Logic and Computer Design Fundamentals

VHDL

Part 3 – Chapter 6 – Finite State Machines

Charles Kime & Thomas Kaminski

Overview

- **Part 1 - VHDL Basics and Types of Descriptions**
- **Part 2 - Behavioral and Hierarchical Description**
- **Part 3 - Finite State Machines**
  - **Finite State Machine (FSM)**
  - **Sequential VHDL - Process Description**
  - **VHDL Keywords and Conditional Constructs**
  - **Example: Process VHDL for a Positive Edge-Triggered D Flip-Flop**
  - **Sequential VHDL for Figure 6-19(a)**
- **Part 4 - Registers and Counters**
- **Part 5 - Algorithmic State Machine Example: Binary Multiplier**

Finite State Machine

- ***Finite State Machine (FSM)*** **– a more theoretical term for sequential circuit, usually, a sequential circuit described at a level higher than structural.**
- **Consists of:**
  - **Inputs**
  - **Outputs**
  - **State**
  - **Next state function, and**
  - **Output function**

Process Description

- So far, have used structural and concurrent (dataflow) descriptions
- The above are limited in complexity and capability of description
- *Process* – can be viewed as a replacement for a concurrent assignment statement that permits more complex descriptions
- A process uses *procedural assignment statements* similar to those in a typical sequential programming language
- Processes are a key element of VHDL for the description of both combinational and sequential circuits.
- Multiple processes can execute concurrently with each other and with concurrent assignment statements

Process Basics

- **Process header:**
  - **Optional: process_label followed by :**
  - **Keyword `process` followed by *sensitivity list* – (list of signals or expressions that cause the process to execute if any one or more change)**
- **Process body may include:**
  - **One or more variable declarations**
    - ***variable* – alternative to signal that is used in statements that execute sequentially rather that concurrently in processes**
    - **A variable is available only within the process where it is declared**
    - **keyword `variable`**
    - **Sequential assignment `:=` used instead of concurrent assignment `<=`**
  - **The actual process beginning with `begin` and ending with `end;` or `end process;`**

New Control Flow Constructs

- There are a number of new control flow constructs, two of which are:
  - if-then-else
  - case
- if-then-else syntax:

```
if condition then
    sequence of statements
{elsif condition then
    sequence of statements}
else
    sequence of statements
end if
```

in which the { } indicate that the enclosed statements can appear from 0 to any number of times

New Control Flow Constructs (continued)

- **if-then-else Example: (A is a variable and B is a signal)**

```
if X = '1' then
    B <= D;
elsif Y = '0' then
    begin
        A := C;
        B <= A;
    end
else
    B <= E;
end if
```

- **Considering `case` later!**

Sequential VHDL Example: Positive Edge-Triggered D Flip-Flop with Reset

```
library ieee;
use ieee.std_logic_1164.all;
entity dff is
   port(CLK, RESET, D : in std_logic;
        Q : out std_logic);
end dff;

architecture pet_pr of dff is
begin
   if (RESET = '1') then
      Q <= '0';
   elsif (CLK'event and CLK = '1')then
         Q <= D;
   end if;
end process;
end pet_pr;
```

New Flow Control Constructs (continued)

- **case syntax:**

```
case expression is
  {when choices =>
        sequence of statements}
end case;
```

in which the { } indicate that the enclosed statements can appear from 0 to any number of times.

New Flow Control Constructs (continued)

- **case Example:** `Z: std_logic_vector(1:0);`

```
case Z is
    when "00" => B <= D;
    when "01" =>
      A := C;
      B <= A;
    when "10" => B <= A;
    when "11" => B <= E;
end case;
```

Declaration of Type state_type

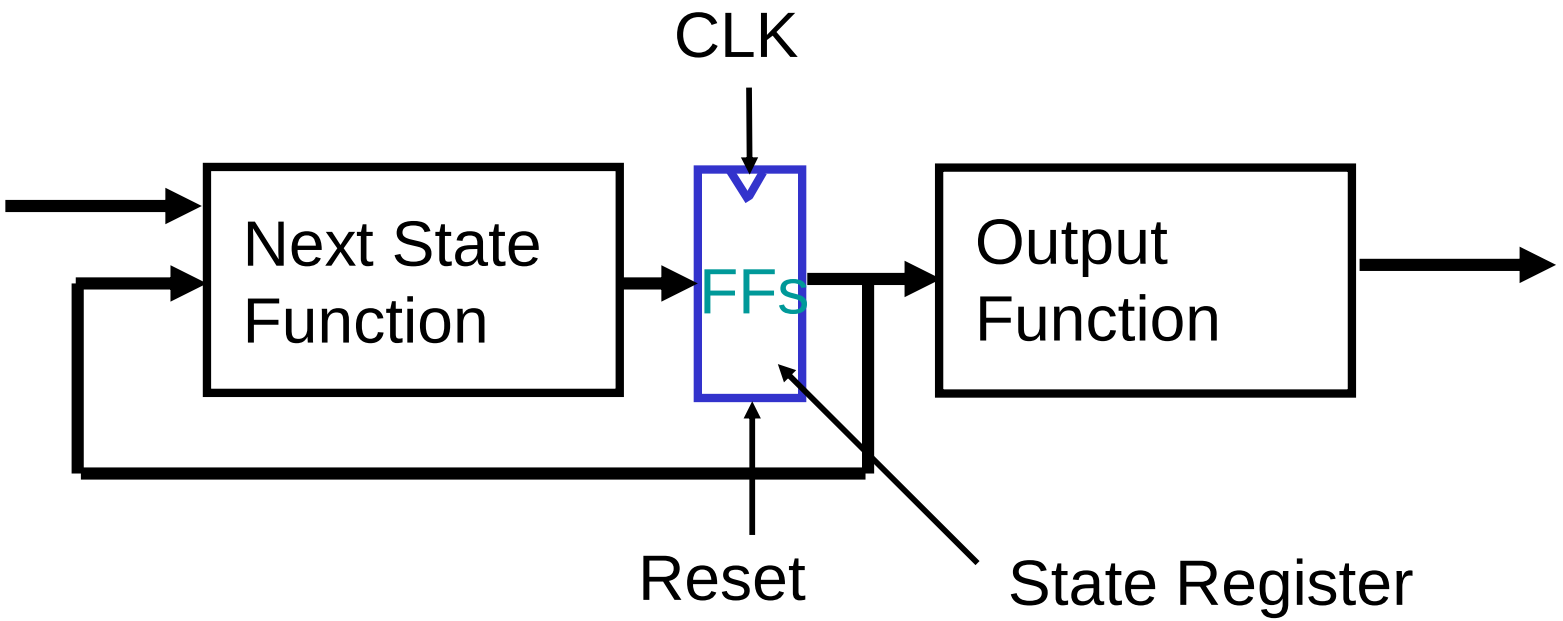
- In VHDL, the user can declare new state types.
- It is useful in describing FSM to declare a type for states.
- Example: FSM has three states with identifiers IDLE, INIT and RUN

```
type state_type is (IDLE, INIT, RUN);
signal state, next_state : state_type;
```

- This permits VHDL descriptions to use states that have no binary codes assigned and no signal of type `std_logic` or `std_logic_vector` declared for representing the register to store them.

Describing Sequential Circuits

- **There are many different ways to organize models for sequential circuits. We will use a model that corresponds to the following diagram:**



- **A process corresponds to each of the 3 blocks in the diagram.**

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;
```

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;
```

Terms of Use

- Return to Title Page