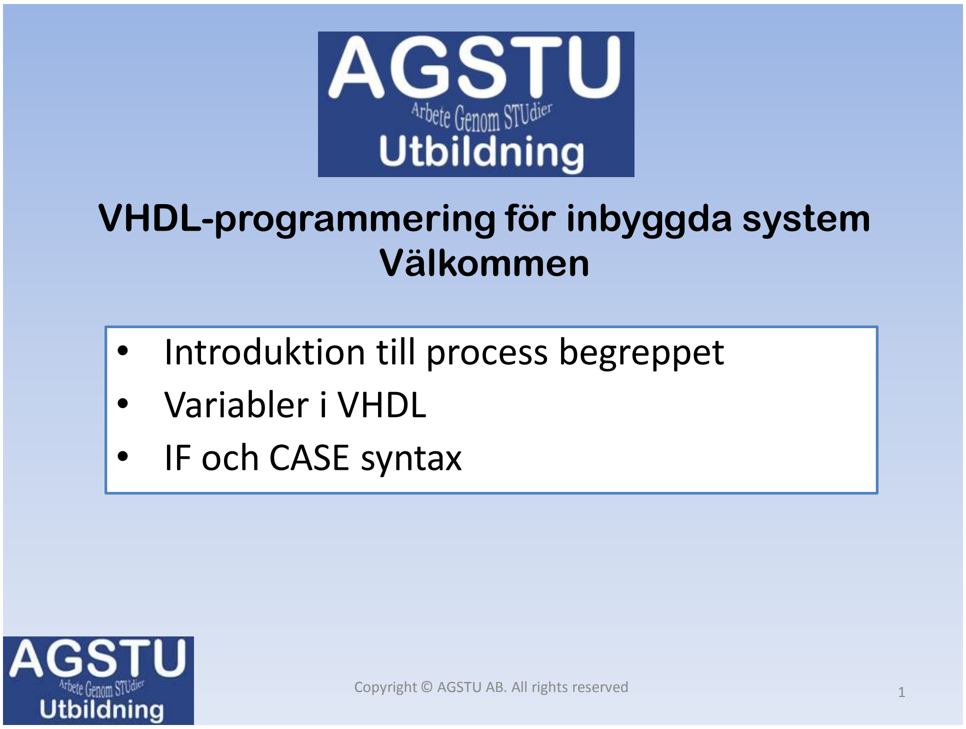
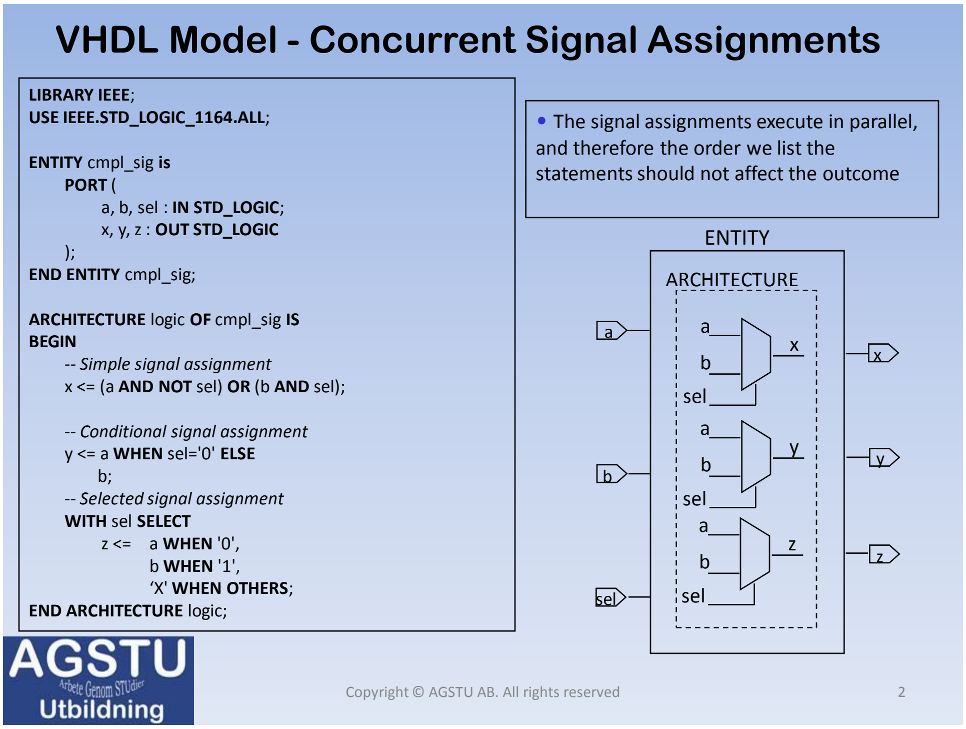


AGSTU Arbete Genom STUdier Utbildning

VHDL-programmering för inbyggda system Välkommen

- Introduktion till process begreppet
- Variabler i VHDL
- IF och CASE syntax



VHDL Model - Concurrent Signal Assignments

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY cmpl_sig is
      PORT (
            a, b, sel : IN STD_LOGIC;
            x, y, z : OUT STD_LOGIC
      );
END ENTITY cmpl_sig;

ARCHITECTURE logic OF cmpl_sig IS
BEGIN
      -- Simple signal assignment
      x <= (a AND NOT sel) OR (b AND sel);

      -- Conditional signal assignment
      y <= a WHEN sel='0' ELSE
            b;
      -- Selected signal assignment
      WITH sel SELECT
            z <=   a WHEN '0',
                   b WHEN '1',
                   'X' WHEN OTHERS;
END ARCHITECTURE logic;
```

- The signal assignments execute in parallel, and therefore the order we list the statements should not affect the outcome

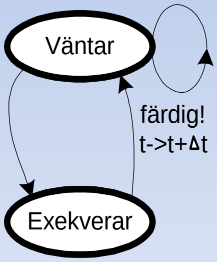
Why use Process

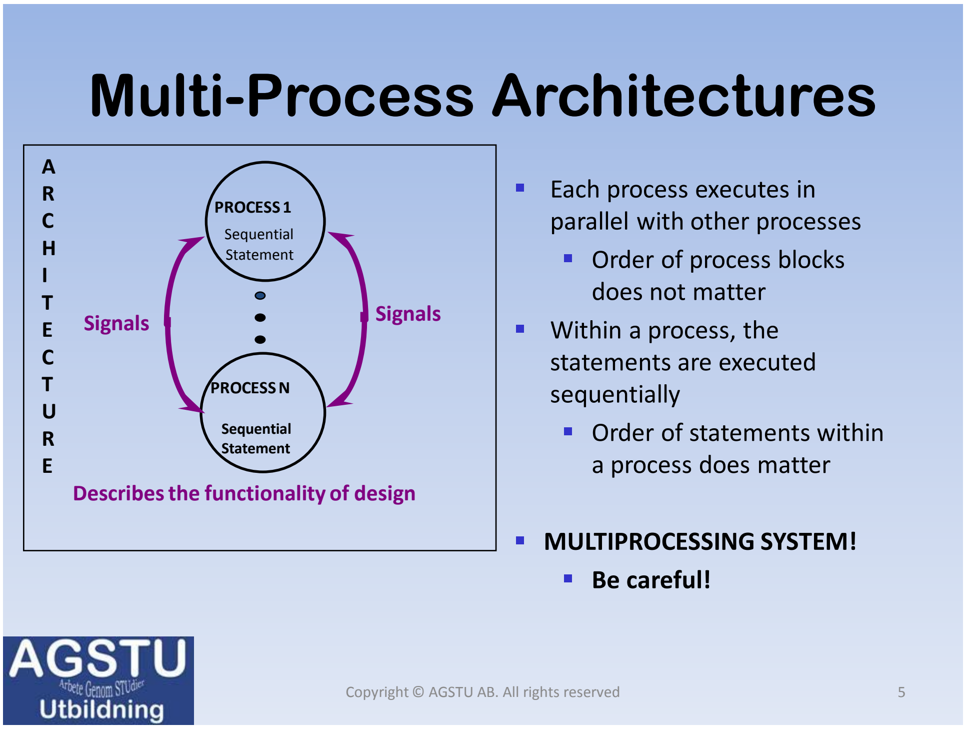
- More design space
  - Sequential execution
  - State machine design
  - Structural design
  - Synchronous design
  - …..

PROCESS Statement

- An architecture can have multiple process statements
- Declaration section allows declaration of local objects and names
- Process contents consist of sequential statements

```
label : PROCESS (sensitivity_list)
     Constant declarations
        Type declarations
        Variable declarations
BEGIN
        Sequential statement #1;
              …
        Sequential statement #n ;
END PROCESS label;
```





Multi-Process Architectures

A R C H I T E C T U R E

PROCESS 1

Sequential Statement

Signals

Signals

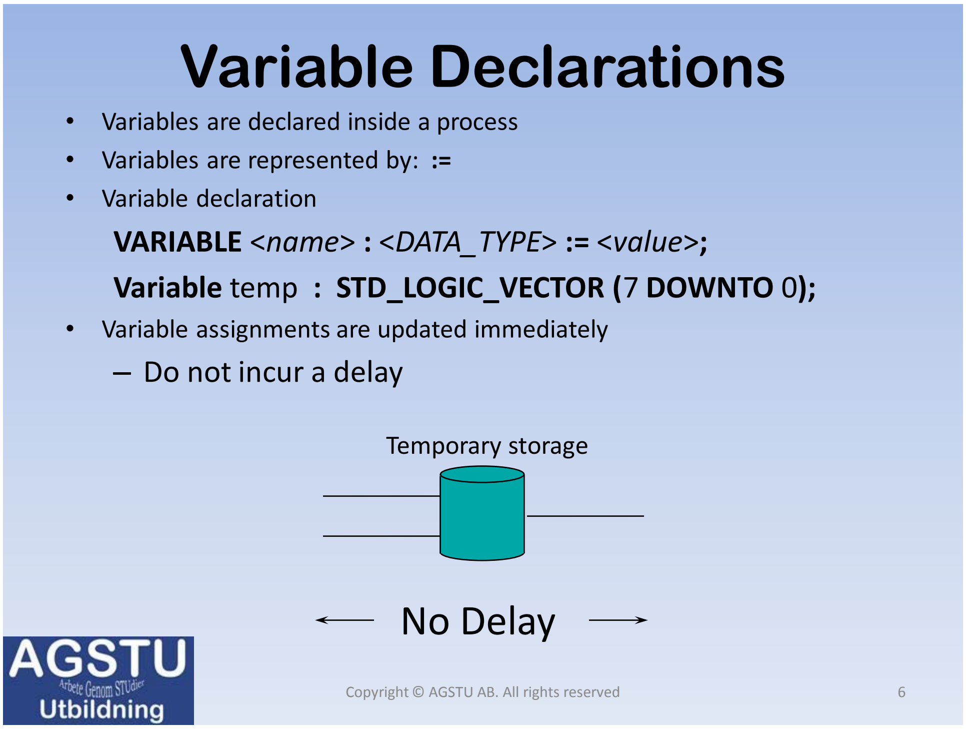
PROCESS N

Sequential Statement

Describes the functionality of design

- Each process executes in parallel with other processes
  - Order of process blocks does not matter
- Within a process, the statements are executed sequentially
  - Order of statements within a process does matter
- **MULTIPROCESSING SYSTEM!**
  - **Be careful!**

AGSTU Arbete Genom STUdier Utbildning



Variable Declarations

- Variables are declared inside a process
- Variables are represented by: **:=**
- Variable declaration

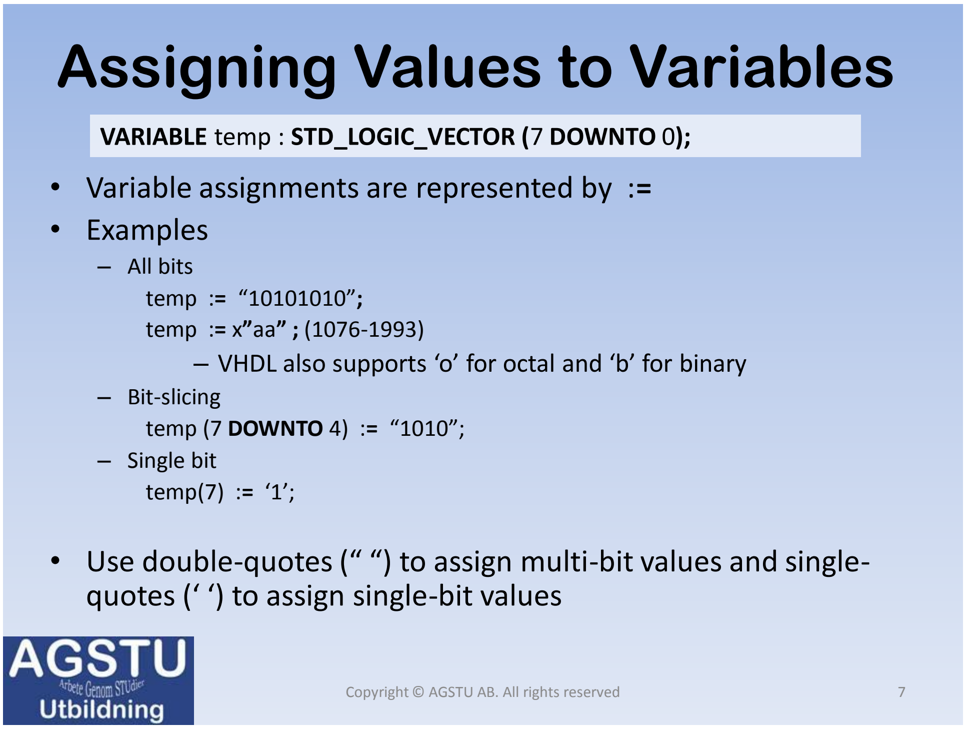
  VARIABLE <*name*> **:** <*DATA_TYPE*> **:=** <*value*>**;**

  Variable temp **: STD_LOGIC_VECTOR (**7 **DOWNTO** 0**);**
- Variable assignments are updated immediately
  – Do not incur a delay

Temporary storage

← No Delay →

AGSTU Arbete Genom STUdier Utbildning



Assigning Values to Variables

VARIABLE temp : **STD_LOGIC_VECTOR** (7 **DOWNTO** 0)**;**

- Variable assignments are represented by **:=**
- Examples
  - All bits

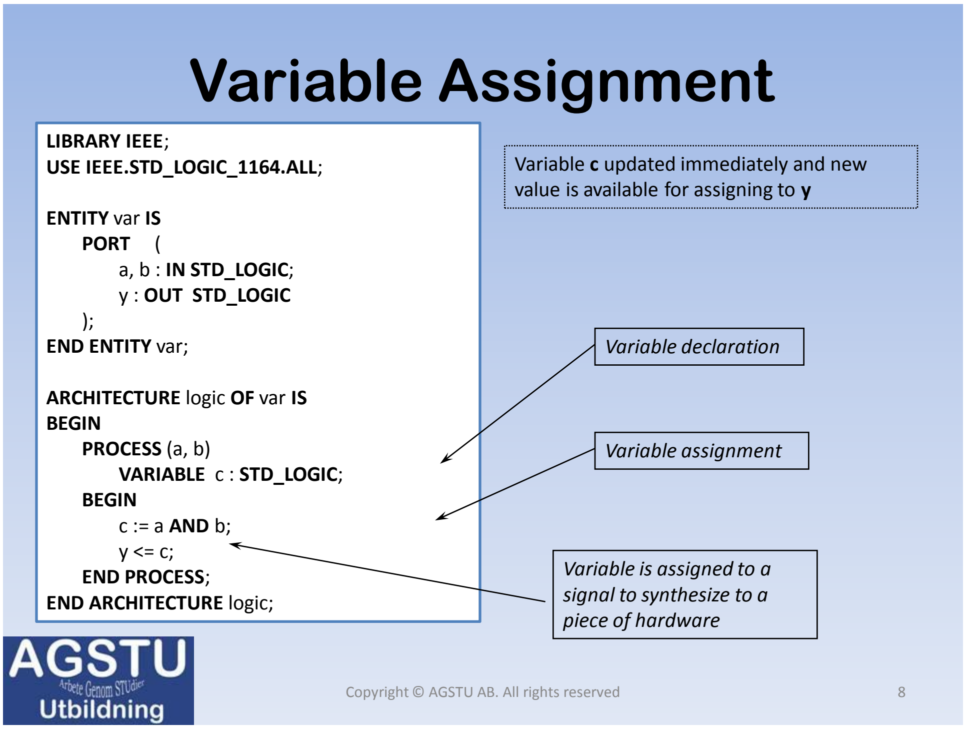
    temp **:=** “10101010”**;**

    temp **:=** x”aa” **;** (1076-1993)
    - VHDL also supports ‘o’ for octal and ‘b’ for binary
  - Bit-slicing

    temp (7 **DOWNTO** 4) **:=** “1010”;
  - Single bit

    temp(7) **:=** ‘1’;
- Use double-quotes (“ “) to assign multi-bit values and single-quotes (‘ ‘) to assign single-bit values

AGSTU
Arbete Genom STUdier
Utbildning



Variable Assignment

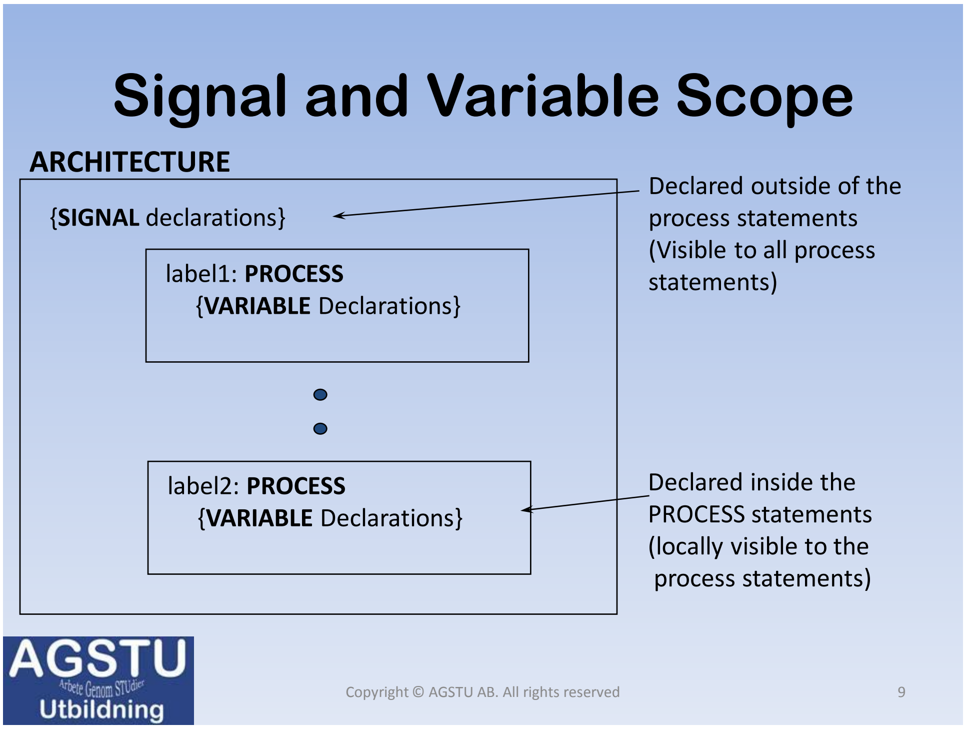
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY var IS
    PORT    (
        a, b : IN STD_LOGIC;
        y : OUT  STD_LOGIC
    );
END ENTITY var;

ARCHITECTURE logic OF var IS
BEGIN
    PROCESS (a, b)
        VARIABLE  c : STD_LOGIC;
    BEGIN
        c := a AND b;
        y <= c;
    END PROCESS;
END ARCHITECTURE logic;
```

Variable **c** updated immediately and new value is available for assigning to **y**

Variable declaration

Variable assignment

Variable is assigned to a signal to synthesize to a piece of hardware



Signal and Variable Scope

ARCHITECTURE

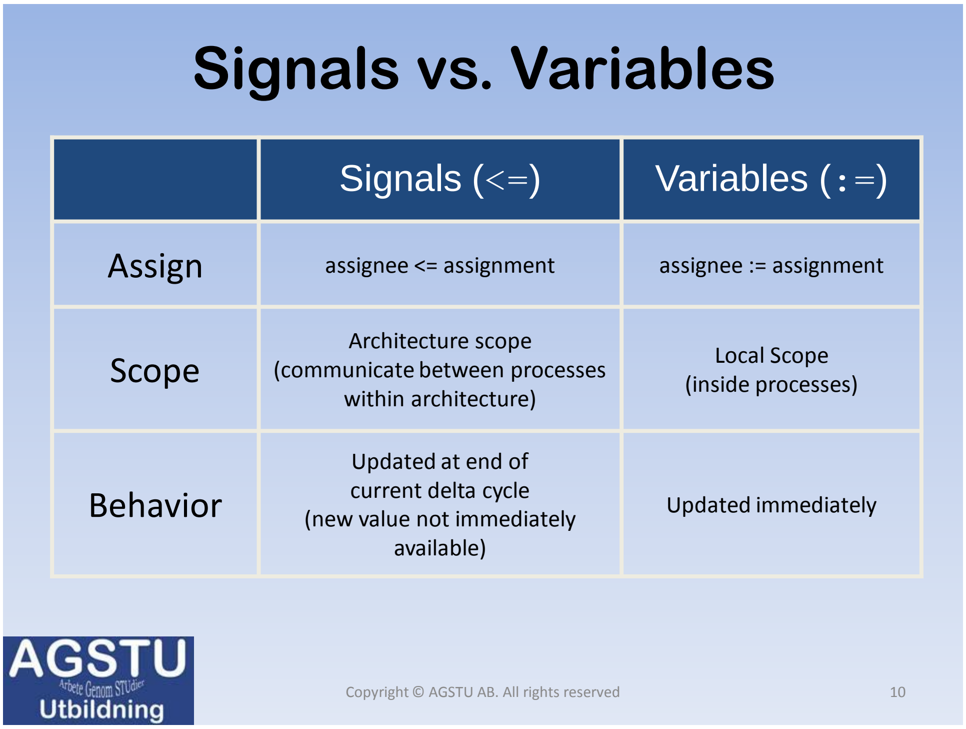
{**SIGNAL** declarations}

label1: **PROCESS**
{**VARIABLE** Declarations}

label2: **PROCESS**
{**VARIABLE** Declarations}

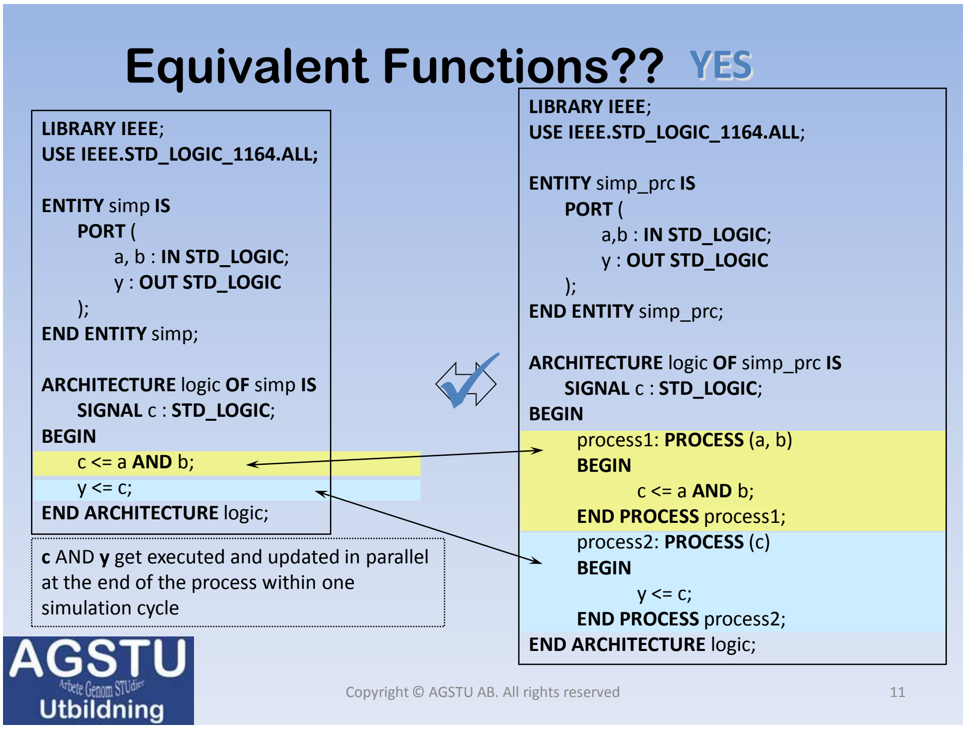
Declared outside of the process statements (Visible to all process statements)

Declared inside the PROCESS statements (locally visible to the process statements)



Signals vs. Variables

	Signals (<=)	Variables (:=)
Assign	assignee <= assignment	assignee := assignment
Scope	Architecture scope (communicate between processes within architecture)	Local Scope (inside processes)
Behavior	Updated at end of current delta cycle (new value not immediately available)	Updated immediately



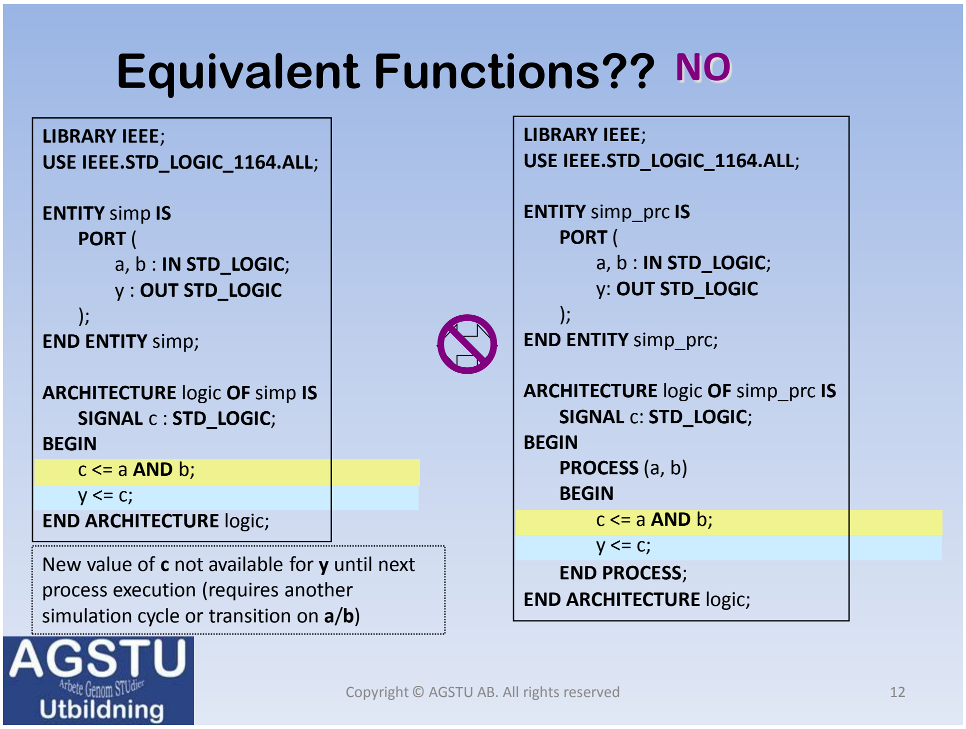
Equivalent Functions?? YES

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY simp IS
      PORT (
            a, b : IN STD_LOGIC;
            y : OUT STD_LOGIC
      );
END ENTITY simp;

ARCHITECTURE logic OF simp IS
      SIGNAL c : STD_LOGIC;
BEGIN
      c <= a AND b;
      y <= c;
END ARCHITECTURE logic;
```

c AND **y** get executed and updated in parallel at the end of the process within one simulation cycle

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY simp_prc IS
      PORT (
            a,b : IN STD_LOGIC;
            y : OUT STD_LOGIC
      );
END ENTITY simp_prc;

ARCHITECTURE logic OF simp_prc IS
      SIGNAL c : STD_LOGIC;
BEGIN
         process1: PROCESS (a, b)
         BEGIN
                  c <= a AND b;
         END PROCESS process1;
         process2: PROCESS (c)
         BEGIN
                  y <= c;
         END PROCESS process2;
END ARCHITECTURE logic;
```

11



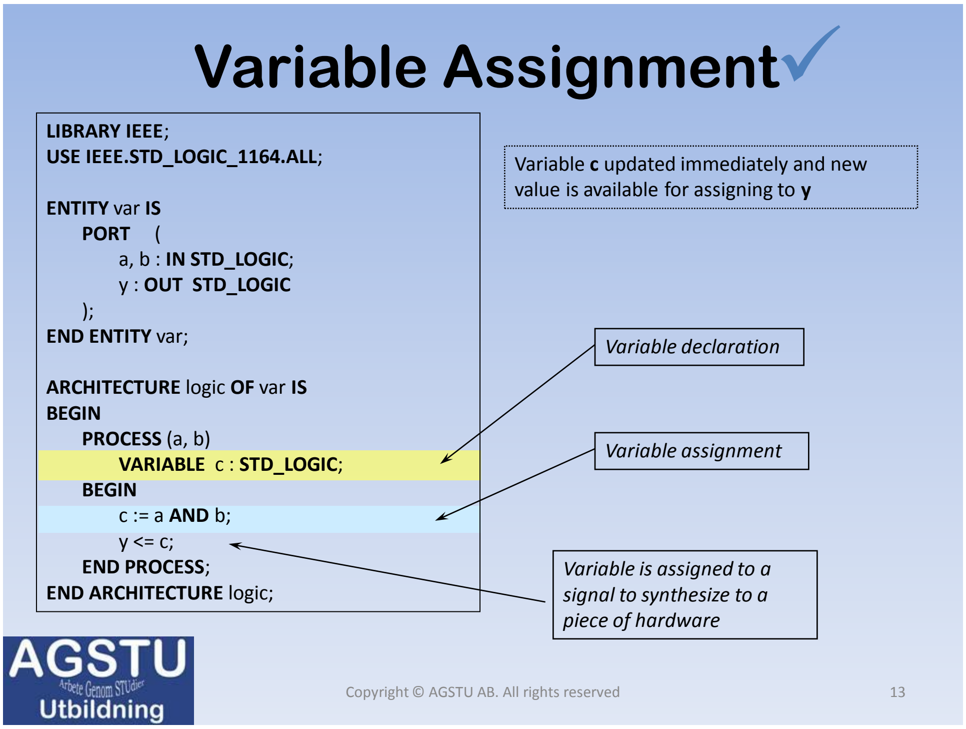
Equivalent Functions?? NO

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY simp IS
      PORT (
            a, b : IN STD_LOGIC;
            y : OUT STD_LOGIC
      );
END ENTITY simp;

ARCHITECTURE logic OF simp IS
      SIGNAL c : STD_LOGIC;
BEGIN
      c <= a AND b;
      y <= c;
END ARCHITECTURE logic;
```

New value of **c** not available for **y** until next process execution (requires another simulation cycle or transition on **a**/**b**)

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY simp_prc IS
      PORT (
            a, b : IN STD_LOGIC;
            y: OUT STD_LOGIC
      );
END ENTITY simp_prc;

ARCHITECTURE logic OF simp_prc IS
      SIGNAL c: STD_LOGIC;
BEGIN
      PROCESS (a, b)
      BEGIN
            c <= a AND b;
            y <= c;
      END PROCESS;
END ARCHITECTURE logic;
```

12



Variable Assignment✓

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY var IS
    PORT    (
        a, b : IN STD_LOGIC;
        y : OUT  STD_LOGIC
    );
END ENTITY var;

ARCHITECTURE logic OF var IS
BEGIN
    PROCESS (a, b)
        VARIABLE  c : STD_LOGIC;
    BEGIN
        c := a AND b;
        y <= c;
    END PROCESS;
END ARCHITECTURE logic;
```

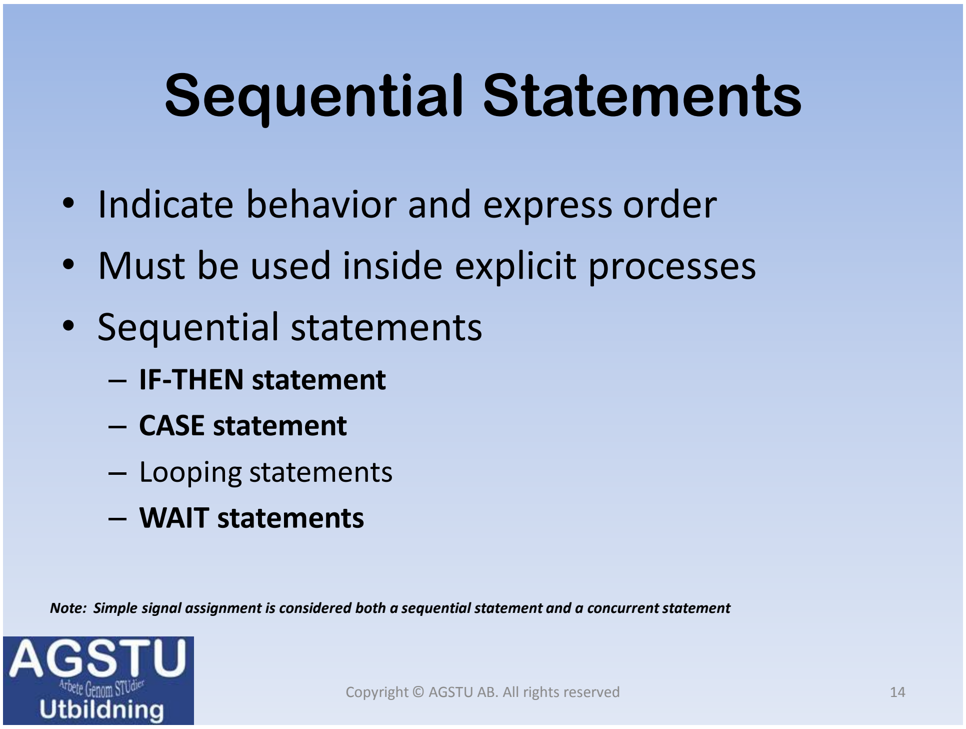
Variable **c** updated immediately and new value is available for assigning to **y**

Variable declaration

Variable assignment

Variable is assigned to a signal to synthesize to a piece of hardware

AGSTU Arbete Genom STUdier Utbildning

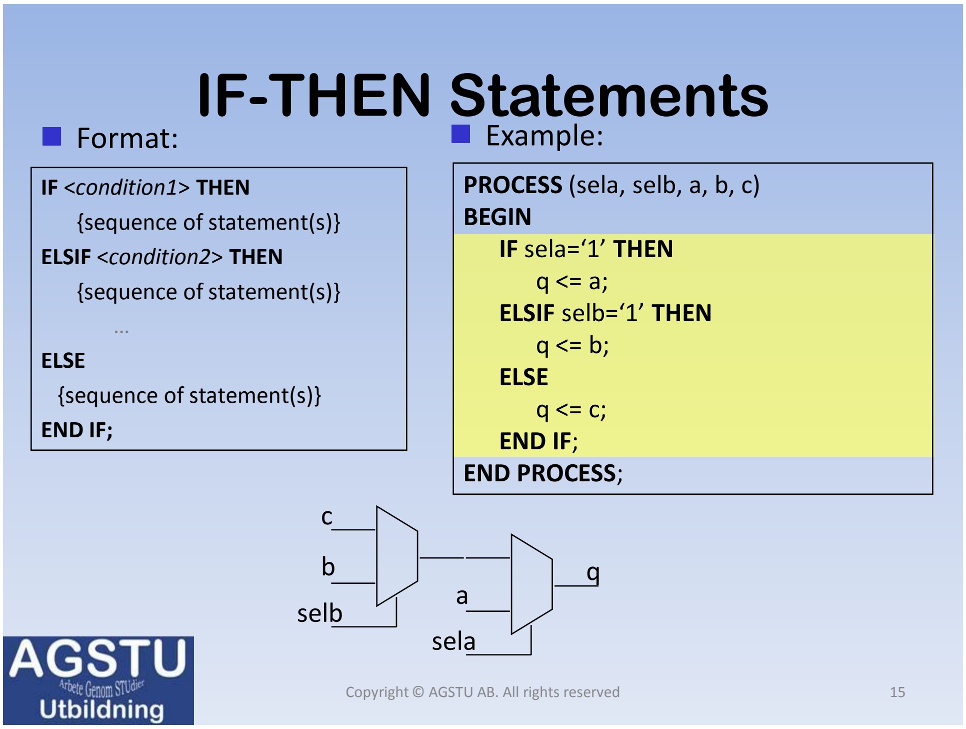


Sequential Statements

- Indicate behavior and express order
- Must be used inside explicit processes
- Sequential statements
  - **IF-THEN statement**
  - **CASE statement**
  - Looping statements
  - **WAIT statements**

Note: *Simple signal assignment is considered both a sequential statement and a concurrent statement*

AGSTU Arbete Genom STUdier Utbildning



IF-THEN Statements

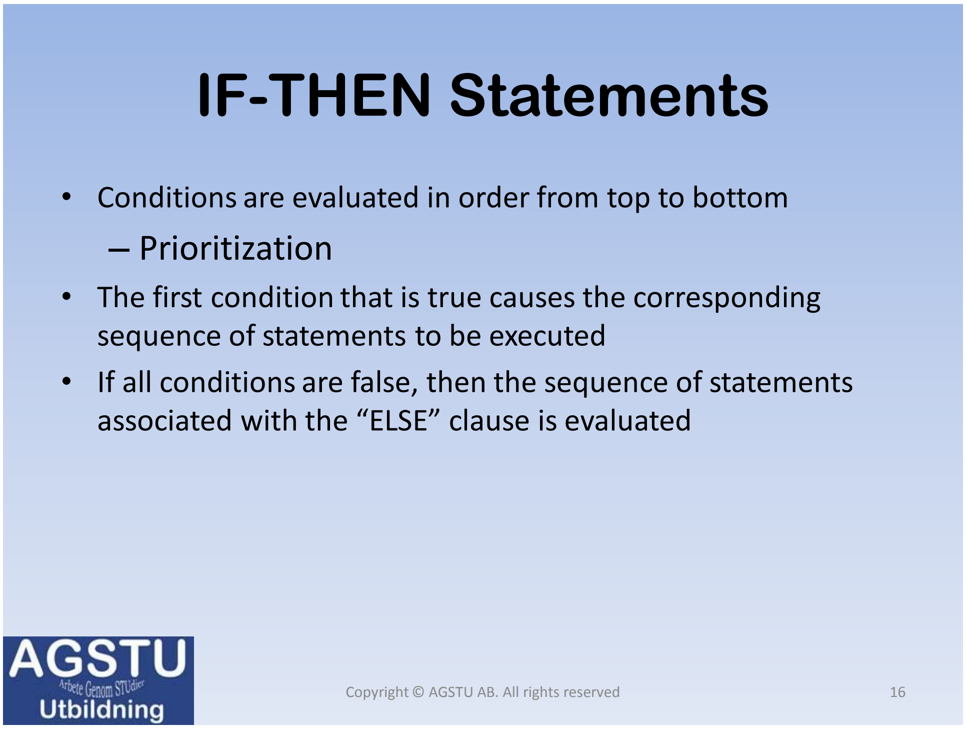
- Format:

```
IF <condition1> THEN
    {sequence of statement(s)}
ELSIF <condition2> THEN
    {sequence of statement(s)}
        …
ELSE
  {sequence of statement(s)}
END IF;
```

- Example:

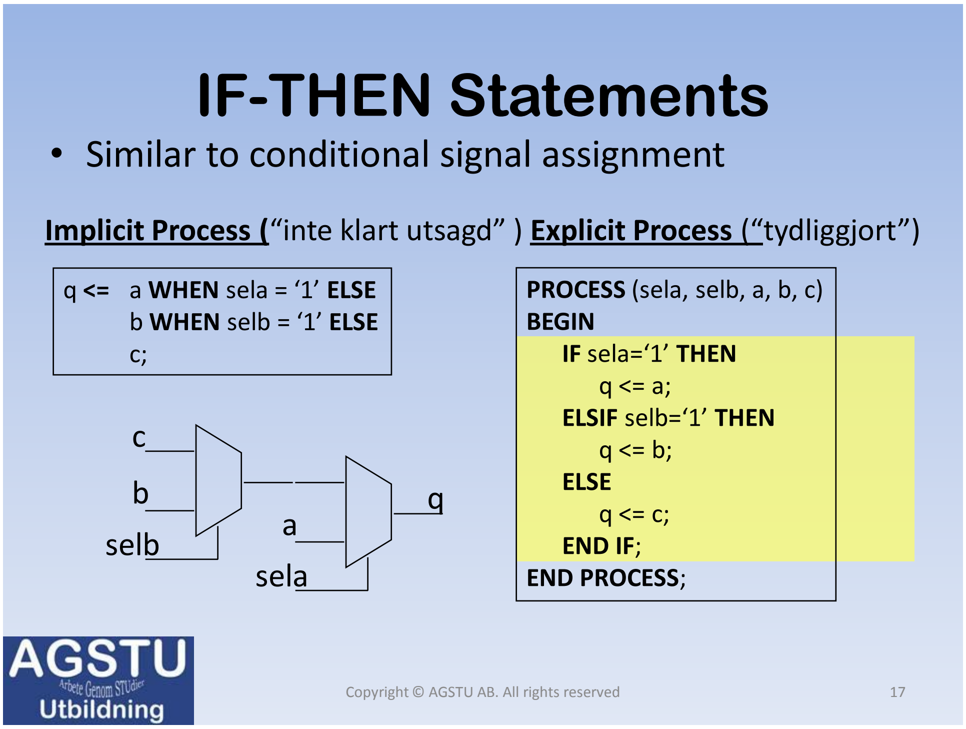
```
PROCESS (sela, selb, a, b, c)
BEGIN
    IF sela='1' THEN
        q <= a;
    ELSIF selb='1' THEN
        q <= b;
    ELSE
        q <= c;
    END IF;
END PROCESS;
```

c
b
selb
a
sela
q



IF-THEN Statements

- Conditions are evaluated in order from top to bottom
  - Prioritization
- The first condition that is true causes the corresponding sequence of statements to be executed
- If all conditions are false, then the sequence of statements associated with the "ELSE" clause is evaluated



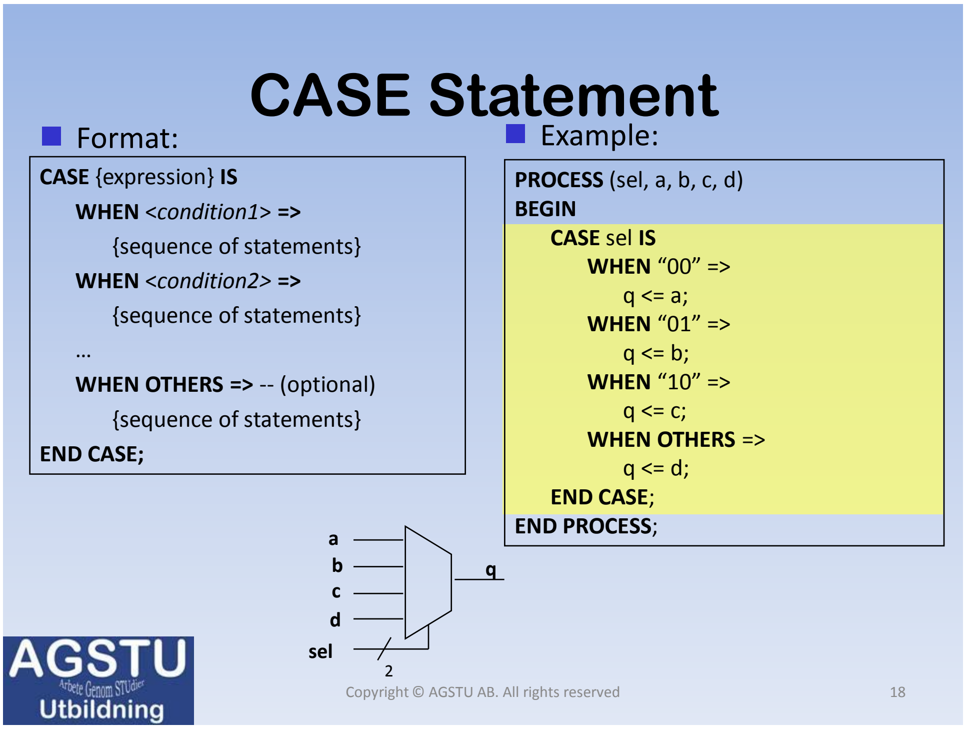
IF-THEN Statements

- Similar to conditional signal assignment

Implicit Process (“inte klart utsagd”) **Explicit Process (**“tydliggjort”)

```
q <=   a WHEN sela = ‘1’ ELSE
       b WHEN selb = ‘1’ ELSE
       c;
```

```
PROCESS (sela, selb, a, b, c)
BEGIN
    IF sela=‘1’ THEN
        q <= a;
    ELSIF selb=‘1’ THEN
        q <= b;
    ELSE
        q <= c;
    END IF;
END PROCESS;
```



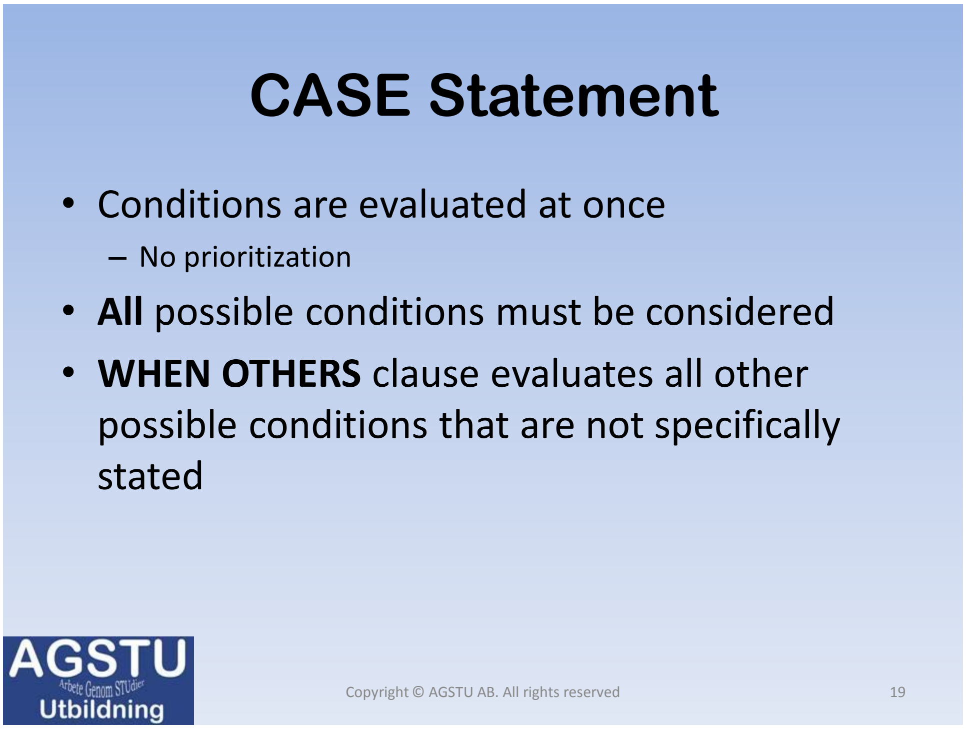
CASE Statement

- Format:

```
CASE {expression} IS
    WHEN <condition1> =>
        {sequence of statements}
    WHEN <condition2> =>
        {sequence of statements}
    …
    WHEN OTHERS => -- (optional)
        {sequence of statements}
END CASE;
```

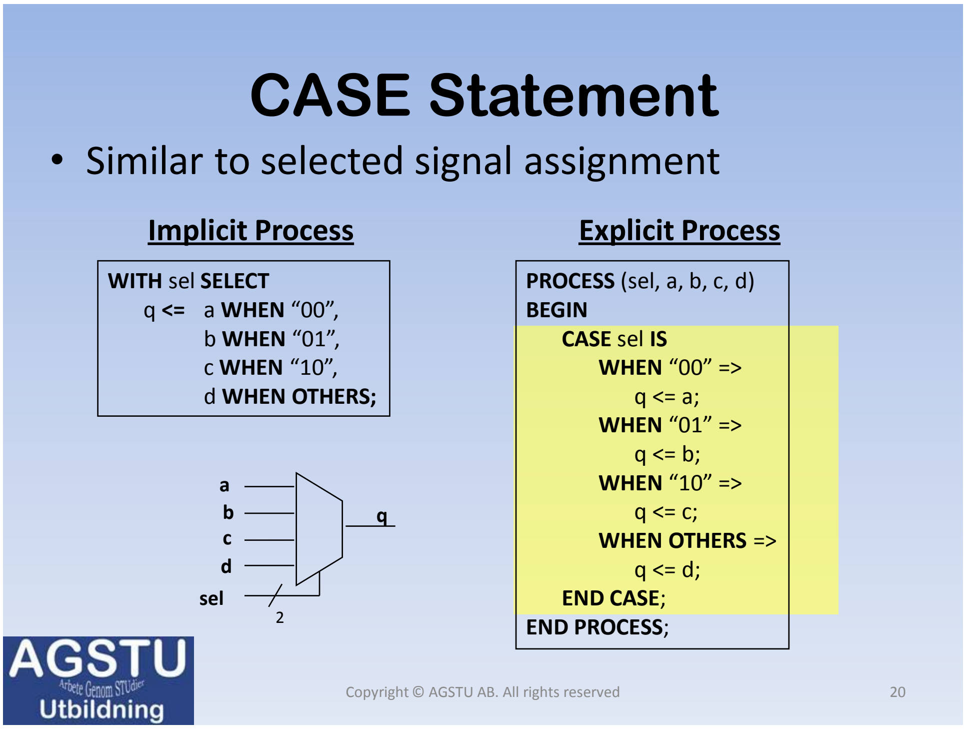
- Example:

```
PROCESS (sel, a, b, c, d)
BEGIN
    CASE sel IS
        WHEN “00” =>
            q <= a;
        WHEN “01” =>
            q <= b;
        WHEN “10” =>
            q <= c;
        WHEN OTHERS =>
            q <= d;
    END CASE;
END PROCESS;
```



CASE Statement

- Conditions are evaluated at once
  - No prioritization
- **All** possible conditions must be considered
- **WHEN OTHERS** clause evaluates all other possible conditions that are not specifically stated



CASE Statement

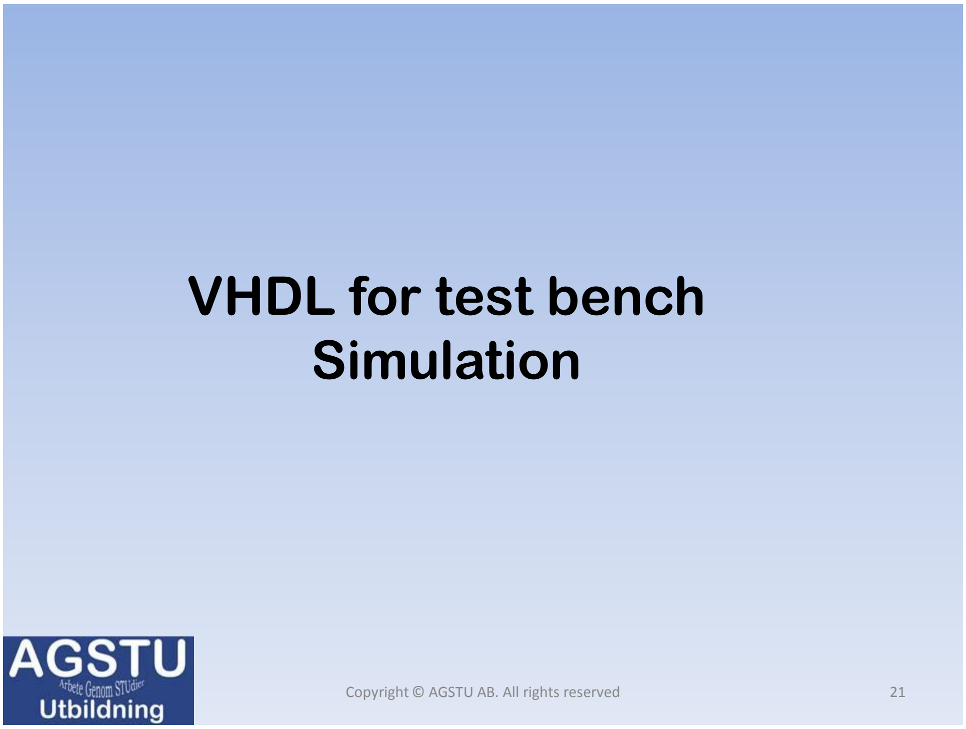
- Similar to selected signal assignment

Implicit Process

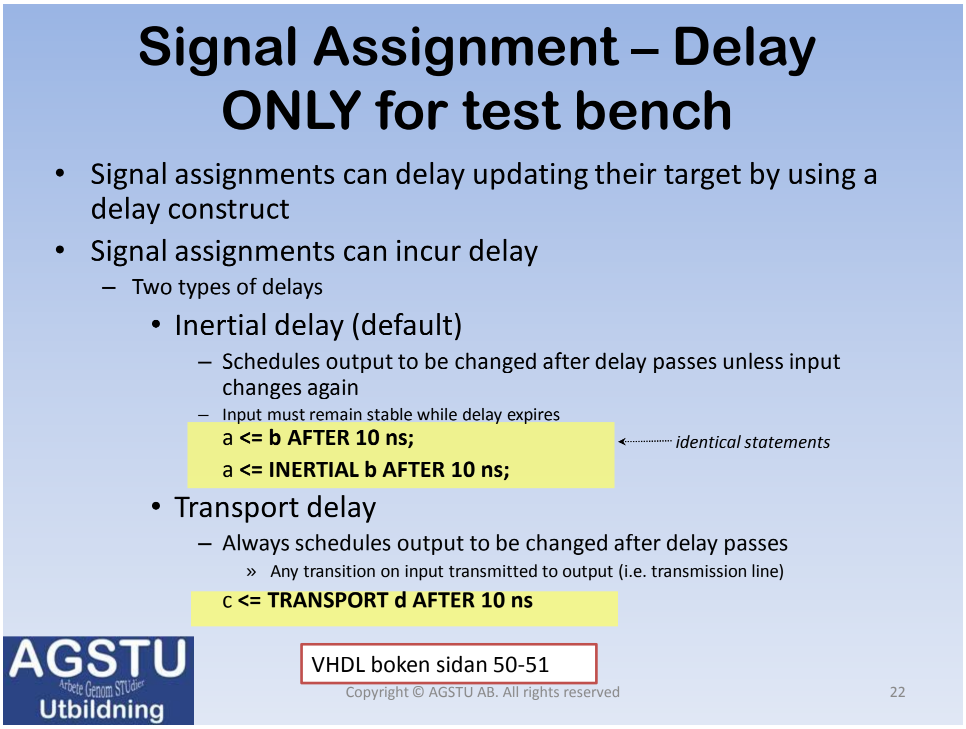
```
WITH sel SELECT
     q <=   a WHEN "00",
            b WHEN "01",
            c WHEN "10",
            d WHEN OTHERS;
```

Explicit Process

```
PROCESS (sel, a, b, c, d)
BEGIN
     CASE sel IS
          WHEN "00" =>
               q <= a;
          WHEN "01" =>
               q <= b;
          WHEN "10" =>
               q <= c;
          WHEN OTHERS =>
               q <= d;
     END CASE;
END PROCESS;
```



VHDL for test bench Simulation



Signal Assignment – Delay ONLY for test bench

- Signal assignments can delay updating their target by using a delay construct
- Signal assignments can incur delay
  - Two types of delays
    - Inertial delay (default)
      - Schedules output to be changed after delay passes unless input changes again
      - Input must remain stable while delay expires

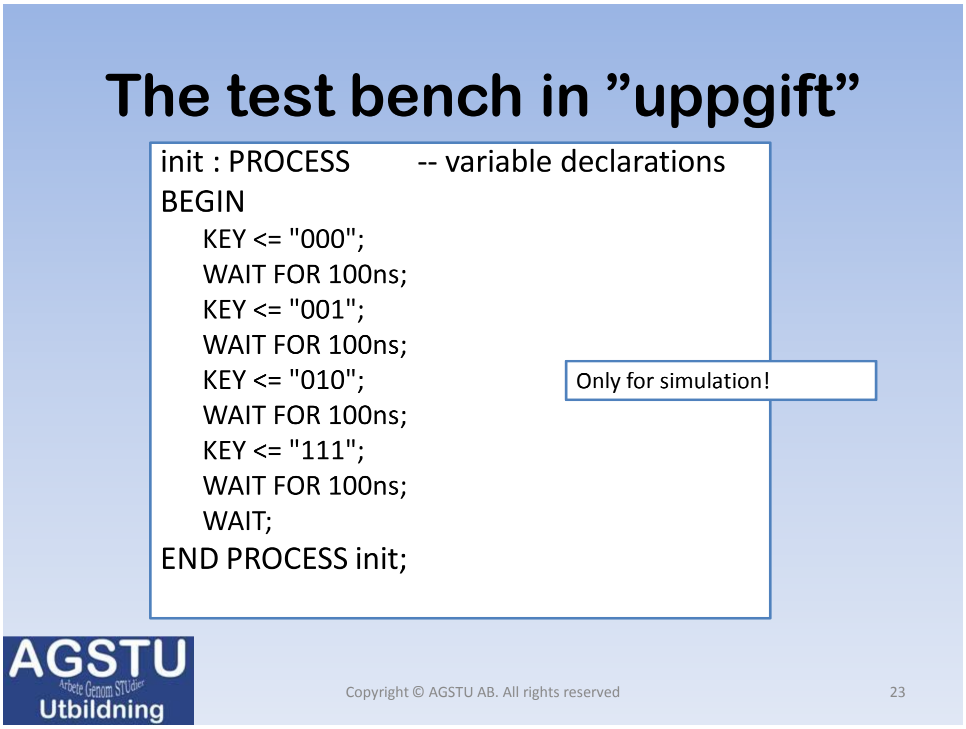
```
a <= b AFTER 10 ns;
a <= INERTIAL b AFTER 10 ns;
```

identical statements

- Transport delay
  - Always schedules output to be changed after delay passes
    - » Any transition on input transmitted to output (i.e. transmission line)

```
c <= TRANSPORT d AFTER 10 ns
```

VHDL boken sidan 50-51

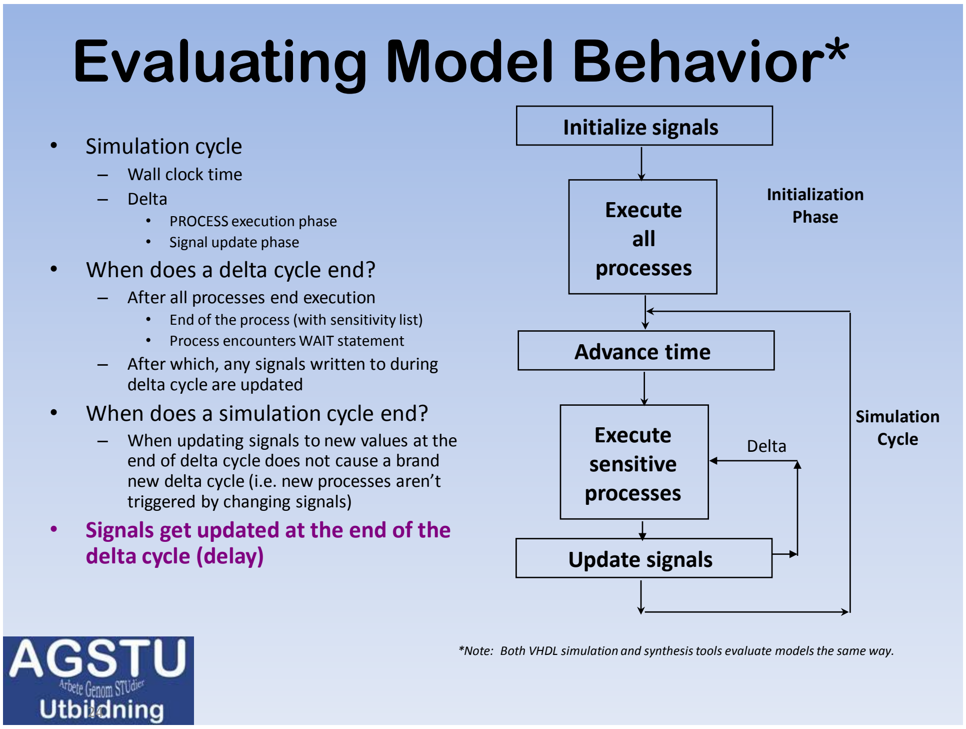


The test bench in ”uppgift”

```
init : PROCESS        -- variable declarations
BEGIN
    KEY <= "000";
    WAIT FOR 100ns;
    KEY <= "001";
    WAIT FOR 100ns;
    KEY <= "010";
    WAIT FOR 100ns;
    KEY <= "111";
    WAIT FOR 100ns;
    WAIT;
END PROCESS init;
```

Only for simulation!

23



Evaluating Model Behavior*

- Simulation cycle
  - Wall clock time
  - Delta
    - PROCESS execution phase
    - Signal update phase
- When does a delta cycle end?
  - After all processes end execution
    - End of the process (with sensitivity list)
    - Process encounters WAIT statement
  - After which, any signals written to during delta cycle are updated
- When does a simulation cycle end?
  - When updating signals to new values at the end of delta cycle does not cause a brand new delta cycle (i.e. new processes aren’t triggered by changing signals)
- **Signals get updated at the end of the delta cycle (delay)**

Initialize signals

Execute all processes

Initialization Phase

Advance time

Execute sensitive processes

Delta

Update signals

Simulation Cycle

**Note: Both VHDL simulation and synthesis tools evaluate models the same way.*

SLUT

The language is very rich and it takes some time to be able to deal with its different types ..but i love it!