

Specification and Simulation of Sequential Designs using VHDL

With thanks to Sudhakar Yalamanchili

Overview

1. Introduction to VHDL

- Use in describing systems

2. Describing & modelling systems

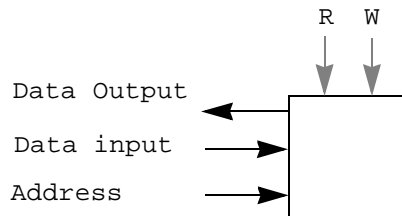
- Interface & behaviour
- Signals, events, timing, concurrency
- Use of discrete event simulation

3. Basic language constructs

- Std_logic resolved type
- Signal assignments
- Delays
- Modelling complex behaviour
- Modelling structure

4. Use of VHDL to model and simulate sequential logic designs

Modeling Complex Behavior



Memory Module

```
add R1, R2, R3  
sub R3, R4, R5  
move R7, R3
```

•
•
•

Instruction Set Simulation

- Concurrent signal assignment statements can easily capture the gate level behavior of digital systems
- Higher level digital components have more complex behaviors
 - input/output behavior not easily captured by concurrent signal assignment statements
 - models utilize state information
 - incorporate data structures
- We need more powerful constructs

The Process Statement


```
library IEEE;  
use IEEE.std_logic_1164.all;  
entity mux4 is  
port ( In0, In1, In2, In3 : in std_logic_vector (7 downto 0);  
        Sel: in std_logic_vector(1 downto 0);  
        Z : out std_logic_vector (7 downto 0));  
end mux4;
```

```
architecture behavioral-3 of mux4 is
```

```
begin
```

```
process (Sel, In0, In1, In2, In3)
```

Sensitivity List



```
variable Zout: std_logic;
```

```
begin
```

```
    if (Sel = "00") then Zout:= In0;
```

```
    elsif (Sel = "01") then Zout:= In1;
```

```
    elsif (Sel = "10") then Zout:= In2;
```

```
    else Zout:= In3;
```

```
    end if;
```

```
    Z <= Zout;
```

Variable Assignment



```
end process;
```

```
end behavioral;
```

The Process Construct

- Statements in a process are executed sequentially
- A process body is structured much like conventional C or Pascal programs
 - declaration and use of variables
 - *if-then, if-then-else, case, loop* and *while* constructs
 - process can contain signal assignment statements
- A process executes concurrently with other concurrent signal assignment statements
- A process takes 0 simulation time to execute and may schedule events in the future: we can think of a process as a complex signal assignment statement!

The Wait Statement

```
library IEEE;  
use IEEE.std_logic_1164.all;  
entity dff is  
port (D, Clk : in std_logic;  
      Q, Qbar : out std_logic);  
end dff;
```

```
architecture behavioral of dff is
```

```
begin
```

```
output: process
```

```
begin
```

```
wait until (Clk'event and Clk = '1'); -- wait for edge
```


```
    Q <= D after 5 ns;
```

```
    Qbar <= not D after 5 ns;
```

```
end process output;
```

```
end behavioral;
```

*signifies a value change
on signal clk*



- wait for *<time expression>*, wait on *<signal>*, wait until *<boolean expression>*

The Wait Statement

- A process can have multiple wait statements
- A process cannot have both a wait statement and a sensitivity list (it should have one or the other)

VHDL Identifiers, Objects, Types, & Operators

- **Identifiers**

- Used as names for variables, signals, constants, and design units such as entities, architectures, etc.
- Composed of alphanumeric characters and underscore
- Must start with a letter; may not end with underscore
- Are not case-sensitive

Data Objects

- **Classes: signals, variables, constants, and files**
- **Range of permissible values determined by type**
 - **Signals represent wires**
 - **Signals differ from variables in that:**
 - **Signals are scheduled to receive values at some time by the simulator, and one can schedule multiple values at distinct instants in the future**
 - **Variables are assigned during execution of assignment statements and can only be assigned one value at a time – variables are essentially equivalent to their conventional programming language counterparts**
 - **Constants must be declared & initialized at start of simulation & cannot be modified during the simulation**

Data Types

- **Specify the range of values an object may take and the set of operations that can be performed on it:**

Type	Range of values	Example declaration
integer	implementation dependent	signal index: integer :=0;
real	implementation dependent	variable val: real :=1.0;
boolean	(TRUE,FALSE)	variable test: boolean :=TRUE;
character	defined in package STANDARD	variable term: character := '@';
bit	0, 1	signal ln1: bit := '0';
bit_vector	array of bit	variable PC: bit_vector (31 downto 0);
time	implementation dependent	variable delay: time :=25 ns;
string	array of char	variable name: string (1 to 10):="model name";
natural	0 to maxint	variable index: natural :=0;
positive	1 to maxint	variable index: positive :=1;

Enumerated Types

- The standard set provided by VHDL can be augmented through user-defined types
- Consider trouble with “bit” signals, thus we have:

```
type std_ulogic is ('U','0','1',  
                   'Z','W','L','H','-');
```

then we can declare

```
signal carry:std_ulogic:='U';
```

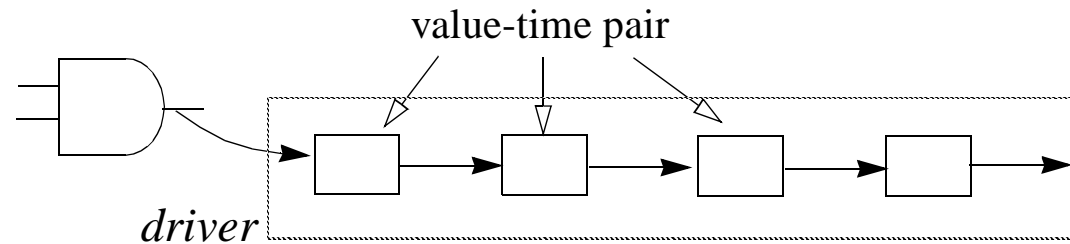
- The definition explicitly enumerates all possible values that an object of this type can assume

Attributes

- Data can be obtained about VHDL items such as types, arrays and signals.

object'attribute

- Example: consider the implementation of a signal



- what types of information about this signal are useful?
 - occurrence of an event
 - elapsed time since last event
 - previous value

Value Kind Attributes

- Return a constant value

type statetype **is** (state0, state1, state2 state3);

- state_type'**left** = state0
- state_type'**right** = state3

Value attribute	Value
type_name' left	returns the left most value of type_name in its defined range
type_name' right	returns the right most value of type_name in its defined range
type_name' high	returns the highest value of type_name in its range
type_name' low	returns the lowest value of type_name in its range
array_name' length	returns the number of elements in the array array_name

Function Kind Attributes

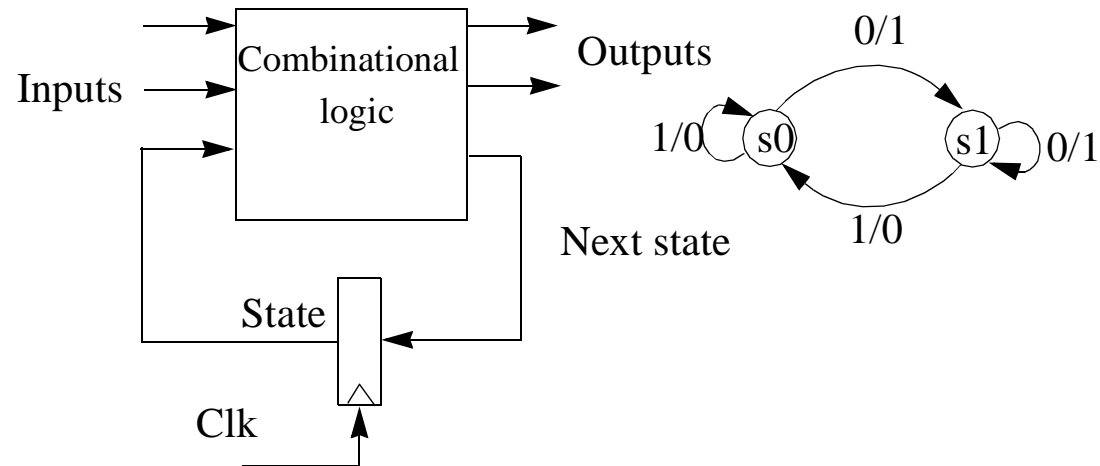
- Use of attributes invokes a function call which returns a value if (`clk'event` and `clk = '1'`)

function call

- Function signal attributes

Function attribute	Function
<code>signal_name'event</code>	Return a Boolean value signifying a change in value on this signal
<code>signal_name'active</code>	Return a Boolean value signifying an assignment made to this signal. This assignment may not be a new value.
<code>signal_name'last_event</code>	Return the time since the last event on this signal
<code>signal_name'last_active</code>	Return the time since the signal was last active
<code>signal_name'last_value</code>	Return the previous value of this signal

State Machines



- Basic components
 - combinational component: output function and next state function
 - sequential component
- Natural process implementation

Example: State Machine

```
library IEEE;  
use IEEE.std_logic_1164.all;  
entity state_machine is  
port(reset, clk, x : in std_logic;  
      z : out std_logic);  
end state_machine;  
  
architecture behavioral of state_machine is  
type statetype is (state0, state1);  
signal state, next_state : statetype := state0;  
begin  
  comb_process: process (state, x)  
  begin  
    --- process description here  
  end process comb_process;  
  
  clk_process: process  
  begin  
    -- process description here  
  end process clk_process;  
end behavioral;
```

Example: Output and Next State Functions

```
comb_process: process (state, x)
begin
  case state is -- depending upon the current state
  when state0 => -- set output signals and next state
    if x = '0' then
      next_state <= state1;
      z <= '1';
    else next_state <= state0;
      z <= '0';
    end if;
  when state1 =>
    if x = '1' then
      next_state <= state0;
      z <= '0';
    else next_state <= state1;
      z <= '1';
    end if;
  end case;
end process comb_process ;
```

- Combination of the next state and output functions

Example: Clock Process

```
clk_process: process
begin
wait until (clk'event and clk = '1'); -- wait until the rising edge
  if reset = '1' then -- check for reset and initialize state
    state <= statetype'left;
  else state <= next_state;
  end if;
end process clk_process;
end behavioral;
```

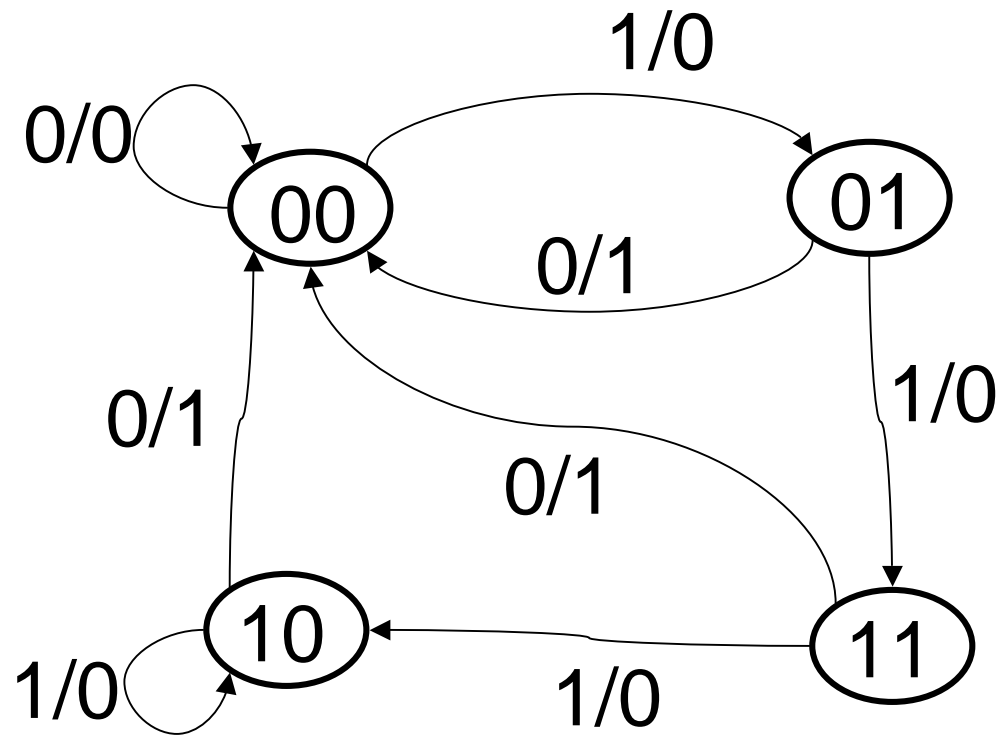
- Use of synchronous reset to initialize into a known state

Three process state machine model

- It is also common, and perhaps clearer to split the VHDL description of a finite state machine into three processes:
 1. A process describing the *next state* function;
 2. A process describing the *output* function; and
 3. A process describing the *clock or state update* function.
- This model form splits the combinational logic into the first two processes

Example

- The following example from Mano & Kime, Fig 6-19(a) illustrates the three process modelling approach by describing the state diagram below



Sequential VHDL Example: Figure 6-19(a)

- VHDL for the sequential circuit (FSM) in Fig. 6-19(a) follows

```
library ieee;
```

```
use ieee.std_logic_1164.all;
```

```
entity fig619a is
```

```
    port (CLK, RESET, X: in std_logic;
```

```
          Z: out std_logic);
```

```
end fig619a;
```

```
architecture sequential of fig619a is
```

```
    type state_type is (S0, S1, S2, S3);
```

```
    signal state, next_state: state_type;
```

```
begin
```

Sequential VHDL Example: Figure 6-19(a) (continued)

```
state_register: process (CLK, RESET)
begin
    if(RESET = '1') then
        state <= S0;
    elsif (CLK'event and CLK = '1') then
        state <= next_state;
    end if;
end process;

next_state_function: process (X, state)is
begin
    case state is
        when S0 =>
            if X = '1' then next_state <= S1;
            else next_state <= S0;
            end if;
    end case;
end process;
```

Sequential VHDL Example: Figure 6-19(a) (continued)

```
when S1 =>
    if X = '1' then next_state <= S3;
    else next_state <= S0;
    end if;
when S2 =>
    if X = '1' then next_state <= S2;
    else next_state <= S0;
    end if;
when S3 =>
    if X = '1' then next_state <= S2;
    else next_state <= S0;
    end if;
when others => next_state <= S0; -- Returns to S0 for
                                -- non-binary combinations
                                -- containing one or more X,Z,U,etc.

end case;
end process;
```

Sequential VHDL Example: Figure 6-19(a) (continued)

```
output_function: process (X, state) is
begin
  case state is
    when S0 => Z <= '0';
    when S1 => if X = '1' then Z <= '0';
               else Z <= '1'; end if;
    when S2 => if X = '1' then Z <= '0';
               else Z <= '1'; end if;
    when S3 => if X = '1' then Z <= '0';
               else Z <= '1'; end if;
    when others => Z <= '0'; -- Changes Z to 0 for non-binary
                           -- combinations containing one or more X,Z,U,etc.
  end case;
end process;
end architecture;
```