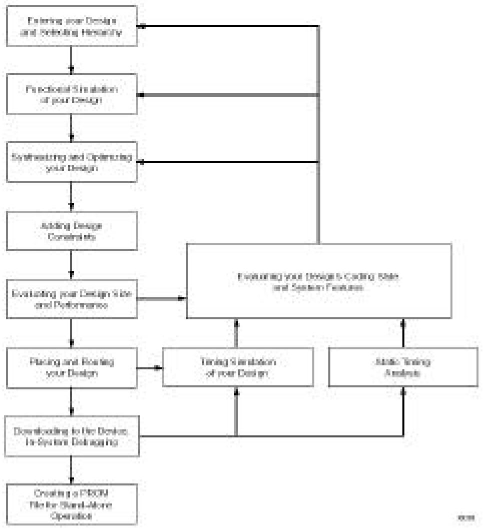
CPE 626
Advanced VLSI Design
Lecture 7: VHDL Synthesis

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

Design Flow Overview


Entering your Design and Selecting Hierarchy
Functional Simulation of your Design
Synthesizing and Optimizing your Design
Adding Design Constraints
Evaluating your Design Size and Performance
Evaluating your Design's Coding Style and System Features
Placing and Routing your Design
Timing Simulation of your Design
Static Timing Analysis
Downloading to the Device, In-System Debugging
Creating a PROM File for Stand-Alone Operation

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

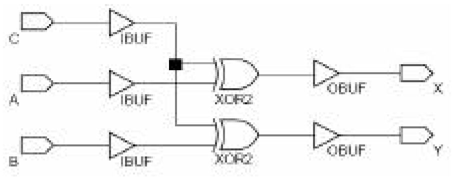
```
Entity alu is
port(
  A : in STD_LOGIC_VECTOR(3 downto 0);
  B : in STD_LOGIC_VECTOR(3 downto 0);
  CLK : in STD_LOGIC;
  C : out STD_LOGIC_VECTOR(3 downto 0)
);
end alu;
architecture BEHAVIORAL of alu is
-- dummy signal
  signal C_INT : STD_LOGIC_VECTOR(3 downto 0);
begin
  C <= C_INT;
  process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

Ö

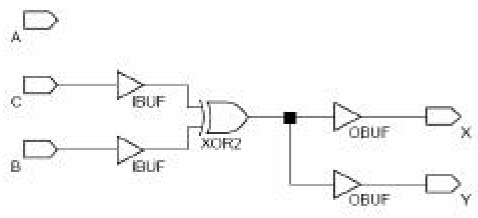
Signals vs. Variables for Synthesis

```
-- 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

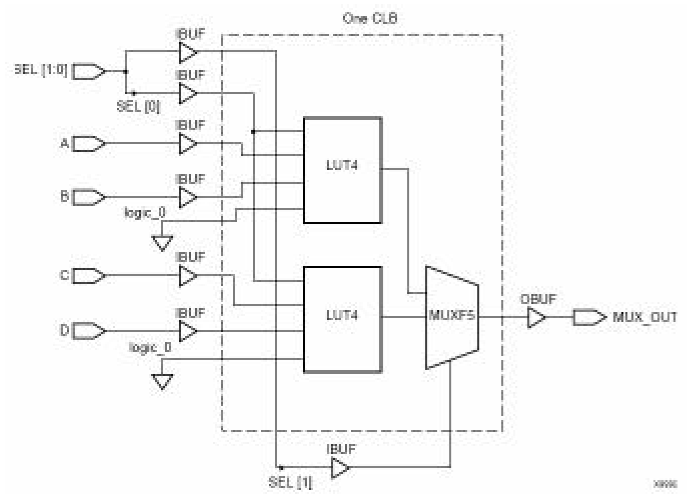
- If statement generally produces priority-encoded logic
  - can contain a set of different expressions
- Case statement generally creates balanced logic
  - evaluated against a common controlling expression
- In general, use the Case statement for complex decoding and use the If statement for speed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



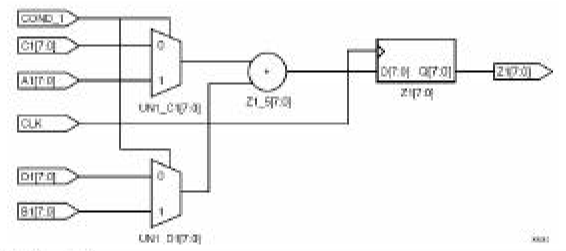
Resource Sharing

- 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)

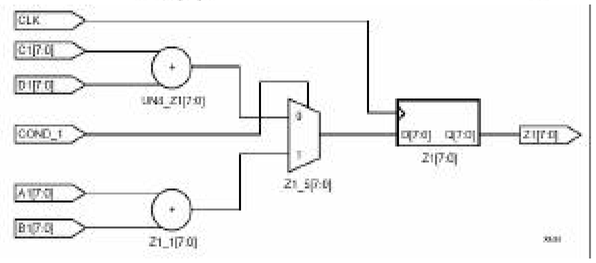
Resource Sharing: VHDL Example

```
-- 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

Registers: Asynchronous Preset/Set

```
-- FF_EXAMPLE.VHD
-- Example of Implementing Registers
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity ff_example is
port ( RESET, SET, CLOCK, ENABLE: in STD_LOGIC;
 D_IN: in STD_LOGIC_VECTOR (7 downto 0);
 A_Q_OUT: out STD_LOGIC_VECTOR (7 downto 0);
 B_Q_OUT: out STD_LOGIC_VECTOR (7 downto 0);
 C_Q_OUT: out STD_LOGIC_VECTOR (7 downto 0);
 D_Q_OUT: out STD_LOGIC_VECTOR (7 downto 0);
 E_Q_OUT: out STD_LOGIC_VECTOR (7 downto 0));
end ff_example;
architecture BEHAV of ff_example is
begin
 -- D flip-flop
 FF: process (CLOCK)
 begin
  if (CLOCK’event and CLOCK=’1’) then
    A_Q_OUT <= D_IN;
  end if;
 end process; -- End FF
```

```
 -- Flip-flop with asynchronous reset
 FF_ASYNC_RESET: process (RESET, CLOCK)
 begin
   if (RESET = ’1’) then
     B_Q_OUT <= “00000000”;
   elsif (CLOCK'event and CLOCK='1') then
     B_Q_OUT <= D_IN;
end if;
-- Flip-flop with asynchronous set
 FF_ASYNC_SET: process (SET, CLOCK)
 begin
   if (SET = '1') then
     C_Q_OUT <= “11111111”;
   elsif (CLOCK'event and CLOCK = '1') then
     C_Q_OUT <= D_IN;
   end if;
 end process; -- End FF_ASYNC_SET
```

Registers: Asynchronous Preset/Set

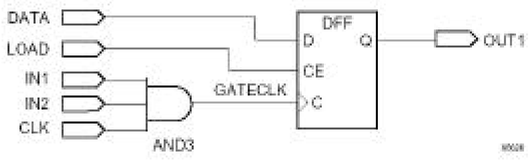
```
 -- Flip-flop with asynchronous reset
-- and clock enable
FF_CLOCK_ENABLE: process (ENABLE, RESET,
    CLOCK)
begin
  if (RESET = ’1’) then
    D_Q_OUT <= “00000000”;
  elsif (CLOCK'event and CLOCK='1') then
    if (ENABLE = '1') then
      D_Q_OUT <= D_IN;
    end if;
  end if;
end process; -- End FF_CLOCK_ENABLE
```

```
-- Flip-flop with asynchronous reset
-- asynchronous set and clock enable
FF_ASR_CLOCK_ENABLE: process (ENABLE,
    RESET, SET, CLOCK)
begin
  if (RESET = '1') then
    E_Q_OUT <= "00000000";
  elsif (SET = '1') then
    E_Q_OUT <= "11111111";
  elsif (CLOCK'event and CLOCK='1') then
    if (ENABLE = '1') then
       E_Q_OUT <= D_IN;
    end if;
  end if;
end process; -- End FF_ASR_CLOCK_ENABLE
end BEHAV;
```

Use Clock_Enable instead gated Clock

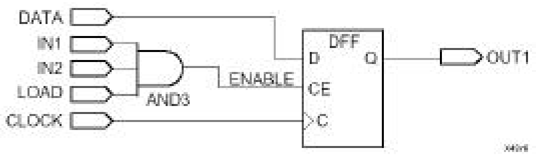
```
-- Better implementation is to use --
-- clock enable rather than gated clock --
-------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity gate_clock is
  port (IN1,IN2,DATA,CLK,LOAD: in
    STD_LOGIC;
       OUT1: out STD_LOGIC);
end gate_clock;
architecture BEHAVIORAL of gate_clock is
  signal GATECLK: STD_LOGIC;
begin
  GATECLK <= (IN1 and IN2 and CLK);
  GATE_PR: process (GATECLK,DATA,LOAD)
  begin
    if (GATECLK'event and GATECLK='1')
      if (LOAD = '1') then
        OUT1 <= DATA;
      end if;
   end if;
  end process; -- End GATE_PR
end BEHAVIORAL;
```

- This implementation introduces
  - Glitches
  - Clock delays
  - Clock skew



Use Clock Enable Instead Gated Clocks

```
-- CLOCK_ENABLE.VHD
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity clock_enable is
  port (IN1,IN2,DATA,CLOCK,LOAD: in STD_LOGIC;
       DOUT: out STD_LOGIC);
end clock_enable;
architecture BEHAV of clock_enable is
  signal ENABLE: STD_LOGIC;
begin
  ENABLE <= IN1 and IN2 and LOAD;
  EN_PR: process (ENABLE,DATA,CLOCK)
  begin
    if (CLOCK'event and CLOCK='1') the
      if (ENABLE = '1') then
         DOUT <= DATA;
      end if;
   end if;
  end process; -- End EN_PR
end BEHAV;
```



HDL Coding Techniques for Common Building Blocks

Multiplication in FPGAs

- http://www.andraka.com/multipli.htm

Multiplier

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity mult is
    port(
          A : in std_logic_vector(17 downto 0);
          B : in std_logic_vector(17 downto 0);
          RES : out std_logic_vector(35 downto 0)
    );
end mult;
```

```
architecture archi of mult is
begin
    RES <= A * B;
end archi;
```

- **Device utilization summary:**

  Selected Device : 3s50tq144-5

Number of Slices:	**19**	**out of**	**768**	**2%**
Number of 4 input LUTs:	**36**	**out of**	**1536**	**2%**
Number of bonded IOBs:	**72**	**out of**	**97**	**74%**
Number of MULT18X18s:	**3**	**out of**	**4**	**75%**

- **Delay = 14.975ns (Levels of Logic = 25)**

Inferring Pipelined Multiplier

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

entity multpipe is
  generic(
    A_port_size : integer := 18;
    B_port_size : integer := 18
  );
  port(
    clk : in std_logic;
    A : in unsigned (A_port_size-1 downto 0);
    B : in unsigned (B_port_size-1 downto 0);
    MULT : out unsigned ( (A_port_size+B_port_size-1) downto 0));
end multpipe;
```

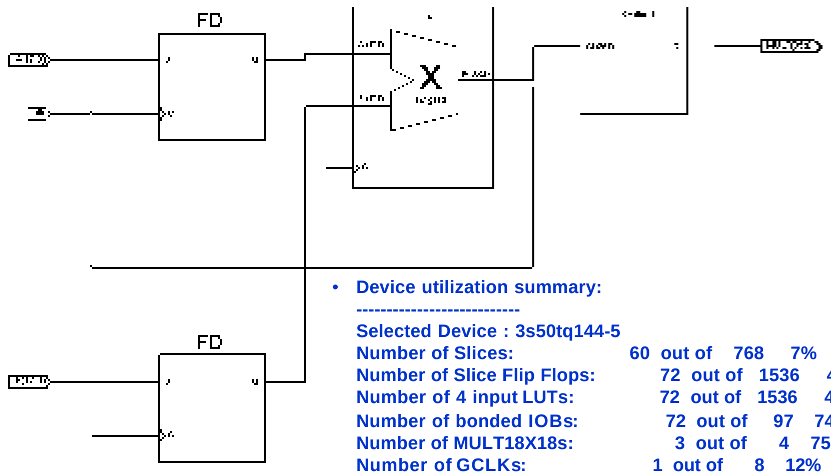
Inferring Pipelined Multiplier (cont’d)

```
architecture Behavioral of multpipe is
 signal a_in, b_in : unsigned (A_port_size-1 downto 0);
   signal mult_res : unsigned ( (A_port_size+B_port_size-1) downto 0);
   signal pipe_1,
          pipe_2,
          pipe_3 : unsigned ((A_port_size+B_port_size-1) downto 0);
begin
   mult_res <= a_in * b_in;

   process (clk)
    begin
     if (clk'event and clk='1') then
            a_in <= A; b_in <= B;
            pipe_1 <= mult_res;
            pipe_2 <= pipe_1;
            pipe_3 <= pipe_2;
            MULT <= pipe_3;
     end if;
    end process;
end Behavioral;
```

=> Four pipeline stages

Pipelined Multiplier: RTL Schematic



- Device utilization summary:

  Selected Device : 3s50tq144-5

  Number of Slices: 60 out of 768 7%

  Number of Slice Flip Flops: 72 out of 1536 4%

  Number of 4 input LUTs: 72 out of 1536 4%

  Number of bonded IOBs: 72 out of 97 74%

  Number of MULT18X18s: 3 out of 4 75%

  Number of GCLKs: 1 out of 8 12%
- Delay: 9.824ns (Levels of Logic = 23)

Inferring PM: Alternative Description #1

```
architecture beh of mult is
  signal a_in, b_in : unsigned (A_port_size-1 downto 0);
  signal mult_res : unsigned ((A_port_size+B_port_size-1) downto 0);
  signal pipe_2,
         pipe_3 : unsigned ((A_port_size+B_port_size-1) downto 0);
begin
  process (clk)
  begin
    if (clk'event and clk='1') then
      a_in <= A; b_in <= B;
      mult_res <= a_in * b_in;
      pipe_2 <= mult_res;
      pipe_3 <= pipe_2;
      MULT <= pipe_3;
   end if;
  end process;
end beh;
```

Inferring PM: Alternative Description #2

```
architecture beh of mult is
  signal a_in, b_in : unsigned (A_port_size-1 downto 0);
  signal mult_res : unsigned ((A_port_size+B_port_size-1) downto 0);
  type pipe_reg_type is array (2 downto 0) of unsigned
     ((A_port_size+B_port_size-1) downto 0);
  signal pipe_regs : pipe_reg_type;
begin
  mult_res <= a_in * b_in;
  process (clk)
  begin
    if (clk'event and clk='1') then
      a_in <= A; b_in <= B;
      pipe_regs <= mult_res & pipe_regs(2 downto 1);
      MULT <= pipe_regs(0);
    end if;
  end process;
end beh;
```

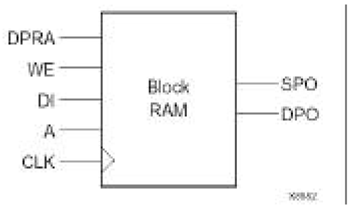
RAMs/ROMs

- XST offers an automatic RAM recognition
  - Keep you HDL code technology independent
- XST can infer distributed or Block RAM
- Parameters
  - Synchronous write, Write enable, RAM enable, Asynchronous or synchronous read, Reset of the data output latches, Data output reset, Single, dual or multiple-port read, Single-port write

Dual-port Block RAM

```
-- Only XST supports RAM inference
-- Infers Dual Port Block Ram

 entity dpblockram is
 port (clk  : in std_logic;
   we   : in std_logic;
   a    : in std_logic_vector(4 downto 0);
   dpra : in std_logic_vector(4 downto 0);
   di   : in std_logic_vector(3 downto 0);
   spo  : out std_logic_vector(3 downto 0);
   dpo  : out std_logic_vector(3 downto 0));
 end dpblockram;
```



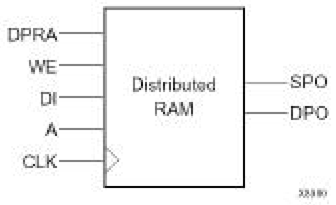
Dual-port Block RAM

```
architecture syn of dpblockram is
 type ram_type is array (31 downto 0) of std_logic_vector (3 downto 0);
 signal RAM : ram_type;
 signal read_a : std_logic_vector(4 downto 0);
 signal read_dpra : std_logic_vector(4 downto 0);
 begin
 process (clk)
 begin
   if (clk'event and clk = '1') then
        if (we = '1') then
                 RAM(conv_integer(a)) <= di;
        end if;
        read_a <= a;
        read_dpra <= dpra;
   end if;
 end process;

 spo <= RAM(conv_integer(read_a));
 dpo <= RAM(conv_integer(read_dpra));
end syn;
```

Dual-port Distributed RAM

```
entity dpdistram is
port (clk  : in std_logic;
  we   : in std_logic;
  a    : in std_logic_vector(4 downto 0);
  dpra : in std_logic_vector(4 downto 0);
  di   : in std_logic_vector(3 downto 0);
  spo  : out std_logic_vector(3 downto 0);
  dpo  : out std_logic_vector(3 downto 0));
end dpdistram;
```



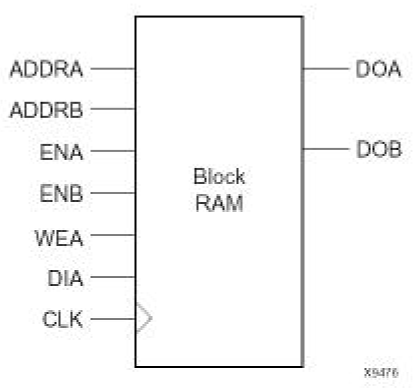
Dual-port Distributed RAM

```
architecture syn of dpdistram is
type ram_type is array (31 downto 0) of std_logic_vector (3 downto 0);
signal RAM : ram_type;

begin
process (clk)
begin
  if (clk'event and clk = '1') then
       if (we = '1') then
               RAM(conv_integer(a)) <= di;
       end if;
  end if;
end process;
spo <= RAM(conv_integer(a));
dpo <= RAM(conv_integer(dpra));
end syn;
```

Dual-port RAM with Enable on Each Port

Write VHDL description.


ADDRA
ADDRB
ENA
ENB
WEA
DIA
CLK
Block RAM
DOA
DOB
X9476