# 6.111 Lecture # 6

## VHDL Statements

### Concurrent

- **Signal assignment**
- **Instantiation**
- **When-Else**
- **With-Select-When**
- **Process (used as wrapper for sequential statements)**

### Sequential (must be inside a process)

- **Signal assignment**
- **If-Then-Elsif-Else**
- **Case-When**

## Concurrent Statements

**Signal assignment:**

```
outc <= ina AND (inb OR inc);
```

**Instantiation:**

```
h1: halfadd PORT MAP (a => ina, b => inb,
                      sum => s1, c => s3);        -- named  association
```

**Or:**

```
h1: halfadd PORT MAP (ina, inb, sum, c);    -- positional   association
```

**With-Select-When** (Note: always use OTHERS as there are values other than '0' and '1')
Select conditions must be mutually exclusive and exhaustive

```
WITH inc SELECT
    outc <= ina WHEN '0',
            inb WHEN '1',
            inb WHEN OTHERS;
```

## Process Statement

**A process is concurrent with other concurrent statements**

**Multiple processes are possible in an architecture**

**A process is a wrapper for sequential statements**

- **Sequential statements model combinational or synchronous logic (or both)**
- **Statements within a process are ‘executed’ sequentially (but use care in interpreting this statement)**
- **Signal assignments can be both sequential and concurrent**

- **'Variables' may be declared within a process (more later)**
- **Signals must be a declared outside of the process**

**Process Syntax:**

```
label:    PROCESS (sensitivity list)
      VARIABLE -- declarations
BEGIN
  -- sequential statements
END PROCESS label;
```

**Process label and variable declarations are optional:**

```
PROCESS (sensitivity list)
BEGIN
      -- sequential statements
END PROCESS;
```

**Sensitivity list is there for simulation:**

- **When simulated, the process is executed when something in that list changes**
- **Sensitivity list is not synthesized in actual logic (except for the use of clocks)**

## Sequential Statements

### Signal Assignment

```
outc <= ina AND (inb or inc);
```

### If-then-elsif-else

```
IF      inc = '0' THEN outc <= ina;
ELSIF   inc = '1' THEN outc <= inb;
                  ELSE outc <= inc;
END IF;
```

### Case-When:

```
CASE inc IS
    WHEN '0'          => outc <= ina;
    WHEN '1'          => outc <= inb;
    WHEN OTHERS       => outc <= ind;
END CASE;
```

**There are other sequential statements which we won't use. See the book if you are curious (various LOOP statements, etc.)**

## Basic Operators

**Logical (use ieee.std_logic_1164.all)**

**AND, OR, NAND, NOR, XOR, XNOR, NOT**

**Relational (use ieee.std_logic_1164.all)**

**=, /=, <, >, <=, >=**
**(note that <= and => have other meanings too)**

**Arithmetic:**

**+, - (* too, but it can't be synthesized)**
**- is defined for unary arithmetic too**

**Concatenation -- defined for strings and signal values**

**&**

## Simulation vs. Synthesis

**Synthesis is to produce hardware that does what the statements specify**
**(processes as well as other concurrent statements)**

**Simulation is used to produce outputs from specified input signals**
**Concurrent statements are evaluated whenever any input changes**
**If evaluation of a concurrent statement changes the input to a concurrent statement, that statement is evaluated**

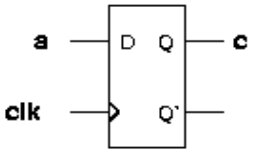
**Processes: you must list conditions that initiate evaluation of the process**
**(we use the sensitivity list for this purpose)**
**All signals in the process are updated when the process finishes**
**(NOT when each sequential statement is executed)**

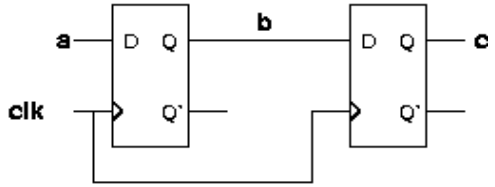
**Synthesis ignores this sensitivity list**

What will synthesis do with this?

```
library ieee;
use ieee.std_logic_1164.all;
entity reg is

  port (
    a, clk : in std_logic;
    c      : out std_logic);

end reg;
architecture top of reg is
signal b : std_logic;
begin  -- top

reg2:  process (clk)
begin  -- process
  if rising_edge(clk) then
    b <= a;
    c <= b;
  end if;
end process;


end top;
```

Will this generate one or two flip flops?



**Well, here is what it did, from the .rpt file:**

DESIGN EQUATIONS (16:16:54)

c.D =
    b.Q

c.C =
    clk

b.D =
    a

b.C =
    clk

# Implicit Memory - Latch example

```
library ieee;
use ieee.std_logic_1164.all;
entity reg is
        port (d, g: in std_logic;
                  q      : out std_logic);
end reg;

architecture top of reg is

begin
        process(d, g)
        begin
                IF g = '1' then q <= d;
                        -- notice there is no ELSE
                end IF;
end process;
end top;
-- this produces q = /g * q + d * g  (mind your g's and q's!)
```

Explicit Memory: Latch (same function, different architecture)

```
library ieee;
use ieee.std_logic_1164.all;
entity reg is
        port (d, g: in std_logic;
                q      : out std_logic);
end reg;

architecture top of reg is
        signal s1: std_logic;

begin
        q <= s1;
        s1 <= d when g = '1' else s1;
end top;
```

Explicit Memory: Latch (same function, yet another architecture)
(note this is more verbose than the prior)

```
library ieee;
use ieee.std_logic_1164.all;
entity reg is
        port (d, g: in std_logic;
                q      : out std_logic);
end reg;

architecture top of reg is
        signal s1: std_logic;

begin
        process (s1, d, g)
                begin
                        if g = '1' then s1 <= d;
                        else s1 <= s1;
                        end if;
        end process;
end top;
```

## Clocked Register (implicit memory): This is a T-ff

```
library ieee;
use ieee.std_logic_1164.all;
entity clked_t is
        port (t, clk : in std_logic;
                  q          : out std_logic);
end clked_t;

architecture top of clked_t is
        signal s1: std_logic;
begin
        q <= s1;
        process (clk, s1)
        begin
        if rising_edge(clk) then
                if t = '1'
                        then s1 <= NOT s1;
                end if;
        end if;
        end process;
end top;
-- this produces q.D = t*/q.Q + /t*q.Q
--                      and   q.C = clk
```

## Clocked Register (explicit memory): This is a T-ff

```
library ieee;
use ieee.std_logic_1164.all;
entity clked_t is
        port (t, clk : in std_logic;
                  q          : out std_logic);
end clked_t;

architecture top of clked_t is
        signal s1: std_logic;
begin
        q <= s1;
        process (clk, s1)
        begin
        if rising_edge(clk) then
                if t = '1'
                        then s1 <= NOT s1;
                        else s1 <= s1;
                end if;
        end if;
        end process;
end top;
-- this produces q.D = t*/q.Q + /t*q.Q
--                      and   q.C = clk
```

## Example: build a counter

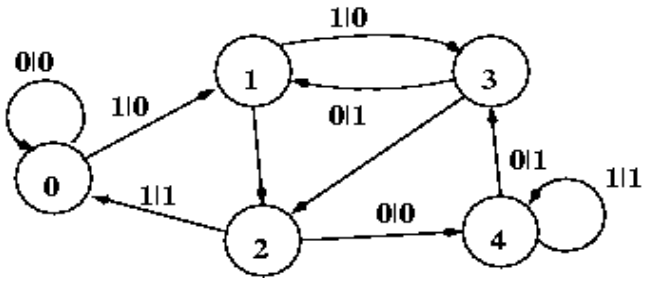
```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity ctr is
        generic (width: integer := 4); -- allows to change width easily

port(
        clk: in std_logic;
        n_clr, n_ld, enp, ent: in std_logic;
        data in std_logic_vector (width-1 downto 0);
        cnt out std_logic_vector (width-1 downto 0);
        rco out std_logic);
end ctr;
```

**Note we are using generic to define a number which we can easily redefine once we get the thing working. This is the entity for this counter. The architecture comes next.**

```
Architecture behavioral of ctr is
signal intcnt, allones: std_logic_vector (width-1 downto 0);
begin
clocked: process (clk)
begin
        if rising_edge(clk) then
                if n_clr = '0' then
                        intcnt <= (others => '0');
                elsif n_ld = '0' then
                        intcnt <= data;
                elsif (enp = '1') and (ent = '1') then
                        intcnt <= intcnt +1;
                end if;
        end if;
end process clocked;
allones <= (others => '1');
zco <= '1' when ((ent = '1') AND (intcnt = allones))
                  else '0';
cnt <= intcnt;
end behavioral;
```

Construction of a finite state machine in VHDL

Binary divide by 5, bitwise. This is a simple FSM for which current state is the running remainder. Output is the bit by bit dividend.




**Here is a suitable entity declaration:**

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity divby5 is port
                (
                 x, clk        : in std_logic;
                 y      : out std_logic);
end divby5;
```

```
architecture state_machine if divby5 is
        type StateType is (state0, state1, state2, state3, state4);
        signal p_s, n_s : StateType;
begin
        fsm: process (p_s, x)
        begin
                case p_s is
                        when state0 => y <= '0';
                                if x = '1' then
                                        n_s <= state1;
                                else
                                        n_s <= state0;
                                end if;
                        when state1 => y <= '0';
                                if x = '1' then
                                        n_s <= state3;
                                else
                                        n_s <= state2;
                                end if;
                        when state2 =>
                                if x = '1' then
                                        n_s <= state0;
                                        y <= '1';
                                else
```

Continued next slide...

```
                        n_s <= state4;
                                y <= '0';
                        end if;
                when state3 => y <= '1';
                        if x = '1' then
                                n_s <= state2;
                        else
                                n_s <= state1;
                        end if;
                when state4 => y <= '1';
                        if x = '1' then
                                n_s <= state4;
                        else
                                n_s <= state3;
                        end if;
                when others => n_s <= state0; -- avoid trap states
        end case
    end process fsm;
    state-clocked     : process (clk)
        begin
                if rising_edge(clk) then
                        p_s <= n_s;
                end if;
        end process state_clocked;
end architecture state_machine;
```