VHDL modeling style

- Synthesis semantics: Ch. 10

Simulation
- Defined in the Language Reference Manual
- IEEE Standard is universally adopted

Synthesis
- Specific to each vendor
- A great deal of consensus exists on major points
- Each vendor has unique features

Synthesis overview

- Questions:
  - What type of hardware can implement
    - A signal?
    - A variable?
    - A process?
  - When does a VHDL description require only combinational logic to be implemented in hardware? Sequential logic?
  - How many bits are required to represent a signal?

- Understanding how VHDL is converted into hardware helps us write descriptions that do what we expect when synthesized.

Synthesis Example

Here’s a concurrent signal assignment:

```vhdle
entity syn is
  port (a, b, c: in std_logic;
       z: out std_logic);
end syn;

architecture first of syn is
begin
  z <= a and b and c;
end first;
```

Same thing, with a variable assignment for an intermediate value:

```vhdle
entity syn is
  port (a, b, c: in std_logic;
       z: out std_logic);
end syn;

architecture second of syn is
begin
  process (a,b,c)
  variable var: std_logic;
  begin
    var := b and c;
    z <= a and var;
  end process;
end second;
```

Screwy example...

Same thing, in a screwy but still valid way:

```vhdle
entity syn is
  port (a, b, c: in std_logic;
       z: out std_logic);
end syn;

architecture third of syn is
begin
  process (a,b,c)
  variable var: std_logic;
  begin
    if (b'event)then
      var := b and c;
    elsif (c'event) then
      var := b and c;
    end if;
    z := a and var;
  end process;
end third;
```

Moral: You can't ignore all of the warnings...

Creating RTL descriptions

- What’s going on here?
  - Some constructs imply sequential hardware, while others imply only combinational hardware.

- Is it safe to say that processes always imply sequential hardware, and concurrent signal assignments always imply combinational hardware?
Creating RTL descriptions

A concurrent signal assignment that requires sequential hardware:

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

entity syn2 is
  port (a, b, c: in std_logic;
       z: inout std_logic);
end syn2;

architecture cond of syn2 is
begin
  z <= a when b='1'
       else b when c='1'
       else z;
end cond;
```

And we've already seen a process that requires only combinational hardware:

```vhdl
architecture second of syn is
begin
  process (a,b,c)
  variable var: std_logic;
  begin
    var := b and c;
    z <= a and var;
  end process;
end second;
```

Process: Combinational or Sequential?

Determining if a process requires sequential hardware:

- Are there multiple wait statements?
- Are variables/signals used before being assigned?
- Are output signals target of assignment whatever branch is activated?

A more realistic example

```vhdl
entity mistake is port (inp: in std_logic_vector (1 downto 0);
                       outp: out std_logic_vector (3 downto 0);
                       even, odd: out std_logic);
end mistake;

architecture behave of mistake is
begin
  process (inp)
  begin
    case inp is
    when "00" => outp <= "0001";
       even <= '1';
       odd <= '0';
    when "01" => outp <= "0010";
       even <= '0';
       odd <= '1';
    when "10" => outp <= "0100";
       even <= '1';
--    odd <= '0';
    when "11" => outp <= "1000";
       even <= '0';
       odd <= '1';
    end case;
  end process;
end behave;
```

Synplicity Warning: “Latch generated from process for signal odd, probably caused by a missing assignment in an if or case stmt”

With assignment commented out
entity D_LATCH is
  port (GATE, DATA: in STD_LOGIC;
Q: out STD_LOGIC);
end D_LATCH;
architecture BEHAV of D_LATCH is
begin
LATCH: process (GATE, DATA)
begin
if (GATE = '1') then
Q <= DATA;
ext if;
end process; -- end LATCH
end BEHAV;

Incomplete conditional expressions always result in
latches. If a CLB has no built-in latches, it must use the
look up tables to implement it.

Code Template 1
if (RESET = '1') then     -- or (RESET = '0')
-- perform asynchronous reset and initialization
elsif (CLK'EVENT and CLK = '1')
-- or (CLK'EVENT and CLK = '0')
-- load flip-flops and registers
end if;

Code Template 2
wait until (CLK'EVENT and CLK = '1');
if RESET = '1' then --Synchronous Reset
-- load flip-flops and registers

Sensitivity Lists for Processes

Simulation Semantics
• Sensitivity list specifies only those signals that trigger
  execution of the process.
• Other signals result in inefficient simulation

Synthesis Semantics
• Sensitivity list specifies all signals that are read by the
  process, whether or not events on the signal trigger
  execution of the process

entity T_FF is
  port (RESET, T, CLK: in STD_LOGIC; QOUT: out STD_LOGIC);
end T_FF;
architecture ALG of T_FF is
begin
signal Q: STD_LOGIC;
process (RESET, T, CLK)
begin
if (RESET = '1') then
Q <= '0';
ext if;
end process; -- end LATCH
QOUT <= Q;
end ALG;

T Flip Flop Model
That Simulates
Correctly

Simulation Response of T Flip-flop Models

entity T_FF2 is
  port (RESET, T, CLK: in STD_LOGIC; QOUT: out STD_LOGIC);
end T_FF2;
architecture ALG of T_FF2 is
begin
signal Q: STD_LOGIC;
process (RESET, T, CLK)
begin
if (RESET = '1') then
Q <= '0';
ext if;
elsif (CLK'EVENT and CLK = '1') then
if T = '1' then
Q <= not Q;
ext if;
end if;
end process; -- end LATCH
QOUT <= Q;
end ALG;

T Flip Flop Model
That Simulates
Incorrectly
DO YOU SEE WHY?
Sensitivity Lists for Processes

Circuit synthesized by Synpify for both cases.

Asynchronous state machines

- Asynchronous state machines don’t have a “clock” signal
- Synplicity can’t handle asynchronous state machines
  - Don’t use them.
  - Error: “Found combinational loop”
  - Example from Synplicity documentation:

```vhdl
entity async is
  port (output : inout std_logic ;
  g, d : in std_logic ) ;
end async ;
architecture async1 of async is
begin
  output <= (((((g and d) or (not g)) and output) or d) and
  output);
end async1;
```

- Don’t try this at home!

Modeling Combinational Logic

Logic Operators

- Parenthesis are required between
  - Different logic operators (except NOT)
  - Non-associative operators

<table>
<thead>
<tr>
<th>Legal Expressions</th>
<th>Illegal Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z &lt;= A and B and C;</td>
<td>Z &lt;= A nand B nand C</td>
</tr>
<tr>
<td>Z &lt;= (A nand B) nand C;</td>
<td>Z &lt;= A and B or C</td>
</tr>
<tr>
<td>Z &lt;= (A and B) or C;</td>
<td></td>
</tr>
<tr>
<td>Z &lt;= not A and B;</td>
<td></td>
</tr>
<tr>
<td>Z &lt;= (not A) and B;</td>
<td></td>
</tr>
</tbody>
</table>

PORT mode buffer

- Avoid use of PORT mode buffer in designs to be synthesized
- If you need to read a port signal of mode OUT, use a dummy internal signal as illustrated on the next slide.
entity ALU is port (…
    C: out Std_Logic_Vector (3 downto 0);
…);
architecture BEHAVIORAL of ALU is
signal C_INT: Std_Logic_Vector (3 downto 0);
begins
process (CLK) begin
    if (CLK’event and CLK='1' then
        C_INT <= A + B + C_INT;
    end if; end process;
    C <= C_INT;
end BEHAVIORAL;

entity XOR_SIG is
    port (A, B, C: in STD_LOGIC;
    X, Y: out STD_LOGIC);
end XOR_SIG;
architecture POORLY_WRITTEN_CODE of XOR_SIG is
    signal D: STD_LOGIC;
begins
    SIG: process (A,B,C)
      begin
        D <= A; -- ignored !!
        X <= C xor D;
        D <= B; -- overrides !!
        Y <= C xor D;
    end process;
end POORLY_WRITTEN_CODE;

entity XOR_VAR is
    port (A, B, C: in STD_LOGIC;
    X, Y: out STD_LOGIC);
end XOR_VAR;
architecture BETTER_CODE of XOR_VAR is
begin
    VAR: process (A,B,C)
      variable D: STD_LOGIC;
      begin
        D := A;
        X <= C xor D;
        D := B;
        Y <= C xor D;
    end process;
end BETTER_CODE;

process (R) begin
    if R(3) = '1' then
        Y <= "1000";
    elsif R(3 downto 2) = "01" then
        Y <= "0100";
    elsif R(3 downto 1) = "001" then
        Y <= "0010";
    elsif R(3 downto 0) = "0001" then
        Y <= "0001";
    else
        Y <= "0000";
    end if; end process;

-- Synthesis of Signals and Variables

entity NESTED_IF is
    port (ADDR_A: in std_logic_vector (1 downto 0); -- ADDRESS
    ADDR_B: in std_logic_vector (1 downto 0); -- ADDRESS
    ADDR_C: in std_logic_vector (1 downto 0); -- ADDRESS
    ADDR_D: in std_logic_vector (1 downto 0); -- ADDRESS
    RESET: in std_logic;
    CLK : in std_logic;
    DEC_Q: out std_logic_vector (5 downto 0)); -- Dec.OUTPUT
end NESTED_IF;

entity NESTED_IF is
    port (ADDR_A: in std_logic_vector (1 downto 0); -- ADDRESS
    ADDR_B: in std_logic_vector (1 downto 0); -- ADDRESS
    ADDR_C: in std_logic_vector (1 downto 0); -- ADDRESS
    ADDR_D: in std_logic_vector (1 downto 0); -- ADDRESS
    RESET: in std_logic;
    CLK : in std_logic;
    DEC_Q: out std_logic_vector (5 downto 0)); -- Dec.OUTPUT
end NESTED_IF;
architecture XILINX of NESTED_IF is
begin
NESTED_IF: process (CLK)
begin
if (CLK'event and CLK = '1') then
if (RESET = '0') then
if (ADDR_A = "00") then
DEC_Q(5 downto 4) <= ADDR_D;
DEC_Q(3 downto 2) <= "01";
DEC_Q(1 downto 0) <= "00";
if (ADDR_B = "01") then
DEC_Q(3 downto 2) <= unsigned(ADDR_A) + '1';
DEC_Q(1 downto 0) <= unsigned(ADDR_B) + '1';
if (ADDR_C = "10") then
DEC_Q(5 downto 4) <= unsigned(ADDR_D) + '1';
if (ADDR_D = "11") then
DEC_Q(5 downto 4) <= "00";
end if;  -- End of ADDR_D
else
DEC_Q(5 downto 4) <= ADDR_D;
end if;  -- End of ADDR_C
end if;  End of ADDR_B
else  -- ADDR_A
DEC_Q(5 downto 4) <= ADDR_D;
DEC_Q(3 downto 2) <= ADDR_A;
DEC_Q(1 downto 0) <= unsigned(ADDR_B) + '1';
end if; -- End of ADDR_A
else  -- RESET
DEC_Q <= "000000";
end if; -- End of RESET
end if; -- End of CLK
end process;
end XILINX;

Nested If

architecture XILINX of IF_CASE is
begin
IF_CASE: process (CLK)
begin
if (CLK'event and CLK = '1') then
if (RESET = '0') then
case ADDR_ALL is
when "00011011" =>
DEC_Q(5 downto 4) <="00";
DEC_Q(3 downto 2) <= unsigned(ADDR_A) + '1';
DEC_Q(1 downto 0) <= unsigned(ADDR_B) + '1';
when "000110--" =>
DEC_Q(5 downto 4) <= unsigned(ADDR_D) + '1';
DEC_Q(3 downto 2) <= unsigned(ADDR_A) + '1';
DEC_Q(1 downto 0) <= unsigned(ADDR_B) + '1';
when "0001----" =>
DEC_Q(5 downto 4) <= ADDR_D;
DEC_Q(3 downto 2) <= unsigned(ADDR_A) + '1';
DEC_Q(1 downto 0) <= unsigned(ADDR_B) + '1';
when others =>
DEC_Q(5 downto 4) <= ADDR_D;
DEC_Q(3 downto 2) <= "01";
DEC_Q(1 downto 0) <= "00";
end case;
else  -- RESET
DEC_Q <= "000000";
end if; -- End of ADDR_B
end if; -- End of CLK
end process;
end XILINX;

Decoder Design With CASE

Implementation of CASE Statement Decoder
entity IF_EX is
port (SEL: in STD_LOGIC_VECTOR(1 downto 0);
A,B,C,D: in STD_LOGIC;
MUX_OUT: out STD_LOGIC);
end IF_EX;
architecture BEHAV of IF_EX is
begin
IF_PRO: process (SEL,A,B,C,D)
begin
if (SEL="00") then MUX_OUT <= A;
elsif (SEL="01") then MUX_OUT <= B;
elsif (SEL="10") then MUX_OUT <= C;
elsif (SEL="11") then MUX_OUT <= D;
else MUX_OUT <= '0';
end if;
end process; --END IF_PRO
end BEHAV;

entity CASE_EX is
port (SEL: in STD_LOGIC_VECTOR(1 downto 0);
A,B,C,D: in STD_LOGIC;
MUX_OUT: out STD_LOGIC);
end CASE_EX;
architecture BEHAV of CASE_EX is
begin
CASE_PRO: process (SEL,A,B,C,D)
begin
case SEL is
when "00" => MUX_OUT <= A;
when "01" => MUX_OUT <= B;
when "10" => MUX_OUT <= C;
when "11" => MUX_OUT <= D;
when others => MUX_OUT <= '0';
end case;
end process; end BEHAV;

Summary

- Looked at examples of how a synthesis tool generates RTL hardware from VHDL descriptions
- Rules for determining if a process generates sequential hardware
- Combinational hardware examples

Next time: Xilinx FPGA's
Reading for next time:
- Textbook: pp. 381-392
- Skim Spartan II data sheets (Spartan_II_functional.pdf and Spartan_II_introduction.pdf) on Assignments/reading web page.