CSU22022 Computer Architecture I

Eighth Lecture - Process

Michael Manzke

2023-2024

Trinity College Dublin

VHDL Process

- CSA models are close to the hardware
- Difficult to simulate CSA models of large complex systems at gate level
- To increase the level of abstraction while preserving external event, 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

… but

Processes
may not be used in design files
for the CSU22022
Coursework!
Exceptions are simulation files
and memory in design files

Random-access Memory

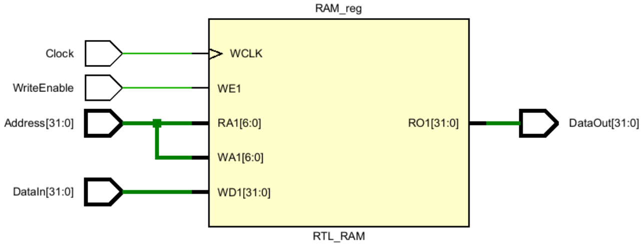

Figure 1: A Process implements this Random-access Memory

Random-access Memory VHDL Code - One

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity CPU_RAM_XXXXXXXX is
    Port ( Clock : in STD_LOGIC;
           Address : in STD_LOGIC_VECTOR (31 downto 0);
           DataIn : in STD_LOGIC_VECTOR (31 downto 0);
           WriteEnable : in STD_LOGIC;
           DataOut : out STD_LOGIC_VECTOR (31 downto 0));
end CPU_RAM_XXXXXXXX;
```

Listing 1: entity CPU_RAM_XXXXXXXX

Random-access Memory VHDL Code - Two

```vhdl
architecture Behavioral of CPU_RAM_XXXXXXXX is

-- we use the least significant 7 bit of the address
type RAM_array is array(0 to 127) of STD_LOGIC_VECTOR (31
    downto 0);

signal RAM : RAM_array:=(
X"00000000",-- 00
X"00000001",-- 01
X"00000002",-- 02
X"00000003",-- 03
```

Listing 2: type RAM_array is array(0 to 127) of STD_LOGIC_VECTOR (31 downto 0);

Random-access Memory VHDL Code - Three

```
-- Machine code
-- example studentID 87654321
-- your machine code starts at digit 3 of your ID = 4
-- Opcode = digit 3 = 4
-- DR = digit 2 = 3
-- SA = digit 1 = 2
-- SB = digit 0 = 1
--       Opcode            DR        SA        SB
  "0000000000000100"&"00011"&"00010"&"00001",-- 04
  "0000000000000101"&"00100"&"00011"&"00010",-- 05
  "0000000000000110"&"00101"&"00100"&"00011",-- 06
  "0000000000000111"&"00110"&"00101"&"00100",-- 07
  "0000000000001000"&"00111"&"00110"&"00101",-- 08
```

Listing 3: machine code

Random-access Memory VHDL Code - Two

```
          .
          .
          .
X"0000007A",-- 7A
X"0000007B",-- 7B
X"0000007C",-- 7C
X"0000007D",-- 7D
X"0000007E",-- 7E
X"0000007F" -- 7F
);
```

Listing 4: End of RAM_array

Random-access Memory VHDL Code - Two

```
begin

process (Clock)
begin
   if Clock'event and Clock='1' then
     if WriteEnable='1' then
       RAM(to_integer(unsigned(Address(6 downto 0)))) <=
    DataIn after 2ns;
     end if;
   end if;
end process;

DataOut <= RAM(to_integer(unsigned(Address(6 downto 0))))
    after 2ns;

end Behavioral;
```

Listing 5: Process that writes into the RAM_array

VHDL Process - one

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

VHDL Process - Two

- 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

VHDL Process - Three

- A process can make assignments to signals decared externally
- Propagation delay is taken into account:
  - RAM (to_integer (unsigned (Address (6 downto 0)))) <= DataIn after 2 ns ;
- 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
  - process (Clock)
- A process in a testbench has no sensitivity list. Therefore, runs constantly.
  - See 3-to-8-Line-Decoder listing 6

3-to-8-Line-Decoder Simulation Code One

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Decoder_3_to_8_Line_TB is
--  we don't need ports
end Decoder_3_to_8_Line_TB;

architecture Simulation of Decoder_3_to_8_Line_TB is

   -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Decoder_3_to_8_Line
    Port ( x, y, z : in STD_LOGIC;
           D0, D1, D2, D3, D4, D5, D6, D7 : out STD_LOGIC);
    END COMPONENT;
```

Listing 6: 3-to-8-Line-Decoder_TB entity

3-to-8-Line-Decoder Simulation Code Two

```
    --Inputs Signals
    signal x_TB, y_TB, z_TB : STD_LOGIC := '0';

    --Output Signal
    signal D0_TB, D1_TB, D2_TB, D3_TB : STD_LOGIC := '0';
    signal D4_TB, D5_TB, D6_TB, D7_TB : STD_LOGIC := '0';

    -- StudentID e.g. 26 33 57 25(DEC) = 1 91 D9 ED(HEX)
    constant StudentID : STD_LOGIC_VECTOR (27 downto 0) := x"
     191D9ED";
```

Listing 7: signal and constant

3-to-8-Line-Decoder Simulation Code Three

```vhdl
begin

   -- Instantiate the Unit Under Test (UUT)
   uut: Decoder_3_to_8_Line PORT MAP (
          x => x_TB,
          y => y_TB,
          z => z_TB,
          D0 => D0_TB,
          D1 => D1_TB,
          D2 => D2_TB,
          D3 => D3_TB,
          D4 => D4_TB,
          D5 => D5_TB,
          D6 => D6_TB,
          D7 => D7_TB
        );
```

Listing 8: Port map for the Decoder_3_to_8_Line entity

3-to-8-Line-Decoder Simulation Code Four

```
stim_proc: process
   begin
      x_TB <= '0'; y_TB <= '0'; z_TB <= '0'; -- case A
      wait for 60 ns;
      x_TB <= '0'; y_TB <= '0'; z_TB <= '1'; -- case B
      wait for 60 ns;
      x_TB <= '0'; y_TB <= '1'; z_TB <= '0'; -- case C
      wait for 60 ns;
      x_TB <= '0'; y_TB <= '1'; z_TB <= '1'; -- case D
      wait for 60 ns;
      x_TB <= '1'; y_TB <= '0'; z_TB <= '0'; -- case E
      wait for 60 ns;
      x_TB <= '1'; y_TB <= '0'; z_TB <= '1'; -- case F
      wait for 60 ns;
      x_TB <= '1'; y_TB <= '1'; z_TB <= '0'; -- case G
      wait for 60 ns;
      x_TB <= '1'; y_TB <= '1'; z_TB <= '1'; -- case H
      wait for 60 ns;
   end process;
end Simulation;
```