Cosa è la Sintesi ?

- Sintesi è un termine generale che descrive il processo di trasformazione di un modello solitamente descritto in HDL, da un livello di astrazione ad uno più basso e più dettagliato.

- Queste trasformazioni sono finalizzate a raggiungere degli obiettivi di progetto (e.g., area, speed, power dissipation) soddisfacendo alcuni vincoli (e.g., I/O rates, sample period)

Livelli di sintesi

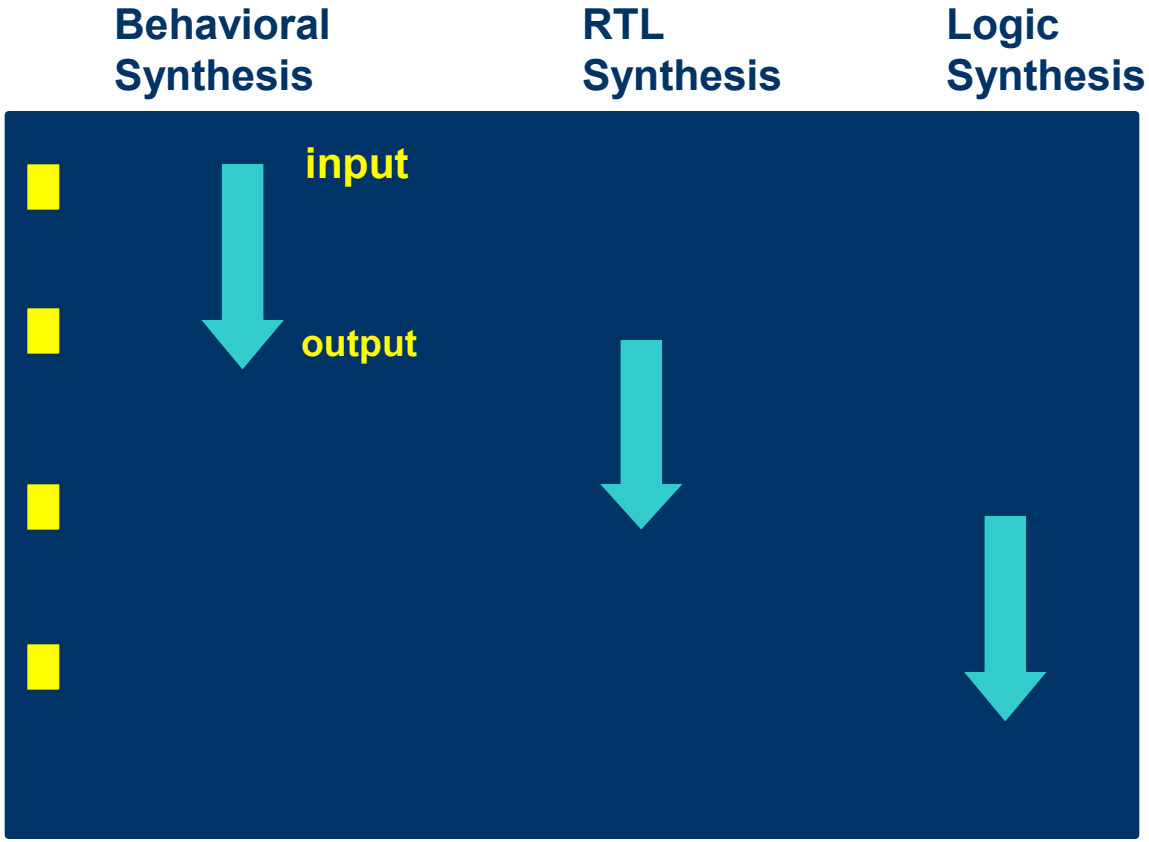
- Behavioral Synthesis: sintesi di una descrizione algoritmica di alto livello
  - Richiede “scheduling” e “assignment”
- Sintesi RTL (Register-Transfer Level) : sintesi di una descrizione definita a livello di registri
- Sintesi Logica: sintesi a partire da una descrizione in logica booleana

[Parker84]

Livelli di sintesi

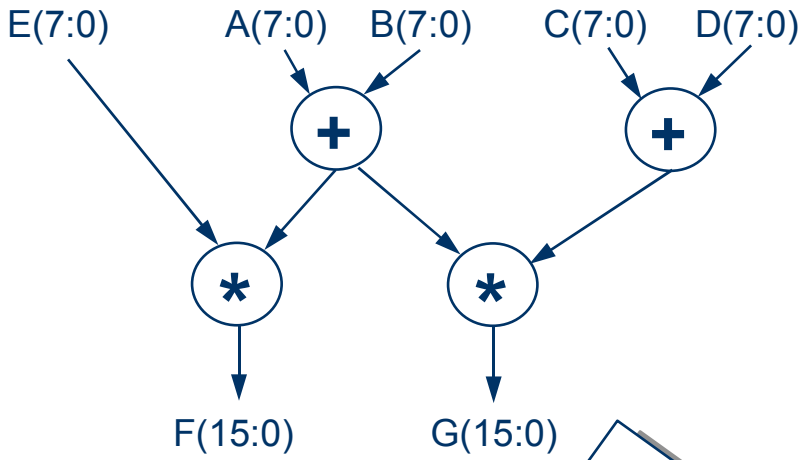
VHDL Descriptions at:

- Algorithmic Level
- Combinational logic, clocked registers, state machines, memories
- Boolean equations
- Structural gate net lists, technology libraries

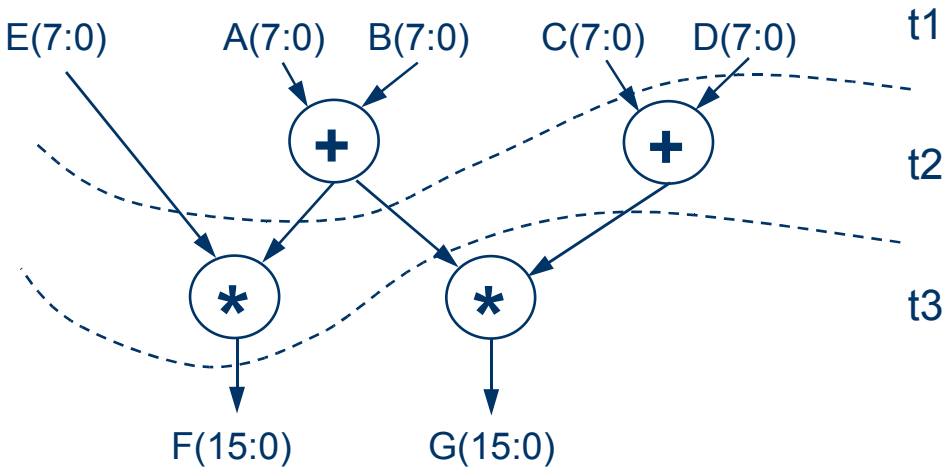


[Madisetti95]

Behavioral Synthesis

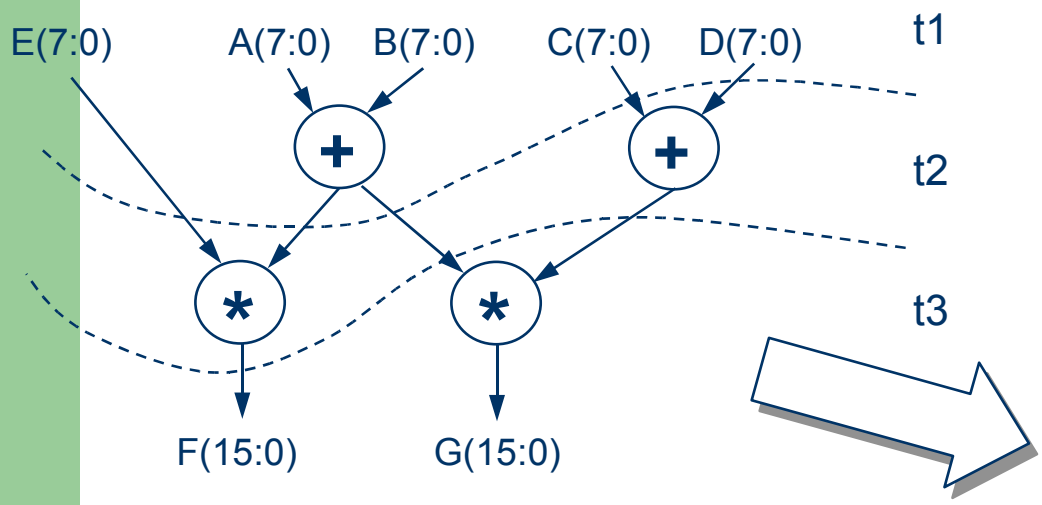


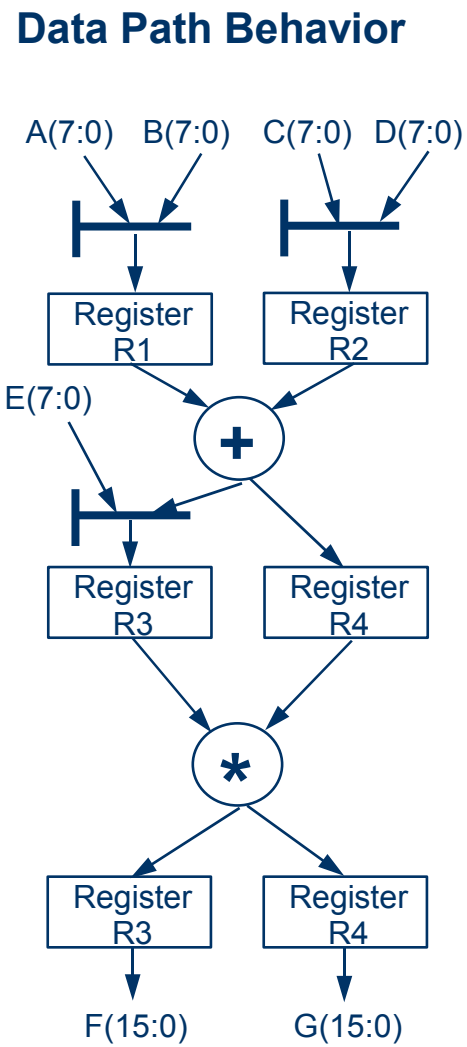

```
f <= (a + b) * e;
g <= (a + b) * (c + d);
```

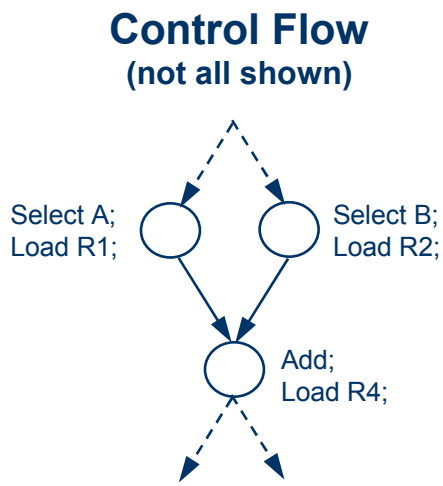


[Parker84]

Behavioral Synthesis (Cont.)

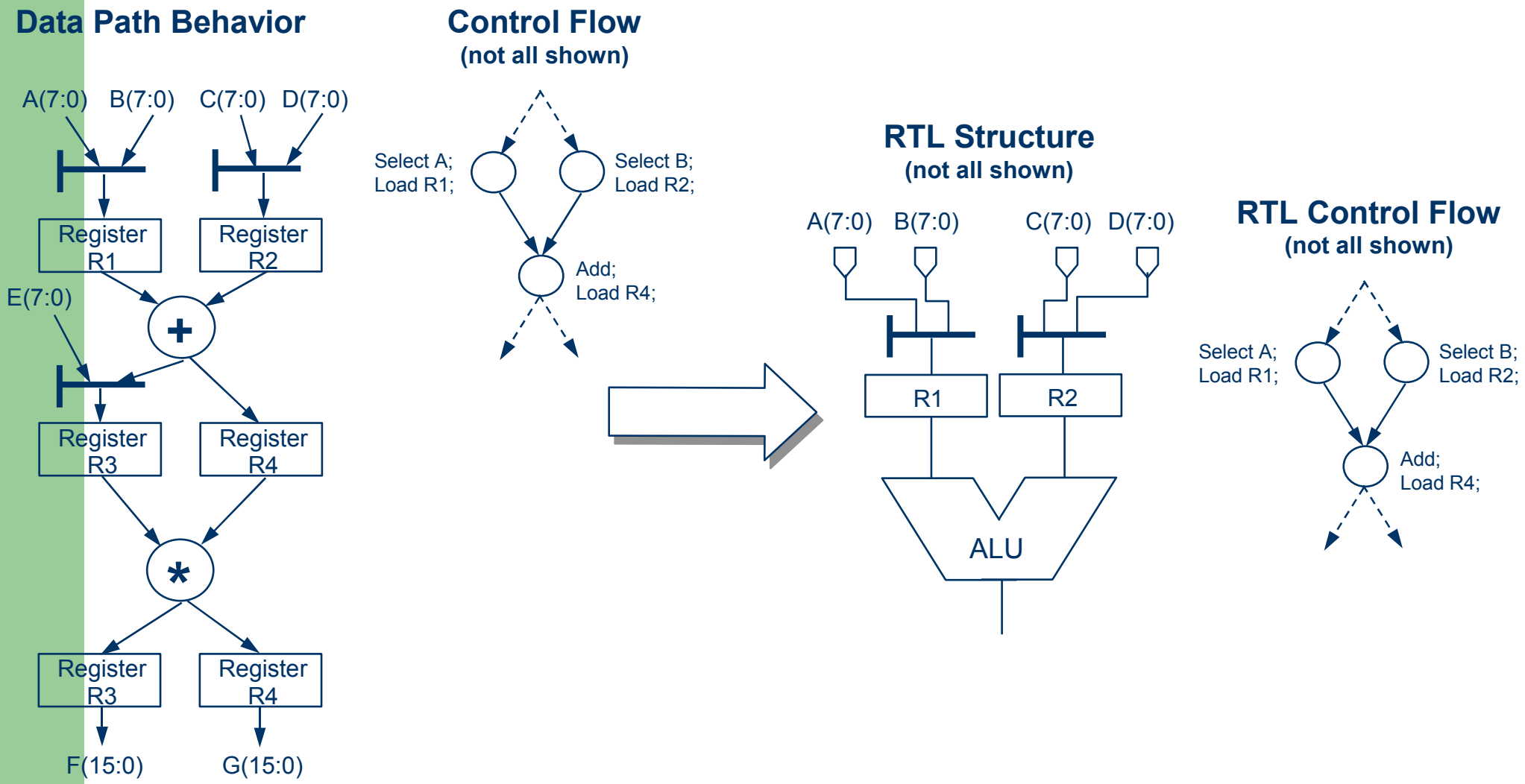






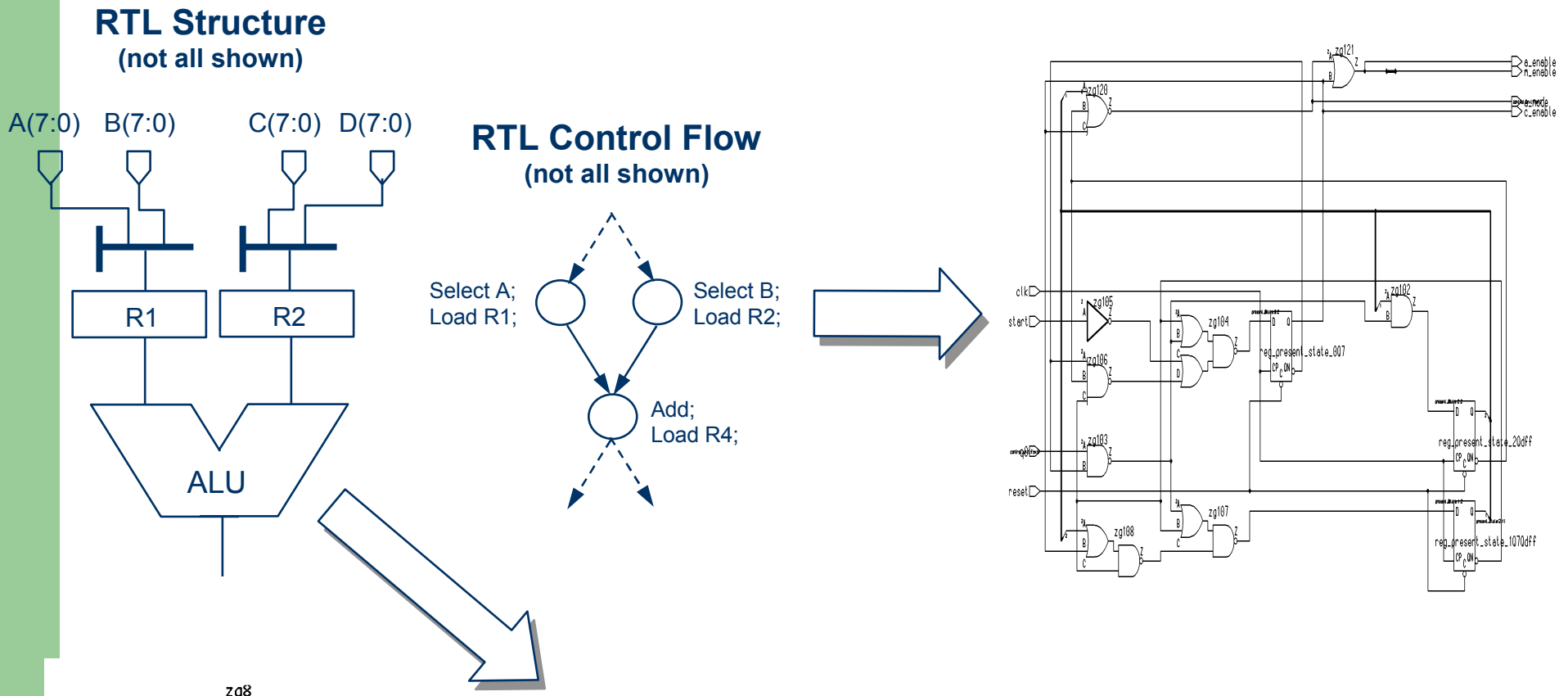
[Parker84]

RTL Level Synthesis



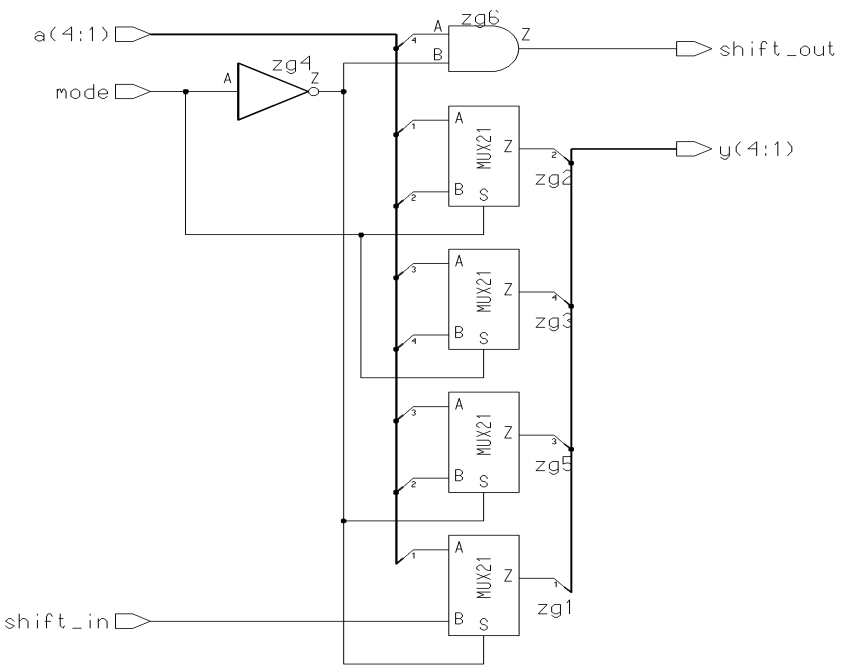
[Parker84]

Logic Synthesis



[Parker84]

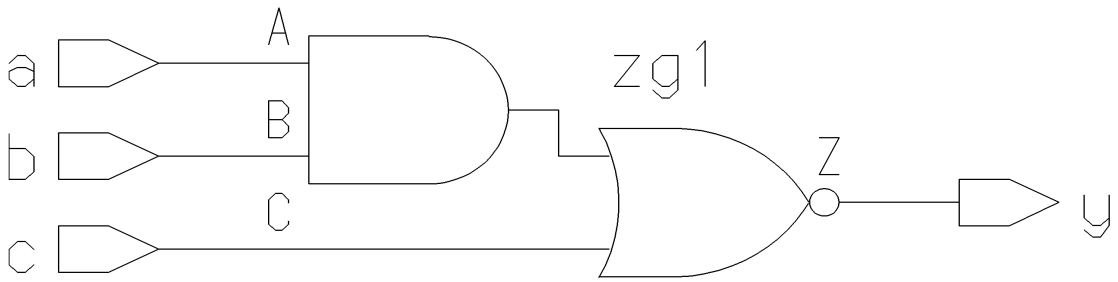
Sintesi di Loop Sequenziali




```
ARCHITECTURE behavior OF shift4 IS
  SIGNAL in_temp  : std_logic_vector(5
DOWNTO 0);
  SIGNAL out_temp : std_logic_vector(5
DOWNTO 1);
  BEGIN
   in_temp(0) <= shift_in;
   in_temp(4 DOWNTO 1) <= a;
   in_temp(5) <= '0';
    comb : PROCESS(mode,in_temp,a)
      BEGIN
        FOR i IN 1 TO 5 LOOP
          IF(mode = '0') THEN
            out_temp(i) <= in_temp(i-1);
          ELSE
            out_temp(i) <= in_temp(i);
          END IF;
        END LOOP;
    END PROCESS comb;
    y <= out_temp(4 DOWNTO 1);
    shift_out <= out_temp(5);
END behavior;
```

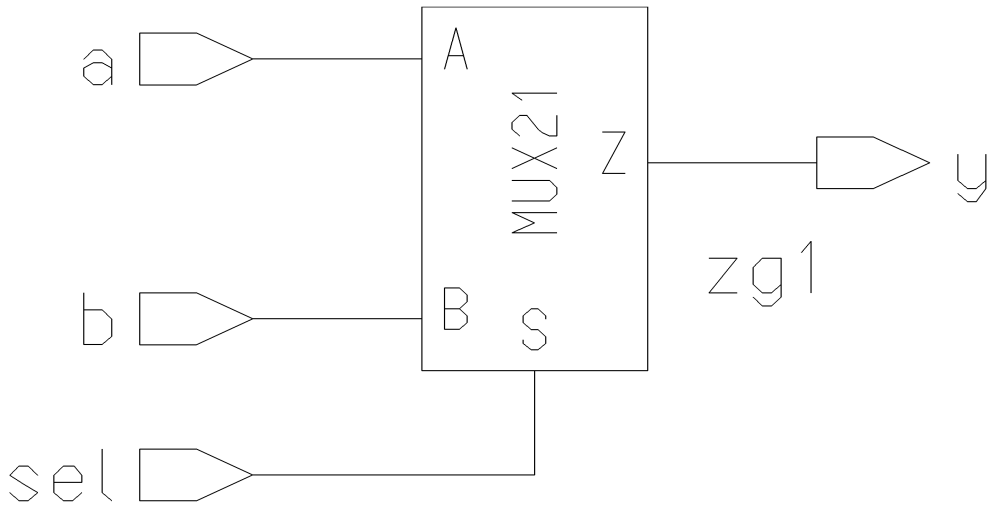
Process Statements Example

```
ENTITY aoi_process is
  PORT(a : IN  std_logic;
       b : IN  std_logic;
       c : IN  std_logic;
       y : OUT std_logic);
END aoi_process;

ARCHITECTURE behavior OF aoi_process IS
  SIGNAL zg1 : std_logic;
  BEGIN
    comb : PROCESS(a,b,c,zg1)
      BEGIN
        zg1 <= a AND b;
        y <= not(zg1 or c);
    END PROCESS comb;
END behavior;
```



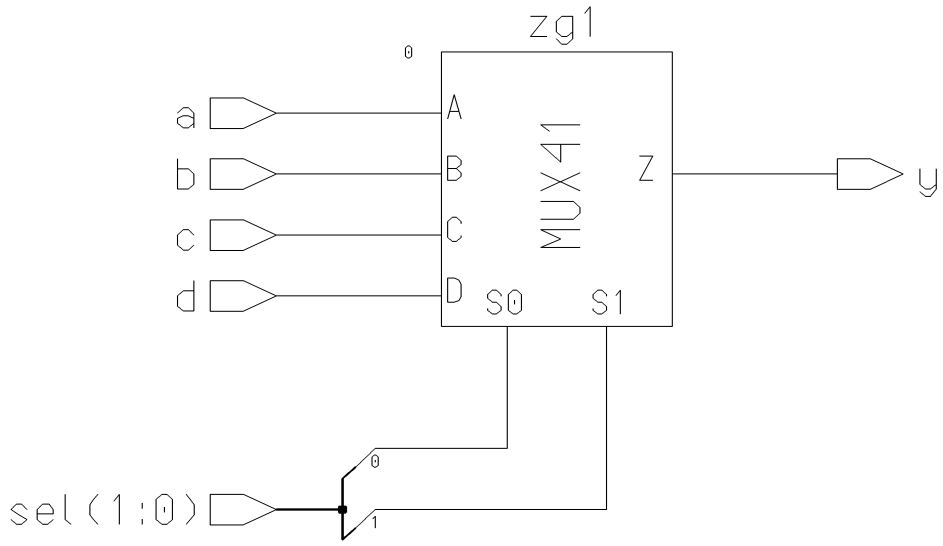
Conditional Signal Assignment

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

ENTITY mux2 is
  PORT(a   : IN  std_logic;
       b   : IN  std_logic;
       sel : IN std_logic;
       y   : OUT std_logic);
END mux2;

ARCHITECTURE behavior OF mux2 IS

  BEGIN
   y <= a WHEN (sel = '0') ELSE
        b WHEN (sel = '1') ELSE
        'X';
END behavior;
```



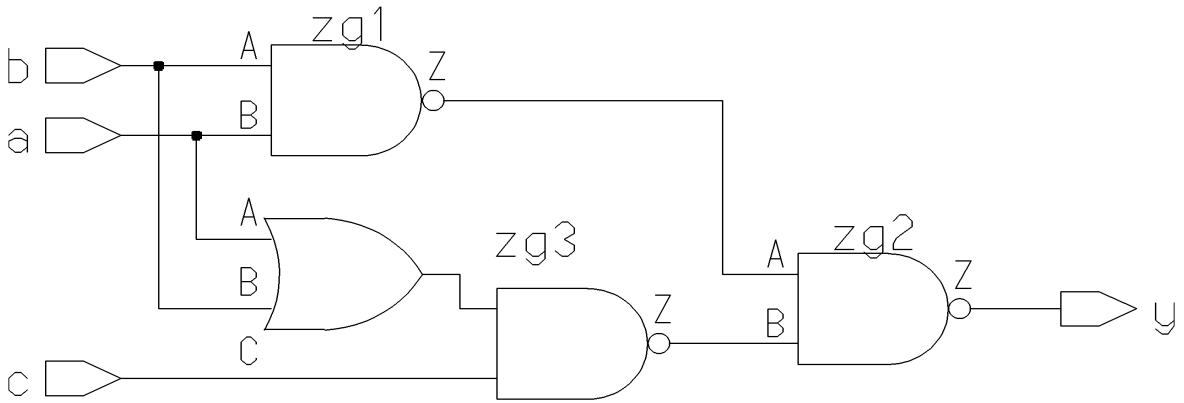
Selected Signal Assignment

```
ENTITY mux4 is
  PORT(a   : IN  std_logic;
       b   : IN  std_logic;
       c   : IN  std_logic;
       d   : IN  std_logic;
       sel : IN std_logic_vector(1 DOWNTO 0);
       y   : OUT std_logic);
END mux4;

ARCHITECTURE behavior OF mux4 IS
  BEGIN
    WITH sel SELECT
      y <= a WHEN "00",
           b WHEN "01",
           c WHEN "10",
           d WHEN "11",
         'X' WHEN OTHERS;
END behavior;
```



Procedures and Functions

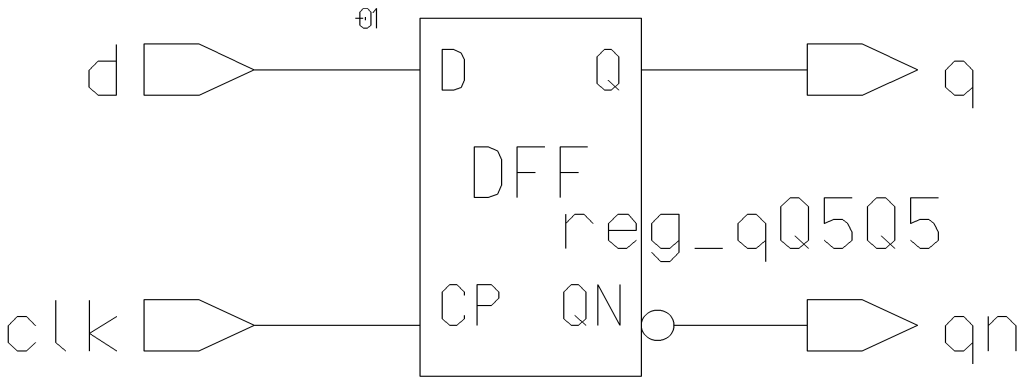
```
PACKAGE BODY logic_package IS
  FUNCTION majority(in1, in2, in3 : std_logic) RETURN std_logic IS
    VARIABLE result : std_logic;
    BEGIN
      IF((in1 = '1' and in2 = '1') or (in2 = '1' and in3 = '1') or
         (in1 = '1' and in3 = '1')) THEN
        result := '1';
      ELSIF((in1 = '0' and in2 = '0') or (in2 = '0' and in3 = '0') or
         (in1 = '0' and in3 = '0')) THEN
        result := '0';
      ELSE result := 'X';
      END IF;
      RETURN result;
  END majority;
```




```
USE work.logic_package.all;

ENTITY voter IS
   PORT(a : IN  std_logic;
        b : IN  std_logic;
        c : IN  std_logic;
        y : OUT std_logic);
END voter;

ARCHITECTURE maj OF voter IS
  BEGIN
    y <= majority(a,b,c);
END maj;
```

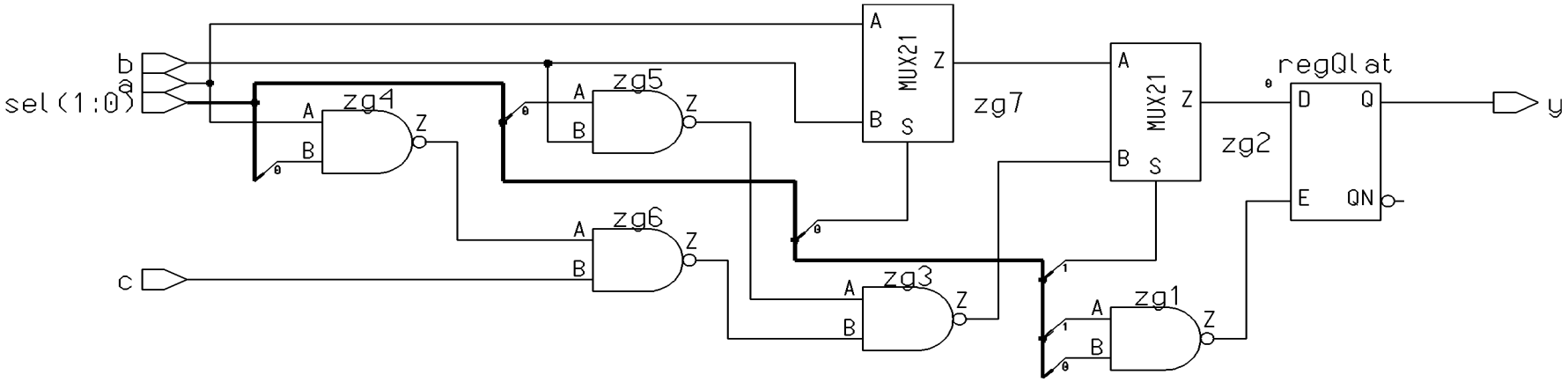
Edge Sensitive D Flip-Flop

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

ENTITY d_ff is
  PORT(d   : IN  std_logic;
       clk : IN  std_logic;
       q   : INOUT std_logic;
       qn  : OUT std_logic);
END d_ff;

ARCHITECTURE behavior OF d_ff IS
  BEGIN
    seq : PROCESS(clk)
      BEGIN
        IF(rising_edge(clk))
THEN
          q <= d;
        END IF;
    END PROCESS seq;
    qn <= not q;
END behavior;
```



Inferring Latches

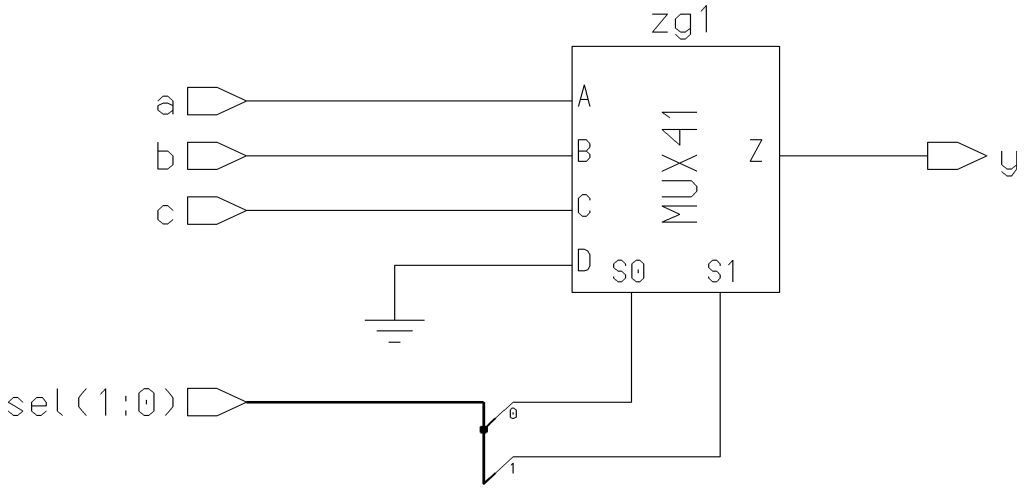
```
ARCHITECTURE behavior OF mux3_seq IS
  BEGIN
    comb : PROCESS(a,b,c,sel)
      BEGIN
        CASE sel IS
          WHEN "00" => y <= a;
          WHEN "01" => y <= b;
          WHEN "10" => y <= c;
          WHEN OTHERS => --empty
        END CASE;
    END PROCESS comb;
END behavior;
```



Avoiding Latches

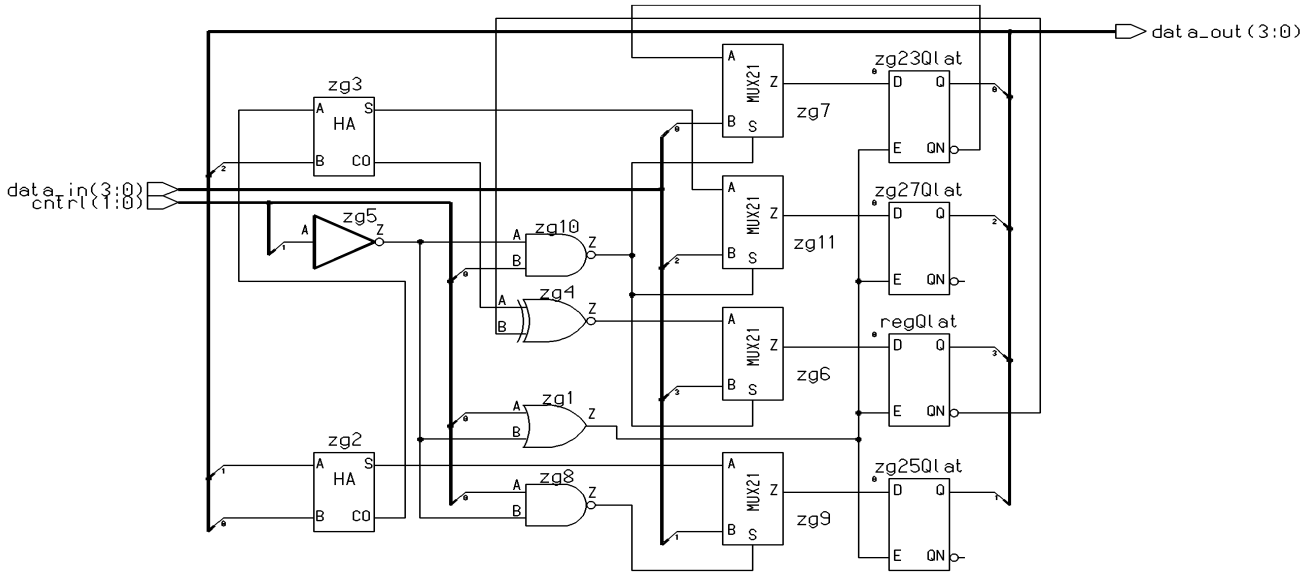
- Assigning values of “don’t care” (‘X’ or ‘-’) in these cases can avoid this

```
ARCHITECTURE behavior OF mux3 IS

  BEGIN
    comb : PROCESS(a,b,c,sel)
      BEGIN
        CASE sel IS
          WHEN "00" => y <= a;
          WHEN "01" => y <= b;
          WHEN "10" => y <= c;
          WHEN OTHERS => y <= 'X';
        END CASE;
    END PROCESS comb;
END behavior;
```

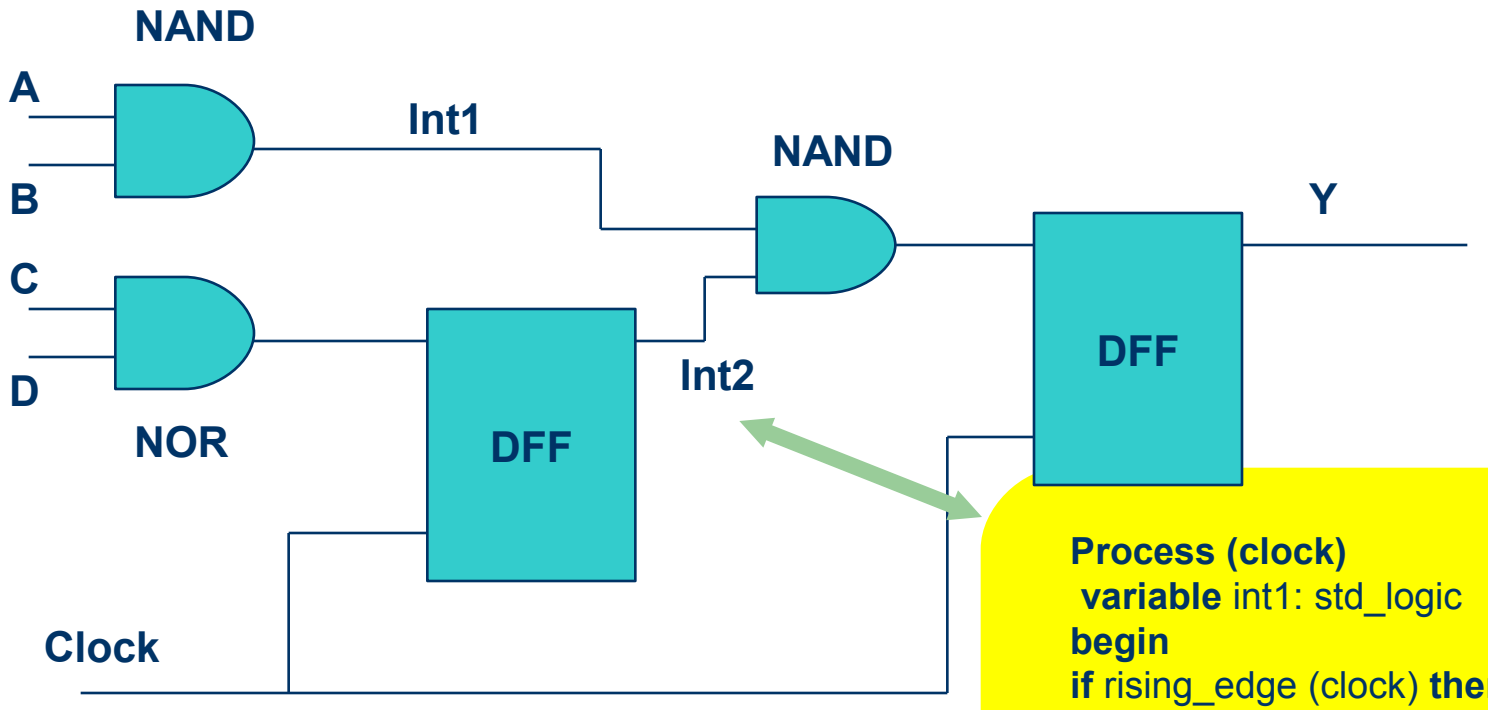


Inferring Latches in Complex Behaviors

```
ARCHITECTURE rtl OF pc_comb IS
  SIGNAL pc : unsigned(3 DOWNTO 0);
  BEGIN
    one : PROCESS(data_in, cntrl, pc)
      BEGIN
        CASE cntrl IS
          WHEN "01" => pc <= (pc + "0001");
          WHEN "10" => pc  <= pc;
          WHEN OTHERS => pc <= data_in;
        END CASE;
    END PROCESS one;
    DATA_OUT <= PC;
END rtl;
```



Synchronous RTL Synthesis Example



Note:**variable** int1 does not infer FF (because it is assigned to before being read, however Y and int2 infer FFs. Sensitivity list only contains the clock and asynchronous set/reset (if present). A, B, C, are not included.

```
Process (clock)
 variable int1: std_logic
begin
if rising_edge (clock) then
int1 := A nand B;
int2 <= C nor D;
Y <= int2 nand int1 ;
end if ; end process;
```

Logic Level Optimizations

Common operations at this level include *factorization* and *flattening,* to name a few. Extensive literature on optimization at this level exists.

Original equation

Y = A.B.C
Y2 = Y + A.B.D
Y3 = A.B + C + D

Factorize

M = A.B
Y = M.C
Y2 = Y + M.D
Y3 = M + C + D

Six gates

Flatten

Y2 = A.B.C + A.B.D
Y3 = A.B + C + D

Factorize

M = A.B,
N = C + D, Y2 = M.N
Y3 = M + N

Four gates