• VHDL-based design and synthesis

1. Description of basic primitives
2. Behavioral models of systems
3. The standard synthesis package IEEE Std 1076.3-1997
4. The VHDL RTL synthesis standard IEEE Std 1076.6-
   -1999

1. Description of basic primitives

- **Concurrent assignments and processes:**

➢ **Concurrent assignments:** A change is projected onto the signal placed on the left hand side of the assignment whenever there is a change on the signals placed on the right hand side

a <= b xor c;

➢ **Processes:** The changes on the signals take place when the is resumed and before its next activation. The activation of a process is a consequence of changes in signals

• *Equivalence:*

```
a <=    '1' when (b=c)
        else '0';
```

```
process (b,c)
begin
  if (b=c) then
     a <= '1';
  else
     a <= '0';
  end if;
end process;
```

```
with b select
      a <=     "00" when "01",
               "01" when "11",
               "11" when others;
```

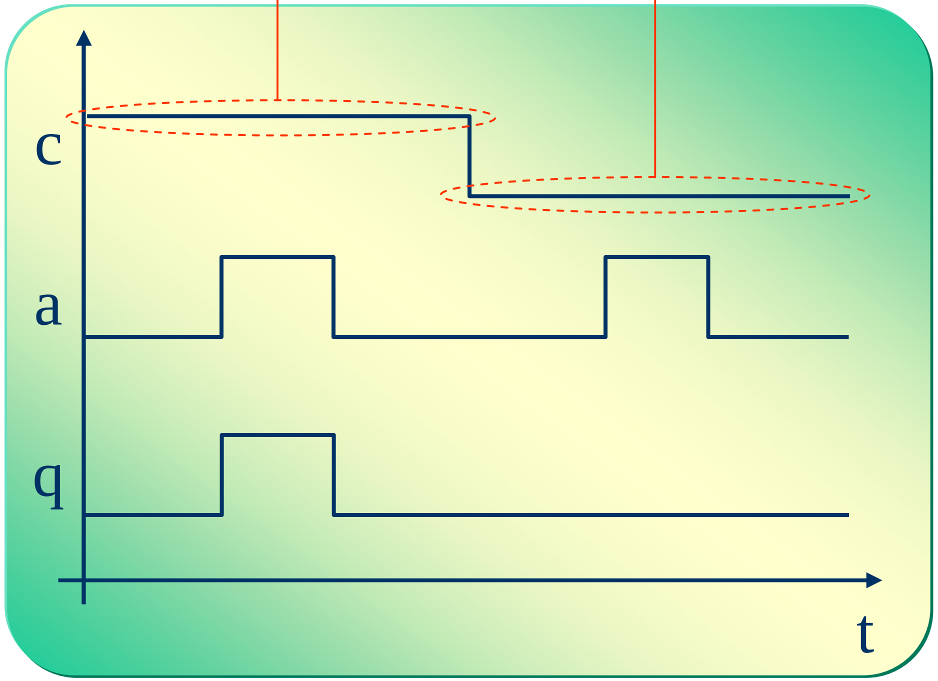
```
process(b)
begin
  case b is
    when "01" => a<= "00";
    when "11" => a<= "01";
    when others => a<= "11";
  end case;
end process;
```

- ***Memory on processes:***

```
process(a,c)
begin
        if (c='1') then
                q <= a;
        end if;
end process;
```

c='1', changes on **a** are reflected on **q**

c='0', **q** holds the previous value



• General description rules:

- **Combinational logic:**
  - Concurrent assignments (recommended)
  - Processes (sensitivity list, default case)

- **Sequential logic:**
  - Processes with **wait** primitive $\Rightarrow$ Flipflops
  - Processes with sensitivity list $\Rightarrow$ Latches

• Description of combinational logic:

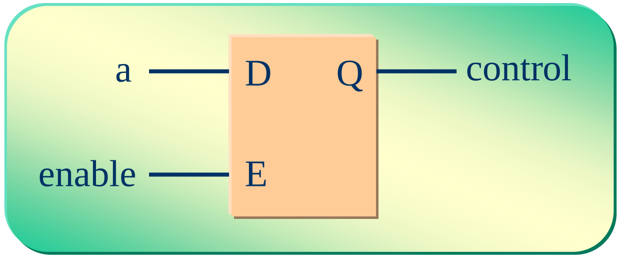
n m … b a	f
0 0 … 0 0	f_0
0 0 … 0 1	f_1
0 0 … 1 0	f_2
0 0 … 1 1	f_3
…	
1 1 … 1 1	f_n

```
f <=      f0 when (n&m&...&b&a="00...00")
          else f1 when (n&m&...&b&a="00...01")
          else f2 when (n&m&...&b&a="00...10")
          else f3 when (n&m&...&b&a="00...11")
          ...
          else fn;
```

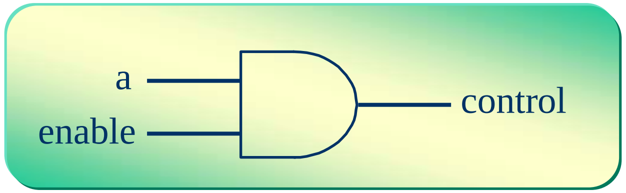
```
with (n&m&...&b&a) select
      f <=      f0 when "00...00",
                f1 when "00...01",
                f2 when "00...10",
                f3 when "00...11",
                ...
                fn when others;
```

• *Feedback in concurrent assignments:*

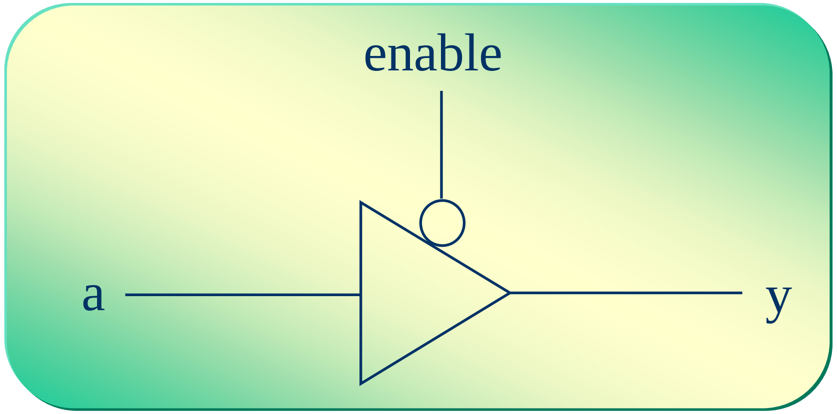
control <= a when (enable=‘1’)
else control;



control <= a when (enable=‘1’)
else ‘0’;



• *Tri-state buffers:*




```
y <=   a when (enable='0')
       else 'Z';
```

std_logic type

ieee library

• Description of sequential logic:

➢ Definition of clock signals:

➢ **Wait primitive:**

```
process
begin
        wait until (clk'event and clk='1');
...
```

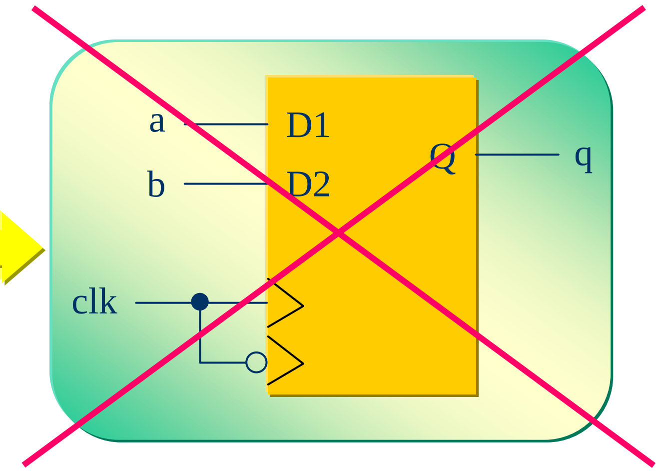
not clk'stable and clk='1'

➢ **Sensitivity list (asynchronous control signals):**

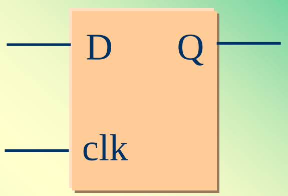
```
process(clk)
begin
        if (clk'event and clk='1')
...
```

- ***Multiple clock signals:***

```
m_clock: process(clk)
begin
   if (clk'event and clk='0') then
         q <= a;
   elsif (clk'event and clk='1') then
         q <= b;
   end if;
end process;
```



- ***Sequential primitives active by level (latches):***

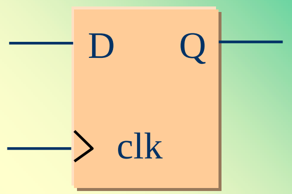


clk	D	Q	Q^+
1	-	-	D
0	-	-	Q

Mode **transparent**

```
latch_p:process(d,clk)
begin
        if (clk='1') then
                q <= d;
        end if;
end process;
```

• Sequential primitives active by edge (flipflops):



clk	D	Q	Q^+
1	-	-	Q
0	-	-	Q
↑	-	-	D
↓	-	-	Q

```
flip_p1: process(clk)
begin
        if (clk'event and clk='1') then
                q <= d;
        end if;
end process;


flip_p2:process
begin
        wait until (clk'event and clk='1');
                q <= d;
end process;
```

not (clk'***stable***) ***and*** clk='1'

- **Control signals:**

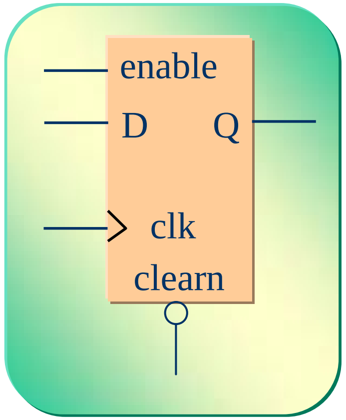
Asynchronous reset

```
async_flip: process(clk,clearn)
begin
    if (clearn='0') then
        q <= '0';
    elsif (clk'event and clk='1') then
        q <= d;
    end if;
end process;
```

Synchronous reset

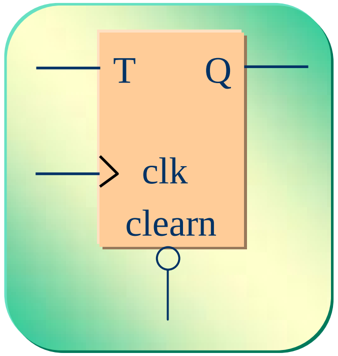
```
sync_flip: process
begin
    wait until (clk'event and clk='0');
    if (clearn='0') then
        q <= '0';
    else
        q <= d;
    end if;
end process;
```

• *E-type register (enable):*



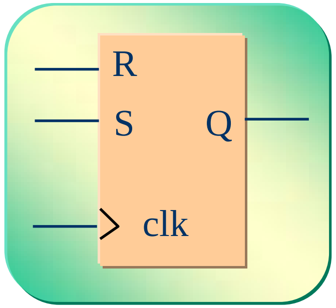

```
E_flipflop: process
begin
    wait until (clk'event and clk='1');
    if (clearn='0') then
        Q <= '0';
    elsif (enable='1') then
        Q <= D;
    end if;
end process;
```

• T-type register (toggle):




```
T_flipflop: process
begin
    wait until (clk'event and clk='1');
    if (clearn='0') then
        Q <= '0';
    elsif (T='1') then
        Q <= not Q;
    end if;
end process;
```

- ***RS-type register (Reset/Set):***

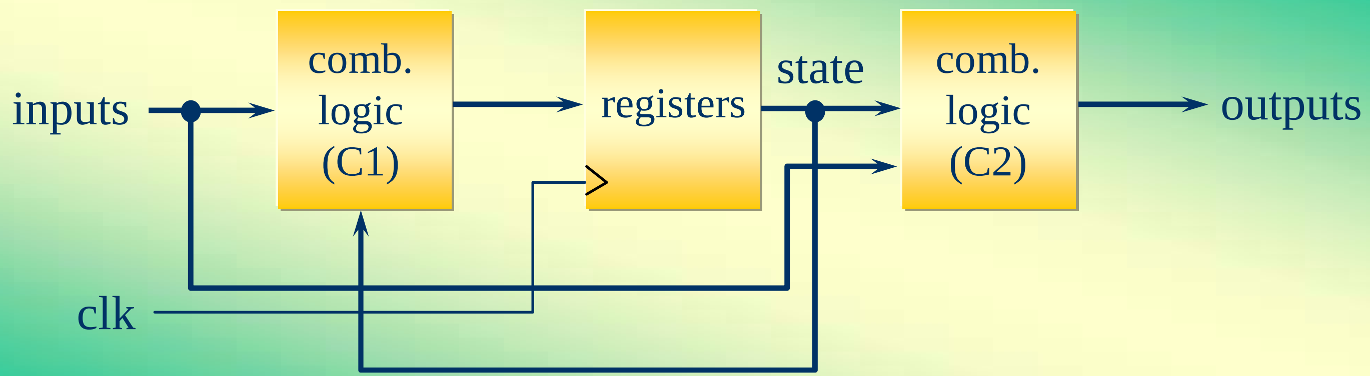



```
RS_flipflop: process
begin
    wait until (clk'event and clk='1');
    if (R='1') then
        Q <= '0';
    elsif (S='1') then
        Q <= '1';
    end if;
end process;
```

- ***Finite state machines:***

➢ Systems with finite memory (number of states)

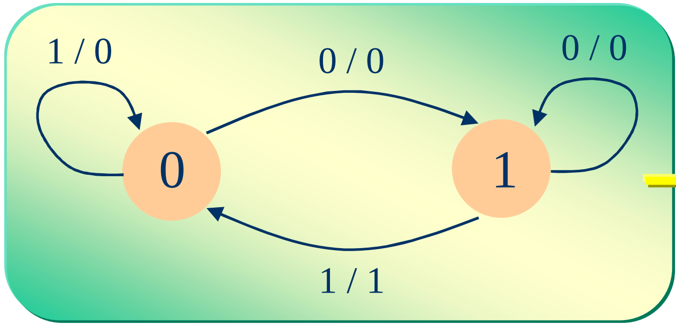
➢ The state of the system and its outputs depend on its previos state and its inputs



• *Structure of a finite state machine:*

```
architecture description of fsm is
type state is (s0, s1, ..., sn);
signal current_state, future_state: state;
begin
      future_state <=     s0 when (inputs="..." and current_state="...")
C1                        else s1 when ...
                          else ...;
      registers: process
      begin
         wait until (clk'event and clk='1');
         if (clearn='0') then
                  current_state <= s0;
         else
                  current_state <= future_state;
         end if;
      end process;
      outputs <=          "..." when (inputs="..." and current_state="...")
C2                        else "..." when ...
                          else ...;
```

- **Example:** Detector for the sequence "01"




```
architecture fsm of detector is
type state is (s0, s1);
signal current_state, future_state: state;
begin
    future_state <=   s0 when (data_in='1')
                      else s1;
    registers: process
    begin
        wait until (clk'event and clk='1');
        if (clearn='0') then
            current_state <= s0;
        else
            current_state <= future_state;
        end if;
    end process;
    detect <= '1' when ((current_state=s1) and
                 (data_in='1'))
                else '0';
end fsm;
```

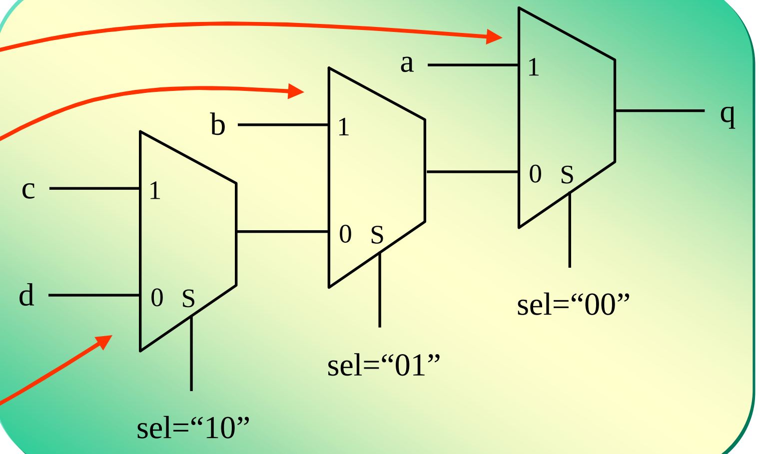
2. Behavioral models of systems

- **Variables vs Signals**

Signals	*Variables*
➢ Declared in **architecture**	➢ Declared in **process, block,** function, procedure
➢ Assignment **with delay**	➢ Assignment **without delay**
➢ Updated **at the end of a process**	➢ Updated **immmediatley**
➢ **Slow** simulation	➢ **Fast** simulation

• *If primitive as a priority encoder (I):*

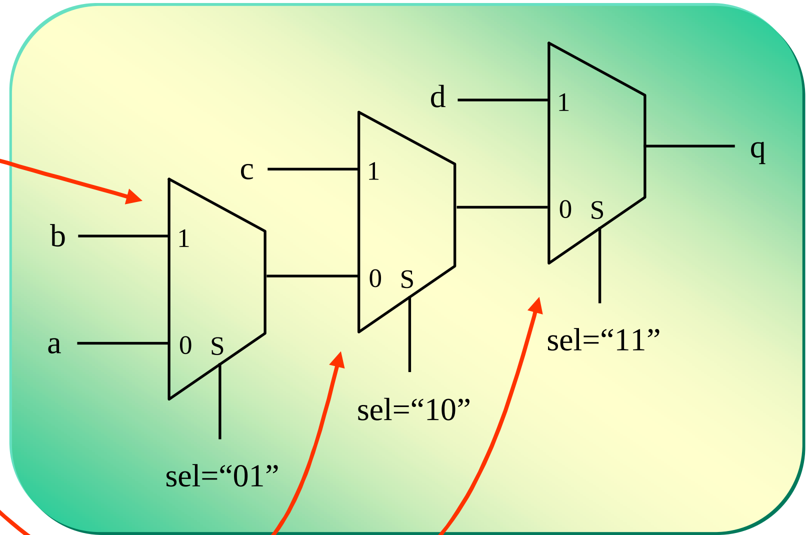
```
process (a,b,c,d,sel)
begin
        if (sel="00") then
                q <= a;
        elsif (sel="01") then
                q <= b;
        elsif (sel="10") then
                q <= c;
        else q <= d;
        end if;
end process;
```




```
q <=    a when (sel="00")
        else b when (sel="01")
        else c when (sel="10")
        else d;
```

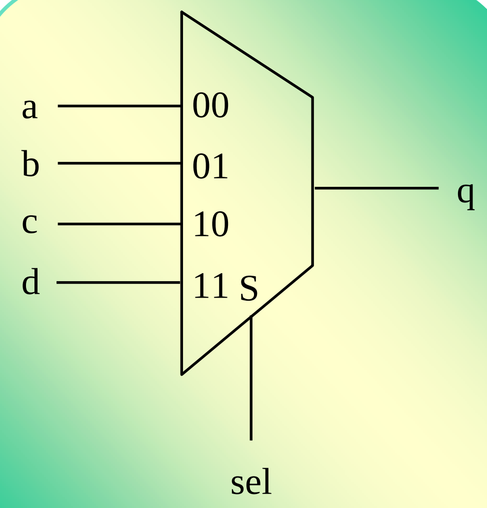
• If primitive as a priority encoder (II):

```
process (a,b,c,d,sel)
begin
        q <= a;
        if (sel="01") then
                q <= b;
        end if;
        if (sel="10) then
                q <= c;
        end if;
        if (sel="11") then
                q <= d;
        end if;
end process;
```



• *Case primitive as a multiplexer:*

```
process (a,b,c,d,sel)
begin
        case sel is
                when "00" => q <= a;
                when "01" => q <= b;
                when "10" => q <= c;
                when others => q <= d;
        end case;
end process;
```

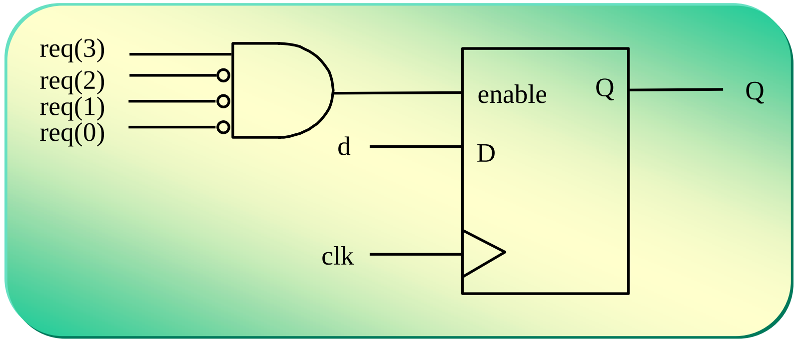



```
with sel select
        q <=    a when "00",
                b when "01",
                c when "10",
                d when others;
```

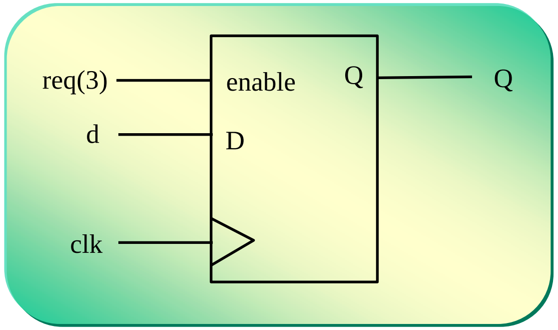
- ***Case:*** Complex decoding
- ***If:*** Selection with critical delay

• Incomplete decoding:

```
if (clk'event and clk='1')
        if (req="1000") then
                q <= d;
        end if;
end if;
```

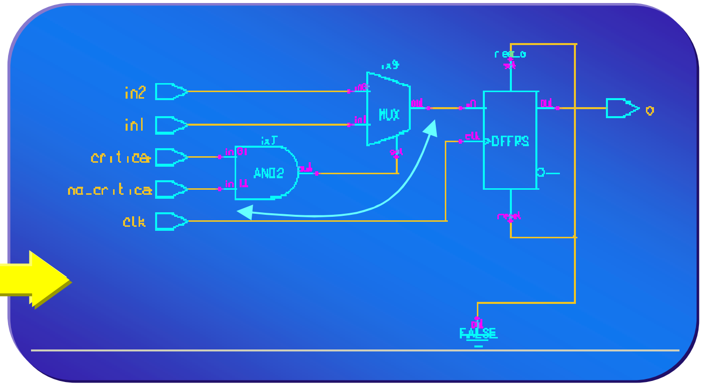



```
if (clk'event and clk='1')
        if (req(3)='1') then
                q <= d;
        end if;
end if;
```

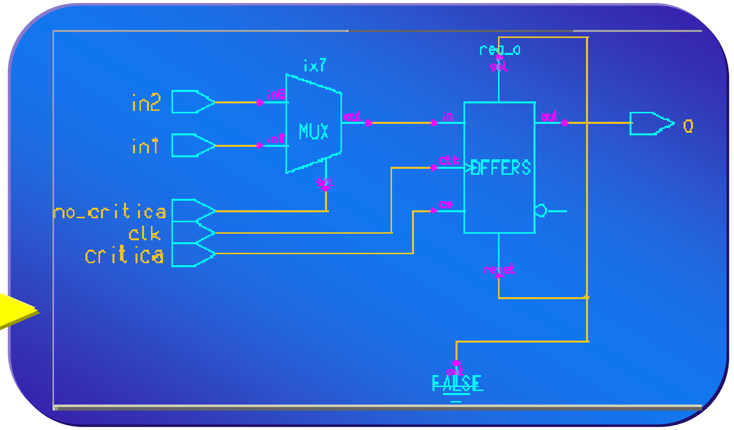


• Signals with critical delay:

```
if (clk'event and clk='1') then
    if (crtica='1' and no_critica='1') then
        o <= in1;
    else
        o <= in2;
    end if;
end if;
```

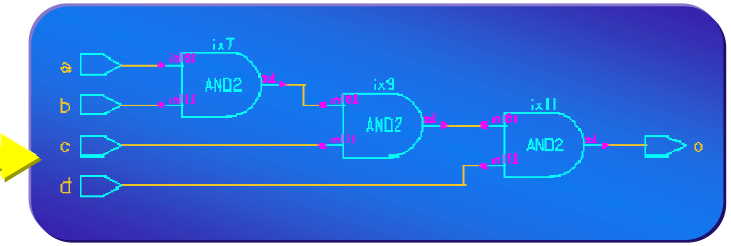



```
if (clk'event and clk='1') then
    if (crtica='1') then
        if (no_critica='1') then
            o <= in1;
        else
            o <= in2;
        end if;
    end if;
end if;
```

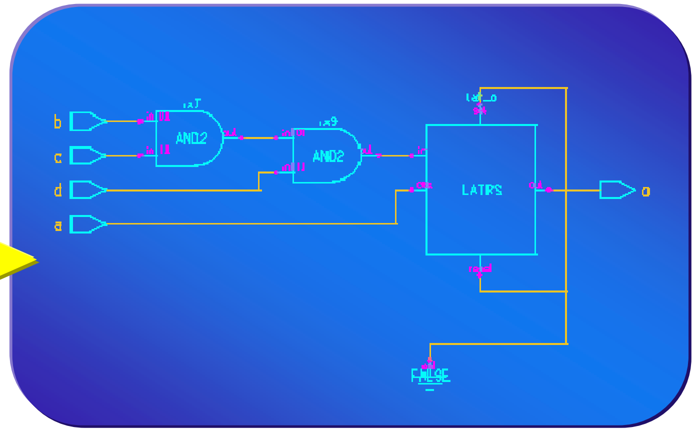


• Latch inference (I):

```
process(a,b,c,d)
begin
    if (a='1' and b='1' and c='1' and d='1') then
        o <= '1';
    else
        o <= '0';
    end if;
end process;
```

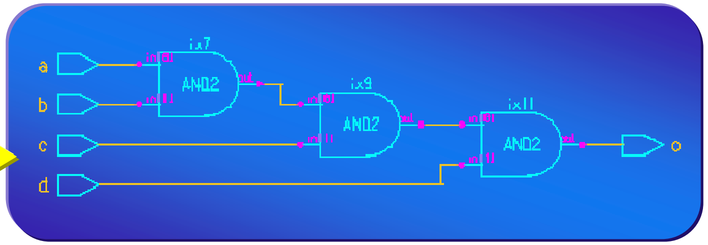



```
process(a,b,c,d)
begin
    if (a='1') then
        if (b='1' and c='1' and d='1') then
            o <= '1';
        else
            o <= '0';
        end if;
    end if;
end process;
```

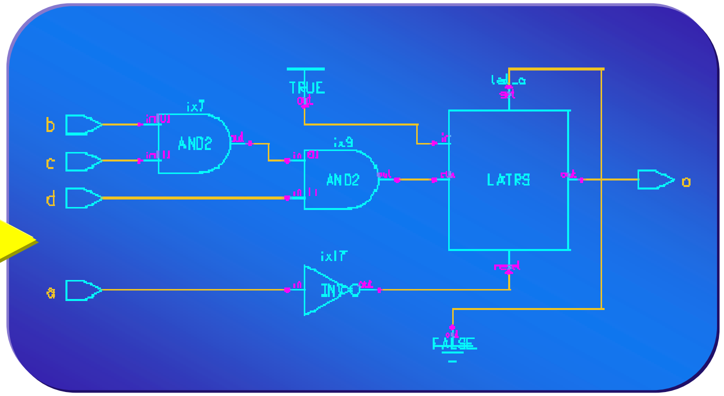


• Latch inference (II):

```
process(a,b,c,d)
begin
    if (a='1' and b='1' and c='1' and d='1') then
        o <= '1';
    else
        o <= '0';
    end if;
end process;
```

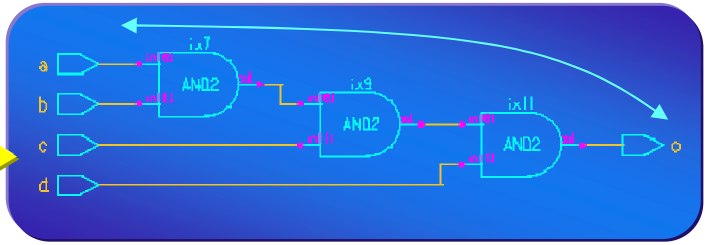



```
process(a,b,c,d)
begin
    if (a='1') then
        if (b='1' and c='1' and d='1') then
            o <= '1';
        end if;
    else
        o <= '0';
    end if;
end process;
```

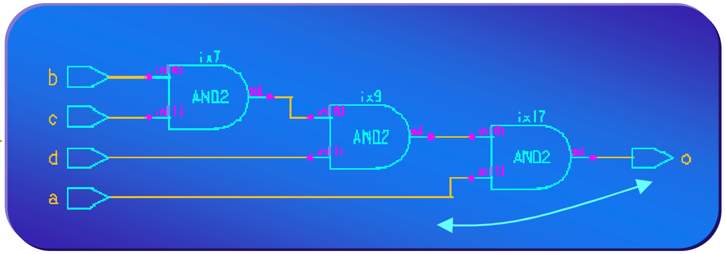


• Priority for critical signals (I):

```
process(a,b,c,d)
begin
    if (a='1' and b='1' and c='1' and d='1') then
        o <= '1';
    else
        o <= '0';
    end if;
end process;
```

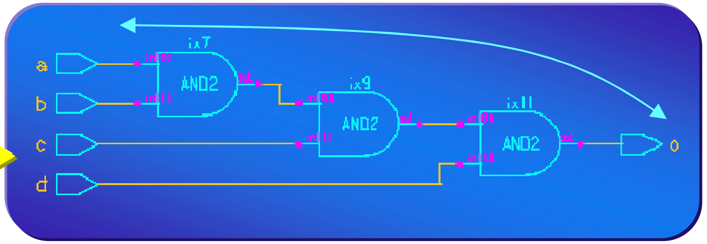



```
process(a,b,c,d)
begin
    if (a='1') then
        if (b='1' and c='1' and d='1') then
            o <= '1';
        else o <= '0';
        end if;
    else
        o <= '0';
    end if;
end process;
```

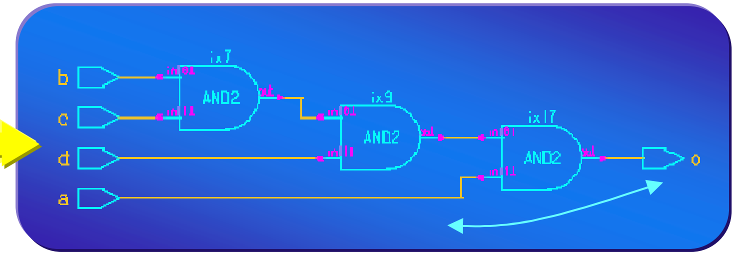


• Priority for critical signals (II):

```
process(a,b,c,d)
begin
    if (a='1' and b='1' and c='1' and d='1') then
        o <= '1';
    else
        o <= '0';
    end if;
end process;
```

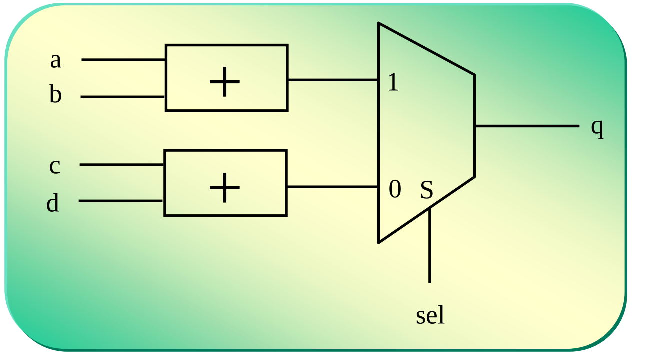



```
process(a,b,c,d)
begin
    if (a='1' and (b='1' and c='1' and d='1')) then
        o <= '1';
    else
        o <= '0';
    end if;
end process;
```

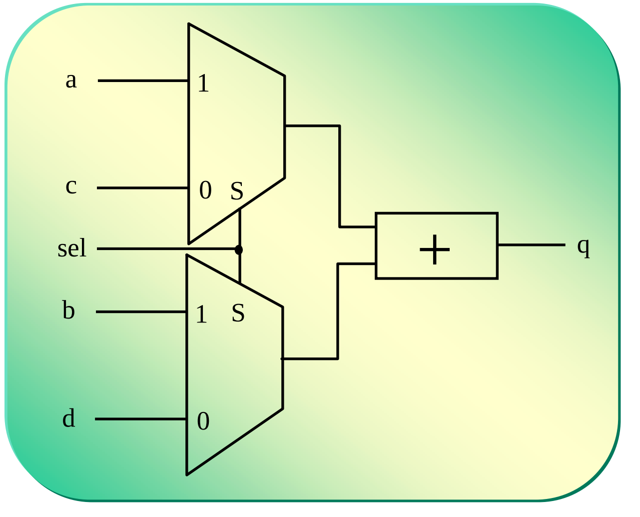


• Resource sharing:

```
if (sel)
        q <= a+b;
else
        q <= c+d;
end if;
```




```
if (sel)
        op1 <= a;
        op2 <= b;
else
        op1 <= c;
        op2 <= d;
end if;
q <= op1+op2;
```



• Scheduling of operations:

- sel1, sel2 **mutually exclusive**

```
if (sel1) then
        q <= a+b;
elsif (sel2) then
        q <= a+c;
end if;
```

- 1 control level
- 1 adder

```
if (sel1) then
        q <= a+b;
else
        if (sel2) then
                q <= a+c;
        end if;
end if;
```

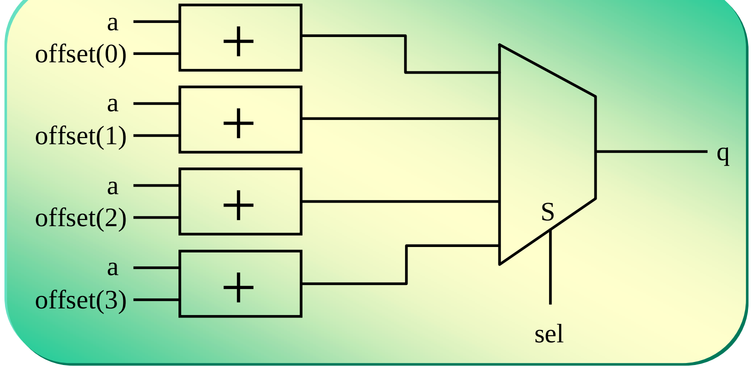
- 1-2 control levels
- 1 adder

```
if (sel1) then
        q <= a+b;
end if;
if (sel2) then
        q <= a+c;
end if;
```

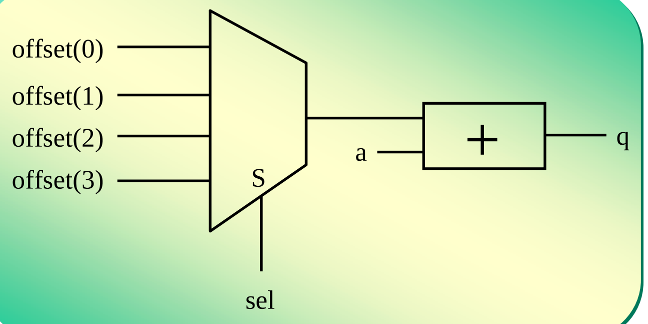
- 3-4 control levels
- 1-2 adders

• Operations inside loops:

```
for i in 0 to 3 loop
        if (sel(i)='1') then
                q <= a + offset(i);
        end if;
end loop;
```

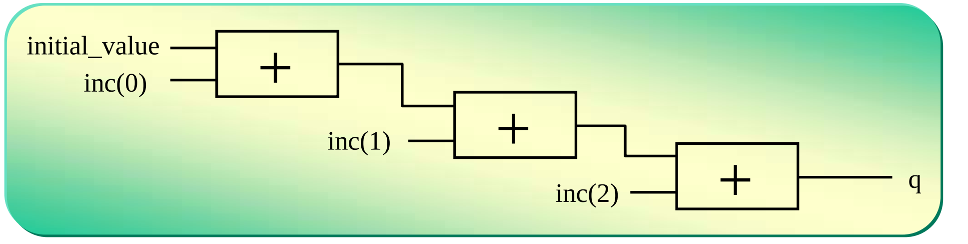



```
for i in 0 to 3 loop
        if (sel(i)='1') then
                add_offset <= offset(i);
        end if;
end loop;
q <= a+add_offset;
```

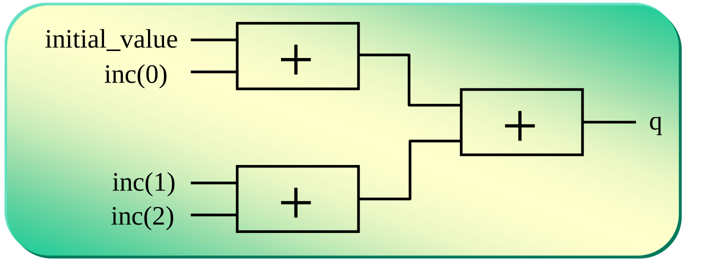


• Loop expansion:

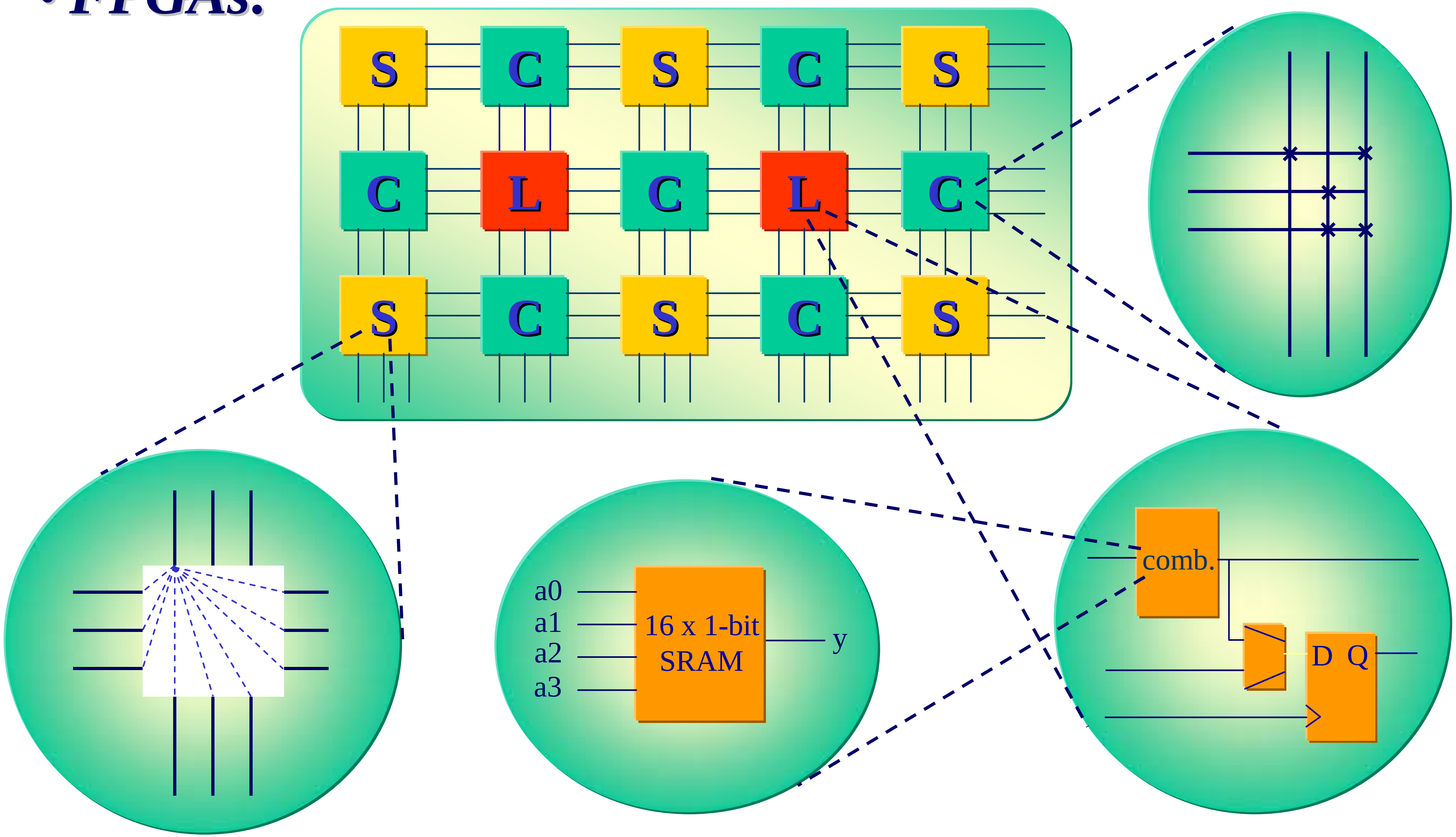
```
q <= initial_value;
for i in 0 to 2 loop
        q <= q+inc(i);
end loop;
```




```
q <=   ((initial_value+inc(0)) +
        (inc(1)+inc(2)));
```



• **FPGAs:**



• Synthesis principles for FPGAs:

- **Limited resources:**
  - combinational elements
  - sequential elements
  - connectivity
- **Execution delay:**

$$t_{total} = t_{component} + t_{routing} + t_{switch}$$

$t_{routing}$, $t_{switch} >> t_{component}$, **and unknown before the physical design (P & R)** → Values to minimise

- ***Minimisation of the connectivity delay:***

  - **Use specific hardware macros:**

    - Adders / subtractors
    - Shifters / rotators
    - Counters
    - Shift registers
    - Memory blocks

  - **One-hot encoding for finite state machines**

3. The standard synthesis package IEEE 1076.3-97

- Interpretation for synthesis of the types *BIT*, *BOOLEAN* and *STD_ULOGIC*
- Definition of the function *STD_MATCH*
- Definition of functions to handle signal edges
- Definition of function for representing signed and unsigned values, as well as conversion functions between types

- **type** STD_ULOGIC **is** (‘1’, ‘0’, ‘H’, ‘L’, ‘U’, ‘X’, ‘W’, ‘-’, ‘Z’);

  STD_ULOGIC → resolved → STD_LOGIC

  ‘1’, ‘0’, ‘H’, ‘L’ → logical values

  ‘U’, ‘X’, ‘W’, ‘-’, ‘Z’ → metalogical values

- **Interpretation of logical values:**
  - **Interpretation as a value 0:**
    - The value ‘0’ of type BIT
    - The value FALSE of type BOOLEAN
    - The values ‘0’ and ‘L’ of type STD_ULOGIC
  - **Interpretation as a value 1:**
    - The value ‘1’ of type BIT
    - The value TRUE of type BOOLEAN
    - The values ‘1’ and ‘H’ of type STD_ULOGIC
- **Interpretation of metalogical values:**
  - **Relational expressions:** In an equality relation, if one operand is a metalogical value and the other is not a static value, the result should be the value FALSE

- **Interpretation of metalogical values:**
  - ***Case* primitives:** If a metalogical value occurs in a choice, the synthesis tool shall interpret the choice as one that can never occur
  - **Arithmetic, logic and shift operations:** The synthesis tool shall treat the operation as an error

- **Interpretation of the high impedance ('Z') value:**
  - If the value 'Z' occurs in an assignment, the synthesis tool shall convert the assignment in a tri-state buffer
  - If the value 'Z' occurs in any construction different from an assignment, the synthesis tool shall treat it as a metalogical value

- **The STD-MATCH function:**
  - ➢ If this function is applied to two arguments of type STD_ULOGIC, it will return the TRUE value if:
    - Both values are well-defined and are the same, or
    - One value is ‘0’ and the other is ‘L’, or
    - One value is ‘1’ and the other is ‘H’, or
    - At least one of the values is ‘-’
  - ➢ If this function is applied to two arguments that are vectors of type STD_ULOGIC, it will return the TRUE value if:
    - The operands have the same length, and
    - STD_MATCH applied to each pair of matching elements returns TRUE
- **Edge detection:** *rising_edge(signal), falling_edge(signal)*

- **Standard arithmetic packages:**
  - **NUMERIC_BIT:** Based on the BIT type
  - **NUMERIC_STD:** Based on the STD_LOGIC type

- **New types:**
  - **UNSIGNED:** Unsigned interger, with the most significant bit on the left
  - **SIGNED:** Interger in 2's complement, with the most significant bit on the left

4. The VHDL RTL synthesis standard IEEE 1076.6-1999

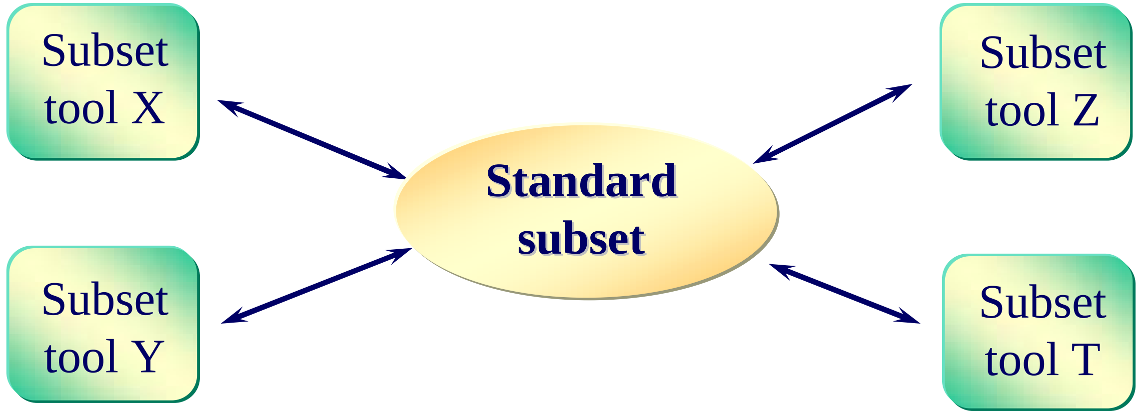
- **Approved on July de 1998 (IEEE Std 1076.6-1999)**
- **Motivation for VHDL: Modelling of electronic systems**

Syntactic and semantic flexibility

- **Impossible to cover the whole language**
- **Tool-dependent synthesis**

Non-portable synthesis

- **Motivation for a standard synthesis subset:**
  - **Portability between tools**



  - **Design portability (specification language):**

• The VHDL RTL synthesis standard (IEEE 1076.6)

• RTL description:

- Data transfer between memory elements (registers)
- Combinational logic

• Supported data types:

- BIT, BOOLEAN, BIT_VECTOR, CHARACTER, STRING, INTEGER (1076-2000)
- STD_ULOGIC, STD_ULOGIC_VECTOR, STD_LOGIC, STD_LOGIC_VECTOR (1164-1993)
- SIGNED, UNSIGNED (1076.3-97)

- ***Sequential primitives active by edge (flipflops):***
  - **Clock signal:** Types BIT, STD_ULOGIC, STD_LOGIC
  - **Specification of a clock (rising) edge:**

- **With the *if* statement:**
  - **rising_edge**(clock)
  - clock**'event** and clock='1'
  - clock='1' and clock**'event**
  - **not** clock**'stable** and clock='1'
  - clock='1' and **not** clock**'stable**

- **Specification of a clock (rising) edge:**
  - **With the *wait until* statement:**
    - **rising_edge**(clock)
    - clock=‘1’
    - clock**’event** and clock=‘1’
    - clock=‘1’ and clock**’event**
    - **not** clock**’stable** and clock=‘1’
    - clock=‘1’ and **not** clock**’stable**
- **Only one clock edge per process** is allowed
- The **wait until** is not allowed within procedures

• Modelling:

```
process(clock)
begin
    if (clock'event and clock='1') then
        q <= d;
    end if;
end process;
```

```
process
begin
    wait until (clock='1');
        q <= d;
end process;
```

- **One** per process
- **First** statement

```
process
    variable var: unsigned(3 downto 0);
begin
    wait until (clock ='1');
        var := var +1;
        count <= var;
end process;
```

Variable read before being assigned

- **Asynchronous control signals:**

```
process(clk, set, reset)
begin
        if (reset='1') then
                Q <= '0';
        elsif (set='1') then
                Q <= '1';
        elsif (clk'event and clk='1') then
                Q <= D;
        end if;
end process;
```

• *Sequential primitives active by level (latches):*

➢ **Necessary inference:**

- One signal or variable is assigned in a process without definition of clock signals **and**
- There are executions of the process that do not imply an explicit assignment on the signal or variable

➢ **Probable inference:**

- One signal or variable is assigned in a process without definition of clock signals **and**
- There are executions of the process where the value of the signal or variable is read before it is assigned

- **Tri-state buffers:** Conditional assignment of the value 'Z' **on a signal or variable**

- **Modelling of combinational logic:**
  - Concurrent assignments
  - Assignments within a process that take place with **every** execution of the process

- **Meta-comments:**

```
-- RTL_SYNTHESIS ON
-- RTL_SYNTHESIS OFF
```

- **ENUM_ENCODING attribute:** Used to instruct the synthesis tools the coding to be used for the elements that constitute an enumerated type (finite state machines)

attribute ENUM_ENCODING: string;

type state is (s0, s1, s2, s3);

attribute ENUM_ENCODING **of** state: **type is** “0001 0010 0100 1000”;