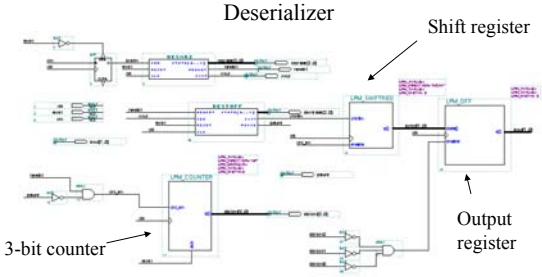
Advanced VHDL

- In this class, have used VHDL primarily finite state machines (random logic + state registers)
- Have used schematic capture + LPMs for datapaths
  - Schematics are nice because a picture can convey the design structure
  - But schematic files are non-portable, file format is specific to Altera Maxplus
- Can use VHDL for entire design including datapath, and connections
  - More portable than schematics, easier to modify
  - Many tools only accept textual representations of designs (Verilog, VHDL, other hardware description languages).

Deserializer



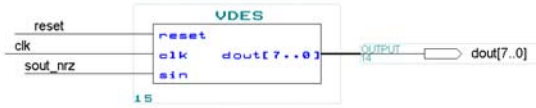
Goal: Replace Deserializer with a single VHDL file, will include 'desnrz', 'destuff' as components.

Entity Declaration

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

Provides arithmetic functionality needed by counter

```
entity vdes is
  port (  reset,clk,sin :      in std_logic;
          dout: out std_logic_vector(7 downto 0)
       );
 end vdes;
```



Architecture – Component Declarations

desnrz, *destuff* are described by separate VHDL files, will be used as components within 'vdes.vhd'.

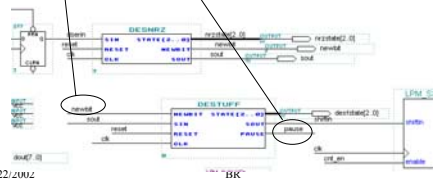
Component declarations are basically just copies of entity definitions.

```
architecture a of vdes is
-- component declarations
 component destuff
  port (  newbit,sin:  in std_logic;
          reset,clk :      in std_logic;
          state:  out std_logic_vector(2 downto 0);
          sout,pause: out std_logic );
 end component;
 component desnrz
  port (  sin:  in std_logic;
         reset,clk :      in std_logic;
          state:  out std_logic_vector(2 downto 0);
         newbit,sout: out std_logic
       );
 end component;
```


Signals

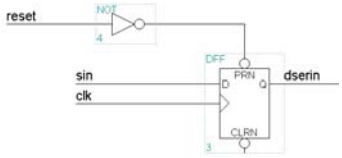
Need to declare signals that are used to tie components/processes together

```
-- signals
  signal dserin : std_logic;
  signal dshift : std_logic_vector(7 downto 0);
  signal  nrzstate: std_logic_vector(2 downto 0);
  signal newbit, sout: std_logic;
  signal desfstate: std_logic_vector(2 downto 0);
  signal shiftin, pause: std_logic;
  signal dbitcnt: std_logic_vector(2 downto 0);
 signal cnt_en : std_logic;
```



Input DFF for *sin*

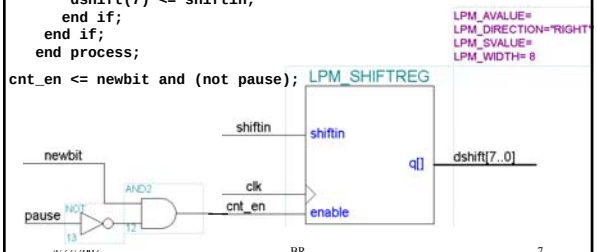
```
-- input DFF
  process(reset,clk)
  begin
    if (reset = '1') then
      dserin <= '1';
    elsif (clk'event and clk = '1') then
      dserin <= sin;
    end if;
 end process;
```



Shift Register

```
-- shift register
  process(clk)
  begin
    if (clk'event and clk = '1') then
      if (cnt_en = '1') then
       dshift(6 downto 0) <= dshift(7 downto 1);
       dshift(7) <= shiftin;
      end if;
    end if;
   end process;
cnt_en <= newbit and (not pause);
```

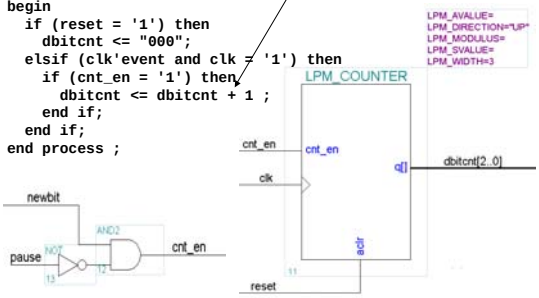
Note that this is a right shift, and that *shiftin* comes into MSB.



3-bit Counter

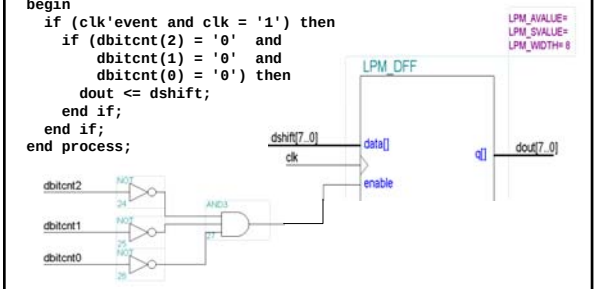
```
-- 3 bit counter
  process(reset,clk)
  begin
    if (reset = '1') then
      dbitcnt <= "000";
    elsif (clk'event and clk = '1') then
      if (cnt_en = '1') then
        dbitcnt <= dbitcnt + 1 ;
      end if;
    end if;
  end process ;
```

Increment by 1



Output Register

```
-- output register
  process(reset,clk)
  begin
    if (clk'event and clk = '1') then
      if (dbitcnt(2) = '0'  and
          dbitcnt(1) = '0'  and
          dbitcnt(0) = '0') then
        dout <= dshift;
      end if;
    end if;
  end process;
```



desnrz Component Instantiation, Connection

```
-- desnrz
u1:desnrz
    port map( sin => dserin, clk => clk, reset => reset,
              state => nrzstate, newbit => newbit,
              sout => sout);
```

Component type

Component Pin name

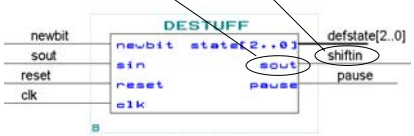
Signal(net) name

Component name (must be unique in VHDL file)

destuff Component Instantiation, Connection

```
-- destuff
u2:destuff
     port map (clk => clk, reset => reset,
               newbit => newbit, sin => sout,
               state => desfstate,
               sout => shiftin,
               pause => pause);
```

Mapping of component pins to signals (nets)



Other Comments

- The *des* schematic used LPM components such as LPM_DFF, LPM_SHIFTREG, LPM_COUNTER
  - These could have been called as 'components' from VHDL instead of describing their functionality via VHDL processes
  - Their functionality is so simple, was easier to simply use VHDL processes
- For more complex blocks such as SRAMs, Multipliers, etc, that correspond directly to FPGA resources, would have used the LPM versions
  - Instantiate the LPMs directly in the VHDL code same as *desnrz*, *destuff*
  - Component declarations are predeclared, need to include the following in the VHDL file:

```
library lpm;
use lpm.lpm_components.all;
```