Synthesis of VHDL

Egbert Molenkamp
Department of Electrical Engineering, Mathematics and Computer Science
University of Twente
PO Box 217
7500 AE Enschede
the Netherlands
email: e.molenkamp@utwente.nl

Contents

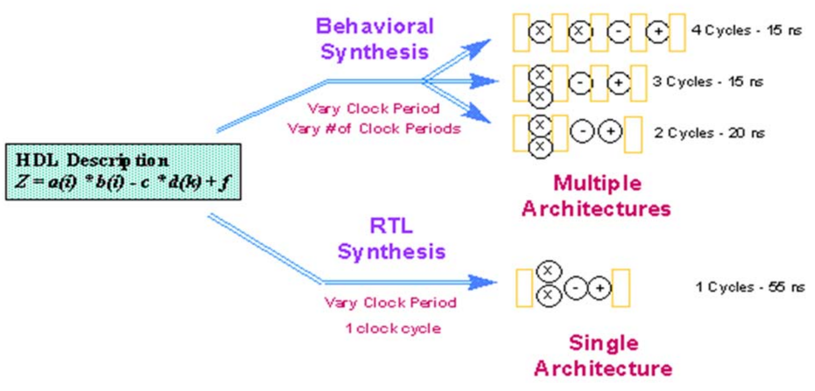
Synthesis tools

- Which subset is supported?
- How good is it supported?
- Is your VHDL description input for different synthesis tools?

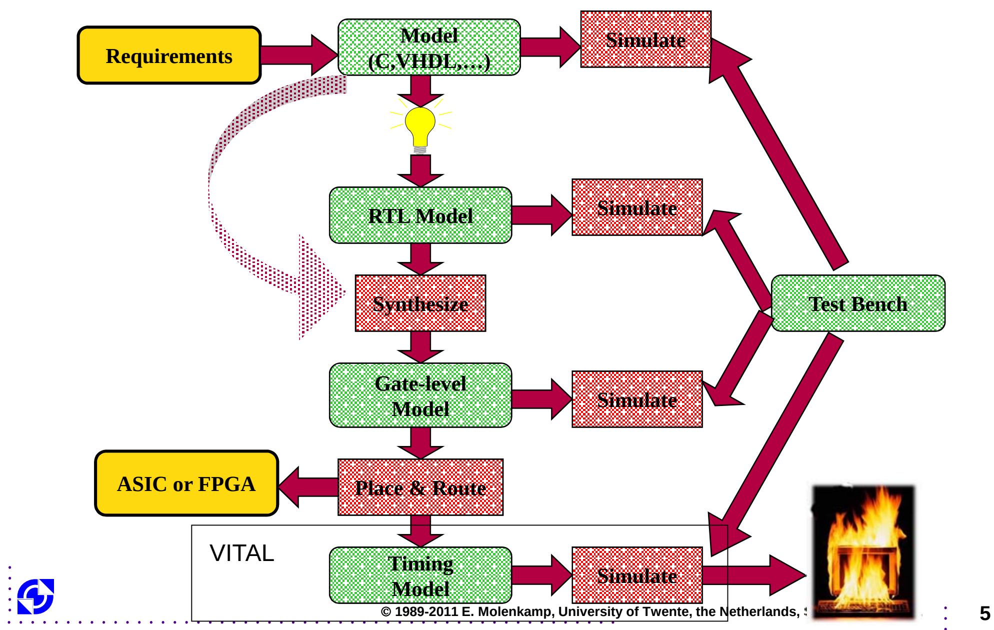
$$\begin{tikzcd}[row sep=large]
\text{Abstract Level, e.g. multiple waits} \arrow[d] \\
\text{RTL Level, logic synthesis} \arrow[d] \\
\text{Realization}
\end{tikzcd}$$

Study the new features that are supported and read the tips!

Behavioral and RTL Synthesis

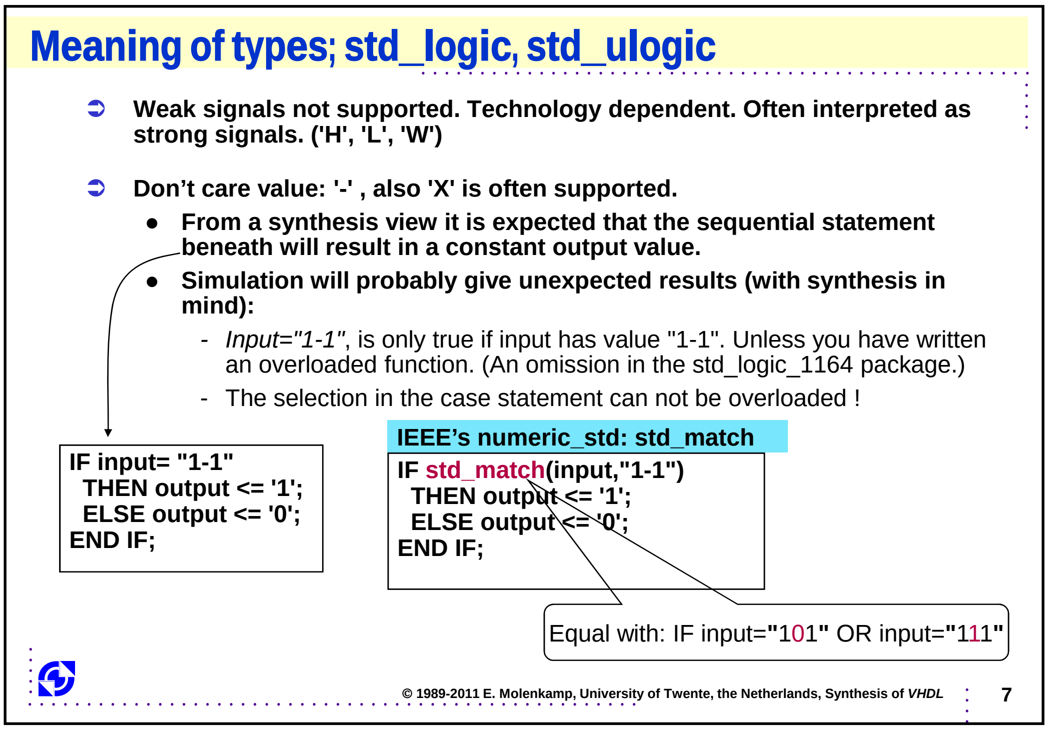


Design methodology



Globally the supported subset for synthesis

- Not:
  - time: y<= x after 10 ns; wait for 10 ns;
  - files
  - dynamic structures (pointers)
  - initial value of a variable in process declaration
  - Initial value of a signal
- Restricted:
  - recursion of functions/procedures
  - asynchrone designs
  - meaning of types
  - Initial values of variables in subprograms
- Supported:
  A lot!!



Meaning of types; std_logic, std_ulogic

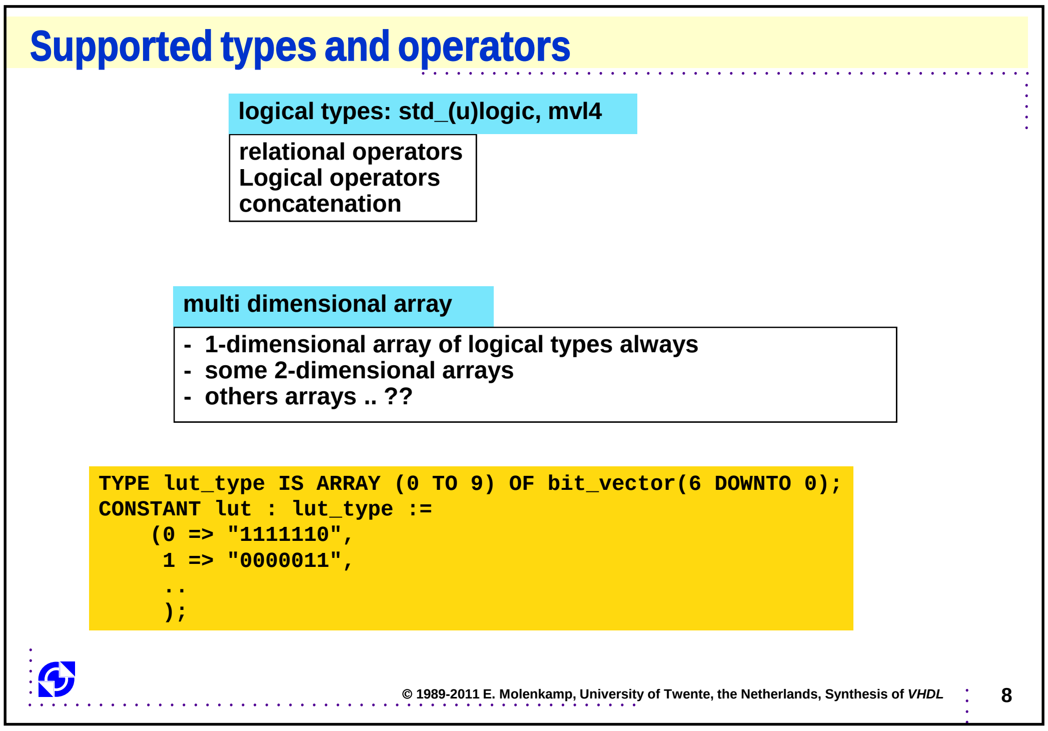
- ➲ **Weak signals not supported. Technology dependent. Often interpreted as strong signals. ('H', 'L', 'W')**
- ➲ **Don't care value: '-' , also 'X' is often supported.**
  - **From a synthesis view it is expected that the sequential statement beneath will result in a constant output value.**
  - **Simulation will probably give unexpected results (with synthesis in mind):**
    - *Input="1-1"*, is only true if input has value "1-1". Unless you have written an overloaded function. (An omission in the std_logic_1164 package.)
    - The selection in the case statement can not be overloaded !

```
IF input= "1-1"
  THEN output <= '1';
  ELSE output <= '0';
END IF;
```

IEEE's numeric_std: std_match

```
IF std_match(input,"1-1")
  THEN output <= '1';
  ELSE output <= '0';
END IF;
```

Equal with: IF input="101" OR input="111"



Supported types and operators

logical types: std_(u)logic, mvl4

relational operators
Logical operators
concatenation

multi dimensional array

- **1-dimensional array of logical types always**
- **some 2-dimensional arrays**
- **others arrays .. ??**

```
TYPE lut_type IS ARRAY (0 TO 9) OF bit_vector(6 DOWNTO 0);
CONSTANT lut : lut_type :=
    (0 => "1111110",
     1 => "0000011",
     ..
     );
```

Supported types and operators /2

enumeration

Most tools use as default a binary code.

Sometimes the default is technology dependent.

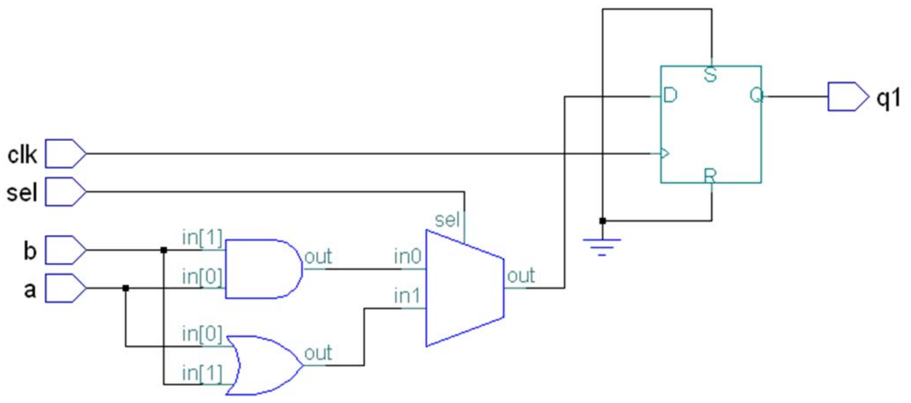
Integer types

Relational operators
Arithmetic operators
- **+, -, ABS**
- ***, /, MOD, REM (often) only if**
  - **right operand a constant and a power of two (for multiplication the right operand often only needs to be a constant), or**
  - **both operands are constants**
  - **some tools support * for any integer**
  - **only a few tools support /, MOD and REM for any integer**
- **** (often) only if**
  - **left operand is 2, or**
  - **both operands are constants**
  - **some tools support ** for any integer.**

Number of bits depends on the RANGE of the integer
- **2's complement in case of negative numbers, otherwise**
- **unsigned**

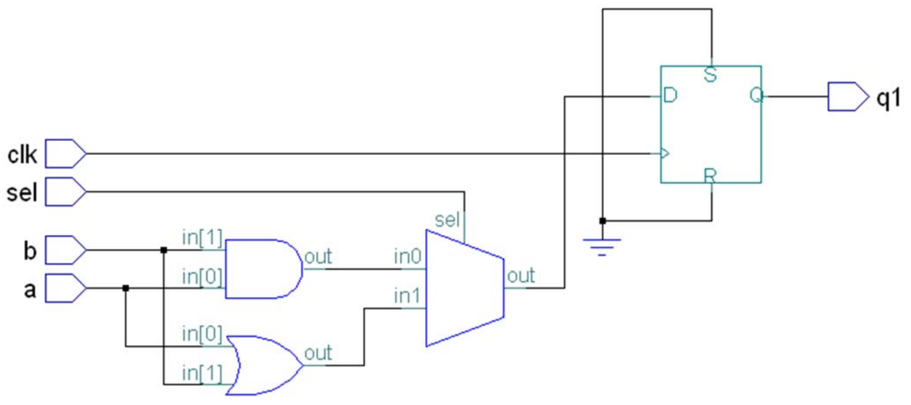
How would you synthesize this?

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY comb IS
  PORT (a, b     : IN std_logic;
           clk, sel : IN std_logic;
           q1       : OUT std_logic);
END comb;

ARCHITECTURE behaviour OF comb IS
  SIGNAL d : std_logic;
BEGIN
  d  <= a or b when sel='1' else
          a and b;

  PROCESS
  BEGIN
    WAIT UNTIL rising_edge(clk);
    q1 <= d;
  END PROCESS;

END behaviour;
```

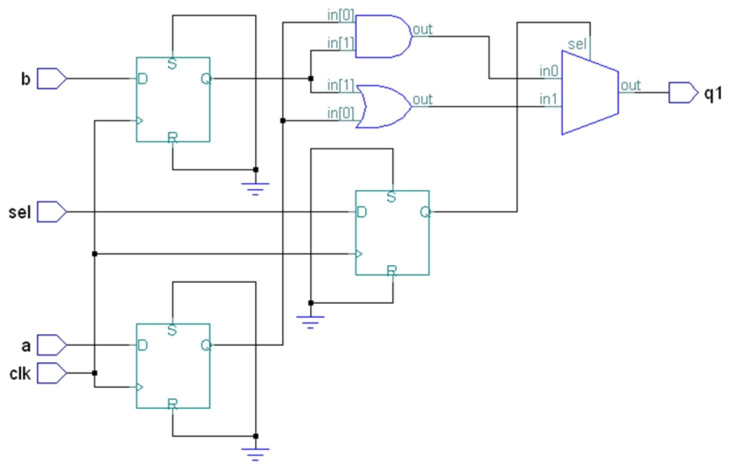


And this ?

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY comb1 IS
  PORT (a, b     : IN std_logic;
             clk, sel : IN std_logic;
             q1       : OUT std_logic);
END comb1;

ARCHITECTURE behaviour OF comb1 IS
BEGIN
  PROCESS
    VARIABLE d : std_logic;
  BEGIN
    WAIT UNTIL rising_edge(clk);
    IF sel='1' THEN
      d := a or b;
    ELSE
      d := a and b;
    END IF;
    q1 <= d;
  END PROCESS;
END behaviour;
```



Is this also correct?



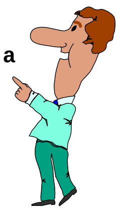
- Late arrival of a, b, sel allowed
- q1 is late available
- How would you write this in VHDL?

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY comb1 IS
  PORT (a, b     : IN std_logic;
            clk, sel : IN std_logic;
            q1       : OUT std_logic);
END comb1;

ARCHITECTURE behaviour OF comb1 IS
  SIGNAL ai, bi, seli : std_logic;
BEGIN

  PROCESS
  BEGIN
    WAIT UNTIL rising_edge(clk);
    ai <= a; bi <= b; seli <= sel;
  END PROCESS;

  q1 <= ai or bi when seli='1' else
           ai and bi;

END behaviour;
```

Synchronous systems

- **No feedback in combinational part**
- **Check on setup/hold time by synthesis tool**
- **Most tools support an explicit process description with:**
  - **As first statement a *wait until clk='1'* (or similar descriptions using *clk'event*)**
  - **Some tools also support multiple wait statements (state is implicit)**
  - **Since signals are used to communicate between processes, for each signal assigned to a register is used. Sometimes this register is removed in a optimization phase.**
  - **Variables are only needed for local use. If a variable is used to remember the previous state a register is used. If the synthesis tool is not sure then a register is used. Always good, but .. hardware overhead. Read the warnings!**

!!



- **Also a process with a sensitivity list is supported**
  - **The sensitivity list contains the clock signal and, if any, asynchronous (p)reset signals.**

Synchronous systems/2

Synchronous design with an asynchronous reset/preset

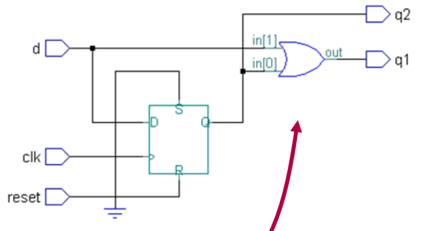
```
PROCESS (reset,preset,clk)
BEGIN
 IF reset='1' THEN
   … reset actions

 ELSIF preset='1' THEN
   … preset actions

 ELSIF clk='1' and clk'EVENT THEN
   … synchronous part

 END IF;
  -- some synthesis tools do not allow
  -- statements here

END PROCESS;
```

```
PROCESS (reset,clk)
BEGIN
 IF reset='1'
  THEN ... reset actions
  ELSIF rising_edge(clk)
   THEN ...
```

Function *rising_edge* is in the IEEE package std_logic_1164.

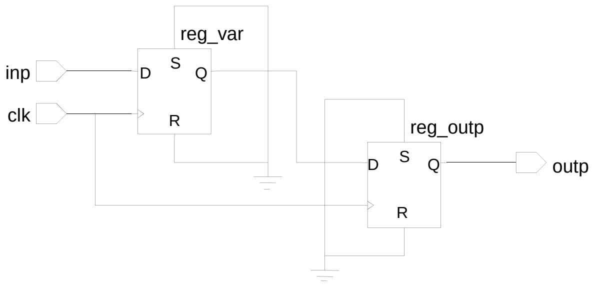
Synchronous systems/3

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY ff IS
  PORT (d,clk,reset : std_logic; q1, q2 : OUT std_logic);
END ff;

ARCHITECTURE test OF ff IS
  SIGNAL q : std_ulogic;
BEGIN
  PROCESS(reset,clk,d,q)
  BEGIN
    IF reset='1' THEN
      q<='0';
    ELSIF rising_edge(clk) THEN
      q<=d;
    END IF;
    q1 <= q OR d;
  END PROCESS;
  q2 <= q;
END test;
```



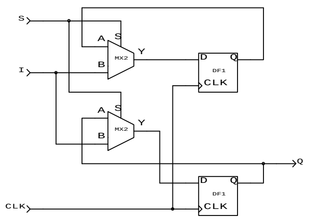
Synchronous systems /4

```
library ieee;
use ieee.std_logic_1164.all;
entity reg_variabele is
  port (inp, clk : in std_logic; outp : out std_logic);
end reg_variabele;

architecture test of reg_variabele is
begin
 process
  variable var : std_logic;
  begin
    wait until clk='1';
    outp <= var;
    var := inp;
  end process;
end test;
```

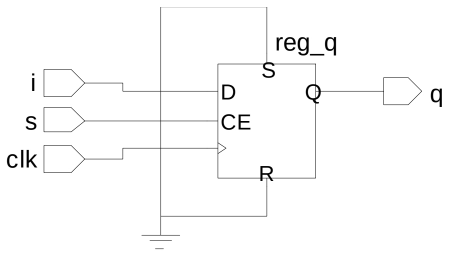


Synchronous systems /5

```
PROCESS
  VARIABLE var : std_logic;
BEGIN
  WAIT UNTIL clk='1';
  IF s='1' THEN
    var := i;
    q <= var;
  END IF;
END PROCESS;
```



Bad synthesis result

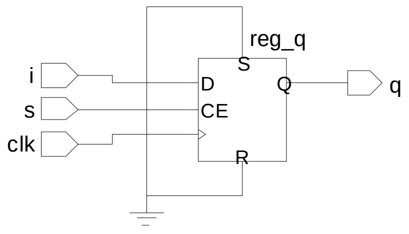


Synchronous systems /6

Solution:

- **Explicitly assign a value after the wait statement to the variable**
- **Not all tools need this, but may have problems again with nested control structures**
- **However for portable descriptions ...**

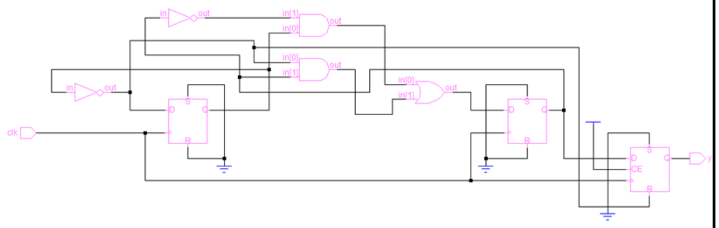
```
PROCESS
  VARIABLE var : std_logic;
BEGIN
  WAIT UNTIL clk='1';
  var := ‘0’; -- guideline for synthesis tool
  IF s='1' THEN
    var := i;
    q <= var;
  END IF;
END PROCESS;
```



Synchronous systems /7

Some synthesis tools can handle multiple wait statements.

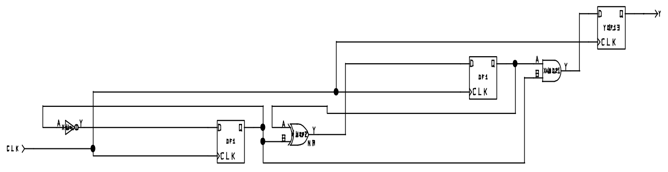
```
ENTITY count4 IS
  PORT (clk : IN bit;
           y   : OUT bit);
END count4;

ARCHITECTURE behaviour OF count4 IS
BEGIN
  PROCESS
  BEGIN
    WAIT UNTIL clk='1';
    y <= '0';
    WAIT UNTIL clk='1';
    y <= '0';
    WAIT UNTIL clk='1';
    y <= '0';
    WAIT UNTIL clk='1';
    y <= '1';
  END PROCESS;
END behaviour;
```



Synchronous systems /8

Alternative description (*Aurora Synthesis* supported this).

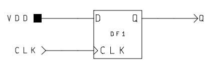
```
 ENTITY count4a IS
  PORT (clk : IN bit;
        y   : OUT bit);
END count4a;

ARCHITECTURE behaviour OF count4a IS
BEGIN
  PROCESS
  BEGIN
    FOR i IN 0 TO 2 LOOP
      WAIT UNTIL clk='1';
      y <= '0';
    END LOOP;
    WAIT UNTIL clk='1';
    y <= '1';
  END PROCESS;
END behaviour;
```

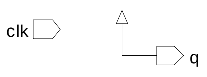


An expensive constant ?

Optimizing the number of flip-flops ?

```
ENTITY const IS
  PORT (clk : IN bit;
        q   : OUT bit);
END const;

ARCHITECTURE behaviour OF const IS
BEGIN
  PROCESS
  BEGIN
    WAIT UNTIL clk='1';
    q <= '1';
  END PROCESS;
END behaviour;
```





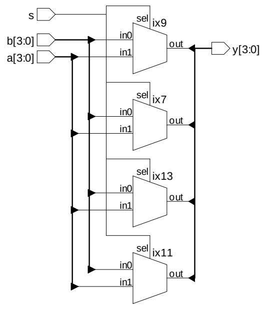
Combinational logic

Written in VHDL as:

- **a concurrent signal assignment statement**
- **an explicit process; required,not sufficient**
  - **after keyword process a sensitivity list required**
  - **all signals read should be in this sensitivity list**
  - **no *'event, rising_edge, falling_edge* is to be used**
  - **all variables should be assigned to before read**

a,b,y of type std_logic_vector(3 downto 0)

```
y <= a WHEN s='1' ELSE
        b;
```



Combinational logic /2

A combinational circuit

- **has 4 inputs with weight 8, 4, 2, and 1.**
  A decimal value (0..10, 11..15 never occurs).
- **The output is '1' if input is 0, 1, 3, 5, 6, 7, 9.**

in3,in2 \ in1,in0	00	01	11	10
00	1	1	1	0
01	0	1	1	1
11	-	-	-	-
10	0	1	-	0

O <= in0 + (in2 * in1) + (/in3 * /in2 * /in1)

What if the specification is changed?

Combinational logic /3

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY combinational IS
  PORT (inp : IN unsigned(3 DOWNTO 0);
              DP  : OUT std_logic);
END combinational;
```

IEEE's numeric_std

type UNSIGNED is array (NATURAL range <>) of STD_LOGIC;

type SIGNED is array (NATURAL range <>) of STD_LOGIC;

and conversion function *to_integer*.

Synopsis std_logic_arith

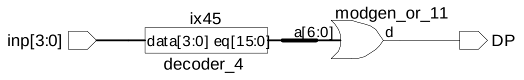
type UNSIGNED is array (NATURAL range <>) of STD_LOGIC;

type SIGNED is array (NATURAL range <>) of STD_LOGIC;

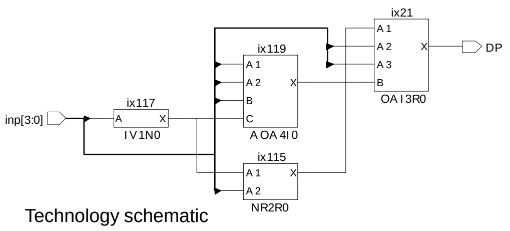
and conversion function *conv_integer*.

Combinational logic /4

```
ARCHITECTURE demo OF combinational IS
BEGIN
  PROCESS (inp)
  BEGIN
    CASE to_integer(inp) IS
      WHEN 0|1|3|5|6|7|9 => DP <= '1';
      WHEN 2|4|8|10      => DP <= '0';
      WHEN OTHERS        => DP <= '-';
    END CASE;
  END PROCESS;
END demo;
```



RTL schematic



Technology schematic

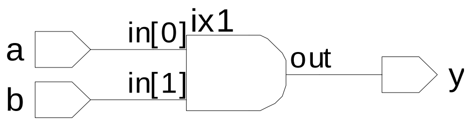
The VHDL specification improves readability, is less error prone, and more easily to change. However, the circuit is too large?

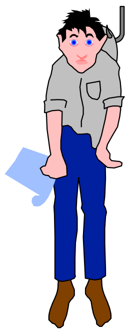
Notice that if input is an unconstrained (!) unsigned it is even more flexible!

Combinational logic /5 .... ?

```
PROCESS (a)
BEGIN
  y <= a AND b;
END PROCESS;
```

Is this your synthesis result?

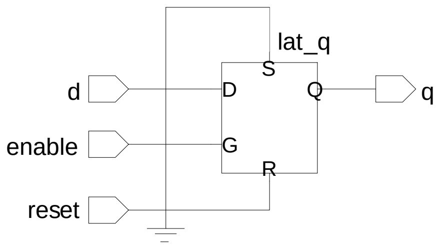




Read the warnings!!!!

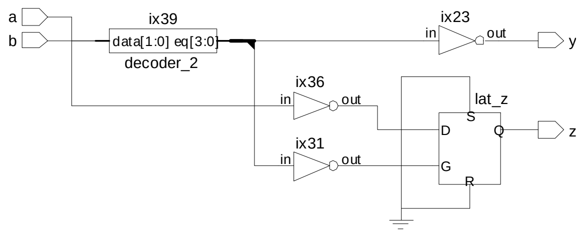
Latches

```
PROCESS(d,enable,reset)
 BEGIN
   IF reset='1' THEN
     q <= '0';
   ELSIF enable='1' THEN
     q <= d;
   END IF;
 END PROCESS;
```



Latches /2

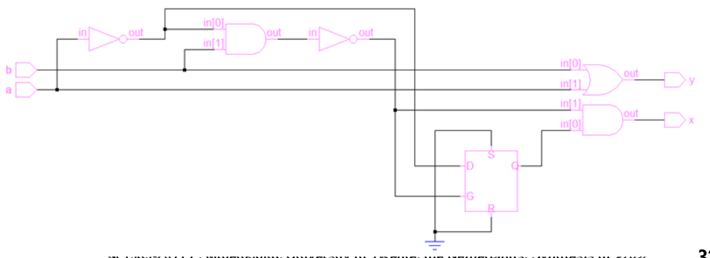
```
PROCESS(a,b)
BEGIN
  CASE std_logic_vector '(a&b) IS
    WHEN "00"        => y <= '0'; z <= '1';
    WHEN "01"        => y <= '1';
    WHEN OTHERS => y <= '1'; z <= '0';
  END CASE;
 END PROCESS;
```



Latches are inferred, and you expected a combinational circuit? Probably not in all branches (if statement, case statement) a value is assigned to the variables and signals.

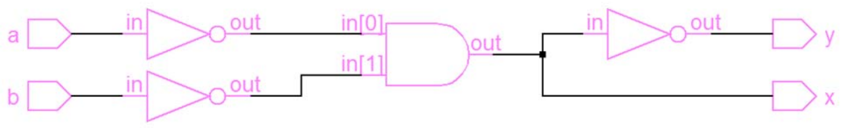
Latches /3

```
ARCHITECTURE behavior OF funny1 IS
  SIGNAL z : std_logic;
BEGIN
  PROCESS(a,b)
  BEGIN
    CASE std_logic_vector '(a&b) IS
      WHEN "00"        => y <= '0'; z <= '1';
      WHEN "01"        => y <= '1';
      WHEN OTHERS => y <= '1'; z <= '0';
    END CASE;
  END PROCESS;
  x <= '0' WHEN std_logic_vector '(a&b)= "01" ELSE
        z;
END behavior;
```



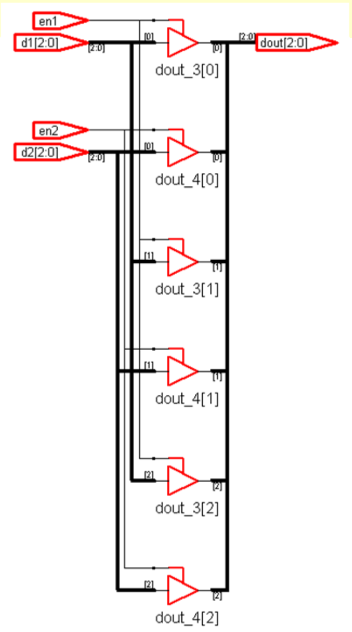
Latches /4

```
ARCHITECTURE behavior OF funny1 IS
  SIGNAL z : std_logic;
BEGIN
  PROCESS(a,b)
    SUBTYPE bv2 IS std_logic_vector(1 DOWNTO 0);
  BEGIN
    CASE bv2'(a&b) IS
      WHEN "00"       => y <= '0'; z <= '1';
      WHEN "01"       => y <= '1'; z <= '-';
      WHEN OTHERS => y <= '1'; z <= '0';
    END CASE;
  END PROCESS;
  x <= '0' WHEN std_logic_vector '(a&b)= "01" ELSE
        z;
END behavior;
```



Tri-states

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY databus_demo IS
  GENERIC (w : integer := 3);
  PORT (d1,d2 : std_logic_vector(w-1 DOWNTO 0);
            en1,en2 : IN std_logic;
            dout : OUT std_logic_vector(w-1 DOWNTO 0));
END  databus_demo;
ARCHITECTURE structure OF databus_demo IS
BEGIN

  PROCESS(d1,en1)
  BEGIN
    IF en1='1' THEN
      dout <= d1;
    ELSE
      dout <= (OTHERS => 'Z');
    END IF;
  END PROCESS;

  dout <= d2 WHEN en2='1' ELSE (OTHERS => 'Z');

END structure;
```


en1
d1[2:0]
en2
d2[2:0]
dout[2:0]
dout_3[0]
dout_4[0]
dout_3[1]
dout_4[1]
dout_3[2]
dout_4[2]