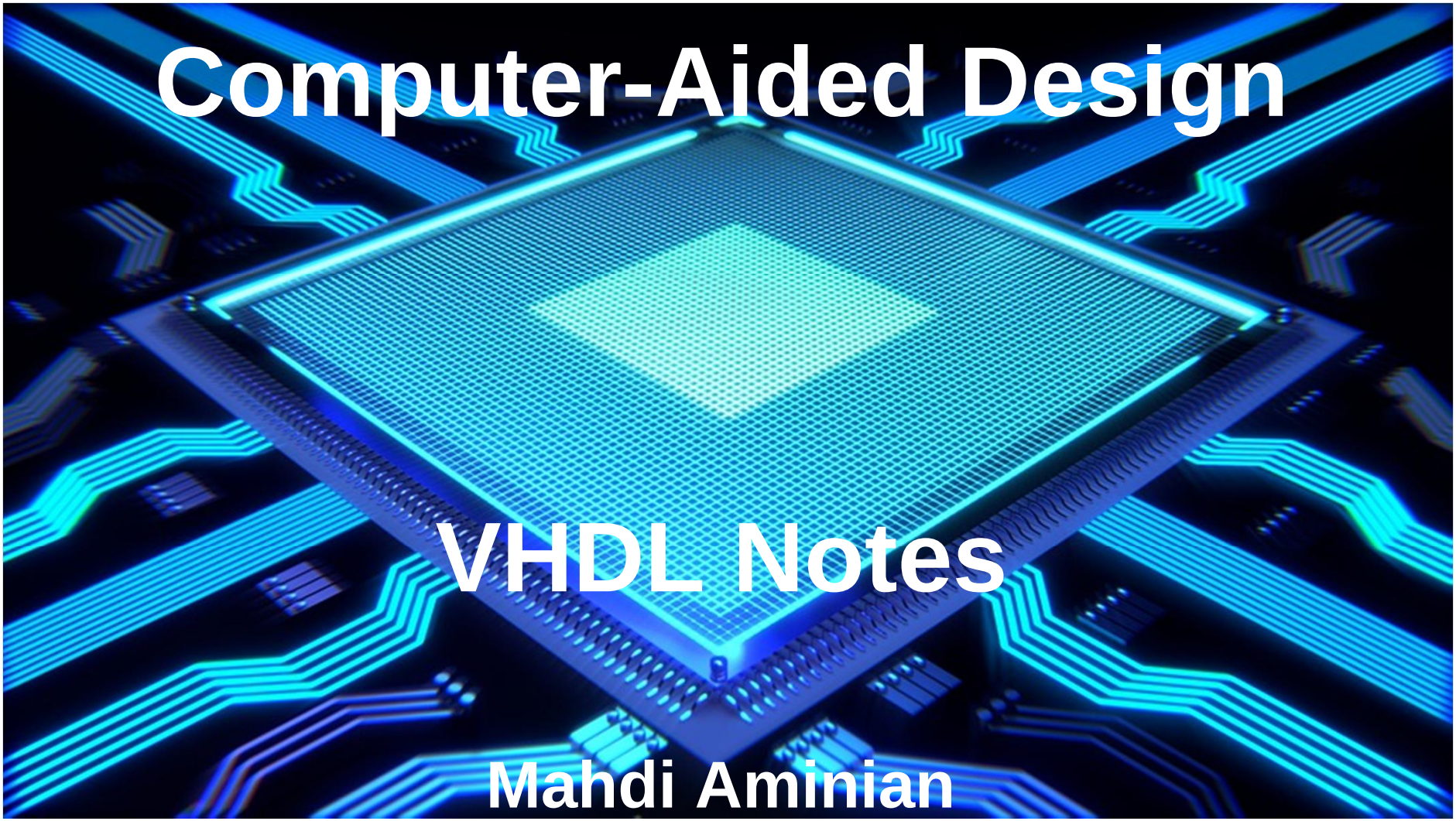


Computer-Aided Design

VHDL Notes

Mahdi Aminian

The University Of Guilan

Rasht - Iran

Some slides courtesy of:

- “Hardware Systems Modeling”, A. Vachoux, EPFL
- CAD slides from Dr. Saheb Zamani

Signed / Unsigned Numbers

Bit-Based Numerical Data Representation

- **Type unsigned and signed**

```
-- unsigned integer value
type unsigned is array (natural range <>) of std_logic;
-- signed integer value
type signed is array (natural range <>) of std_logic;
```

  - Examples:

```
signal au : unsigned(7 downto 0);  -- values 0 to 255
signal as : signed(7 downto 0);    -- values -128 to 127 (2's-complement)
```

- **Supported operators:**

Relational:	= /= < <= > >=
Logical:	and or nand nor not xor xnor
Arithmetic:	abs(1) sign -(1) + - * / rem mod
Shift and rotate:	sll srl sla(2) sra(2) rol ror
Concatenation:	&

(1) *type signed only* (2) *VHDL-2008 only*

```
-- required context clause
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

Shift Operators

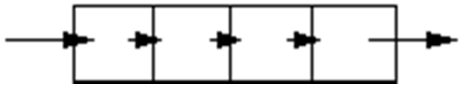
```
signal A_BUS, B_BUS, Z_BUS : bit_vector (3 downto 0);

Z_BUS <= A_BUS sll 2;
Z_BUS <= B_BUS sra 1;
Z_BUS <= A_BUS ror 3;
```

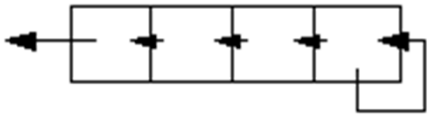
At the end, the first value of the type is used for filling up

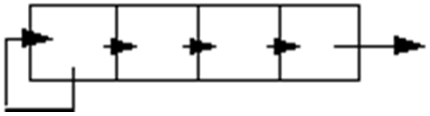
- **Logical Shift**

sll 

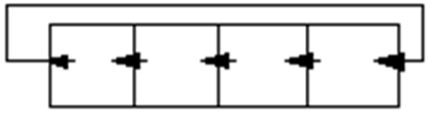
srl 

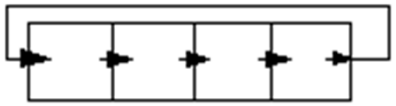
- **Arithmetic Shift**

sla 

sra 

- **Rotation**

rol 

ror 

Working with std_(u)logic(_vector) and (un)signed

- **Given the following declarations (also valid for variables)**

```
signal A_sul  : std_ulogic;
signal B_sl   : std_logic;
signal C_sulv : std_ulogic_vector(7 downto 0);
signal D_slv  : std_logic_vector(7 downto 0);
signal E_uv   : unsigned(7 downto 0);
signal F_sv   : signed(7 downto 0);
```

- **Compatible types**

```
A_sul    <= B_sl;
B_sl     <= A_sul;
A_sul    <= C_sulv(7);
D_slv(2) <= C_sulv(0);
```

```
-- VHDL-2008 only
C_sulv <= D_slv;
D_slv  <= C_sulv;
```

- **Type casting**

```
C_sulv <= std_ulogic_vector(D_slv);
D_slv  <= std_logic_vector(C_sulv);
E_uv   <= unsigned(D_slv);
F_sv   <= signed(D_slv);
```

```
signal D2_slv, D3_slv : std_logic_vector(D_slv'range);
D_slv <= std_logic_vector(unsigned(D2_slv) + unsigned(D3_slv));
```

```
signal C2_sulv, C3_sulv : std_ulogic_vector(7 downto 0);
signal F2_sv : signed(8 downto 0);
F2_sv  <= signed('0' & C2_sulv) - signed('0' & C3_sulv);
```

Working with (un)signed and integer

- **Given the following declarations (also valid for variables)**

```
signal E_uv  : unsigned(7 downto 0);
signal F_sv  : signed(7 downto 0);
signal G_int : integer;
signal H_nat : natural;
```

- **Conversion functions**

```
G_int <= to_integer(E_uv);
H_nat <= to_integer(E_uv);

G_int <= to_integer(F_sv);
```

```
E_uv <= to_unsigned(G_int, E_uv'length);  -- to_unsigned(arg, size)
F_sv <= to_signed(G_int, F_sv'length);    -- to_signed(arg, size)
```

```
signal G2_int, G3_int : integer;
E_uv <= to_unsigned((G2_int + G3_int), E_uv'length);
F_sv <= to_signed((G2_int - G3_int), F_sv'length);
```

```
signal F2_sv, F3_sv : signed(7 downto 0);
G_int <= to_integer(F2_sv + F3_sv);
```

Variable

1-Bit Full Adder: Algorithmic Architecture

- **Variable objects can only be declared and used in a process**
  - It is forbidden to declare signals in a process
- **Sequential statements**
  - Variable assignment statement uses the delimiter ":="
- **Variables get their new values immediately**
- **Variables retain their state between process activations**

```
architecture algo of fadd1 is
   use ieee.numeric_std.all;
begin
   process (opa, opb, cin)
      variable tmp : unsigned(1 downto 0);
   begin
      tmp := (others => '0');
      if opa = '1' then tmp := tmp + 1; end if;
      if opb = '1' then tmp := tmp + 1; end if;
      if cin = '1' then tmp := tmp + 1; end if;
      if tmp > 1 then
         cout <= '1';
      else
         cout <= '0';
      end if;
      if tmp mod 2 = 0 then
         sum <= '0';
      else
         sum <= '1';
      end if;
   end process;
end architecture algo;
```

Signals or Variables?

Signals	Variables
Represent hardware signals	Represent (temporary) storage
Global scope (design entity)	Local scope (process)
Complex data structure (driver)	Memory location only
Assignment with "<="	Assignment with ":="
Updated in one step after all processes have finished executing	Updated immediately after the assignment
Most appropriate for expressing concurrent hardware behavior	Most appropriate for expressing algorithmic (procedural) behavior

- Signals(variables) can be assigned to variables(signals) of compatible types

Loop Example

```
architecture sfor of reduced_xor is
   signal tmp : std_logic_vector(3 downto 0);
begin
   process (a, tmp)
   begin
      tmp(0) <= a(0);
      for i in 1 to 3 loop
         tmp(i) <= a(i) xor tmp(i-1);
      end loop;
      z <= tmp(3);
   end process;
end architecture sfor;
```

```
entity reduced_xor is
   port (
      signal a : in  std_logic_vector(3 downto 0);
      signal z : out std_logic);
end entity reduced_xor;
```

```
architecture sfor2 of reduced_xor is
begin
   process (a)
      variable tmp : std_logic;
   begin
      tmp := a(0);
      for i in 1 to 3 loop
         tmp := a(i) xor tmp;
      end loop;
      z <= tmp;
   end process;
end architecture sfor2;
```

```
entity reduced_xor is
   port (
      signal a : in  std_logic_vector(3 downto 0);
      signal z : out std_logic);
end entity reduced_xor;
```

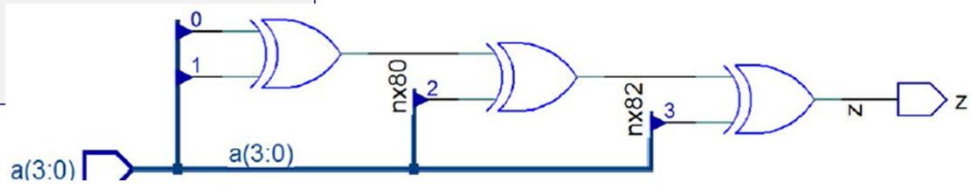
a(3:0)

Loop Example

```
architecture sfor of reduced_xor is
   signal tmp : std_logic_vector(3 downto 0);
begin
   process (a, tmp)
   begin
      tmp(0) <= a(0);
      for i in 1 to 3 loop
         tmp(i) <= a(i) xor tmp(i-1);
      end loop;
      z <= tmp(3);
   end process;
end architecture sfor;
```

```
entity reduced_xor is
   port (
      signal a : in  std_logic_vector(3 downto 0);
      signal z : out std_logic);
end entity reduced_xor;
```

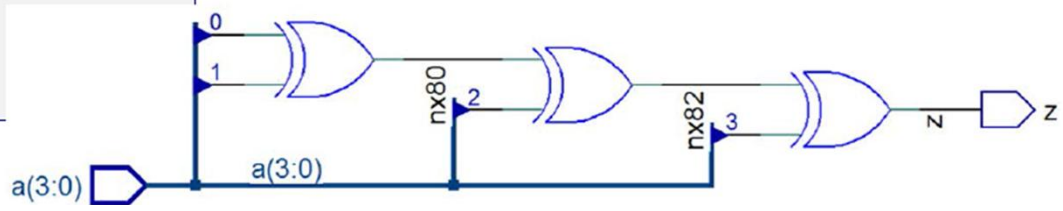
$$z = a(3) \oplus a(2) \oplus a(1) \oplus a(0)$$




```
architecture sfor2 of reduced_xor is
begin
   process (a)
      variable tmp : std_logic;
   begin
      tmp := a(0);
      for i in 1 to 3 loop
         tmp := a(i) xor tmp;
      end loop;
      z <= tmp;
   end process;
end architecture sfor2;
```

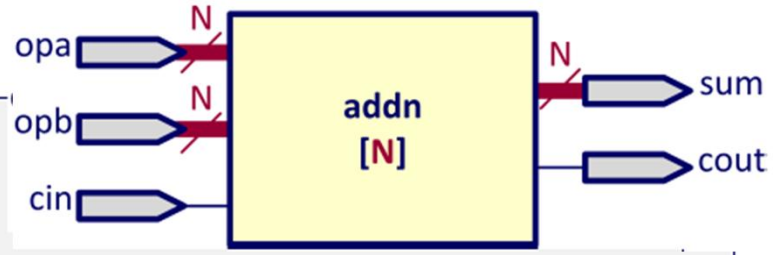
```
entity reduced_xor is
   port (
      signal a : in  std_logic_vector(3 downto 0);
      signal z : out std_logic);
end entity reduced_xor;
```

$$z = a(3) \oplus a(2) \oplus a(1) \oplus a(0)$$



Generic

Generic N-bit Adder: Dataflow Architecture




```
library ieee;
use ieee.std_logic_1164.all;
entity addn is
   generic (NBITS : positive := 8);  -- bus size, default value of 8
   port (
      signal opa, opb : in  std_logic_vector(NBITS-1 downto 0);
      signal cin      : in  std_logic;
      signal sum      : out std_logic_vector(NBITS-1 downto 0);
      signal cout     : out std_logic);
end entity addn;

architecture dfl of addn is
   signal cc : std_logic_vector(NBITS downto 0);  -- internal carry chain
begin
   cc(0) <= cin;
   STAGES : for i in 0 to NBITS-1 generate
      PS : sum(i)  <= opa(i) xor opb(i) xor cc(i);
      PC : cc(i+1) <= (opa(i) and opb(i)) or (cc(i) and (opa(i) or opb(i)));
   end generate STAGES;
   cout <= cc(NBITS);
end architecture dfl;
```

Generic N-Bit Adder: Structural Architecture

```
architecture str of addn is
   signal cc : std_logic_vector(NBITS downto 0);  -- internal carry chain
begin
   cc(0) <= cin;
   STAGES : for i in NBITS-1 downto 0 generate
      FA1 : entity work.fadd1(dfl)
         port map (
            opa  => opa(i),
            opb  => opb(i),
            cin  => cc(i),
            sum  => sum(i),
            cout => cc(i+1));
   end generate STAGES;
   cout <= cc(NBITS);
end architecture str;
```

- **Duplicated elements are now direct instances of design entities fadd1(dfl)**

Instantiation of a 32-bit Adder

```
library ieee;
use ieee.std_logic_1164.all;
entity hw_system is
   port (...);
end entity hw_system;

architecture str of hw_system is
   ...
begin
   ...
   ADD32 : entity work.addn(dfl)   -- or addn(algo) or addn(str)
      generic map (NBITS => 32)
      port map (...);
   ...
end architecture str;
```

- **Actual values of instance parameters are defined by a generic map clause**
  - Could be omitted as a default value of 8 is defined in the entity addn
  - Sometimes a better practice is to have generic parameters without a default value

Synthesis Notes

[Synthesis] Simple Concurrent Signal Assignment

- **Delay clauses are ignored**

```
sum  <= opa xor opb xor cin after 2 ns;
```

- **Multiple element waveforms are not allowed**

```
s <= '1', '0' after 10 ns, '1' after 18 ns, '0' after 25 ns;
```

- **Supported signal assignment:**

```
signal-name <= value-expression ;
```

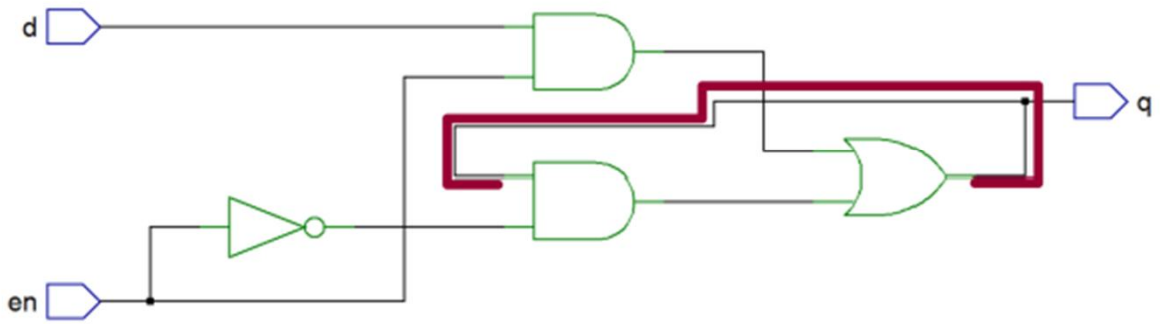
- **Examples:**

```
status <= '1';
even <= (p1 and p2) or (p3 and p4);
```

[Synthesis] Avoid Zero-Delay Feedback Loops

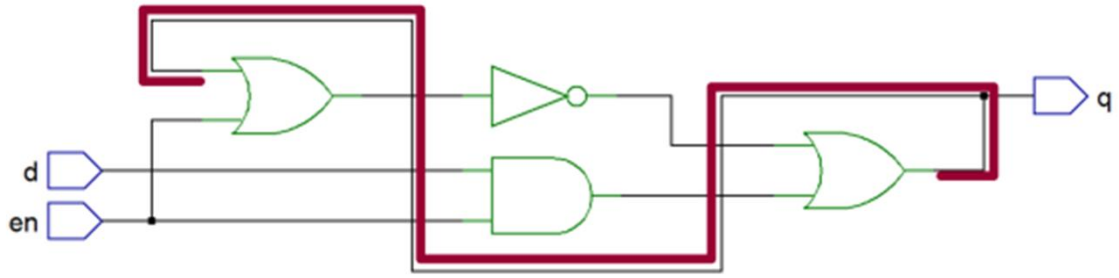
- **Bad expression of state retention**

```
q <= (q and (not en)) or (d and en);
```



- **Signal oscillation**

```
q <= ((not q) and (not en)) or (d and en);
```



[Synthesis] Combinational Logic from a Process Statement

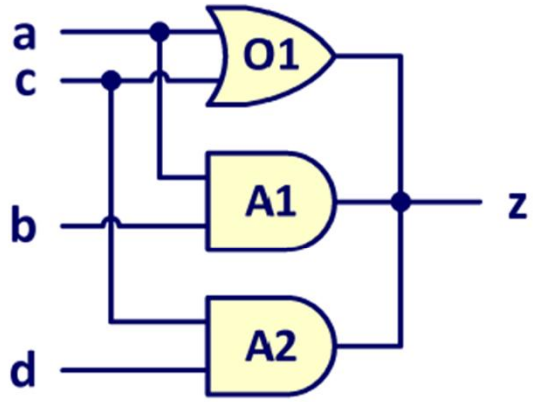
- **A process expresses a combinational behavior if and only if all following conditions are met:**
  1. The process has a sensitivity list
  2. All signals that are read in the process are in the sensitivity list
  3. The process does not declare variables or, if it does, variables are always assigned before they are read
  4. All variables or signals are assigned in every branch of the execution flow in the process body (if or case statement)

- **Otherwise state or memory is inferred**

[Synthesis] Avoid Multiple Concurrent Signal Assignments

```
entity multiple is
   port (
      signal a, b, c, d : in  std_logic;
      signal z          : out std_logic);
end entity multiple;

architecture dfl of multiple is
begin
   O1 : z <= a or c;
   A1 : z <= a and b;
   A2 : z <= c and d;
end architecture dfl;
```



- **Legal VHDL but incorrect design**
  - Signal z may get the value 'X' in simulation
  - Inferred hardware may not work properly
- **Use of unresolved type std_ulogic would make the analysis to fail**
- **Rather use single-source signals with a mux or three-state buffers**

Simple Sequential Signal Assignment

- Same restrictions as for simple concurrent signal assignment
- BUT, if multiple signal assignments to the same signal, only the last active one counts

```
entity multiple is
   port (
      signal a, b, c, d : in  std_logic;
      signal z          : out std_logic);
end entity multiple;
```

```
architecture proc of multiple is
begin
   process (a, b, c, d)
   begin
      z <= a or c;
      z <= a and b;
      z <= c and d;
   end process;
end architecture proc;
```

=

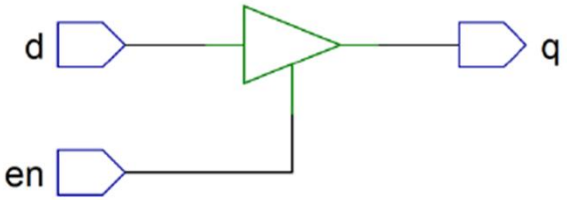
```
architecture proc of multiple is
begin
   process (a, b, c, d)
   begin
      z <= c and d;
   end process;
end architecture proc;
```

[Synthesis] 'Z' Interpretation

- **High-impedance or open circuit**
  - **Three-state buffer**

```
entity bufz is
   port (
      signal en, d : in  std_logic;
      signal q     : out std_logic);
end entity bufz;
```

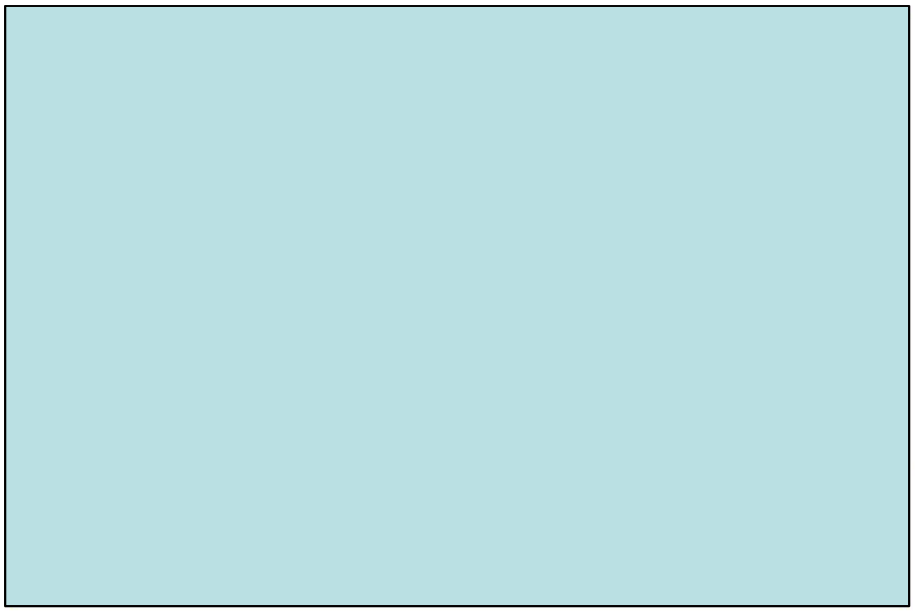
```
architecture bhv1 of bufz is
begin
   process (en, d)
   begin
      if en = '1' then
         q <= d;
      else
         q <= 'Z';
      end if;
   end process;
end architecture bhv1;
```



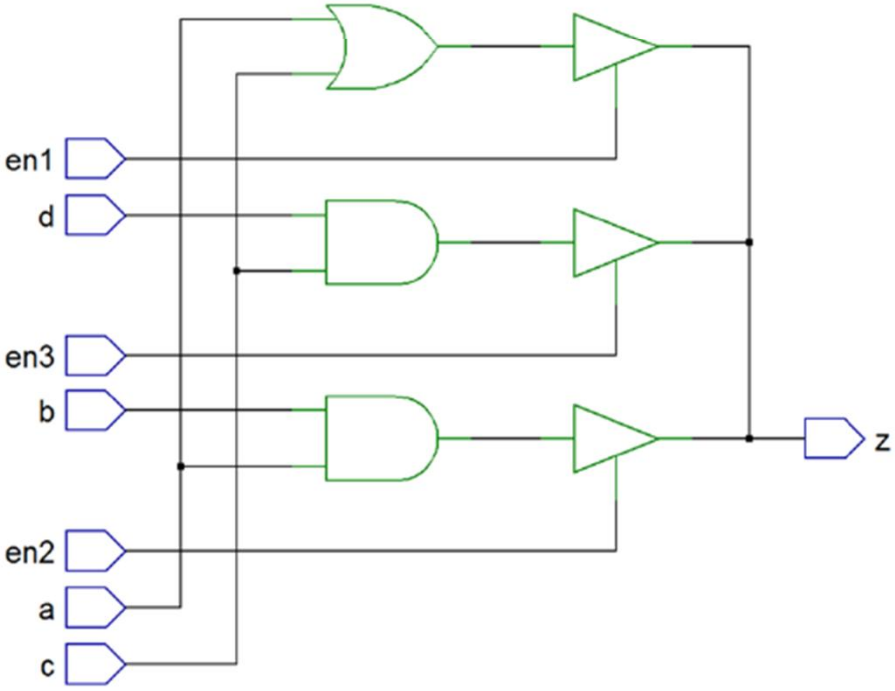

```
architecture bhv2 of bufz is
begin
   process (en, d)
   begin
      q <= 'Z';
      if en = '1' then
         q <= d;
      end if;
   end process;
end architecture bhv2;
```

```
architecture bhv3 of bufz is
begin
   q <= d when en = '1' else
        'Z';
end architecture bhv3;
```

[Synthesis] Multiple Sources and Three-State Buffer



- Separate concurrent statements
- Multiplexer behavior

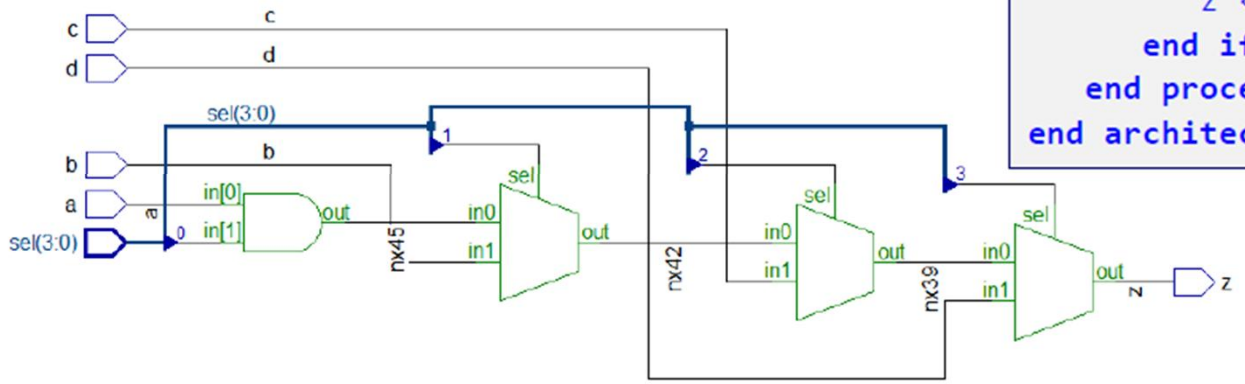


[Synthesis] Sequential If - Priority

```
entity ifstmt is
   port (
      signal a, b, c, d : in std_logic;
      signal sel : in std_logic_vector(3 downto 0);
      signal z : out std_logic);
end entity ifstmt;
```

```
architecture priority of ifstmt is
begin
   process (a, b, c, d, sel)
   begin
      z <= '0';  -- default value
      if sel(0) = '1' then
         z <= a;
      end if;
      if sel(1) = '1' then
         z <= b;
      end if;
      if sel(2) = '1' then
         z <= c;
      end if;
      if sel(3) = '1' then
         z <= d;
      end if;
   end process;
end architecture priority;
```

- **Last if statement has the highest priority => shorter propagation time**



Sequential If Statement Synthesis Pitfalls

- **May infer unwanted memory logic**
- **Redundant test**

```
process (a, b, c, sel1, sel2)
begin
   if sel1 = '1' then
      z <= a;
   elsif sel1 /= '1' and sel2 = '1' then
      z <= b;
   else
      z <= c;
   end if;
end process;
```

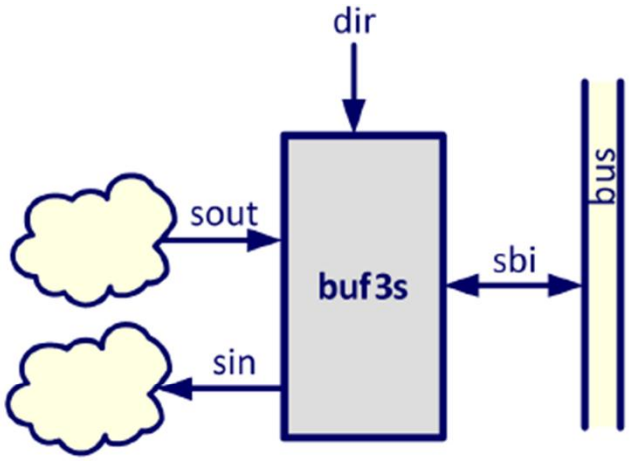
```
process (a, b, sel)
begin
   if sel = '1' then
      z <= a;
   elsif sel /= '1' then
      z <= b;
   end if;
end process;
```

- **Different signal targets**

```
process (a, b, sel)
begin
   if sel = '1' then
      z <= a;
   else
      y <= b;
   end if;
end process;
```

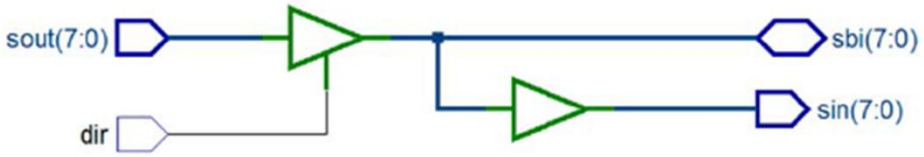
[Synthesis] Bidirectional Bus

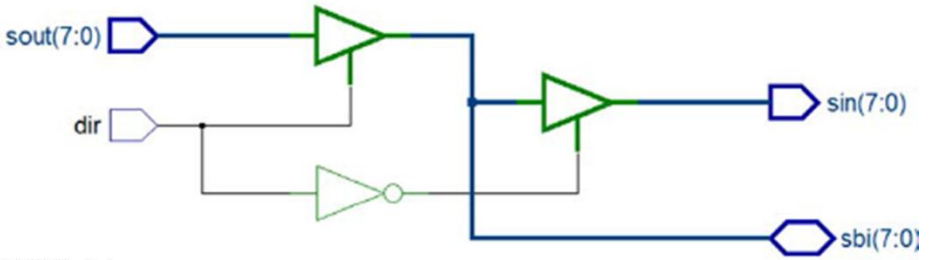
```
entity buf3s is
   port (
      signal sout : in    std_logic_vector(7 downto 0);
      signal sin  : out   std_logic_vector(7 downto 0);
      signal dir  : in    std_logic;
      signal sbi  : inout std_logic_vector(7 downto 0));
end entity buf3s;
```




```
architecture single of buf3s is
begin
   sbi <= sout when dir = '1' else
          (others => 'Z');
   sin <= sbi;
end architecture single;
```

```
architecture dual of buf3s is
begin
   sbi <= sout when dir = '1' else
          (others => 'Z');
   sin <= sbi when dir = '0' else
          (others => 'Z');
end architecture dual;
```





[Synthesis] Simple ALU (1)

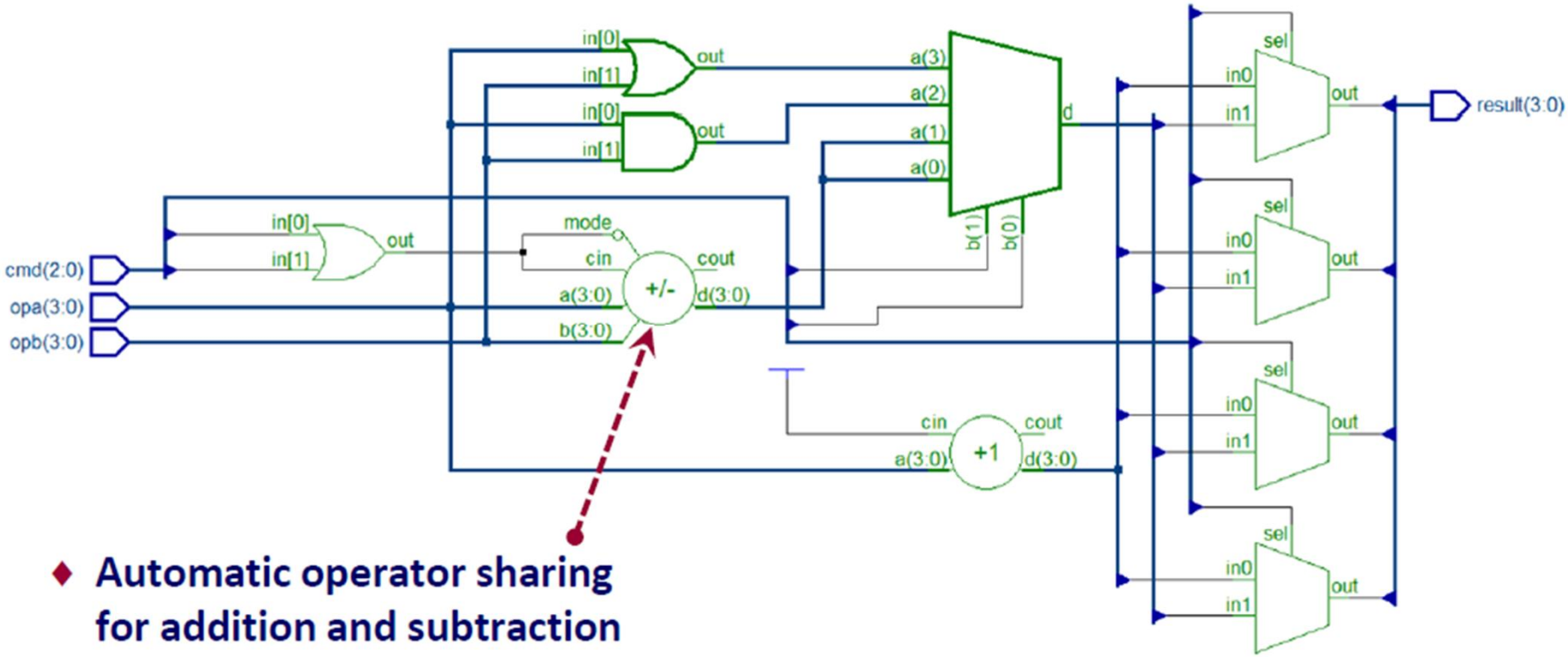
```
library ieee; use ieee.std_logic_1164.all;
entity alu is
   port (
      signal cmd      : in  std_logic_vector(2 downto 0);
      signal opa, opb : in  std_logic_vector(7 downto 0);
      signal result   : out std_logic_vector(7 downto 0));
end entity alu;
```

```
architecture simple_signed of alu is
   use ieee.numeric_std.all;
   signal opas, opbs, inc, sum, diff, res : signed(7 downto 0);
begin
   -- get signed interpretations of input operands
   opas <= signed(opa); opbs <= signed(opb);  -- type casting
   -- compute ALU functions
   inc <= opas + 1; sum <= opas + opbs; diff <= opas - opbs;
   -- select function for signed result
   res <= inc           when cmd(2) = '0' else
          sum           when cmd = "100"  else
          diff          when cmd = "101"  else
          opas and opbs when cmd = "110"  else
          opas or opbs  when cmd = "111"  else
          (others => 'X');
   -- get std_logic_vector interpretation of result
   result <= std_logic_vector(res);  -- type casting
end architecture simple_signed;
```

cmd	result
0--	opa + 1
100	opa + opb
101	opa - opb
110	opa and opb
111	opa or opb

- **Arith operations wrap-round on overflow**

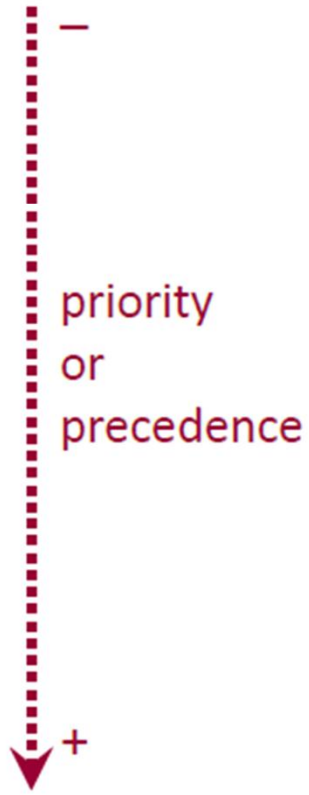
[Synthesis] Simple ALU (2)



- **Automatic operator sharing for addition and subtraction**

[Synthesis] Hardware Realization of Operators

- **Supported in synthesis with limitations**
- **Logical:** or and nor nand xor xnor
- **Relational** [a]**:** = /= < [b] > [b] >= [b] <= [b]
- **Shift & rotate** [c]**:** sll srl sla sra rol ror
- **Addition:** + [b] – [b] & [d]
- **Unary:** + –
- **Multiplication:** * [b],[e],[f] / [g],[h] mod [h] rem [h]
- **Others:** ** [i] abs not



[Synthesis] Hardware Realization of Operators

(a) Result is of type boolean.

(b) Can be shared with another operator of the same kind and same priority level.

(c) Introduced in VHDL-1993. The IEEE 1076.6 standard mentions that they are not supported for synthesis and that the functions SHIFT_LEFT, SHIFT_RIGHT, ROTATE_LEFT and ROTATE_RIGHT from packages NUMERIC_BIT/_STD should be used instead.
When the distance of shift/rotation is constant, the bit rearrangement is hardwired. When the distance is non-constant (e.g. defined by a variable or a signal), a combinational barrel shifter is inferred.

(d) Concatenation operator '&' can be used to emulate shift and rotate operators.

(e) In general infers a combinational circuit. The inference mechanism depends on the tool (e.g. DesignWare from Synopsys).

(f) If the right operand is a multiple of 2, infers a simple left shift.

(g) Right operand must be a power of 2, so a right shift is inferred.

(h) Not recommended to be used in synthesis.

(i) Powered operand must be a constant equal to 2. Some synthesis tools also require that the left operand is a power of two constant value.

Simulation Model

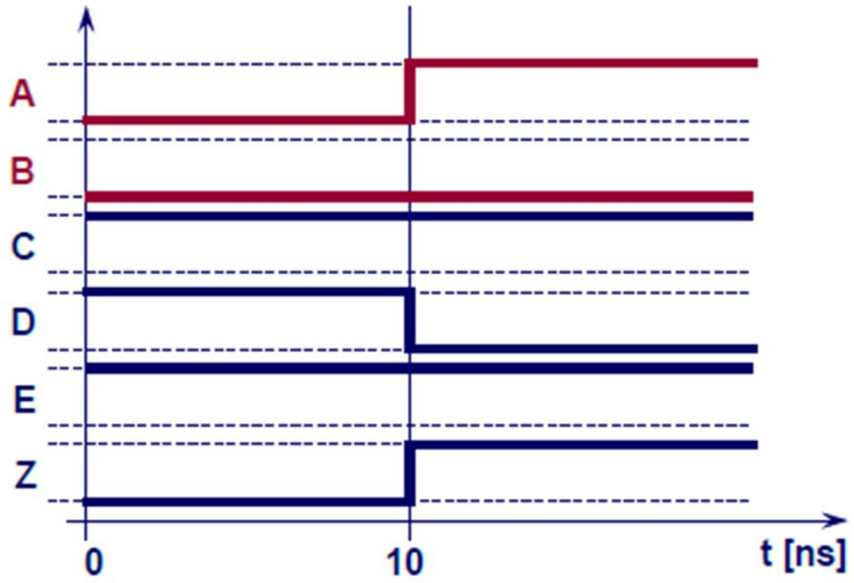
VHDL Simulation Model

- **Initialization phase**
  - **I.1.** Each signal and variable gets a default or defined initial value.
  - **I.2.** Current simulation time Tc := 0.
  - **I.3.** [***Process execution phase***] Each process is executed until it suspends.
  - **I.4.** Computation of the next simulation time Tn, which is the earliest of:
    next time of a pending transaction,
    next time of end of a timeout (wait for), or
    value time'high.
    If Tn = Tc, the next simulation cycle is a *delta cycle*.
- **Simulation cycle**
  - **S.1.** Tc := Tn.
  - **S.2.** [***Signal update phase***] Current values of signals get their new values.
  - **S.3.** [***Process execution phase***] Processes sensitive to signal changes are executed.
  - **S.4.** Tn = next simulation time as in I.4.
    If Tn = time'high and no more pending transactions, simulation stops.
    Otherwise, go to S.1.

Zero Delay Simulation (Delta Cycle)

```
-- Function: Z = A xor B
entity noteq is
   port (
      signal A, B : in  std_logic;
      signal Z    : out std_logic);
end entity noteq;

architecture dfl of noteq is
   signal C, D, E : std_logic;
begin
   P1 : C <= A nand B;
   P2 : D <= A nand C;
   P3 : E <= C nand B;
   P4 : Z <= D nand E;
end architecture dfl;
```



- **Signal drivers (sources): A, B, P1_C, P2_D, P3_E, P4_Z**
  - Sources for signals A and B are assumed to be defined outside the design entity noteq(dfl)

```
A <= '0', '1' after 10 ns;
B <= '0';
```

Zero Delay Simulation (Delta Cycle)

t [ns]	δ	A	B	P1_C	P2_D	P3_E	P4_Z
0	0	'U' [Tc/'0'] [10 ns/'1']	'U' [Tc/'0']	'U'	'U'	'U'	'U'
	1	'0'	'0'	'U' [Tc/'1']	'U' [Tc/'1']	'U' [Tc/'1']	'U'
	2	'0'	'0'	'1'	'1'	'1'	'U' [Tc/'0']
	3	'0'	'0'	'1'	'1'	'1'	'0'
10	0	'1'	'0'	'1' [Tc/'1']	'1' [Tc/'0']	'1'	'0'
	1	'1'	'0'	'1'	'0'	'1'	'0' [Tc/'1']
	2	'1'	'0'	'1'	'0'	'1'	'1'