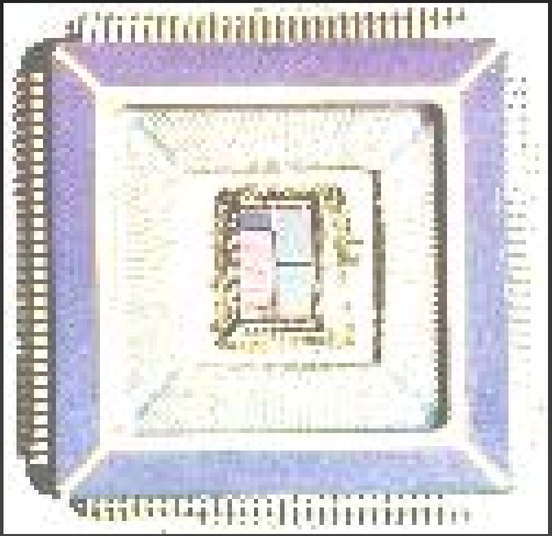


Hardware Modeling

VHDL - Architectures

Vienna University of Technology
Department of Computer Engineering
ECS Group

Contents

- Structural Modeling
  - Instantiation of Components
- Behavioral Modeling
  - Processes
  - Concurrent Signal Assignments
- Mixed Modeling

Ways of Modeling a Circuit

```
architecture xyz of abc is
begin

?

end architecture xyz;
```

Ways of Modeling a Circuit

```
architecture xyz of abc is
begin
```

Behavioral

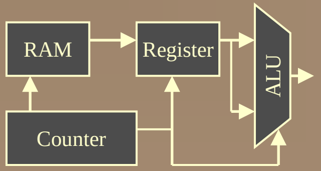
```
if A = `1` then
   B <= B+1
else
   B <= B
end if
```

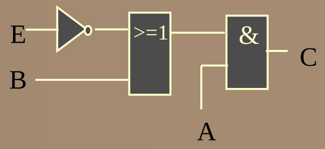
```
D <= NOT E

F <= D OR B

C <= F AND A
```

Structural

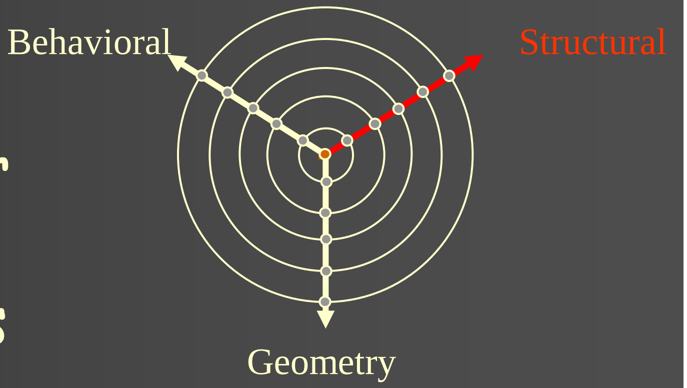




```
end architecture xyz;
```

Structural Modeling

- Instantiation of components
- Either graphically (Schematic) or textually (VHDL, Verilog, …)
- Connections between components created by wires (graphically) or port maps



<u>Attention: Possibility of getting dependent on a certain technology or vendor specific libraries!</u>

Instantiation of Components

- Component Declaration

  Describes the component's interface
  (best to be done within a package, but also possible in the declaration part of an architecture or block)

- Component Instantiation

  Assignment of signals to the component interface

- Component Configuration

  Functionality selection for each component

Declaration of a Component

Architecture

```
architecture struct of abc is
  component one
    generic
    (
      DEPTH : integer := 8
    );
    port
    (
      a, b : in  bit;
      c    : out bit
    );
  end component one;
begin
    ......
end architecture struct;
```

Package

```
package my_pkg is
  component one
    generic
    (
      DEPTH : integer := 8
    );
    port
    (
      a, b : in  bit;
      c    : out bit
    );
  end component one;

  ......
end package my_pkg;
```

Instantiation of a Component (1)

- Three important terms:

formals	Ports and generics of the instantiated entity
locals	Ports and generics of the component declaration
actuals	Signals (parameters) within the architecture

- When instantiated: *Locals* get connected to *actuals* (⇒ port map, generic map)

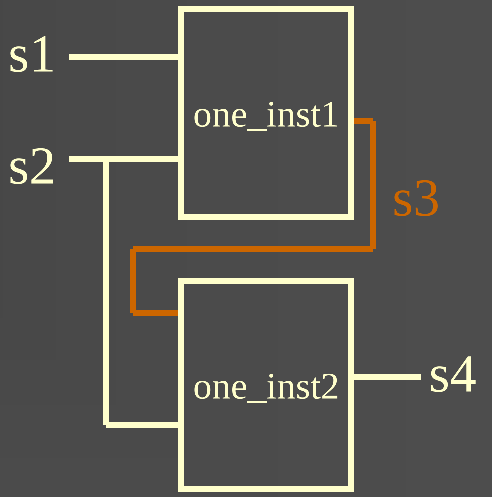
Instantiation of a Component (2)

```
use work.my_pkg.all;

architecture struct of xyz is
  signal s1, s2, s3, s4: bit;
begin
  --named association
  one_inst1 : one
    generic map(DEPTH => 10)
    port map(a => s1, b => s2,
                 c => s3);
  --positional association
  one_inst2 : one
    generic map(10)
    port map(s3, s2, s4);
end architecture struct;
```

Default value is replaced by 10

2 instances of one



Conditional Instantiation

- if-generate statement

```
.....

  if boolean expression generate
    one_inst1 : one
      generic map(DEPTH => 10)
      port map(a => s1, b => s2,
                          c => s3);
  end generate;

.....
```

Parallel Instantiation (1)

- for-generate statement

```
......

  -- s1, s2 and s3 are declared as:
  -- bit_vector(0 to 4)
  for i in 0 to 4 generate
    one_inst1 : one
      generic map(DEPTH => 10)
      port map(a => s1(i), b => s2(i),
                    c => s3(i));
  end generate;

......
```

Parallel Instantiation (2)

```
......
  -- s1, s2 and s3 are declared as:
  -- bit_vector(0 to 4)
  for i in 0 to 4 generate
    signal tmp : bit;
  begin
    one_inst1 : one
      generic map(DEPTH => 10)
      port map(a => s1(i), b => s2(i),
                   c => tmp);

    one_inst2 : one
      generic map(DEPTH => 10)
      port map(a => tmp, b => s2(i),
                   c => s3(i));
  end generate;
......
```

Component Configuration

- Implicit configuration, if only one architecture for the component exists
- Explicit configuration done in the design unit's configuration:

```
configuration xyz_cfg of xyz is
  for struct
    for one_inst1 : one use work.one(struct);
    for one_inst2 : one use work.one(beh);
  end for;
end configuration xyz_cfg;
```

- or directly at instantiation:

```
one_inst1 : entity work.one(struct)
  generic map ...
```

Component Configuration

- Implicit configuration, if only one architecture for the component exists
- Explicit configuration done in the design unit's configuration:

```
configuration xyz_cfg of xyz is
  for struct
    for one_inst1 : one use work.one(struct);
    for one_inst2 : one use work.one(beh);
  end for;
end configuration xyz_cfg;
```

- ~~or directly at instantiation:~~

```
one_inst1 : entity work.one(struct)
  generic map ...
```

Structural Modeling - Summary

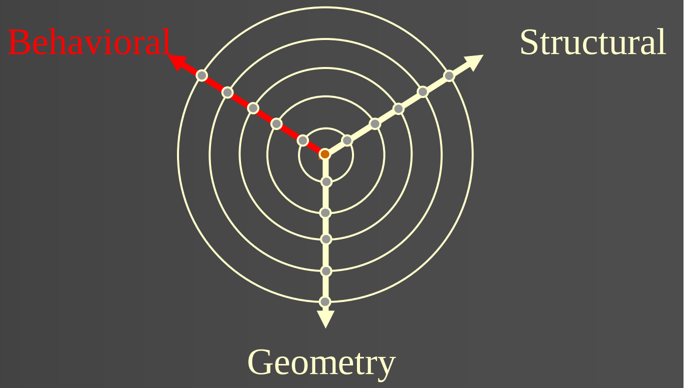
- Three steps
  - Component declaration
  - Component instantiation
  - Component configuration
- Connections between components are modeled using port mapping
- Parameters are set through generic mapping
- Possibility for conditional and parallel instantiation

Contents

- Structural Modeling
  - Instantiation of Components
- Behavioral Modeling
  - Processes
  - Concurrent Signal Assignments
- Mixed Modeling

Behavioral Modeling

- How to model the behavior of hardware?
  - Hardware may work parallel or sequential
  - Input changes may trigger output changes at arbitrary times
  - How to model time and delay?



Behavioral Architecture

```
architecture beh of abc is
begin
```

Time & delay

if ... then

?

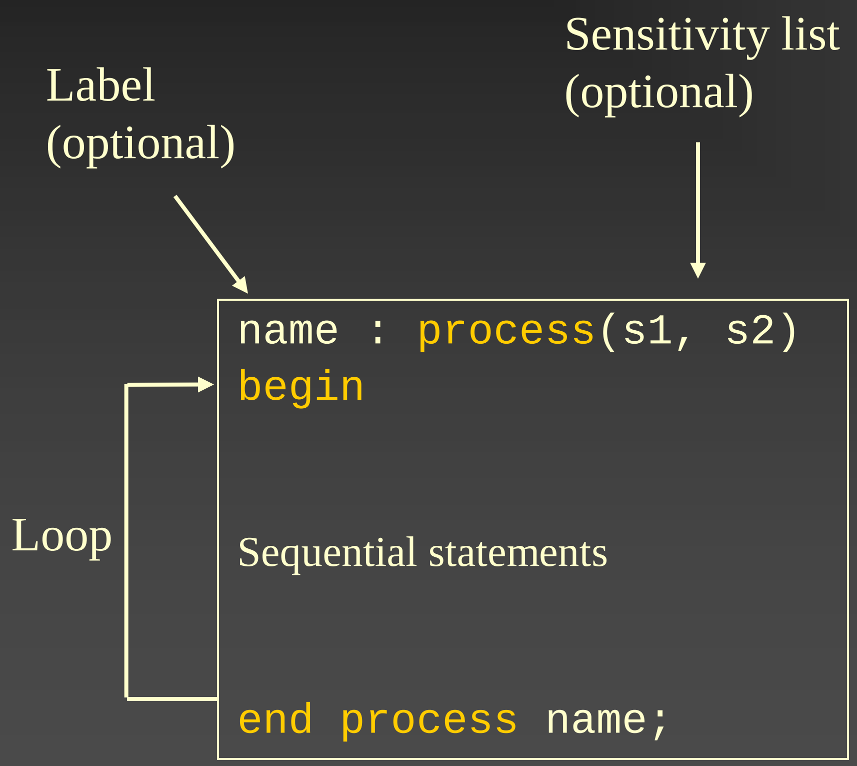
for ... loop

Spikes and races

case ... when

```
end architecture beh;
```

Processes (1)



```
name : process(s1, s2)
begin

end process name;
```

- Statements within a process are executed sequentially
- Execution „comparable“ to an infinite loop
- Execution controlled by the sensitivity list or wait statements
- A single architecture may have multiple, concurrent processes

Processes (2)

- Processes interchange information using signals.
- Assignments on signals are not done immediately, but at the next wait statement

  ⇒ Required for modeling parallel executions
- Signal assignments may trigger additional process executions

Wait Statements

- A single process may have multiple wait statements (simulation)
- or must have exactly a single wait statement (synthesis)

```
architecture beh of abc is
   signal s1, s2, s3 : bit;
begin
  p1: process
  begin
    s2 <= s1;
    s3 <= s2;
    wait on s1, s2;
  end process;
end architecture beh;
```

Simulation of Processes

- Execute all statements until a wait statement or the end of the process is reached
- If the end is reached, continue at the start of the process
- If a wait statement is reached suspend the execution of the process
  - If there is an active process
    - Resume that process
  - Otherwise
    - Increment simulation time

Sensitivity List

Special form of the wait on statement (at the end of the process)!

```
architecture beh of abc is
   signal s1, s2, s3 : bit;
begin
  p1: process (s1, s2)
  begin
    s2 <= s1;
    s3 <= s2;
    ~~wait on s1, s2;~~
  end process;
end architecture beh;
```

Sensitivity List

- Wait on at the end of a process is very common for synthesizable VHDL code!
- Therefore when writing synthesizable VHDL code, only sensitivity lists and no wait statements are normally used.
- For simulation of synthesizable VHDL code, the sensitivity list must contain all signals which are read by the process.
- Synthesis tools ignore the sensitivity list (only warning, if not complete).

Example Execution of a Process

```
architecture beh of abc is
   signal s1, s2, s3 : bit;
begin
   p1 : process(s1)
   begin
      s2 <= s1;
      s3 <= s2;
   end process;
end architecture beh;
```

Assumptions:
Before starting the simulation all signals have the value 0.

- `p1 : process(s1)` → s1 changes from 0 to 1 → Process is started
- `s2 <= s1;` → s2 is marked to become 1 (s1's value)
- `s3 <= s2;` → s3 is marked to become 0 (s2's old value!)
- `end process;` → Implicit wait on statement (sensitivity list)

The new values are assigned to s2 and s3
=> s2 becomes 1, s3 stays 0!

⇒ s2 and s3 have different values after the execution of the process!

W H Y ? ? ?

Example Execution of a Process

```
architecture beh of abc is
  signal s1, s2, s3 : bit;
begin
  p1 : process(s1, s2)
  begin
    s2 <= s1;
    s3 <= s2;
  end process;
end architecture beh;
```

Assumptions:
Before starting the simulation all signals have the value 0.

s1 changes from 0 to 1 → Process is started

s2 is marked to become 1 (s1's value)

s3 is marked to become 0 (s2's old value!)

Implicit wait on statement (sensitivity list)
The new values are assigned to s2 and s3
=> s2 becomes 1, s3 stays 0!

⇒ s2 and s3 have different values after the execution of the process!

Incomplete sensitivity list! s2 is missing

Architecture with 2 Processes

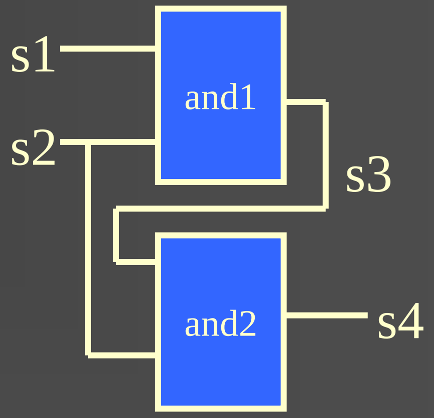
```
architecture beh of abc is
  signal s1, s2, s3, s4: bit;
begin
  and1 : process(s1, s2)
    if s1 = `1` and s2 = `1` then
      s3 <= `1`;
    else
      s3 <= `0` ;
    end if;
  end process and1;

  and2 : process(s2, s3)
    if s2 = `1` and s3 = `1` then
      s4 <= `1`;
    else
      s4 <= `0`;
    end if;
  end process and2;
end architecture beh;
```

sequential

sequential

Parallel



Manual Simulation

```
architecture beh of abc is
  signal s1, s2, s3, s4: bit;
begin
  and1 : process(s1, s2)
    if s1 = `1` and s2 = `1` then
      s3 <= `1`;
    else
      s3 <= `0` ;
    end if;
  end process and1;

  and2 : process(s2, s3)
    if s2 = `1` and s3 = `1` then
      s4 <= `1`;
    else
      s4 <= `0`;
    end if;
  end process and2;
end architecture beh;
```

Assumptions:

s1 = 0
s2 = 1
⇒ s3 = 0
⇒ s4 = 0

Manual Simulation

```
architecture beh of abc is
  signal s1, s2, s3, s4: bit;
begin
  and1 : process(s1, s2)
    if s1 = `1` and s2 = `1` then
      s3 <= `1`;
    else
      s3 <= `0` ;
    end if;
  end process and1;

  and2 : process(s2, s3)
    if s2 = `1` and s3 = `1` then
      s4 <= `1`;
    else
      s4 <= `0`;
    end if;
  end process and2;
end architecture beh;
```

Assumptions:

$s1 = 0$
$s2 = 1$
$\Rightarrow s3 = 0$
$\Rightarrow s4 = 0$

$s1: 0 \rightarrow 1$

Manual Simulation

```
architecture beh of abc is
  signal s1, s2, s3, s4: bit;
begin
  and1 : process(s1, s2)
    if s1 = `1` and s2 = `1` then
      s3 <= `1`;
    else
      s3 <= `0` ;
    end if;
  end process and1;

  and2 : process(s2, s3)
    if s2 = `1` and s3 = `1` then
      s4 <= `1`;
    else
      s4 <= `0`;
    end if;
  end process and2;
end architecture beh;
```

Assumptions:

$s1 = 0$

$s2 = 1$

$\Rightarrow s3 = 0$

$\Rightarrow s4 = 0$

$s1: 0 \rightarrow 1$

$\Rightarrow s3: 0 \rightarrow 1$

Manual Simulation

```
architecture beh of abc is
  signal s1, s2, s3, s4: bit;
begin
  and1 : process(s1, s2)
    if s1 = `1` and s2 = `1` then
      s3 <= `1`;
    else
      s3 <= `0` ;
    end if;
  end process and1;

  and2 : process(s2, s3)
    if s2 = `1` and s3 = `1` then
      s4 <= `1`;
    else
      s4 <= `0`;
    end if;
  end process and2;
end architecture beh;
```

Assumptions:
$s1 = 0$
$s2 = 1$
$\Rightarrow s3 = 0$
$\Rightarrow s4 = 0$

$s1: 0 \rightarrow 1$

$\Rightarrow s3: 0 \rightarrow 1$

$\Rightarrow s4: 0 \rightarrow 1$

Another Example

```
entity abc is
  port
  (
    A, B, C : in  bit;
    Z, R    : out bit
  );
end entity abc;
```

```
architecture beh of abc is
  signal X, Y : bit;
begin
  process(A, B, C)
  begin
     X <= A;
     Y <= B;
     Z <= X and Y;
     Y <= C;
     R <= X and Y;
  end process;
end architecture beh;
```

Which values are assigned to Z and R?

Another Example

```
entity abc is
  port
  (
    A, B, C : in  bit;
    Z, R    : out bit
  );
end entity abc;
architecture beh of abc is
  signal X, Y : bit;
begin
  process(A, B, C)
  begin
     X <= A;
     Y <= B;
     Z <= X and Y;
     Y <= C;
     R <= X and Y;
  end process;
end architecture beh;
```

Which values are assigned to Z and R?

After a single process execution
Z and R are set to X_old and Y_old.

After completing the sensitivity list (adding X and Y), a second execution of the process is triggered.

R and Z are set to A and C after the second iteration.

Attention:
Z is NEVER set to A and B

Simple Signal Assignment Example

- Simple signal assignment is much too complicated:

```
assign : process(s1, s2)
   s3 <= s1 and s2;
end process assign;
```

- Is there an easier way?
- Yes:
  - Short form for signal assignment (named concurrent signal assignment):

```
s3 <= s1 and s2;
```

Concurrent Signal Assignments

- Outside of a process
- Possibility for specifying a delay
- Possibility for conditional assignments
- Possibility for selective assignments
- Parallel execution (short form of a process declaration)

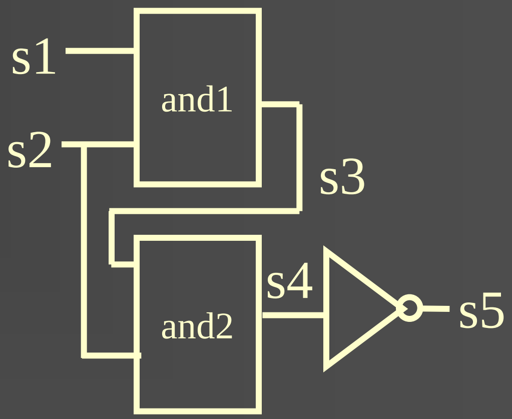
Concurrent Signal Assignments

Unconditional assignment

Conditional Assignment

Selective Assignment

```
architecture beh of abc is
  signal s1, s2, s3, s4, s5: bit;
begin
  and1 : s3 <= s1 and s2;
  and2 : s4 <= `1` when s3=`1` and s2 =`1`
               else `0`;
  inverter : with s4 select
               output <= `1` when `0`,
                         `0` when `1`;
end architecture beh;
```



Signal Delay

- Delays are modeled as follows:

```
s3  <= s1 and s2 after 1 ns;
```

⇒ Applicable only in Behavioral Simulation!

- Not synthesizable, ignored by the synthesis tool.
- After technology mapping, real hardware delays added to simulation netlist.

Behavioral Modeling - Summary

- Based on processes
- Multiple processes per architecture
- Multiple processes executed concurrently
- Processes executed sequentially
- Controlled by wait statements (simulation)
- Simplifications:
  - Sensitivity list instead of wait on
  - Concurrent signal assignments

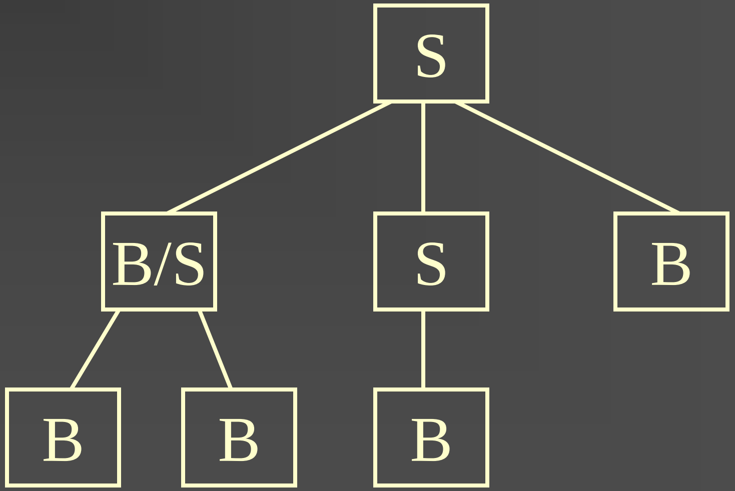
Contents

- Structural Modeling
  - Instantiation of Components
- Behavioral Modeling
  - Processes
  - Concurrent Signal Assignments
- Mixed Modeling

Mixed Modeling (1)

- Behavioral- and structural Descriptions may be mixed in the same architecture
- Descriptions on different layers of abstraction may be mixed

S ... Structural Description
B ... Behavioral Description
S/B .... Mixed Behavioral/ Structural Description

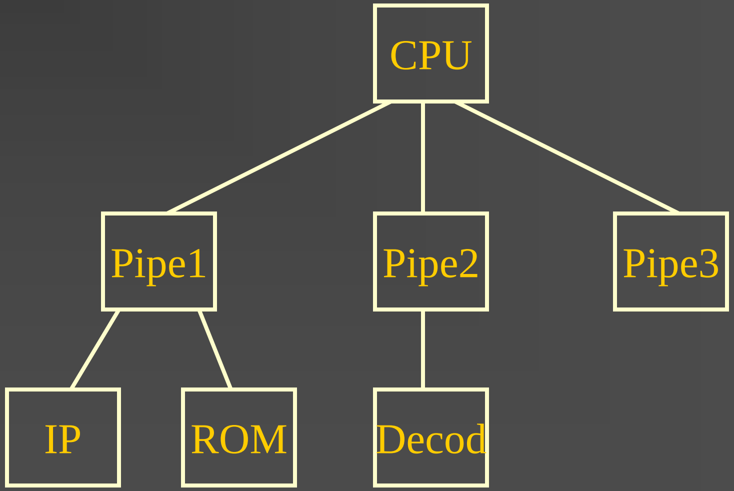


Mixed Modeling (1)

- Behavioral- and structural Descriptions may be mixed in the same architecture
- Descriptions on different layers of abstraction may be mixed

Example of a CPU

Break down into subsystems
⇒ Integration of subsystems as components



Mixed Modell (2)

```
architecture beh of abc is
  signal s1, s2, s3, s4, s5 : bit;
begin
```

Structural

```
  and1 : one
    generic map(DEPTH => 10)
    port map(a => s1, b => s2,
                 c => s3);
```

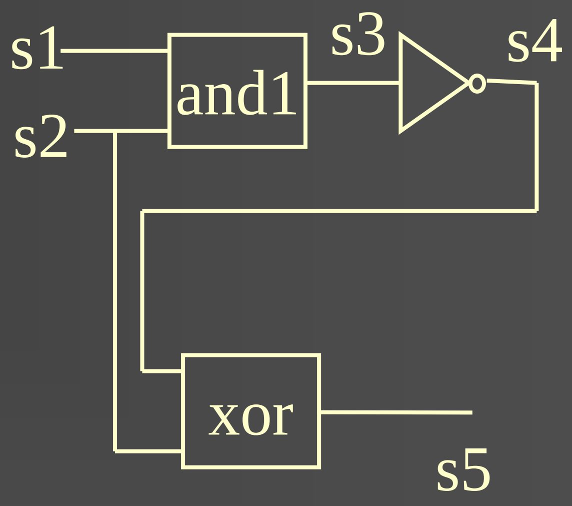
Behavioral
RTL level

```
  process(s3)
  begin
    if s3=`1`  then
      s4 <= `0`;
    else
      s4 <= `1` ;
   end if;
 end process;
```

Behavioral
Logic level

```
 s5  <= s1 xor s4;
```

```
end architecture beh;
```



Summary

```
architecture xyz of abc is
begin
```

Behavioral

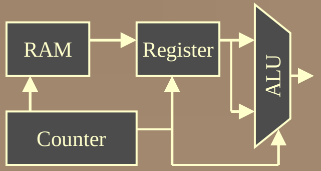
Structural

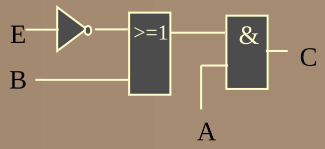
```
if A = `1` then
   B <= B+1
else
   B <= B
end if
```

```
D <= NOT E

F <= D OR B

C <= F AND A
```





```
end architecture xyz;
```

Summary

```
architecture xyz of abc is
begin
```

Behavioral	Structural
Processes • Processes are running in parallel • Statements of a process are executed sequentially •Controlled by wait statements **Simplifications** • Concurrent signal assignments •Unconditional, conditional, selective •Outside of Processes •Sensitivity lists	**Components** • Declaration •Package or architecture • Instantiation •Port mapping, generic mapping • Configuration •Configuration file

```
end architecture xyz;
```