6: INTEGER:= 0;
    begin
        Z6 <= X + Y.X;
    end block B;
    process
        variable X: INTEGER:= 5;
    begin
        Z5 <= X;
        wait;
    end process;
    Z1 <= WORK.SIG.X;
    Z2 <= X;
    Z3 <= R;
    Z4 <= B.X;
```

end Z;

Náhradné hodnoty (Default Values)




```
package Three_level_logic is
    type Three is ('X', '0', '1');
    function Inv (Q: Three) return Three;
end Three_level_logic;
package body Three_level_logic is
    function Inv (Q: Three) return Three is
    begin
        case Q is
            when 'X' =>
                return 'X';
            when '0' =>
                return '1';
            when '1' =>
                return '0';
        end case;
    end Inv;
end Three_level_logic;
use Work.Three_level_logic.all;
entity Inverter is
    port (P1: Three := 'X'; P2: out Three := 'X');
```

```
end Inverter;
use Work.Three_level_logic.all;
entity Buf is
    port (P1: Three := 'X'; P2: out Three := 'X');
end Buf;
architecture Inverter of Inverter is
begin
    P2 <= Inv (P1) after 5 ns;
end Inverter;
architecture Buf of Buf is
begin
    P2 <= P1 after 5 ns;
end Buf;
use Work.Three_level_logic.all;
entity Oscilator is
end Oscilator;
architecture Oscilator of Oscilator is
    component Inverter
        port (P1: Three; P2: out Three);
    end component;
    component Buf
        port (P1: Three; P2: out Three);
    end component;
    for all : Inverter use entity Work.Inverter(Inverter);
    for all : Buf use entity Work.Buf(Buf);
    signal S1 : Three := '0';
    signal S2 : Three := '1';
begin
    U1 : Inverter port map (S1, S2);
    U2 : Buf port map (S2, S1);
end Oscilator;
```

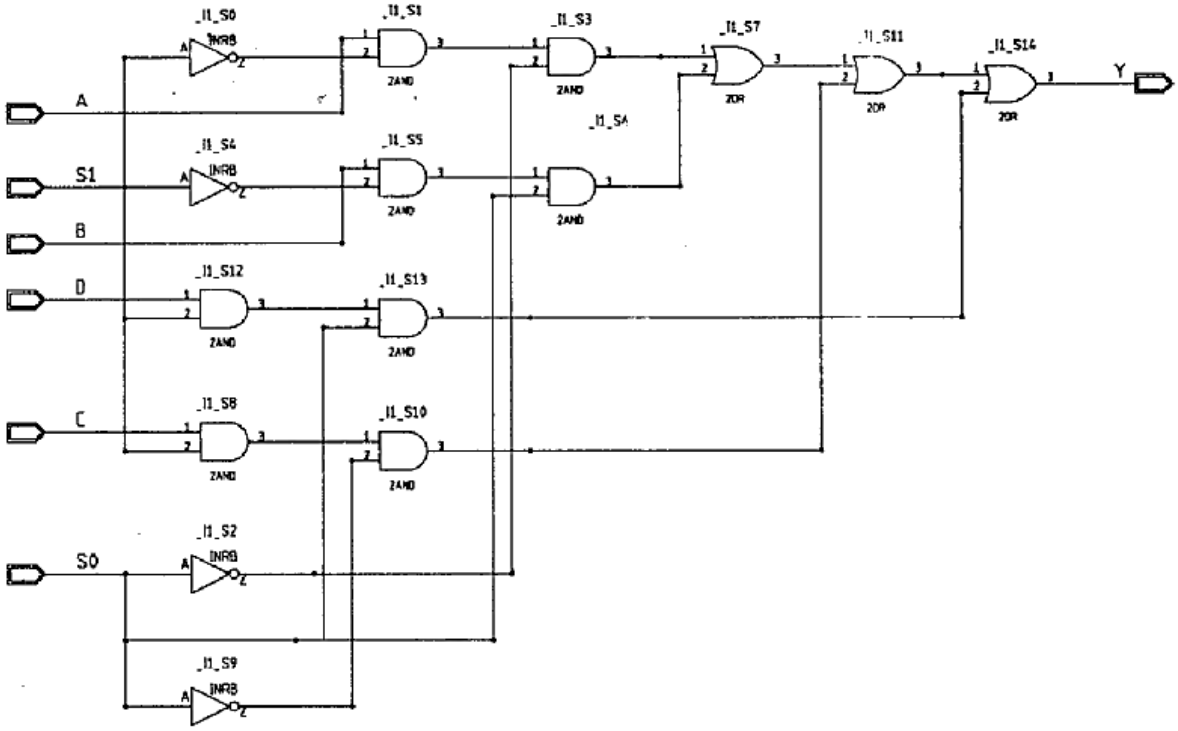
Nekompatibilita typov

```
function Bit4_to_BIT (b: in Bit4) return BIT
is begin
    case b is
        when '0' => return '0';
        when '1' => return '1';
        when 'X' => return '0';
        when 'Z' => return '0';
    end case;
end Bit4_to_BIT;

function BIT_to_Bit4 (b: in BIT) return Bit4
is begin
    case b is
        when '0' => return '0';
        when '1' => return '1';
    end case;
end BIT_to_Bit4;

signal C: Bit;
port (CLEAR: Bit4…..);
port map (CLEAR => BIT_to_Bit4( C) ……);

signal FF_OUT: Bit;
port (Q: inout Bit4…..);
port map (Bit4_to_BIT( Q) => BIT_to_Bit4( FF_OUT)….);
```

Návrh kombinačnej logiky a automatická syntéza

Multiplexor

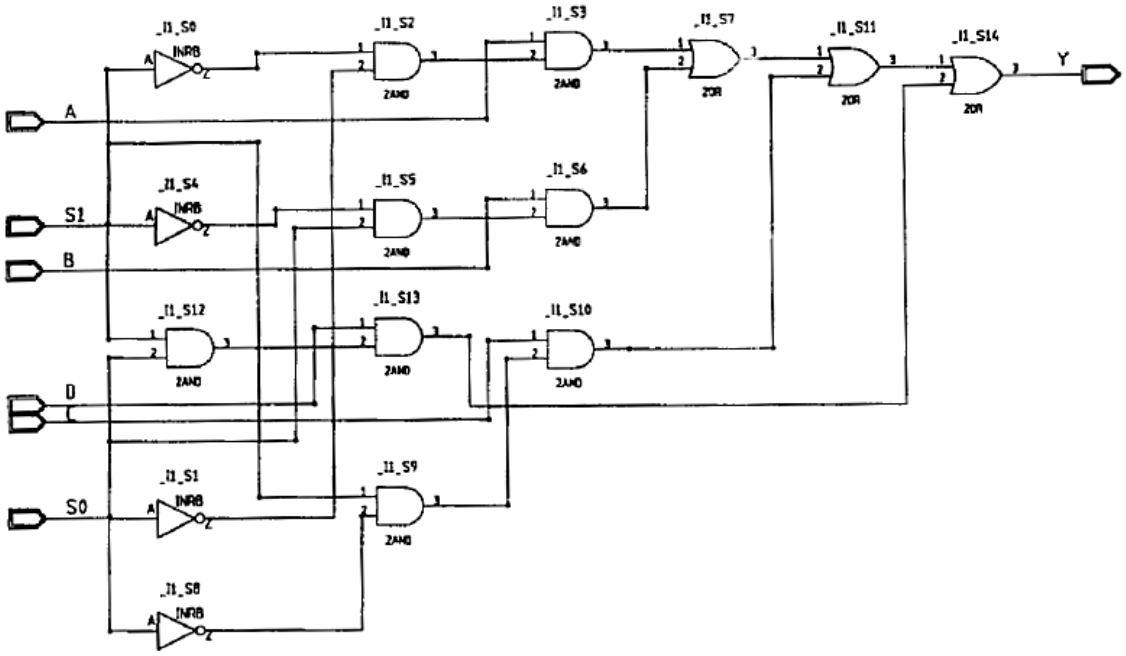
```
entity MUX4TO1 is
   port(   A,B,C,D,S0,S1 : in BIT;
           Y : out BIT );
end MUX4TO1;

--  Concurrent signal assignment
architecture DATAFLOW1 of MUX4TO1 is
begin
   -- parentheses required when mixing operators of equal
precedence
   Y<= (A and not S1 and not S0) or (B and not S1 and S0) or
      (C and S1 and not S0) or (D and S1 and S0);
```



DATAFLOW1 synthesized circuit.

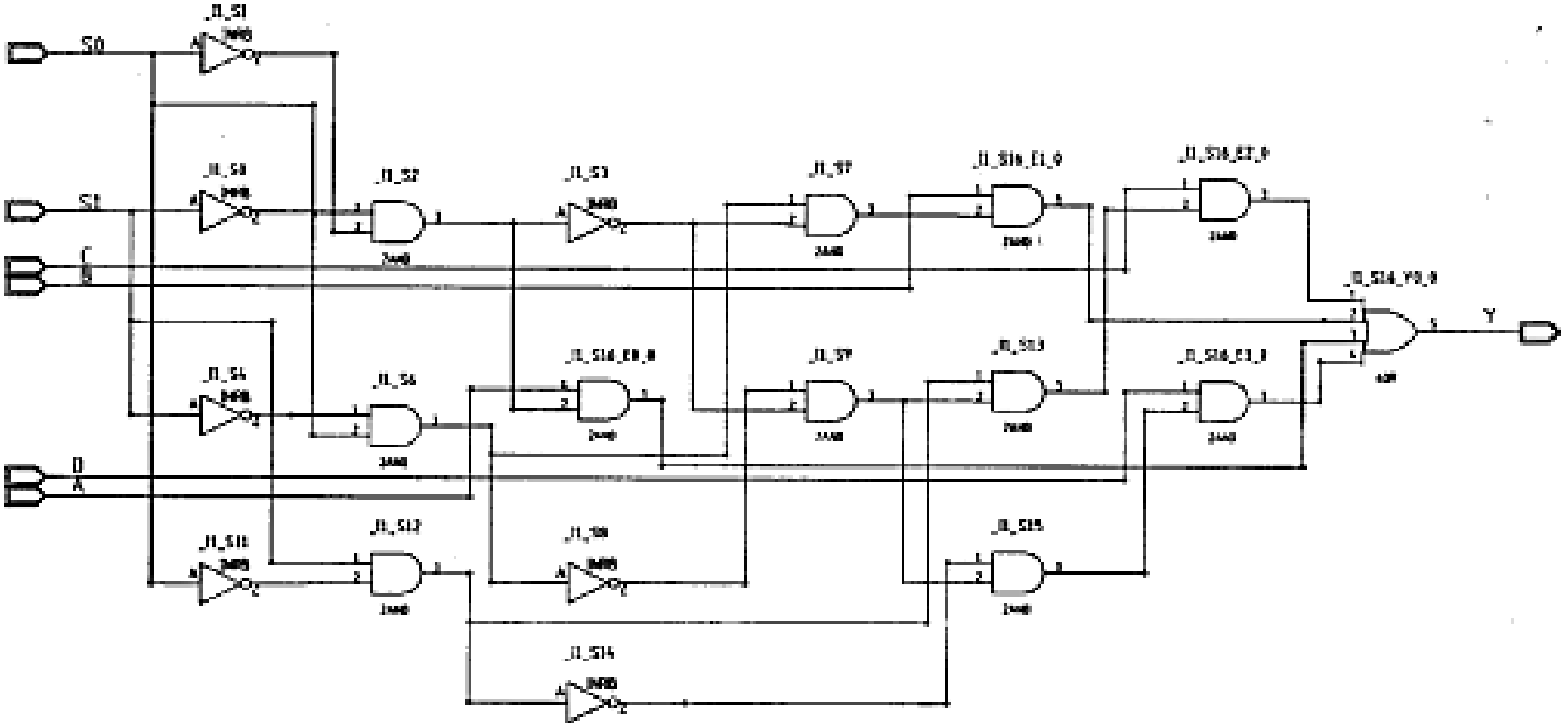
```
end DATAFLOW1;
```

```
--   Concurrent signal assignment with additional parentheses
architecture DATAFLOW2 of MUX4TO1 is
begin
   -- extra parentheses control the order of evaluation/structure of the
circuit
   Y<= (A and (not S1 and not S0)) or (B and (not S1 and S0)) or
      (C and (S1 and not S0)) or (D and (S1 and S0));
```



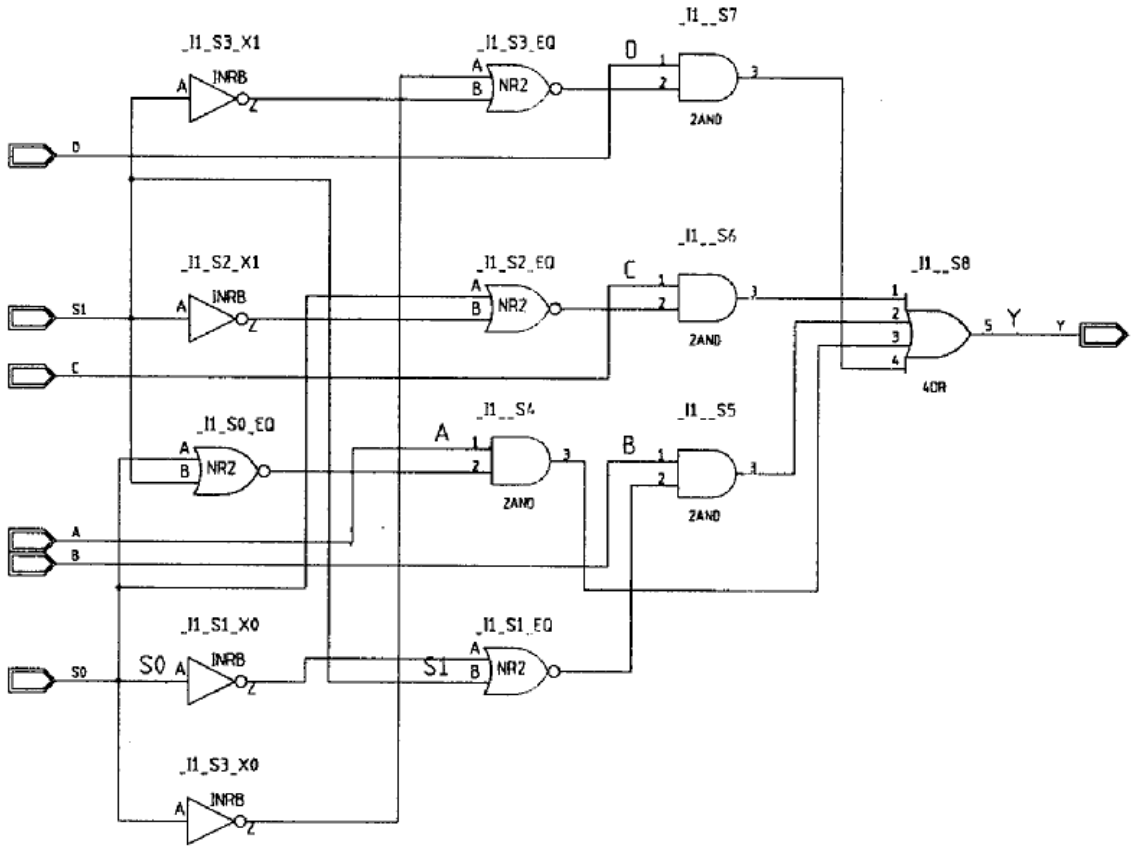
DATAFLOW2 synthesized circuit.

```
end DATAFLOW2;

--   Conditional signal assignment
architecture DATAFLOW3 of MUX4TO1 is
begin
   -- each branch condition may be different
   Y<= A when (S1='0' and S0='0') else
      B when (S1='0' and S0='1') else
      C when (S1='1' and S0='0') else
      D;
end DATAFLOW3;
```

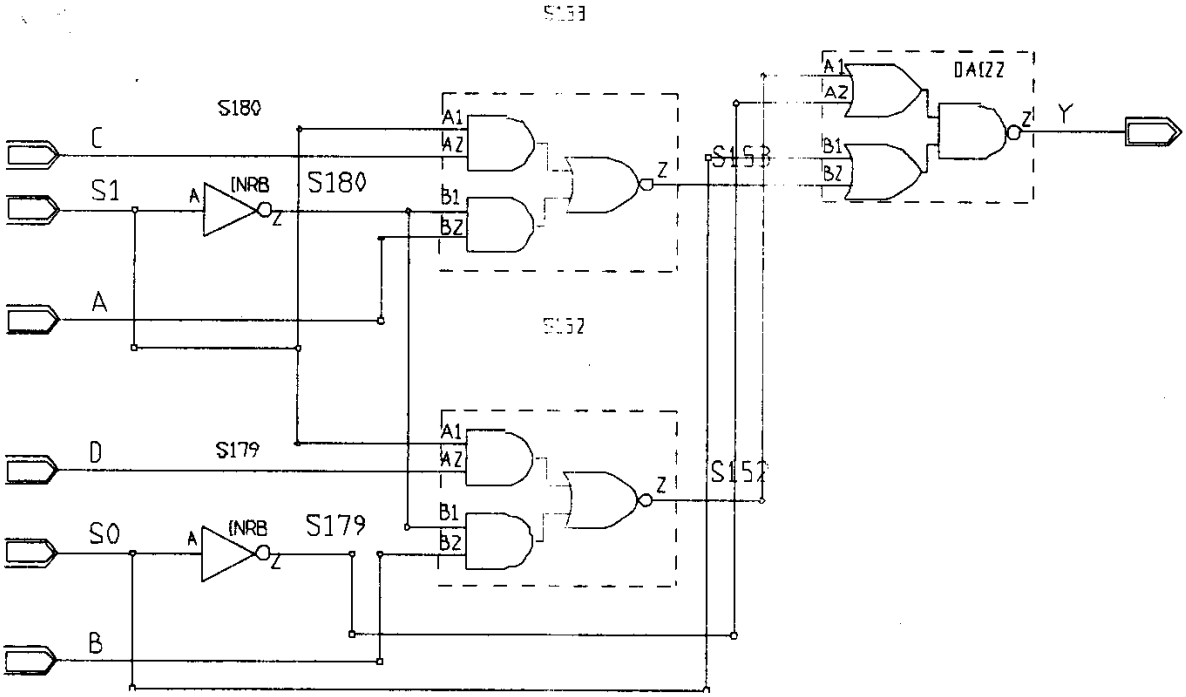


DATAFLOW3 synthesized circuit.

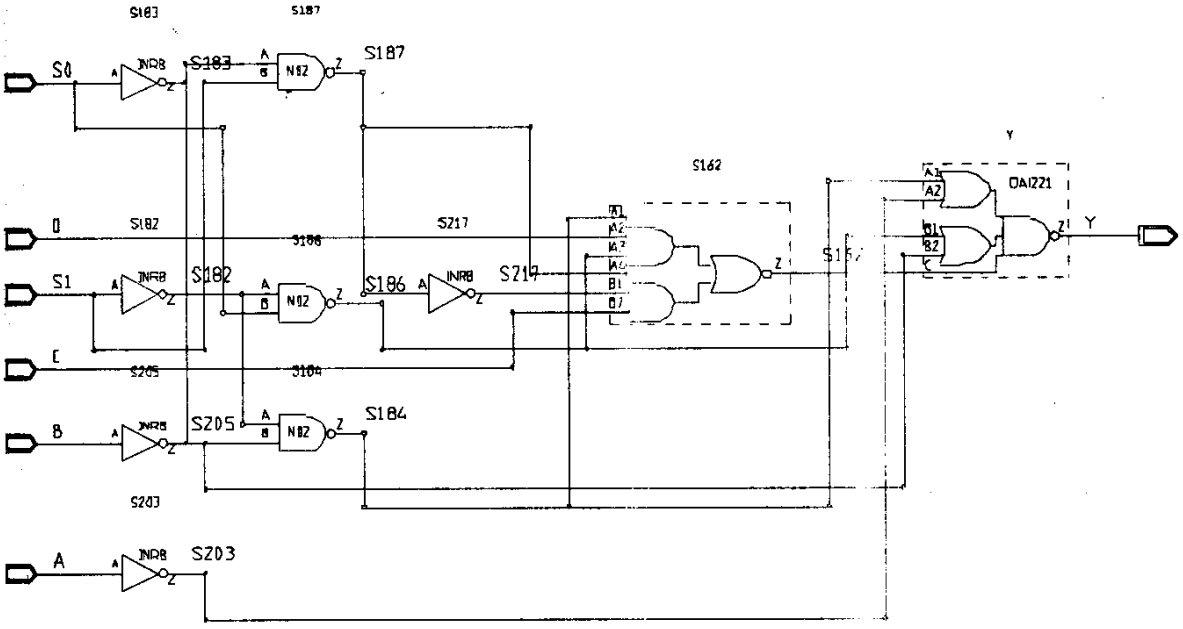
```
--   Selected signal assignment
architecture DATAFLOW4 of MUX4TO1 is
signal SEL : BIT_VECTOR(1 downto 0);
begin
   -- each branch tests for a different value of the same condition
   with SEL select
      Y <=   A when 00,
             B when 01,
             C when 10,
             D when 11;
   SEL <= S1&S0;
end DATAFLOW4;
```



DATAFLOW4 synthesized circuit.



DATAFLOW1, DATAFLOW2 and DATAFLOW4 optimized circuit.

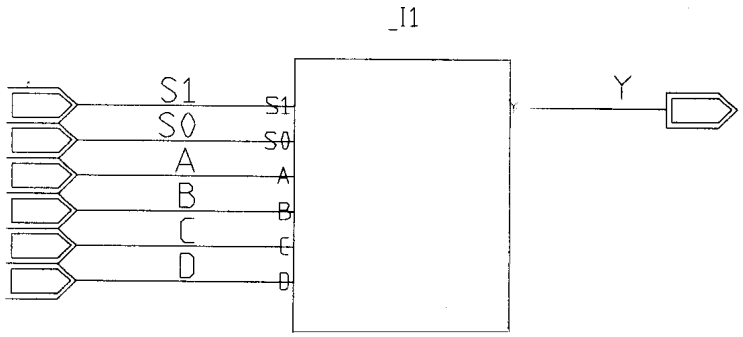


DATAFLOW3 optimized circuit.

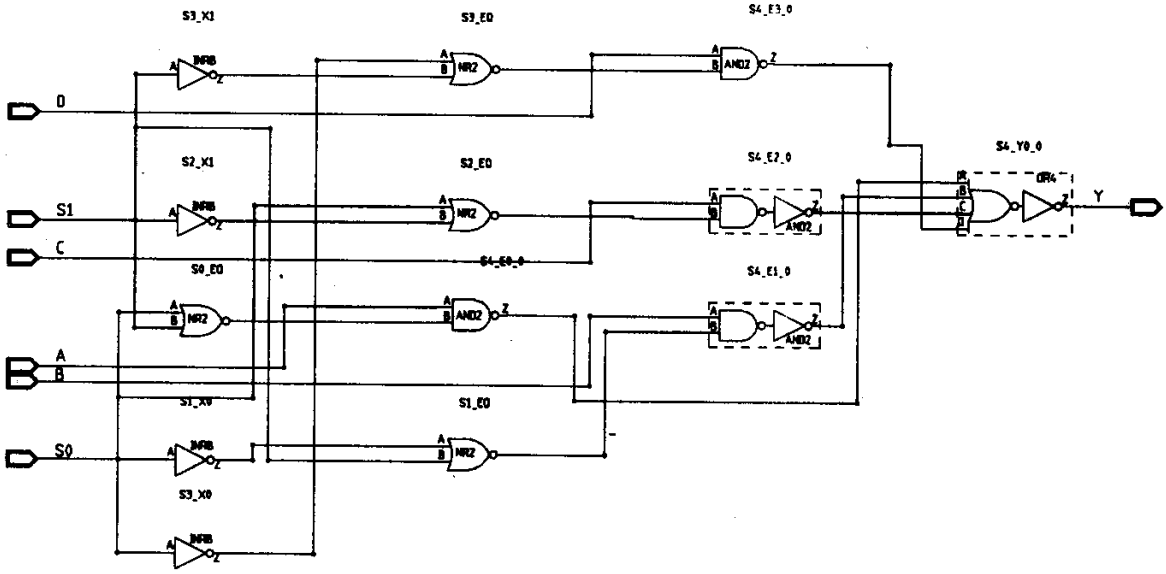
Table 4.4 Logic synthesis and optimization statistics for the dataflow style architecture

Parameters *Temperature: 25.0 °C* *Voltage: 5.0 V*	*Synthesis* *Library: generic*	*Optimization* *Type: area* *Level: low*
Circuit: MUX4TO1-DATAFLOW1		
Cells	15	5
Transistors	74	28
Area	9154.08	3344.76
Longest input to output delay	(S1 to Y) 3.171 ns	1.568 ns
Circuit : MUX4TO1-DATAFLOW2		
Cells	15	5
Transistors	74	28
Area	9154.08	3344.76
Longest input to output delay	(S0 to Y) 3.171 ns	1.568 ns
Circuit: MUX4TO1-DATAFLOW3		
Cells	19	10
Transistors	90	44
Area	11266.56	5809.32
Longest input to output delay	(S0 to Y) 4.619 ns	2.967 ns
Circuit: MUX4TO1-DATAFLOW4		
Cells	13	5
Transistors	58	28
Area	7393.68	3344.76
Longest input to output delay	(S1 to Y) 2.292	1.568

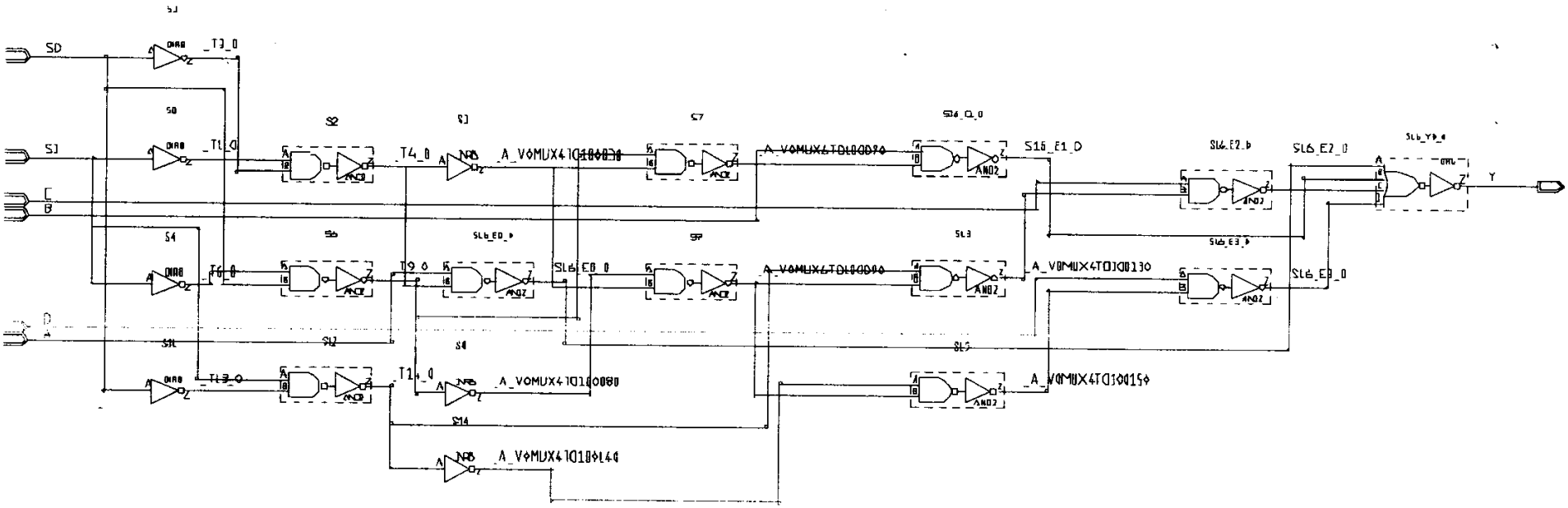
```
entity MUX4TO1 is
   port (A, B, C, D : in BIT;
         S : in BIT_VECTOR(1 downto 0); -- array type inputs
         Y : out BIT );
end MUX4TO1;

architecture CASE1 of MUX4TO1 is
begin
   process (A,B,C,D,S(1 downto 0))
   begin
      case S is
         when "00" =>  Y <= A;
         when "01" =>  Y <= B;
         when "10" =>  Y <= C;
         when "11" =>  Y <= D;
      end case;
   end process;
end CASE1;
```



Top-level circuit for CASE and IF_ELSE architectures.



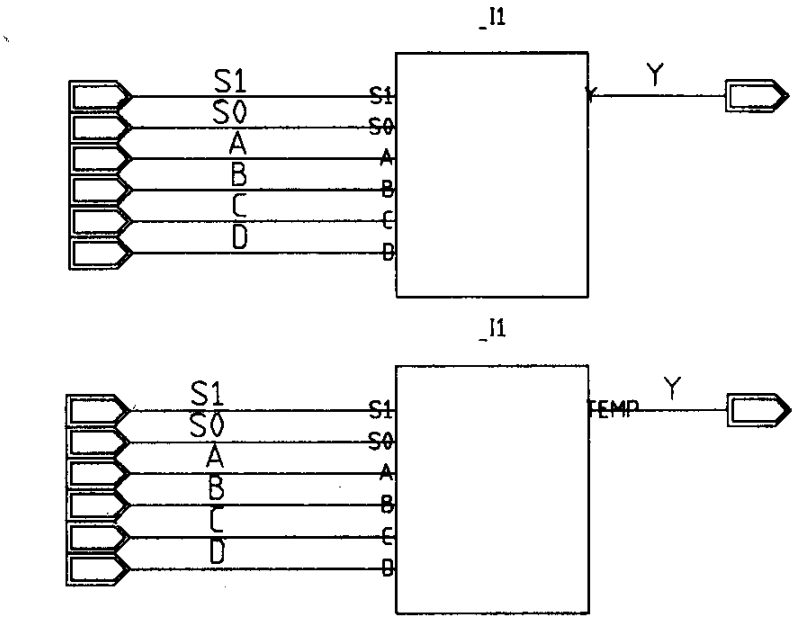
Second level of the synthesized CASE1 circuit.



Second level of the synthesized IF_ELSE circuit.

```
architecture IF_ELSE of MUX4TO1 is
begin
   process (A,B,C,D,S(1 downto 0))
   begin
      if (S(1)='0' and S(0)='0') then
         Y <= A;
      elsif (S(1)='0' and S(0)='1') then
         Y <= B;
      elsif (S=“10“) then
         Y <= C;
      else
         Y <= D;
      end if;
   end process;
end IF_ELSE;


-- The Case statement using a variable in the process
architecture CASE2 of MUX4TO1 is
begin
   process (A,B,C,D,S(1 downto 0))
   variable TEMP : BIT;
   begin
      case S is
         when "00" =>  TEMP := A;
         when "01" =>  TEMP := B;
         when "10" =>  TEMP := C;
         when "11" =>  TEMP := D;
      end case;
      -- The variable is not visible outside of the process.
      Y <= TEMP;
   end process;
end CASE2;
```

```
-- The Case statement using a local signal
architecture CASE3 of MUX4TO1 is
   signal TEMP : BIT;
begin
   process (A,B,C,D,S(1 downto 0))
   begin
      case S is
         when "00" =>  TEMP <= A;
         when "01" =>  TEMP <= B;
         when "10" =>  TEMP <= C;
         when "11" =>  TEMP <= D;
      end case;
   end process;
-- A concurrent signal assignment is used to pass the result out to Y.
   Y <= TEMP;
end CASE3;
```



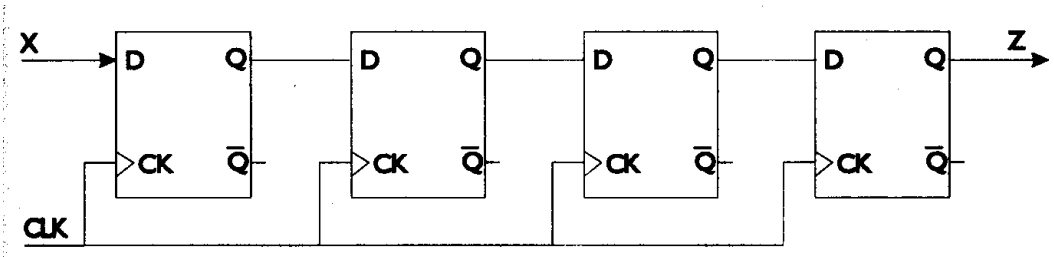
Top level circuits for the CASE2 and CASE3 architectures.

Table 4.5 Logic synthesis and optimization statistics for the behavioural style architecture

Parameters *Temperature: 25.0 °C* *Voltage: 5.0 V*	*Synthesis* *Library : generic*	*Optimization* *Type: area* *Level: low*
Circuit: MUX4TO1-CASE1		
Cells	13	5
Transistors	58	28
Area	7393.68	3344.76
Longest input to output delay	(S0 to Y) 2.292	1.568 ns
Circuit: MUX4TO1-IF_ELSE		
Cells	19	10
Transistors	90	44
Area	11266.56	5809.32
Longest input to output delay	(S0 to Y) 4.619 NS	2.967 ns
Circuit: MUX4TO1-CASE2		
Cells	13	5
Transistors	58	28
Area	7393.68	3344.76
Longest input to output delay	(S0 to Y) 2.292 NS	1.561 ns
Circuit: MUX4TO1-CASE3		
Cells	13	5
Transistors	58	28
Area	7393.68	3344.76
Longest input to output delay	(S0 to Y) 2.292 NS	1.561 ns

Návrh sekvenčnej logiky

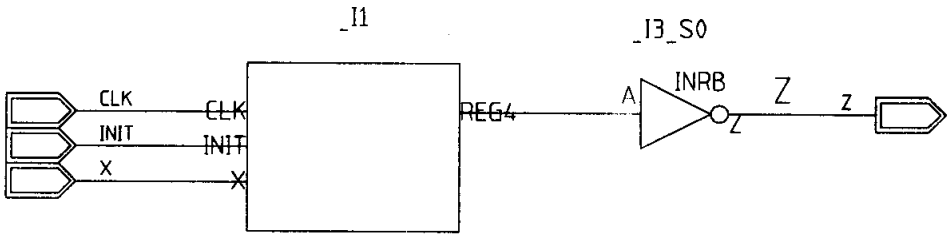
4-bitový posuvný register



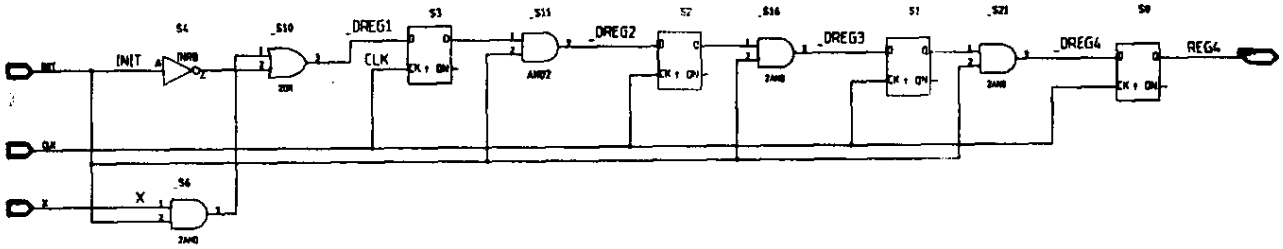
Simple 4-bit shift register.

```
--balík analyzovaný do knižnice USER_UTILS
package NEW_TYPE_DEFS is
   subtype HALF_REG is BIT_VECTOR(1 to 4);
end NEW_TYPE_DEFS;

library USER_UTILS;
use USER_UTILS.NEW_TYPE_DEFS.all;


entity FOUR_SHIFT is
   port (X : in BIT;
         CLK, INIT : in BIT;
         Z : out BIT );
end FOUR_SHIFT;


architecture STRING_ASSIGN of FOUR_SHIFT is
signal REG : HALF_REG;
begin
   process (X, CLK, INIT)
   begin
      if (not CLK'stable and CLK='1') then
         if INIT='1' then
            REG <= X & REG(1 to 3);
         else
            REG <= "1000";
         end if;
      end if;
   end process;
   Z <= not REG(4);
end STRING_ASSIGN;
```

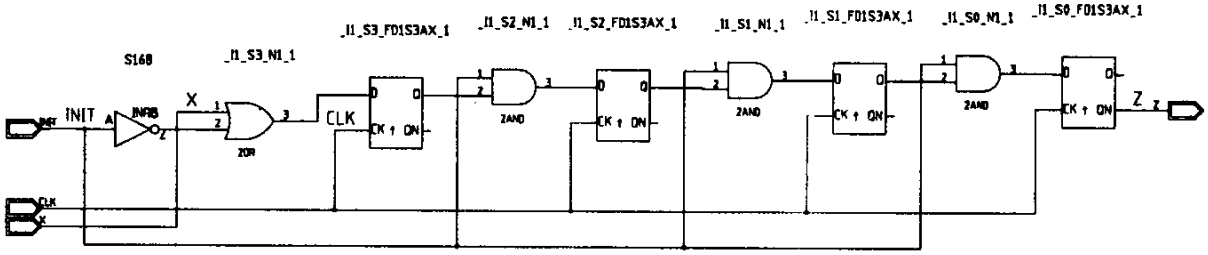
```
-- edge-triggered synchronous process
architecture SYNC_WAIT of FOUR_SHIFT is
signal REG : HALF_REG;
begin
   process
   begin
      wait until (CLK'event and CLK='1');
      if (INIT='1') then
         REG <= X & REG(1 to 3);
      else
         REG <= "1000";
      end if;
   end process;
   Z <= not REG(4);
end SYNC_WAIT;
```



Top-level synthesized circuits for the architectures STRING_ASSIGN, ASYNC_IF1, ASYNC_IF2 and SYNC_WAIT.

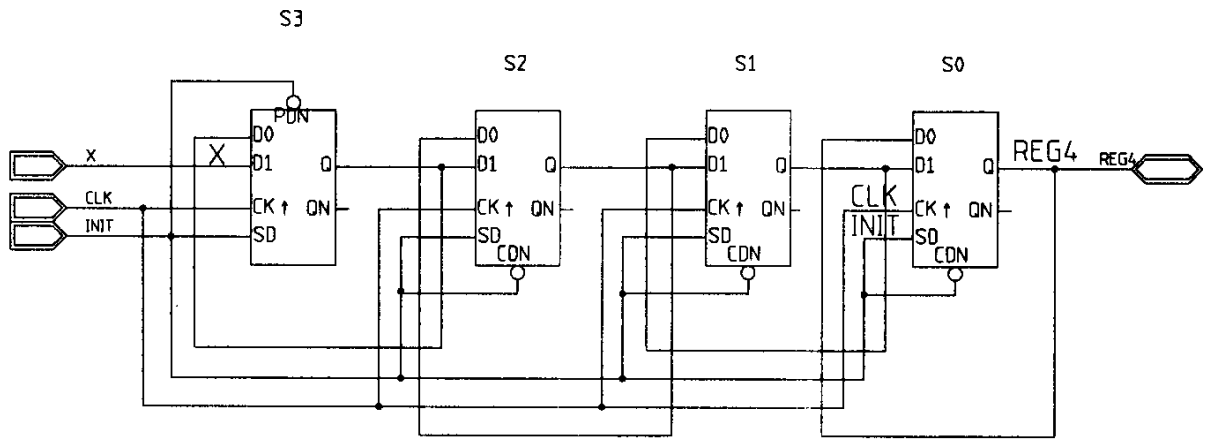


Synthesized circuit described by the process in both the STRING_ASSIGN and SYNC_WAIT architectures.

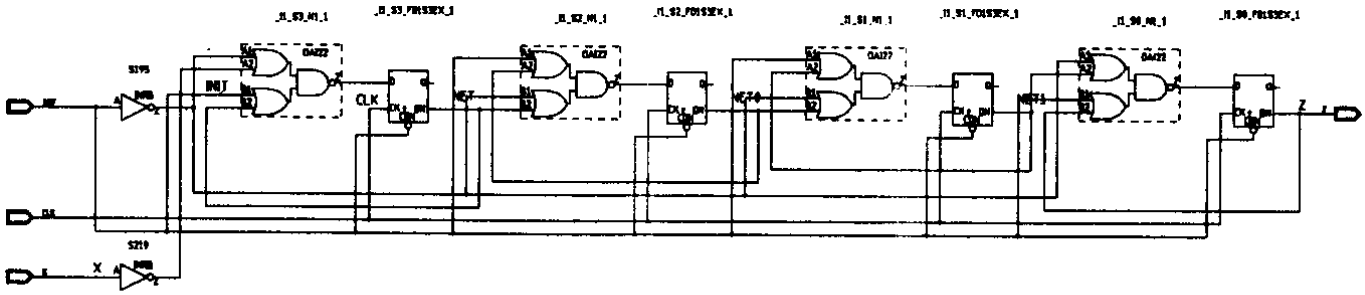


Optimized circuit for the STRING_ASSIGN and SYNC_WAIT architectures. Each two-level hierarchy has been flattened.

```
-- Independent asynchronous initialization
architecture ASYNC_IF1 of FOUR_SHIFT is
signal REG : HALF_REG;
begin
   process (X, CLK, INIT)
   begin
      if (INIT='0') then
         REG <= (1, others=>0);
      end if;
      if (CLK'event and CLK='1') then
         if (INIT='1') then
            REG <= X & REG(1 to 3);
         end if;
      end if;
   end process;
   Z <= not REG(4);
end ASYNC_IF1;
```

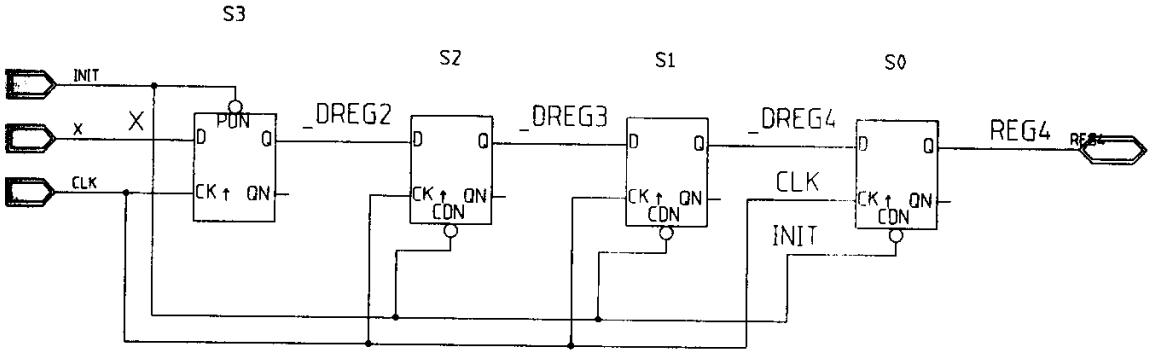


Synthesized circuit described by the process in the ASYNC_IF1 architecture.

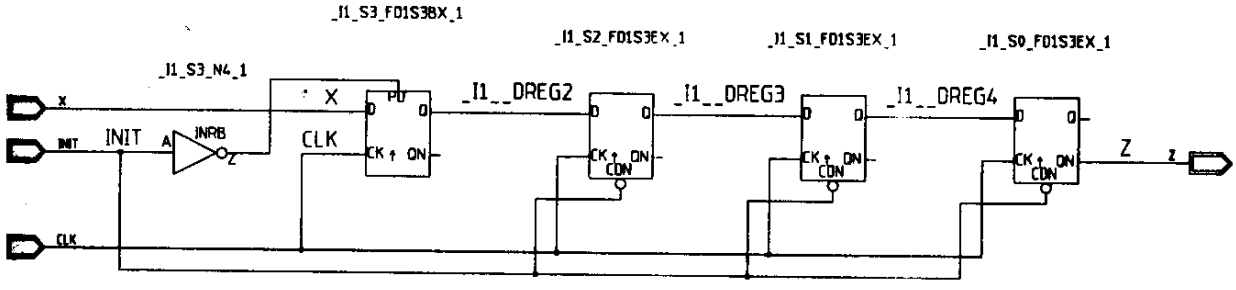


Optimized circuit (with flattened hierarchy) for the ASYNC_IF1 architecture.

```
-- Asynchronous initialization combined with the synchronous section
architecture ASYNC_IF2 of FOUR_SHIFT is
signal REG : HALF_REG;
begin
   process (X, CLK, INIT)
   begin
      if (INIT='0') then
         REG <= (1,0,0,0);
      elsif (CLK'event and CLK='1') then
         REG <= X & REG(1 to 3);
      end if;
   end process;
   Z <= not REG(4);
end ASYNC_IF2;
```



Synthesized circuit described by the process in the ASYNC_IF2 architecture.

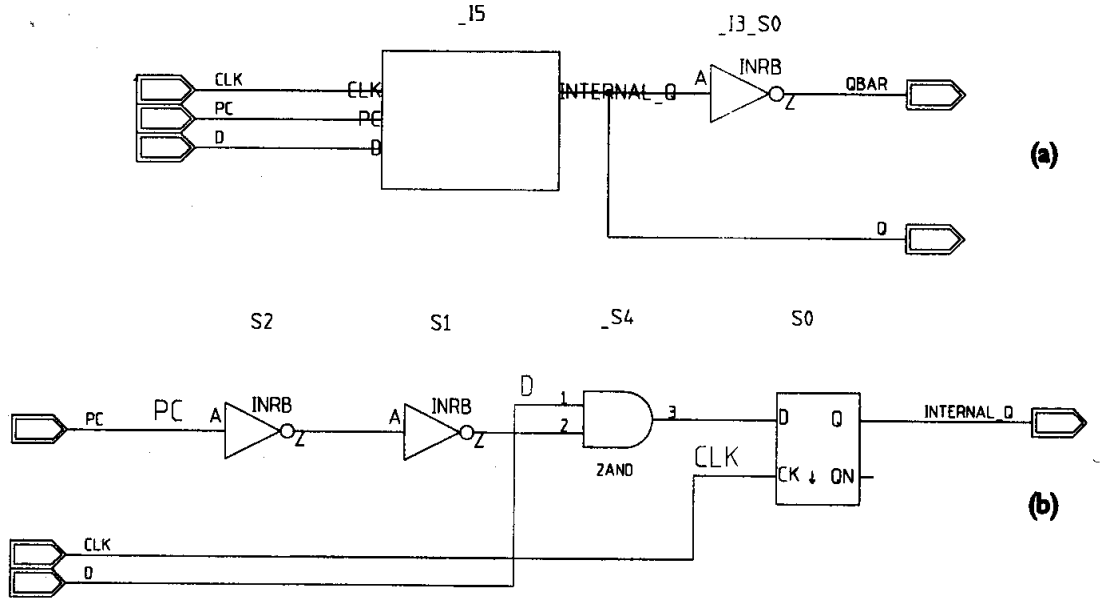


Optimized circuit (with flattened hierarchy) for the ASYNC_IF2 architecture.

Table 5.1 Logic synthesis and optimization statistics for two synchronous and two asynchronous behavioural style architectures

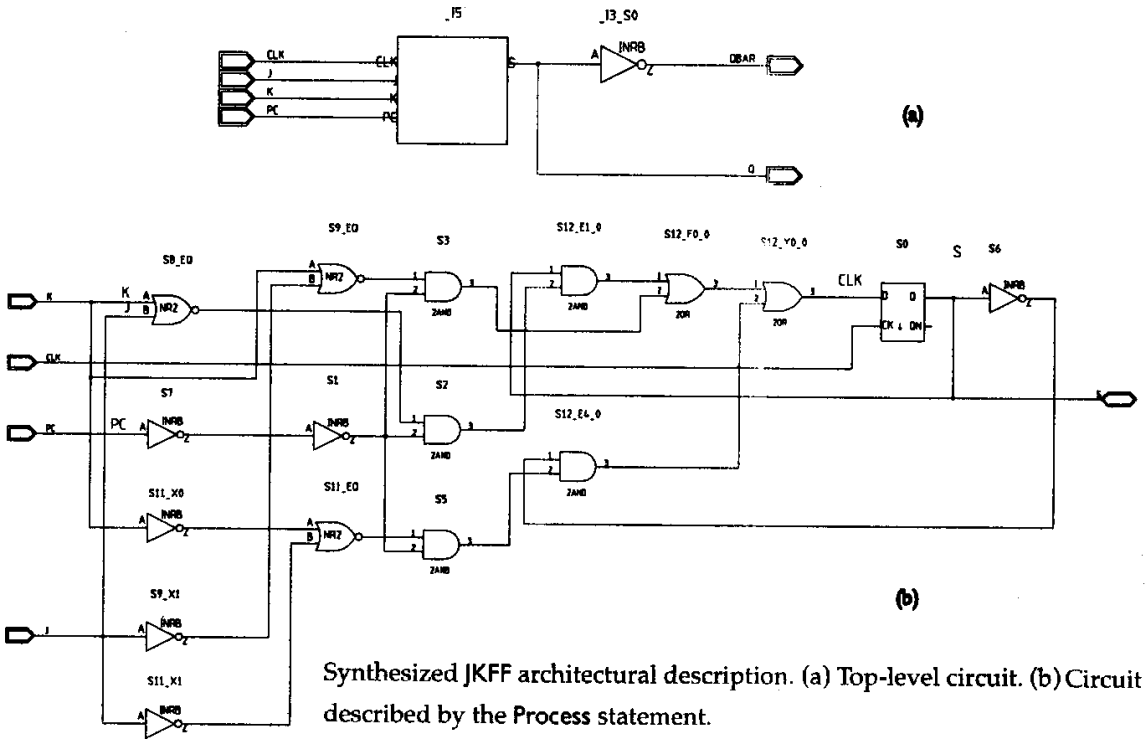
Parameters *Temperature: 25.0 °C* *Voltage: 5.0 V*	*Synthesis* *Library: generic*	*Optimization* *Type: area* *Level: low*
Circuit: FOUR_SHIFT-STRING_ASSIGN FOUR_SHIFT-SYNC_WAIT		
Cells	11	9
Transistors	106	98
Area	13379	12322
Longest latch to latch delay	2.297	No change
Circuit: FOUR_SHIFT-ASYNC_IF1		
Cells	5	10
Transistors	138	124
Area	15843	15491
Longest latch to latch delay	2.775	3.082
Circuit: FOUR_SHIFT-ASYNC_IF2		
Cells	5	No change
Transistors	92	90
Area	11794	11618
Longest latch to latch delay	1.813	No change

```
entity DFF is
   port (D : in BIT;
         CLK, PC : in BIT;
         Q, QBAR : out BIT);
end DFF;

-- D-type with synchronous clear, behavioral description
architecture BEHAVIOR of DFF is
signal INTERNAL_Q : BIT;
begin
   -- outputs assigned concurrently (outside) the process
   Q <= INTERNAL_Q;
   QBAR <= not INTERNAL_Q;
   process
   begin
      wait until (CLK'event and CLK='0');
      if (PC='0') then
         INTERNAL_Q <= '0'
      else
         INTERNAL_Q <= D;
      end if;
   end process;
end BEHAVIOR;
```



Synthesized DFF architectural description. (a) Top-level circuit. (b) Circuit described by the Process statement.

```
entity JKFF is
   port (J, K : in BIT;
         CLK, PC : in BIT;
         Q, QBAR : out BIT);
end JKFF;

architecture BEHAVIOR of JKFF is
signal S : BIT;
begin
   Q <= S;
   QBAR <= not S;
   process
   variable JK : BIT_VECTOR(0 to 1);
   begin
      wait until (CLK'event and CLK='0');
      JK := J & K;
      if (PC='0') then
         S <= '0';
      else
         case JK is
            when "00" =>S <= S;
            when "10" =>S <= '1';
            when "01" =>S <= '0';
            when "11" =>S <= not S;
         end case;
      end if;
   end process;
end BEHAVIOR;
```



Synthesized JKFF architectural description. (a) Top-level circuit. (b) Circuit described by the Process statement.

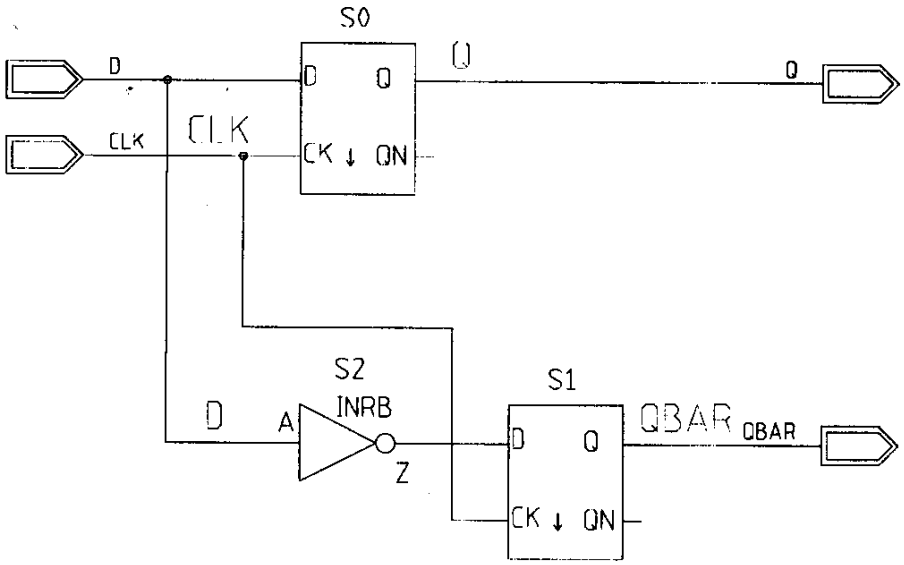
Table 5.3 Logic synthesis and optimization statistics for the D- and JK-type user-defined components

Parameters *Temperature: 25.0 °C* *Voltage: 5.0 V*	*Synthesis* *Library: generic*	*Optimization* *Type: area* *Level: low*
Circuit: DFF-BEHAVIOUR		
Cells	5	2
Transistors	30	24
Area	4048	2992
Longest input to latch delay	1.942	1.304
Circuit: JKFF-BEHAVIOUR		
Cells	18	4
Transistors	86	34
Area	11266	4224
Longest input to latch delay	4.214	2.294

Table 5.4 Logic synthesis and optimization statistics for the structural style architectures. Manually instantiated D-type and regular/irregular generated JK-type structures. All components are the optimized versions

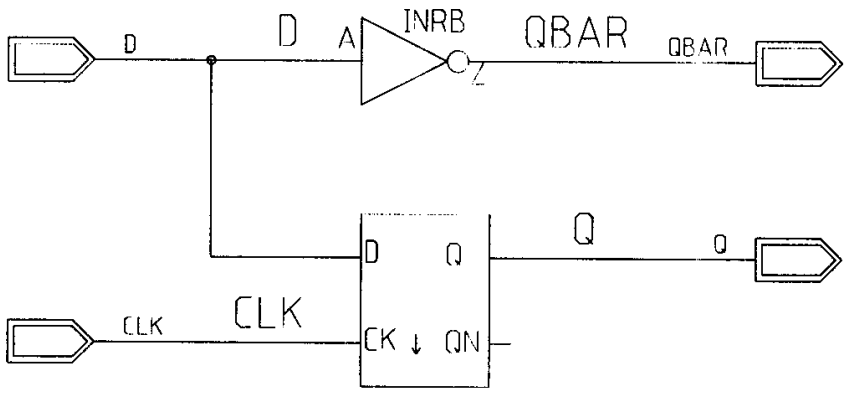
Parameters *Temperature: 25.0 °C* *Voltage: 5.0 V*	*Synthesis* *Library: generic*	*Optimization* *Type: area* *Level: low*
Circuit: FOUR_SHIFT-STRUCTURAL		
Cells	8	No change
Transistors	96	No change
Area	11970	No change
Longest latch to latch delay	2.684	No change
Circuit: FOUR_SHIFT-REG_GEN FOUR_SHIFT-IRREG_GEN		
Cells	17	13
Transistors	138	122
Area	17251	15139
Longest latch to latch delay	4.353	4.240

```
architectgure BAD1 of DFF is
begin
   process
   begin
      wait until (CLK'event and CLK= '0');
      Q <= D;
      QBAR <= not D;
   end process;
end BAD1;
```



Synthesized circuit produced by BAD1.

```
architectgure BAD2 of DFF is
begin
   process
   begin
      wait until (CLK'event and CLK= '0');
      Q <= D;
   end process;
   QBAR <= not D;
end BAD2;
```



Synthesized circuit produced by **BAD2**.

```
architectgure BAD_JK of JKFF is
signal S, SBAR : BIT;
begin
   process
   variable JK : BIT_VECTOR(0 to 1);
   begin
      wait until (CLK'event and CLK= '0');
      JK:= J&K;
      case JK is
         when "00" => S<= S;
         when "10" => S<= '1';
         when "01" => S<= '0';
         when "11" => S<= not S;
      end case;
      SBAR <= not S;
   end process;
   Q <= S;
   QBAR <= SBAR;
end BAD_JK;
```