CPE 626
Advanced VLSI Design
Lecture 6: VHDL Synthesis

Aleksandar Milenkovic

http://www.ece.uah.edu/~milenka
http://www.ece.uah.edu/~milenka/cpe626-04F/
milenka@ece.uah.edu

Assistant Professor
Electrical and Computer Engineering Dept.
University of Alabama in Huntsville

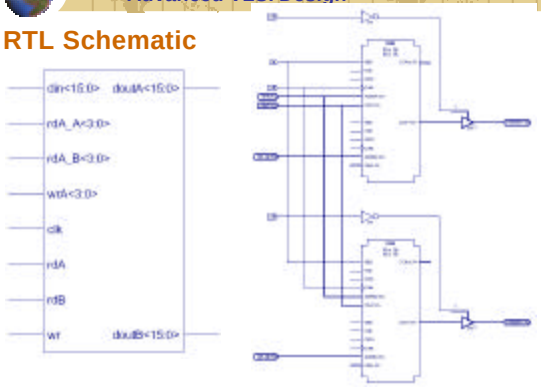
Register File: An Example

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity regfile is
    Port ( clk : in std_logic;
           din : in std_logic_vector(15 downto 0);
           rdA : in std_logic;
           rdB : in std_logic;
           wr : in std_logic;
           rdA_A : in std_logic_vector(3 downto 0);
           rdA_B : in std_logic_vector(3 downto 0);
           wrA : in std_logic_vector(3 downto 0);
           doutA : out std_logic_vector(15 downto 0);
           doutB : out std_logic_vector(15 downto 0));
end regfile;
```


Register File: An Example (cont'd)

```
architecture Behavioral of regfile is

    type memory_array is array(15 downto 0) of std_logic_vector(15 downto 0);
    signal rfile : memory_array := (others => (others => '0'));

begin
          process(clk)
          begin
                    if (clk'event and clk='1') then
                              if (wr = '1') then
                                        rfile(conv_integer(wrA)) <= din;
                              end if;
                    end if;
          end process;

          doutA <= rfile(conv_integer(rdA_A)) when rdA = '1' else (others => 'Z');
          doutB <= rfile(conv_integer(rdA_B)) when rdB = '1' else (others => 'Z');


end Behavioral;
```

RTL Schematic



Testbench: An Example

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY regfile_tbench_vhd_tb IS
END regfile_tbench_vhd_tb;

ARCHITECTURE behavior OF regfile_tbench_vhd_tb IS

    COMPONENT regfile
    PORT(
          clk : IN std_logic;
          din : IN std_logic_vector(15 downto 0);
          rdA : IN std_logic;
          rdB : IN std_logic;
          wr : IN std_logic;
          rdA_A : IN std_logic_vector(3 downto 0);
          rdA_B : IN std_logic_vector(3 downto 0);
          wrA : IN std_logic_vector(3 downto 0);
          doutA : OUT std_logic_vector(15 downto 0);
          doutB : OUT std_logic_vector(15 downto 0)
          );
    END COMPONENT;
```

```
SIGNAL clk :  std_logic := '0';
SIGNAL din :  std_logic_vector(15
downto 0);
SIGNAL rdA :  std_logic;
SIGNAL rdB :  std_logic;
SIGNAL wr :  std_logic;
SIGNAL rdA_A :  std_logic_vector(3
downto 0);
SIGNAL rdA_B :  std_logic_vector(3
downto 0);
SIGNAL wrA :  std_logic_vector(3
downto 0);
SIGNAL doutA :
std_logic_vector(15 downto 0);
SIGNAL doutB :
std_logic_vector(15 downto 0);
```


Testbench: An Example

```
BEGIN

    uut: regfile PORT MAP(
          clk => clk,
          din => din,
          rdA => rdA,
          rdB => rdB,
          wr => wr,
          rdA_A => rdA_A,
          rdA_B => rdA_B,
          wrA => wrA,
          doutA => doutA,
          doutB => doutB
    );


-- *** Test Bench - User Defined Section ***
    clk <= not clk after 100 ns;
```

```
tb : PROCESS
    BEGIN
          din <=
     "1111111111111111";
          wr <= '0' after 0 ns, '1'
     after 200 ns, '0' after 350
     ns;
          wrA <= "0000" after 0 ns,
     "0001" after 500 ns;
          rdA <= '1';
          rdB <= '1';
          rdA_A <= "0000" after 0
     ns, "0001" after 600 ns;
          rdA_B <= "0000" after 0
     ns, "0001" after 600 ns;
       wait; -- will wait forever
    END PROCESS;
-- *** End Test Bench - User
     Defined Section ***


END;
```

Simulation: Post Place & Route VHDL

ISE HDL Options

RAM Style (FPGA only)
Specifies the way in which the macrogenerator implements the RAM macros.
Note This property is available only when the RAM Extraction property is set to True (checkbox is checked).
You can select one of the following three options:
Auto
XST determines the best implementation for each macro.
Distributed
Implements RAM as Distributed RAM.
Block
Implements RAM as Block RAM.
By default, this property is set to Auto.
ROM Extraction (FPGA only)
Specifies whether or not to use a ROM macro inference. By default, this property is set to True (checkbox is checked), and ROM inference is enabled.
ROM Style (FPGA only)
Specifies the way in which the macrogenerator implements the ROM macros.
Note This property is available only when the ROM Extraction property is set to True (checkbox is checked).
You can select one of the following three options:
Auto
XST determines the best implementation for each macro.
Distributed
Implements RAM as Distributed RAM.
Block
Implements RAM as Block RAM.
If the RAM Extraction property is set to False (checkbox is blank), then the RAM Style property is disabled and is not written to the command line.

Design Flow Overview

Schematic Capture

- RTL code
- Carefully Select Design Hierarchy
- Architectural Wizards (Clocking, RocketIO)
- Core Generator

Core Generator

- Design tool that delivers parameterized cores optimized for Xilinx FPGAs
  - E.g., adders, multipliers, filters, FIFOs, memories …
- Core Generator outputs
  - *.EDN file => EDIF (Electronic Data Interchange Format) netlist file; it includes information required to implement the module in a Xilinx FPGA
  - *.VHO => VHDL template file; used as a model for instantiating a module in a VHDL design
  - *.VHD => VHDL wrapper file; provides support for functional simulation

Functional Simulation

- Verify the syntax and functionality
  - Separate simulation for each module => easier to debug
  - When each module behaves as expected, create a test bench for entire design

Coding for Synthesis

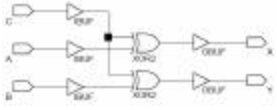
Avoid Ports Declared as Buffers

```
Entity alu is
port(
  A : in STD_LOGIC_VECTOR(3 downto 0);
  B : in STD_LOGIC_VECTOR(3 downto 0);
  CLK : in STD_LOGIC;
  C : out STD_LOGIC_VECTOR(3 downto 0)
);
end alu;
architecture BEHAVIORAL of alu is
-- dummy signal
  signal C_INT : STD_LOGIC_VECTOR(3 downto 0);
begin
  C <= C_INT;
  process begin
    if (CLK'event and CLK='1') then
      C_INT < =UNSIGNED(A) + UNSIGNED(B) +
          UNSIGNED(C_INT);
    end if;
  end process;
end BEHAVIORAL;
```

```
Entity alu is
port(
  A : in STD_LOGIC_VECTOR(3 downto 0);
  B : in STD_LOGIC_VECTOR(3 downto 0);
  CLK : in STD_LOGIC;
  C : buffer STD_LOGIC_VECTOR(3 downto 0) );
end alu;
architecture BEHAVIORAL of alu is
begin
  process begin
    if (CLK'event and CLK='1') then
      C <= UNSIGNED(A) + UNSIGNED(B)
   UNSIGNED(C);
    end if;
  end process;
end BEHAVIORAL;
```

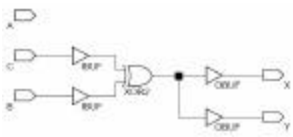

Signals vs. Variables for Synthesis

```
-- XOR_VAR.VHD
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity xor_var is
port (
  A, B, C: in STD_LOGIC;
  X, Y: out STD_LOGIC
);
end xor_var;
architecture VAR_ARCH 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 VAR_ARCH;
```



Signals vs. Variables for Synthesis

```
-- XOR_SIG.VHD
Library IEEE;
use IEEE.std_logic_1164.all;
entity xor_sig is
port (
  A, B, C: in STD_LOGIC;
  X, Y: out STD_LOGIC
);
end xor_sig;
architecture SIG_ARCH of xor_sig is
  signal D: STD_LOGIC;
begin
  SIG:process (A,B,C)
  begin
    D <= A; -- ignored !!
    X <= C xor D;
    D <= B; -- overrides !!
    Y <= C xor D;
  end process;
end SIG_ARCH;
```



Guidelines for synthesis

- VHDL/Verilog are not originally planned as languages for synthesis
  - Many HDL simulation constructs are not supported in synthesis
- Omit “wait for XX” statements
- Omit delay statements “ … after XX;”
- Omit initial values
- Order and grouping of arithmetic statement can influence synthesis

If Statement vs. Case Statement

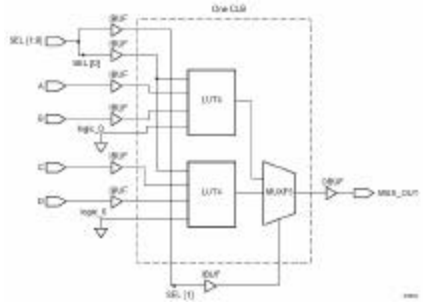
- If statement generally produces priority-encoded logic
  - can contain a set of different expressions
- Case statement generally creates balanced logic
  - evaluated against a common controlling expression
- In general, use the Case statement for complex decoding and use the If statement for speed critical paths

MUX 4-1 Synthesis

```
-- IF_EX.VHD
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
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;
```

```
-- CASE_EX.VHD
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
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 CASE_PRO
end BEHAV;
```


MUX 4-1 Implementation



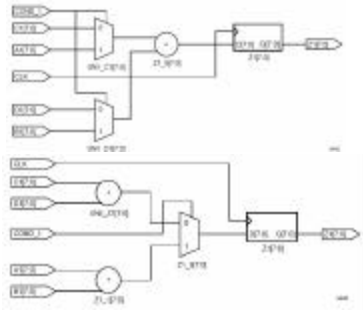
Resource Sharing

- An optimization that uses a single functional block (adder, multiplier, shifter, …) to implement several HDL operators
- Pros: less gates, simpler routing
- Cons: adds extra logic levels => increase delay (avoid sharing on the critical path)

Resource Sharing: VHDL Example

```
-- RES_SHARING.VHD
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
entity res_sharing is
port (
  A1,B1,C1,D1 : in STD_LOGIC_VECTOR (7 downto 0);
  COND_1 : in STD_LOGIC;
  Z1 : out STD_LOGIC_VECTOR (7 downto 0));
end res_sharing;
architecture BEHAV of res_sharing is
begin
  P1: process (A1,B1,C1,D1,COND_1)
  begin
    if (COND_1='1') then
      Z1 <= A1 + B1;
    else
      Z1 <= C1 + D1;
    end if;
   end process; -- end P1
end BEHAV;
```


Implementation: W/WO Resource Sharing



Enabled

Disabled