Useful VHDL

Digital Systems Diagrams
VHDL Design Unit

Interface Specification

Work Specification

to/from other modules

to/from other modules

VHDL Design Unit

Entity Declaration

Architecture Body

to/from other modules

to/from other modules
Basic Digital Logic Functions

- Gates
- Multiplexer
- Decoder
- Adder
- Subtractor

Gates: Simple Signal Assignment

NAND_L <= not ( A_H and B_H and C_H );

AND_H <= A_H and B_H and C_H;

CARRY_H <= ( A_H and B_H ) or
             ( A_H and C_H ) or
             ( B_H and C_H );
Multiplexer: Conditional Signal Assignment

\[
\text{OUTPUT}_H \leftarrow \begin{cases} 
\text{IN}_0_H & \text{when } \text{ABC} = \text{"000" and EN} = \text{'0'} \\
\text{IN}_1_H & \text{when } \text{ABC} = \text{"001" and EN} = \text{'0'} \\
\text{IN}_2_H & \text{when } \text{ABC} = \text{"001" and EN} = \text{'0'} \\
\text{IN}_3_H & \text{when } \text{ABC} = \text{"001" and EN} = \text{'0'} \\
\text{IN}_4_H & \text{when } \text{ABC} = \text{"001" and EN} = \text{'0'} \\
\text{IN}_5_H & \text{when } \text{ABC} = \text{"001" and EN} = \text{'0'} \\
\text{IN}_6_H & \text{when } \text{ABC} = \text{"001" and EN} = \text{'0'} \\
\text{IN}_7_H & \text{when } \text{ABC} = \text{"001" and EN} = \text{'0'} \\
\text{DEFAULT}_H & \text{else}
\end{cases}
\]

Decoder: Conditional Signal Assignment

\[
\text{ENBL}_H \leftarrow \begin{cases} 
\text{‘1’} & \text{when PSR} = \text{ST02} \text{ else ‘0’}
\end{cases}
\]

\[
\text{COUNT}_H \leftarrow \begin{cases} 
\text{‘1’} & \text{when PSR} = \text{ST02 or PSR} = \text{ST04 else ‘0’}
\end{cases}
\]

\[
\text{SYS_RST}_H \leftarrow \begin{cases} 
\text{‘1’} & \text{when CYCLE} < 10 \text{ else ‘0’}
\end{cases}
\]

Adder/Subtractor: Signal Assignment

ADD_OUT <= A_IN + B_IN;
SUB_OUT <= A_IN – B_IN;

ALU_OUT <= A_IN + B_IN + CARY when FUN = “001” else
        A_IN – B_IN – CARY when FUN = “010” else
        A_IN (7) & A_IN (7 downto 1)
        when FUN = “011” else
        (others => ‘0’);

Basic Digital Logic Storage Functions

• Flip-Flop
• Register
• Shift Register
• Counter
• State Register
Flip-Flop: Use Process

entity FF is
  port (
    D_IN  : in  STD_LOGIC;
    D_OUT : out STD_LOGIC;
    CLK   : in  STD_LOGIC;
    CLR   : in  STD_LOGIC
  );
end entity FF;

architecture SIMPLE of FF is

signal INT_FF : STD_LOGIC;
begin

  FF_PROC:
  process ( CLK, CLR ) is
  begin
    if CLR = '1' then
      INT_FF <= '0';
    elsif RISING_EDGE ( CLK ) then
      INT_FF <= D_IN;
    end if;
  end process;

  D_OUT <= INT_FF;

end architecture SIMPLE;
entity REG is
  port (  
    D_IN  : in  STD_LOGIC_VECTOR ( 7 downto 0 );  
    D_OUT : out STD_LOGIC_VECTOR ( 7 downto 0 );  
    CLK   : in  STD_LOGIC;  
    CLR   : in  STD_LOGIC
  );
end entity REG;

architecture SIMPLE of REG is
signal INT_REG : STD_LOGIC_VECTOR ( 7 downto 0 );
begin

REG_PROC:
  process ( CLK, CLR ) is
  begin
    if CLR = '1' then
      INT_REG <= "00000000";
    elsif RISING_EDGE ( CLK ) then
      INT_REG <= D_IN;
    end if;
  end process;

  D_OUT <= INT_REG;

end architecture SIMPLE;
Useful Operator: Concatenation

Task: concatenate "0111" with "1010" to make "01111010"

Operator: &

Let A be STD_LOGIC_VECTOR ( 3 downto 0 ) and A = "0111"
Let B be STD_LOGIC_VECTOR ( 3 downto 0 ) and B = "1010"
Let C be STD_LOGIC_VECTOR ( 7 downto 0 )
then C <= A & B;

Shift with Concatenation Operator

Let C be STD_LOGIC_VECTOR ( 7 downto 0 ) with value "01111010"
Let D be STD_LOGIC_VECTOR ( 7 downto 0 )

Object: let D be C arithmetically shifted right by two bit positions:

D <= C(7) & C(7) & C( 7 downto 2 );
Shift Register: Register with Shift

entity SR_REG is
port (
    D_IN    : in  STD_LOGIC_VECTOR ( 7 downto 0 );
    D_OUT   : out STD_LOGIC_VECTOR ( 7 downto 0 );
    S_IN    : STD_LOGIC;
    LOAD_H  : in  STD_LOGIC;
    SHIFT_H : in  STD_LOGIC;
    CLK     : in  STD_LOGIC;
    CLR     : in  STD_LOGIC
);
end entity SR_REG;

architecture SIMPLE of SR_REG is
signal INT_REG : STD_LOGIC_VECTOR ( 7 downto 0 );
begin
REG_PROC:
    process ( CLK, CLR ) is
        begin
            if CLR = '1' then
                INT_REG <= "00000000";
            elsif RISING_EDGE ( CLK ) then
                if LOAD_H = '1' then
                    INT_REG <= D_IN;
                elsif SHIFT_H = '1' then
                    INT_REG <= S_IN & INT_REG ( 7 downto 1 );
                end if;
            end if;
        end if;
    end process;
    D_OUT <= INT_REG;
end architecture SIMPLE;
More Details

- Enumeration type - lists possible values
- Syntax: `type TYPE_NAME is ( LIST );`
- Examples: `type BIT is ( ‘0’, ‘1’ );`
  `type STATE is (S0, S1, S2, S3, S4 );`
- Position numbers start at 0, increment through the list

Concurrent Statement

- Block statement
- Process statement
- Component instantiation
- Generate statement
- Concurrent signal assignment statement
- Concurrent assertion statement
- Concurrent procedure call statement
Conditional Signal Assignment

- Concurrent statement
- Identify options and waveforms
- Syntax:
  $\text{TARGET} \leftarrow \text{waveform1 when COND1 else}
  \text{waveform2 when COND2 else}
  \text{waveform3 when COND3 else}
  \ldots \text{else}
  \text{waveform}_n$;

Selected Signal Assignment

- Concurrent statement
- Identify options and waveforms
- Syntax:
  with $\text{EXPRESSION}$ select
  $\text{TARGET} \leftarrow \text{waveform1 when } \text{CHOICE1},$
  $\text{waveform2 when } \text{CHOICE2},$
  $\text{waveform3 when } C3 \mid C4,$
  $\text{waveformk when } Ck$;
Process Statement

• Concurrent statement
• Activity in process - sequential
• Delay mechanism user selectable
  – sensitivity list - comma separated list of signals
  – event on any signal in list causes process activity
  – wait statements in process body
• Activity visited once on startup

Process Statement Syntax

process PROC_NAME ( SEN_LIST ) is

{ declaration area }
Begin

{sequential statement}*

end process {PROC_NAME};
Sequential Statements

- Signal Assignment Statement
- If Statement
- Case Statement
- Loop Statement
- Wait statement
- Assertion Statement
- Report Statement

Sequential Statements (Cont.)

- Variable assignment statement
- Procedure Call Statement
- Next Statement
- Exit Statement
- Null Statement
Case Statement Syntax

case SEL_EXP is
  when CHOICE => {sequential stmt}
  when CHOICE | CHOICE => {seq stmt}
  when others => {sequential stmt}
end case ;