Introduction to VHDL for Design and Modeling

Integrated Microelectronics Engineering, Module 1
November 4, 1998

Part 1: VHDL for Design

Ed Casas

VHDL

- a *Very complicated* Hardware Description Language
- luckily, only a small subset is needed for design
- VHDL is used for design (covered this morning) and simulation (covered this afternoon)

Outline

- Introduction (AND gate)
- Vectors and Buses
- Selected Assignment (3-to-8 decoder)
- Conditional Assignment (4-to-3 priority encoder)
- Sequential Circuits (flip-flop)
- State Machines (switch debouncer)
- `Signed` and `Unsigned` Types (3-bit counter)

- Components, Packages and Libraries
- Using Components
- Type Declarations
- Tri-State Buses

First VHDL Example

```
-- An AND gate

library ieee ;
use ieee.std_logic_1164.all;

entity example1 is port (
   a, b: in std_logic ;
   c: out std_logic ) ;
end example1 ;

architecture rtl of example1 is
begin
      c <= a and b ;
end rtl ;
```

VHDL Syntax

- not case sensitive
- comments begin with --
- statements end with ;
- signal/entity names: letter followed by letters, digits, _
- details on library and use statements later

Entities and Architectures

- the entity names the device
- the entity declares the input and output signals
- the architecture describes what the device does
- every statement in the architecture “executes” *concurrently*

Schematic for Example 1

- not surprisingly, synthesizing example1 creates:

Exercise (Expressions)

Exercise: Write the VHDL description of a half-adder, a circuit that computes the sum, `sum`, and carry, `carry`, of two one-bit numbers, a and b.

Vectors

- one-dimensional arrays used to model buses
- usually declared with decreasing indices:

```
a : std_logic_vector (3 downto 0) ;
```

- constants enclosed in double quotes:

```
a <= "0010" ;
```

Vector Operations

- can take "slices" (e.g. `x(3 downto 2)`)
- can concatenate (e.g. `a & b`)
- logic operators (e.g. `a and b`) apply bit-by-bit

Exercise (Vectors)

Exercise: Write a VHDL expression that shifts `x`, an 8-bit `std_logic_vector` declared as `x : std_logic_vector (7 downto 0) ;`, left by one bit and sets the least-significant bit to zero.

Selected Assignment

- models operation of multiplexer
- one value selected by controlling expression
- can implement arbitrary truth table
- always use an `others` clause
- we can declare local signals in architectures

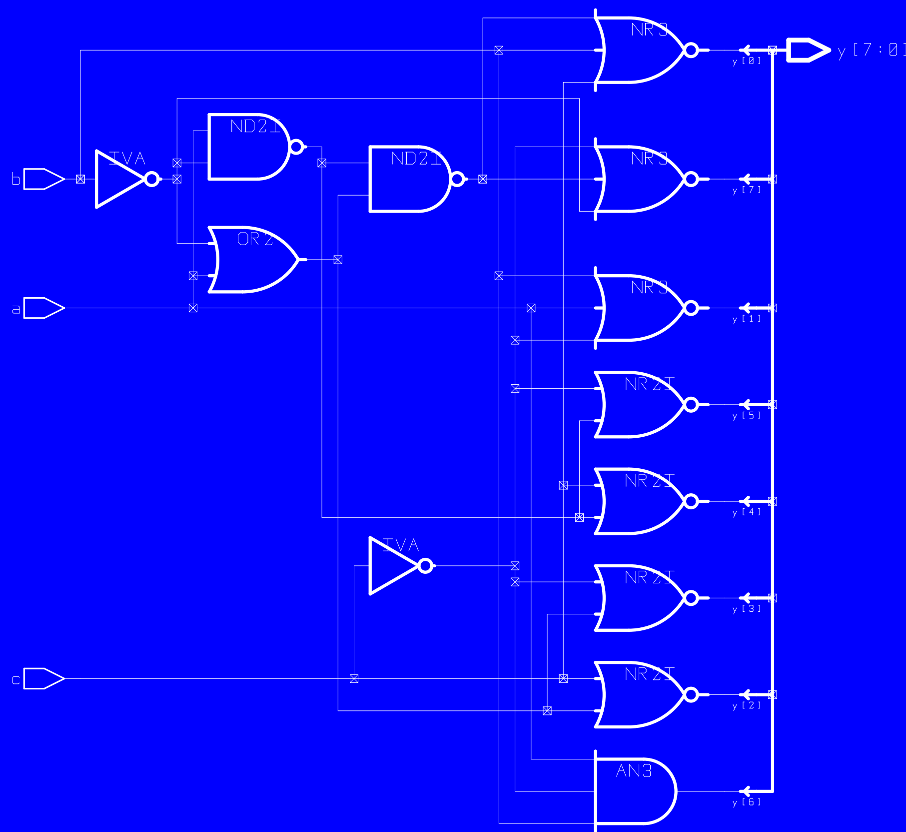
VHDL for 3-to-8 Decoder

```
-- 3-to-8 decoder

library ieee ;
use ieee.std_logic_1164.all;

entity decoder is
  port (
  a, b, c : in std_logic ;
  y : out  std_logic_vector (7 downto 0) ) ;
end decoder ;
```

VHDL for 3-to-8 Decoder (Architecture)

```
architecture rtl of decoder is
  signal abc : std_logic_vector (2 downto 0) ;
begin
  abc <= a & b & c ;

  with abc select y <=
     "00000001" when "000",
     "00000010" when "001",
     "00000100" when "010",
     "00001000" when "011",
     "00010000" when "100",
     "00100000" when "101",
     "01000000" when "110",
     "10000000" when others ;
end rtl ;
```

Schematic of 3-to-8 Decoder



Conditional Assignment

- models `if/then/else`, but is concurrent
- expressions tested in order
- only value for first true condition is assigned

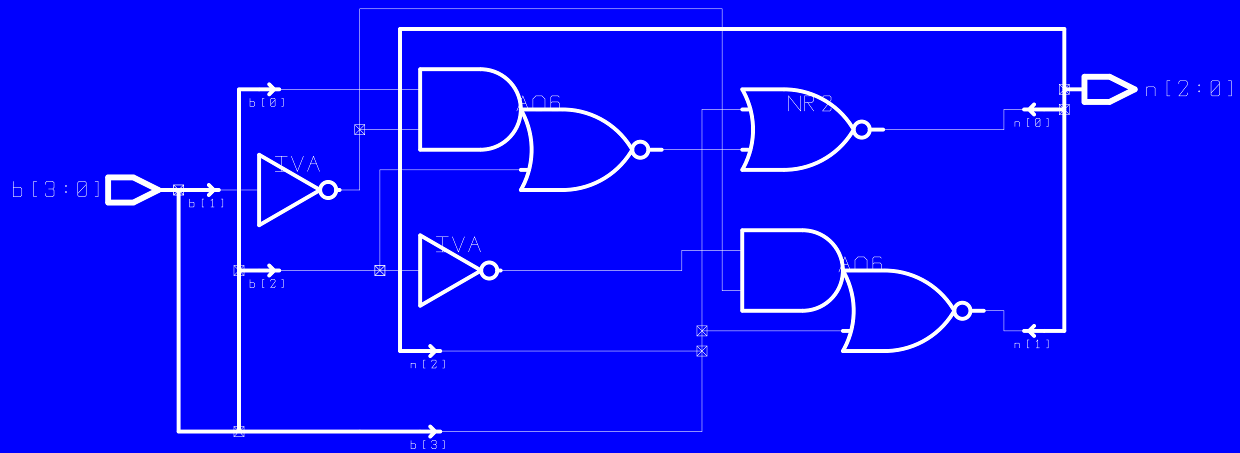
VHDL for 4-to-3 Encoder

```
-- 4-to-3 encoder

library ieee ;
use ieee.std_logic_1164.all ;

entity encoder is port (
   b : in std_logic_vector (3 downto 0) ;
   n : out std_logic_vector (2 downto 0) ) ;
end encoder ;
```

VHDL for 4-to-3 Encoder (Architecture)

```
architecture rtl of encoder is
begin
   n <=
      "100" when b(3) = '1' else
      "011" when b(2) = '1' else
      "010" when b(1) = '1' else
      "001" when b(0) = '1' else
      "000" ;
end rtl ;
```

4-to-3 Encoder



Exercise (Selected Assignment)

Exercise: If we had used a selected assignment statement, how many lines would have been required in the selected assignment?

Sequential Circuits

- the `process` statement can generate flip-flops or registers
- details of `process` covered later

VHDL for D Flip-Flop

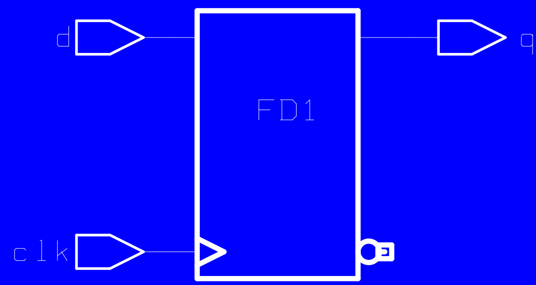
```
-- D Flip-Flop

library ieee ;
use ieee.std_logic_1164.all;

entity dff is
   port (
   d, clk : in std_logic ;
   q : out std_logic ) ;
end dff ;
```

VHDL for D Flip-Flop (Architecture)

```
architecture rtl of dff is
begin
process(clk)
begin
   if clk'event and clk = '1' then
      q <= d ;
   end if ;
end process ;
end rtl ;
```

Schematic of dff

- the synthesized result:



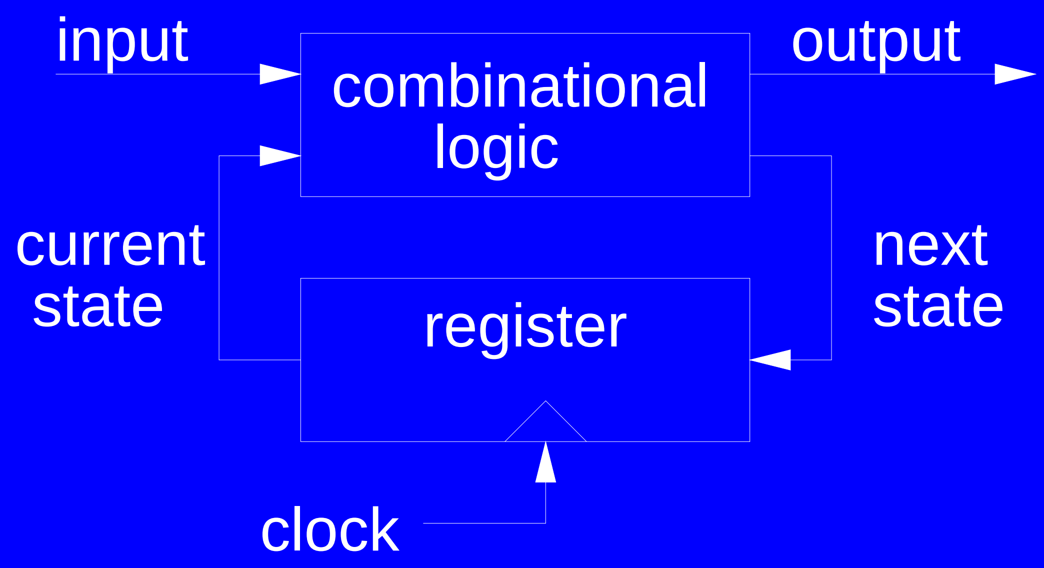
Exercise (Sequential Circuits)

Exercise: What would we get if we replaced d and q with signals of type `std_logic_vector`?

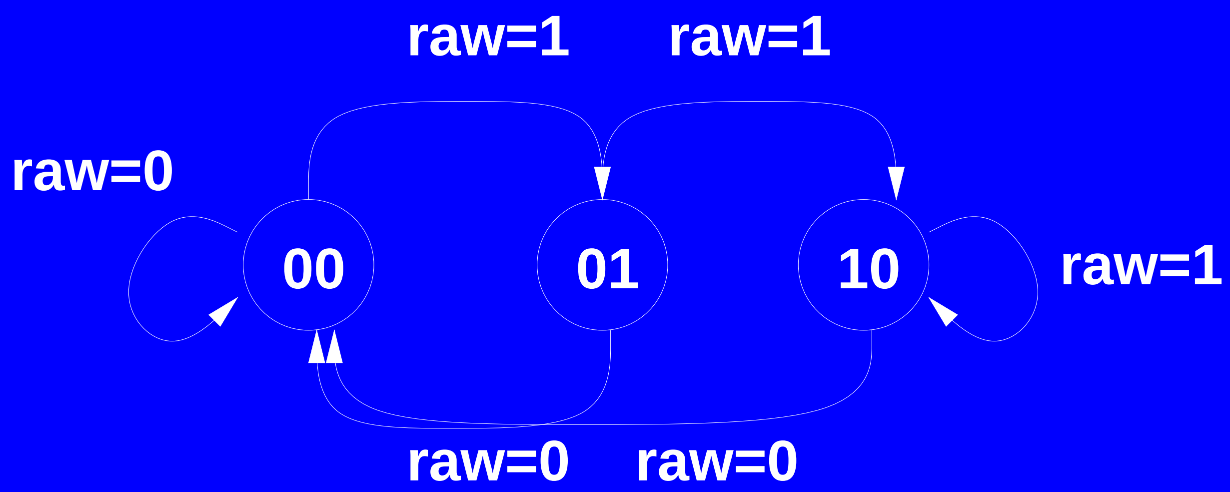
State Machines

- use combinational logic to compute next state and outputs
- use registers to hold current state

Finite State Machine



Switch Debouncer State Diagram



state	output
00	0
01	0
10	1

VHDL for Switch Debouncer

```
-- Switch Debouncer

library ieee ;
use ieee.std_logic_1164.all ;

entity debounce is
   port (
      raw : in std_logic ;
      clean : out std_logic ;
      clk : in std_logic ) ;
end debounce ;
```

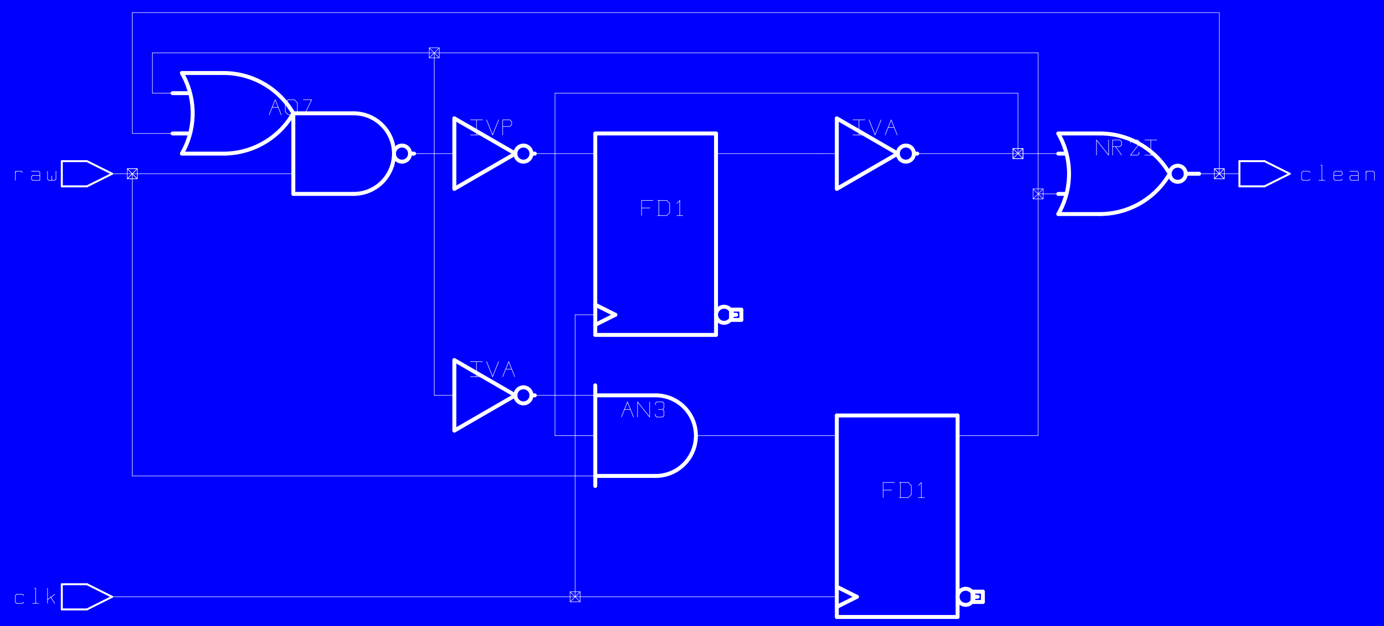
VHDL for Switch Debouncer (Architecture)

```
architecture rtl of debounce is
   signal currents, nexts :
       std_logic_vector (1 downto 0) ;
begin

   -- combinational logic for next state
   nexts <=
      "00" when raw = '0' else
      "01" when currents = "00" else
      "10" ;

   -- combinational logic for output
   clean <= '1' when currents = "10" else '0' ;
```

VHDL for Switch Debouncer (process)

```
    -- sequential logic
    process(clk)
    begin
       if clk'event and clk = '1' then
          currents <= nexts ;
       end if ;
    end process ;

 end rtl ;
```

Schematic of Debouncer



Exercise (State Machine)

Exercise: Identify the components in the schematic that were created (“*instantiated*”) by different parts of the VHDL code.

Arithmetic Types

- allow us to treat logic values as numbers
- arithmetic and comparison operators available
- multiply and divide may not synthesize

VHDL for 3-bit Counter

```
-- 3-bit Counter

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

entity counter is
   port (
      count_out : out unsigned (2 downto 0) ;
      clk : in std_logic ) ;
end counter ;
```

VHDL for Up/Down Counter (Architecture)

```
architecture rtl of counter is
   signal count, nextcount : unsigned (2 downto 0) ;
begin
   nextcount <= count + 1 ;

   process(clk)
   begin
      if clk'event and clk='1' then
         count <= nextcount ;
      end if ;
   end process ;

   count_out <= count ;
end rtl ;
```

Arithmetic Types (ctd)

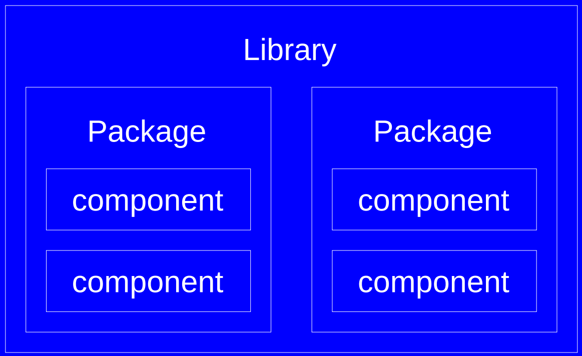
- can use arithmetic types to implement a counter as a state machine
- can't read port signals of type `out`

Exercise (Arithmetic Types)

Exercise: Write the architecture for a 16-bit adder with two `signed` inputs, a and b and a `signed` output c.

Components

- allows design re-use
- like an entity, a component defines interface, not functionality
- definitions usually saved in packages (files)
- packages are stored in libraries (directories)



VHDL for Component Declaration

- package name is `flipflops`
- declares the `rs` component:

```
package flipflops is
   component rs
      port ( r, s : in bit ; q : out bit ) ;
   end component ;
end flipflops ;
```

- compiling this file creates the package

Using Packages and Libraries

- `library` and `use` statements make contents of packages available
- e.g. to access the `DSP` package in the `ALTERA` library:

```
library altera ;
use altera.dsp.all ;
```

Using Components

- a component “instantiation” statement:
  - places a copy of the component in the design
  - is a concurrent statement
  - describes how the component connects to other signals
- is “structural” design

VHDL for XOR2 Component Declaration

- the `xor2` component is described in a package:

```
 -- define an xor2 component in a package

library ieee ;
use ieee.std_logic_1164.all ;

package xor_pkg is
   component xor2
      port ( a, b : in std_logic ; x : out std_logic ) ;
   end component ;
end xor_pkg ;
```

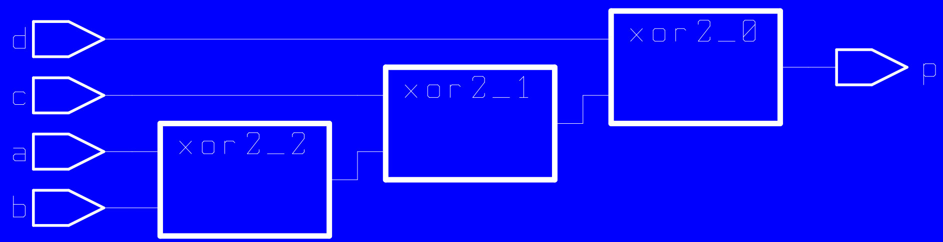
VHDL for Parity Generator

```
-- Parity function built from xor gates

library ieee ;
use ieee.std_logic_1164.all ;

use work.xor_pkg.all ;

entity parity is
   port ( a, b, c, d : in std_logic ; p : out std_logic ) ;
end parity ;
```

VHDL for Parity Generator (Architecture)

```
architecture rtl of parity is
   -- internal signals
   signal x, y : std_logic  ;
begin
   x1: xor2 port map ( a, b, x ) ;
   x2: xor2 port map ( c, x, y ) ;
   x3: xor2 port map ( d, y, p ) ;
end rtl ;
```

Schematic of Parity Generator



Exercise (Component Instantiation)

Exercise: Label the connections within the parity generator schematic with the signal names used in the architecture.

Type Declarations

- can declare new signal types (e.g. new bus widths, enumeration types for state machines)
- usually placed in a package

VHDL for Type Declarations

- to create a package called `dsp_types`:

```
package dsp_types is
   type mode is (slow, medium, fast) ;
   subtype sample is std_logic_vector (7 downto 0) ;
end dsp_types ;
```

Tri-State Buses

- tri-state (high-impedance) often used in buses
- assigning the value `'Z'` to a signal of type `out` tri-states that output

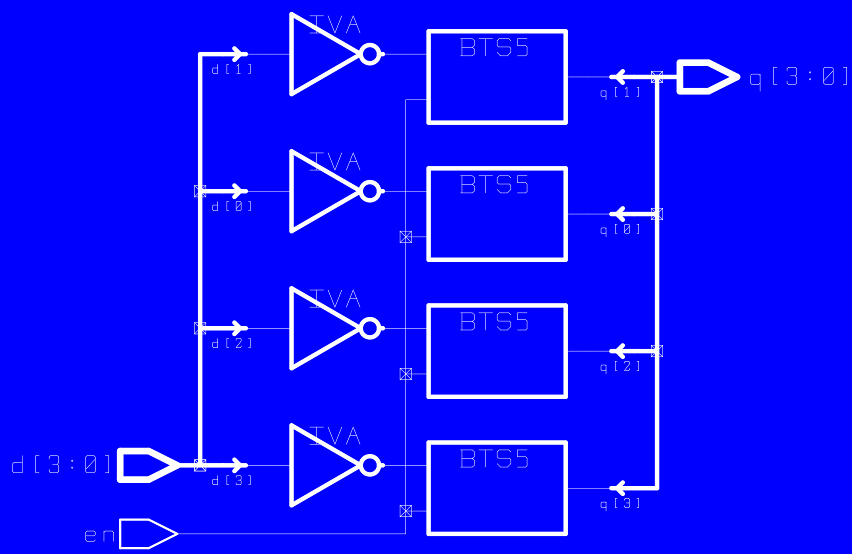
VHDL for Tri-State Buffer

```
-- Tri-State Buffer

library ieee ;
use ieee.std_logic_1164.all ;

entity tbuf is port (
   d : in std_logic_vector (3 downto 0) ;
   q : out std_logic_vector (3 downto 0) ;
   en : in std_logic
   ) ;
end tbuf ;
```

VHDL for Tri-State Buffer (Architecture)

```
architecture rtl of tbuf is
begin
   q <=
      d when en = '1' else
      "ZZZZ" ;
end rtl ;
```

Schematic of Tri-State Buffer



VHDL for Demo Circuit

```
-- demonstration design light chaser

library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.democom.all ;

entity demo is
  port (
  clk : in std_logic ;
  led : out std_logic_vector (7 downto 0) ) ;
end demo;
```

VHDL for Demo Circuit (Architecture)

```
architecture rtl of demo is
  signal count : unsigned (2 downto 0) ;
  signal scount : std_logic_vector (2 downto 0) ;
  signal ledN : std_logic_vector(7 downto 0) ;
begin
  c1: counter port map ( count, clk ) ;
  scount <= std_logic_vector(count);
  d1: decoder port map ( scount(2), scount(1), scount(0), ledN ) ;
  led <= not ledN ;
end rtl ;
```