Constructing VHDL Models with CSA

- List all components (e.g., gate) inclusive propagation delays.
- Identify input/output signals as input/output ports.
- All remaining signals are internal signals.
- Identify type of each internal, input and output signal as e.g. std_logic, std_logic_vector.
- Use the information to complete the template on the following slide.
- If there are N signals and output ports, there will be N CSA statements in the VHDL model.
- CSA statements maintain a close correspondence with the hardware being modelled.
- CSA is only one out of many alternative VHDL constructs.

CSA-VHDL model template

```
library library-name-1, library-name-2;
use library-name-1.package-name.ALL;
use library-name-2.package-name.ALL;
entity entity_name is
    Port (input signals: in type;
          output signal: out type);
end entity_name;
architecture arch_name of entity_name is
-- declare internal signals
signal internal-signal-1: type := initialisation;
signal internal-signal-2: type := initialisation;
Begin
-- specify value of each signal as function of other signals
internal-signal-1 <= simple, conditional, or selected CSA;
internal-signal-2 <= simple, conditional, or selected CSA;

Output signal <= simple, conditional, or selected CSA;
end arch_name;
```

Process Construct

- CSA models close to the hardware.
- Difficult to simulate CSA models of large complex systems at gate level.
- To increase level of abstraction while preserving external event behaviour we need a more powerful language construct.
- The process construct allows us to:
  - Model at a higher level of abstraction.
  - Use conventional programming language constructs.

Memory Module [entity]

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity memory is -- use unsigned for memory address
    Port ( address : in unsigned std_logic_vector(31 downto 0);
           write_data : in std_logic_vector(31 downto 0);
           MemWrite, MemRead : in std_logic;
           read_data : out std_logic_vector(31 downto 0));
end memory;
```

Memory Module [architecture]

```
architecture Behavioral of memory is
type mem_array is array(0 to 7) of std_logic_vector(31 downto 0);
-- define type, for memory arrays
begin
mem_process: process (address, write_data)
-- initialize data memory, X denotes hexadecimal number
variable data_mem : mem_array := (
X"00000000", X"00000000", X"00000000",X"00000000",
X"00000000", X"00000000", X"00000000",X"00000000");
variable addr:integer
begin  -- the following type conversion function is in std_logic_arith
addr:=conv_integer(address(2 downto 0));
if MemWrite ='1' then
data_mem(addr):= write_data;
elsif MemRead='1' then
read_data <= data_mem(addr) after 10 ns;
end if;
end process;
end Behavioral;
```

1. Process Details

- A process is a sequentially executed block of code.
- The VHDL model on the previous two slides consists of one process that is labelled **mem_process:**.
- Similar to conventional block structured programming languages.
- Process begins with a declaration section followed by:
  - begin and end process.
- begin determines start of sequential execution.

2. Process Details

- Data structures may include:
  - Arrays, queues…
- Programs may use standard data types:
  - Integer, character, real number …
- Variable assignment take place immediately
  - Variable assignment :=
- Values assigned to variables are visible to all following statements in the context of this process.
- Control flow within a process is determined by constructs such as:
  - IF-THEN-ELSE, CASE, LOOP

3. Process Details

- A process can make assignments to signals decared externally.
  - read_data <= data_mem(addr) after 10 ns;
- Propagation delay is taken into account.
  - read_data is scheduled to take its new value after the time specified by the after clause has expired.
- The rest of the process executes in zero time with respect to simulation.
- A process is executed if an input signal in the list following the process has changed.
- The list of inputs is called **sensitivity list**.