Digital Systems Design

Review of VHDL for Sequential Circuits

Overview

• Basic storage elements
  – Structural design using library components
  – Behavioral design: D latches and D flip-flops
    • Options including
      – Synchronous and asynchronous reset
      – Multiplexed inputs
      – Enable inputs

• Counters
• Shift Registers
• Arbitrary finite state machines (FSM)
  – Mealy and Moore model designs
Structural VHDL Using a D flip-flop

```vhdl
library ieee;
use ieee.std_logic_1164.all;
library altera;
use altera.maxplus2.all;

entity flipflop is
  port (d, clock : in std_logic;
        reset_n, preset_n : in std_logic;
        q : out std_logic);
end flipflop;

architecture structure of flipflop is
begin
  dff_instance: dff port map (d, clock, reset_n, preset_n, q);
end structure;
```

**reset_n and preset_n: Active low signals**

Code for a gated D latch

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity latch is
  port (d, clk : in std_logic;
        q : out std_logic);
end latch;

architecture behavior of latch is
begin
  process (d, clk)
  begin
    if clk = '1' then
      q <= d;
    end if;
  end process;
end behavior;
```

**USES IMPLIED MEMORY**

If no assignment is made, the last assignment is remembered.
Code for a D flip-flop

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

architecture behavior of flipflop is
begin
  process (clock)
  begin
    if clock'event and clock = '1' then
      q <= d;
    end if;
  end process;
end behavior;
```

POSITIVE EDGE TRIGGERED

Code for a D flip-flop (alternate)

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

architecture behavior of flipflop is
begin
  process (clock)
  begin
    if rising_edge(clock) then
      q <= d;
    end if;
  end process;
end behavior;
```

POSITIVE EDGE TRIGGERED. USE falling_edge FOR NEGATIVE EDGE
### Code for a D flip-flop (alternate)

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

architecture behavior of flipflop is
begin
    process
        begin
            wait until clock'event and clock = '1';
            q <= d;
        end process;
    end behavior;
end flipflop;
```

### D flip-flop with synchronous reset

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity flipflop is
    port ( d, reset_n, clock : in std_logic;
           q         : out std_logic);
end flipflop;
architecture behavior of flipflop is
begin
    process
        begin
            wait until clock'event and clock = '1';
            if reset_n = '0' then
                q <= '0';
            else
                q <= d;
            end if;
        end process;
    end behavior;
end flipflop;
```
D flip-flop with MUX input

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity muxdff is
  port ( d0, d1, sel, clock : in std_logic;
         q       : out std_logic);
end muxdff;

architecture behavior of muxdff is
begin
  process
  begin
    wait until clock'event and clock = '1';
    if sel = '0' then
      q <= d0;
    else
      q <= d1;
    end if;
  end process;
end behavior;
```

This will be our preferred method for creating a D flip-flop or a multibit register with enable.

D flip-flop with enable input

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity flipflop is
  port ( enable, d, clk : in std_logic;
         q       : out std_logic);
end flipflop;

architecture behavior of flipflop is
begin
  process(clk)
  begin
    if enable='0' then null;
    elsif rising_edge(clk) then
      q <= d;
    end if;
  end process;
end behavior;
```

This will be our preferred method for creating a D flip-flop or a multibit register with enable.
D flip-flop with asynchronous reset

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity flipflop is
  port ( reset_n, d, clk : in std_logic;
        q : out std_logic);
end flipflop;
architecture behavior of flipflop is
begin
  process(clk,reset_n)
  begin
    if reset_n = '0' then
      q <= '0';
    elsif rising_edge(clk) then
      q <= d;
    end if;
  end process;
end behavior;
```

This will be our preferred method for creating a D flip-flop or a multibit register with reset.

Counter Modeling with VHDL

- Counters are simple examples of sequential circuits
- Counters can be modeled as arbitrary FSMs, but this is not the most straightforward method of modeling these circuits
- Counters can be easily modeled using basic arithmetic expressions
- Options include:
  - Arithmetic operations on UNSIGNED and SIGNED signals
  - Use of the INTEGER data type
library ieee;
use ieee.std_logic_1164.all;
-- use numeric_std to include
-- signed and unsigned data types
use ieee.numeric_std.all;

entity upcount is
port ( clock, reset_n,e : in std_logic;
q : out unsigned(3 downto 0));
end upcount;

architecture behavior of upcount is
signal count : unsigned(3 downto 0);
begin
process (clock,reset_n)
begin
if reset_n = '0' then
count <= "0000";
elsif (clock'event and clock = '1') then
if e = '1' then
count <= count + 1;
else
count <= count;
end if;
end if;
end process;
q <= count;
end behavior;
library ieee;
use ieee.std_logic_1164.all;

entity count is
port( clock : in std_logic;
     sload : in std_logic;
     -- integer data types are a default size of 32 bits
     -- use a range specifier to limit the number of bits
     -- generated for the register (5 bits in this case)
     data : in integer range 0 to 31;
     result : out integer range 0 to 31;
end count;

architecture rtl of count is
begin
    signal result_reg : integer range 0 to 31;
    process (clock)
    begin
        if (clock'event and clock = '1') then
            if (sload = '1') then
                result_reg <= data;
            else
                result_reg <= result_reg + 1;
            end if;
        end if;
    end process;
    result <= result_reg;
end rtl;
-- vhdl code for an 8-bit shift-left register with a positive-
-- edge clock, asynchronous clear, serial in, and serial out.

library ieee;
use ieee.std_logic_1164.all;

entity shift is
port(
    -- clk is the clock for the shift operation
    -- si is a serial input into the lsb of the shift register
    -- clr is an asynchronous active high clear control signal
    clk, si, clr : in std_logic;
    -- so is a serial output from the msb of the shift register
    so : out std_logic);
end shift;

architecture behavior of shift is
    signal s_reg: std_logic_vector(7 downto 0);
begnin
    process(clk,clr)
    begin
        if(clr='1') then
            s_reg <= "00000000";
        elsif rising_edge(clk) then
            s_reg <= s_reg (6 downto 0) & si;
        end if;
    end process;
    so <= s_reg(7);
end behavior;
Arbitrary FSM design using CAD tools

- VHDL provides a number of constructs for designing finite state machines
- There is not a standard way for defining an FSM
- Basic approach
  - Create a user-defined data type to represent the possible states of an FSM
  - This signal represents the outputs (state variables) of the flip-flops that implement the states in the FSM
  - VHDL compiler chooses the appropriate number of flip-flops during the synthesis process
  - The state assignment can be done by the compiler or can be user specified

User defined data types

- The `TYPE` keyword will be used to define a new data type used to represent states in the FSM

```vhdl
type fsm_state is (a, b, c);
```

A user-defined data type definition  Data type name  Valid values for the data type

Defines a data type (called `fsm_state`) that can take on three distinct values: `a`, `b`, or `c`.

Representing states

- A `SIGNAL` is defined, of the user-defined `fsm_state`, to represent the flip-flop outputs

```vhdl
    type fsm_state is (a, b, c);
    signal y: fsm_state;
```

The signal, `y`, can be used to represent the flip-flop outputs for an FSM that has three states.

Design example

- Create a VHDL description for a circuit that detects a '11' input sequence on an input, `w`

Moore state diagram for the circuit
library ieee;
use ieee.std_logic_1164.all;

entity detect is
  port( clk, reset_n, w : in std_logic;
        z       : out std_logic);
end detect;

architecture behavior of detect is
  type fsm_state is (a,b,c);
  signal y: fsm_state;
begin

  process (reset_n, clk) begin
    if reset_n = '0' then
      y <= a;
    elsif rising_edge(clk) then
      case y is
        when a =>
          if w='0' then
            y <= a;
          else
            y <= b;
          end if;
        when b =>
          if w='0' then
            y <= a;
          else
            y <= c;
          end if;
        when c =>
          if w='0' then
            y <= a;
          else
            y <= c;
          end if;
      end case;
    end if;
  end process;
  z <= '1' when y=c else '0';
end behavior;

Alternative style of VHDL code

- Another form for describing the circuit in VHDL is to define two signals to represent the state of the FSM
  - One signal, \( y_{\text{present}} \), defines the present state of the FSM
  - The second, \( y_{\text{next}} \), defines the next state of the machine
- This notation uses \( y \) (present state) and \( Y \) (next state) for state information
- Two PROCESS statements will be used to describe the machine
  - The first describes the STATE TABLE as a combinational circuit
  - The second describes the flip-flops, stating that \( y_{\text{present}} \) should take on the value of \( y_{\text{next}} \) after each positive clock edge

Alternate VHDL description

```vhdl
architecture behavior of detect is
  type fsm_state is (a,b,c);
  signal y_present, y_next: fsm_state;
begin
  process(w,y_present)
  begin
    case y_present is
    when a =>
      if w='0' then
        y_next <= a;
      else
        y_next <= b;
    when b =>
      if w='0' then
        y_next <= a;
      else
        y_next <= c;
    when others =>
      y_next <= a;
  end case;
end process;
```

Alternate VHDL description

```vhdl
when c =>
  if w='0' then
    y_next <= a;
  else
    y_next <= c;
  end case;
end process;

process(clk, reset_n)
begin
  if reset_n='0' then
    y_present <= a;
  elsif rising_edge(clk) then
    y_present <= y_next;
  end if;
end process;

z <= '1' when y_present=c else '0';
end behavior;
```

Specifying a state assignment

- With the previous designs, state assignment is done by the VHDL compiler
- State assignments can be user specified

```vhdl
architecture behavior of simple is
  type fsm_state is (a, b, c);
  attribute enum_encoding : string;
  attribute enum_encoding of fsm_state : type is "00 01 11";
  signal y_present, y_next : fsm_state;
begin
  or
architecture behavior of simple is
  signal y_present, y_next : std_logic_vector(1 downto 0);
  constant a : std_logic_vector(1 downto 0) := "00";
  constant b : std_logic_vector(1 downto 0) := "01";
  constant c : std_logic_vector(1 downto 0) := "11";
```
A Mealy FSM can be described in a similar manner as a Moore FSM.

The state transitions are described in the same way as the original VHDL example.

The major difference in the case of a Mealy FSM is the way in which the code for the output is written.

Recall the Mealy state diagram for the ‘11’ sequence detector.

VHDL code of a Mealy FSM

```
architecture behavior of detect is
type fsm_state is (a,b);
signal y: fsm_state;
begin
  process(reset_n,clk)
  begin
    if reset_n='0' then
      y <= a;
    elsif rising_edge(clk) then
      case y is
        when a =>
          if w='0' then y<= a;
          else y<= b;
          end if;
        when b =>
          if w='0' then y<= a;
          else y<= b;
          end if;
          end case;
    end if;
  end process;
end behavior;
```

Mealy ‘11’ detector VHDL code

```
architecture behavior of detect is
end case;
end if;
end process;
begin
  process(y,w)
  begin
    case y is
      when a =>
        z <= '0';
      when b =>
        z <= w;
      end case;
    end case;
    end process;
    end behavior;
```

```
architecture behavior of detect is
end case;
end if;
end process;
begin
  process(reset_n,clk)
  begin
    if reset_n='0' then
      y <= a;
    elsif rising_edge(clk) then
      case y is
        when a =>
          if w='0' then y<= a;
          else y<= b;
          end if;
        when b =>
          if w='0' then y<= a;
          else y<= b;
          end if;
          end case;
    end if;
  end process;
end behavior;
```