

VHDL Tutorial

EE3109
Computer Aided Digital Design
Fall 2006

UTEP

Full Custom CMOS design

- Only done when performance is king
- Unit cost is medium - \$50 a chip
- Performance is best – 3 GHz processor
- Development cost (one time cost) is very high
 - 1000 engineers working for 2 years and costed at \$200k/year
 - \$400 million dollar development cost – wow!
 - turn around and sell 100 million chips for \$150 each
 - \$15 billion income – wow! development cost insignificant
 - moral of the story:
 - get a lot of engineers to hand tweak design for max performance

UTEP

ASIC Chips

- Mostly logic design in Verilog and VHDL
- Little transistor-level work
- Synthesis/Place/Route cells from logic library - tools
- Library includes
 - standard cells (i.e. ANDs, OR's, Flip Flops)
 - memories
 - IO blocks (input and output of chip)
 - Phase-Lock Loops
 - probably developed in another company (i.e. Artisan)
- Fabricate in a foundry – Fabless Design House
- 20 to 50 engineers in one year to design chip

VHDL?

- VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
- VHDL is a "Description Language", not a programming language like C, Pascal, Java.
- Originally developed for Department of Defense, released first in 1985.
- Not technology specific but predominantly used for digital system design.

VHDL Capabilities

- Exchange medium between chip vendors and CAD tool users.
- Supports both synchronous and asynchronous timing models.
- Various delay constraints can be described.
- Allows defining new data types.
- IEEE and ANSI standard, therefore very portable.
- Supports a wide range of abstraction levels, design, modeling styles.

VHDL Primary Design Constructs

- Five primary design constructs, also known as "Design Units" used to describe logic
- Entity
 - The Interface of a logic circuit.
- Architecture
 - Particular implementation of an entity
- Configuration
 - Binds entities, architectures, and component declarations
- Package declaration
 - Convenient way to define and group functions, procedures, types, components etc.
- Package body
 - Implementation of the functionality exposed by package declaration.

Design Unit (1) : ENTITY

- Defines the interface of the hardware module to the outside environment in which it is used
- Entity declaration syntax

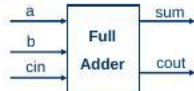
```
entity entity_name is
    generic specifications
    port specifications
begin
    entity statements          --passive statements (generally not used)
end entity_name
```

port – pass declared signals to and from external design

Generic – pass parameters from higher level design to the module

- Example

```
entity full_adder is
    generic (delay : time:=25ns);
    port (a, b, cin :in bit; sum, cout:out bit);
end full_adder
```



Design Unit (1) : ENTITY (2)

Mode in: The port can only be read within the entity and its architectures.

Mode out: This port can only be written.

Mode inout: This port can be read and written. This is useful for modeling bus systems.

Mode buffer: The port can be read and written. Each port must have only one driver.

```
entity DFF is
    -- parameter: width of the data
    generic (width: integer);
    -- input and output signals
    port ( CLK, NR: in bit;
          D: in bit_vector(1 to width);
          Q: out bit_vector(1 to width));
end DFF;
```

Design Unit (2) : ARCHITECTURE

- Functionality description of system declared by entity
- Can be behavioral or structural in terms of simpler components
- Architecture/Entity pair defines a circuit
- Multiple architectures can be defined for an entity
- Architecture declaration syntax

```
architecture architecture_name of entity_name is
    signal declarations
    component declarations
begin
    description statements
end architecture_name
```

signal – internal signals generated inside the module

component – declaring any external entities used in this module

description – behavioral or structural

Design Unit (2) : ARCHITECTURE(2)

- Example

```
architecture behavioral of full_adder is
    signal temp : bit;
begin
    P1: temp<=a xor b;
    P2: sum<=cin xor temp;
    P3: cout<=( and b) or (a and cin) or (b and cin);
end behavioral
```

temp – declared signal

a, b, cin, sum and cout – port signals

P1, P1, P3 are implicit process labels

Topics

- Identifiers
- Data Types
- Objects
- Operators
- Modeling Styles

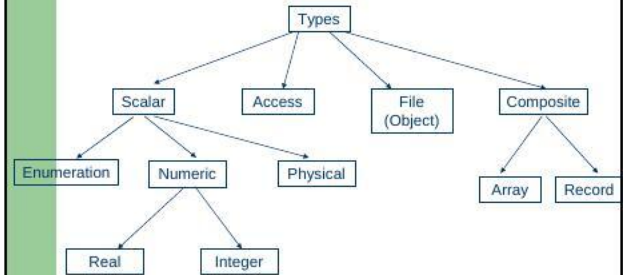
Identifiers

- Signals, Variables, constant names
- Names of entities, architectures, & Packages
 - Letters, numerals, underscores
 - Case insensitive
 - No two consecutive underscores
 - Must begin with a letter
 - No VHDL keyword

Identifiers (contd..)

- mySignal_23 -- normal identifier
- rdy, RDY, Rdy -- identical identifiers
- vector_&_vector -- X : special character
- last of Zout -- X : white spaces
- idle__state -- X : consecutive underscores
- 24th_signal -- X : begins with a numeral
- case, register -- X : VHDL keywords
- <http://www.peakfpga.com/vhdlref/>

VHDL Data types



Scalar Data Types

- **Integer**
 - Minimum range for any implementation as defined by standard: - 2,147,483,647 to 2,147,483,647
 - Example assignment to a variable of type integer :

```

ARCHITECTURE test_int OF test IS
BEGIN
    PROCESS (X)
        VARIABLE a: INTEGER;
    BEGIN
        a := 1; -- OK
        a := -1; -- OK
        a := 1.0; -- illegal
    END PROCESS;
END test_int;
    
```

Scalar Data Types (Cont.)

- **Real**
 - Minimum range for any implementation as defined by standard: - 1.0E38 to 1.0E38
 - Example assignment to a variable of type real :

```

ARCHITECTURE test_real OF test IS
BEGIN
    PROCESS (X)
        VARIABLE a: REAL;
    BEGIN
        a := 1.3; -- OK
        a := -7.5; -- OK
        a := 1; -- illegal
        a := 1.7E13; -- OK
        a := 5.3 ns; -- illegal
    END PROCESS;
END test_real;
    
```

Scalar Data Types (Cont.)

- **Enumerated**
 - User specifies list of possible values
 - Example declaration and usage of enumerated data type :

```

TYPE binary IS ( ON, OFF );
... some statements ...
ARCHITECTURE test_enum OF test IS
BEGIN
    PROCESS (X)
        VARIABLE a: binary;
    BEGIN
        a := ON; -- OK
        ... more statements ...
        a := OFF; -- OK
        ... more statements ...
    END PROCESS;
END test_enum;
    
```

Scalar Data Types (Cont.)

- **Physical**
 - Require associated units
 - Range must be specified
 - Example of physical type declaration :

```

TYPE resistance IS RANGE 0 TO 10000000
UNITS
    ohm; -- ohm
    Kohm = 1000 ohm; -- i.e. 1 KΩ
    Mohm = 1000 kohm; -- i.e. 1 MΩ
END UNITS;
    
```

- Time is the only physical type predefined in VHDL standard

Composite Data Types

• Array

- Used to group elements of the same type into a single VHDL object
- Range may be unconstrained in declaration
 - Range would then be constrained when array is used
- Example declaration for one-dimensional array (vector) :

```
TYPE data_bus IS ARRAY(0 TO 31) OF BIT;
```

```
VARIABLE X : data_bus;
VARIABLE Y : BIT;

Y := X(12); -- Y gets value of element at index 12
```

Composite Data Types (Cont.)

- Example one-dimensional array using DOWNTO :

```
TYPE reg_type IS ARRAY(15 DOWNT0 0) OF BIT;
```

```
VARIABLE X : reg_type;
VARIABLE Y : BIT;

Y := X(4); -- Y gets value of element at index 4
```

- DOWNT0 keyword must be used if leftmost index is greater than rightmost index
- e.g. 'Big-Endian' bit ordering

Composite Data Types (Cont.)

• Records

- Used to group elements of possibly different types into a single VHDL object
- Elements are indexed via field names
- Examples of record declaration and usage :

```
TYPE binary IS ( ON, OFF );
TYPE switch_info IS
    RECORD
        status : BINARY;
        IDnumber : INTEGER;
    END RECORD;

VARIABLE switch : switch_info;
switch.status := ON; -- status of the switch
switch.IDnumber := 30; -- e.g. number of the switch
```

Access Data Type

• Access

- Analogous to pointers in other languages
- Allows for dynamic allocation of storage
- Useful for implementing queues, fifos, etc.

Subtypes

• Subtype

- Allows for user defined constraints on a data type
 - e.g. a subtype based on an unconstrained VHDL type
- May include entire range of base type
- Assignments that are out of the subtype range are illegal
 - Range violation detected at run time rather than compile time because only base type is checked at compile time
- Subtype declaration syntax :

```
SUBTYPE name IS base_type RANGE <user range>;
```

- Subtype example :

```
SUBTYPE first_ten IS INTEGER RANGE 0 TO 9;
```

VHDL Data Types

Bit	'1', '0'	Q<='1'
Bit_vector	(Array of bits)	Data<="0001001"
Boolean	True, False	Q <= True;
Integer	-1, 2, 0, 99	Count <= Count +2;
Real	1.0, 1.0E5	V1 = V2/5.2
Time	7ns, 1us, 100ps	Q<='1' after 6ns;
Character	'a', 'g', '2', '\$'	Char <= 'x';
String	(Array of characters)	Msg = "MEM";

VHDL Objects

- Four types of objects in VHDL
 - Constants
 - Variables
 - Signals
 - Files
- The scope of an object is as follows :
 - Objects declared in a package are available to all VHDL descriptions that use that package
 - Objects declared in an entity are available to all architectures associated with that entity
 - Objects declared in an architecture are available to all statements in that architecture
 - Objects declared in a process are available only within that process

VHDL Objects - Constant

- Name assigned to a specific value of a type
- Allow for easy update and readability
- Declaration of constant may omit value so that the value assignment may be deferred
- Declaration syntax :
- Declaration examples :

```
CONSTANT constant_name : type_name [:= value];
```

```
CONSTANT PI : REAL := 3.14;  
CONSTANT SPEED : INTEGER;
```

Variables

- Convenient for local storage
- value can be changed & no time dimension associated
- operations on variables are instantaneous
- Declaration Syntax:

```
VARIABLE variable_name : type_name [:= value];
```

- Declaration Examples:

```
VARIABLE code : BIT_VECTOR(3 DOWNT0 0) := "0000";  
VARIABLE freq : INTEGER;
```

Signals

- Communication between VHDL components
- Value can be changed
- Time dimension associated
- Signal is driven by one or more drivers
- Declaration Syntax:

```
SIGNAL signal_name : type_name [:= value];
```

- Declaration Examples:

```
SIGNAL brdy : BIT;  
brdy <= '0' AFTER 5ns, '1' AFTER 10ns;
```

- Difference between variables and signals is the assignment delay

Files

- Note that files are defined a fourth data type initially, but were reclassified as objects in VHDL in 1993
- Files provide a way for a VHDL design to communicate with the host environment
- File declarations make a file available for use to a design
- Files can be opened for reading and writing
- The package STANDARD defines basic file I/O routines for VHDL types
- The package TEXTIO defines more powerful routines handling I/O of text files

User defined data types

- User can define their own data type
- Declaration Syntax:

```
TYPE data_type IS type_name;
```

- Declaration Examples

```
TYPE state_type IS (S0, S1, S2, S3, S4);  
signal state_name : state_type := S1;
```

VHDL Operators

- Logical - not, and, or, nand, nor, xor
- Relational - =, /=, <, <=, >, >=
- Shift - sll, srl, sla, sra, rol, ror
- Adding - +, -, &
- Multiplying - *, /, mod, rem
- Miscellaneous- **, abs
- Signing - +, -

VHDL Control Statements

- Similar to C or PASCAL.
- All are sequential
 - If...end if
 - case
 - loop
 - wait
 - exit
 - next
 - return

If ... End If

- evaluated from top to bottom

```

If conditional_expression then
  -- sequential statements
[elseif conditional_expression then]
  -- sequential statements
[else]
  -- sequential statements
end if;
  
```

- example

```

if a<0 then
  level:=1;
elseif a>1000 then
  level:=3;
else
  level:=2;
end if;
  
```

Case

- Mutually exclusive sets of choices
- *others* is used to cover all remaining values

```

case expression is
  when choice1 => sequential statements;
  when choice2 => sequential statements;
  [when others => sequential statements;]
end case;
  
```

- example

```

case x(0 to 1) is
  when "00" => z <='0';
  when "01" => z <='1';
  when others => z <='Z';
end case;
  
```

Modeling Styles

- Structural
 - Describe the circuit in terms of its interconnected components
 - Very low level (transistor) or a very high level description (block diagram)
- Behavioral
 - Define the operation of a circuit over time
 - Sequential statements inside a process
 - Examples include state diagrams, timing diagrams etc
- Dataflow
 - Define how data flows in the system
- Mixed
 - Combination of other styles.

Structural

- Focus is on how components are interconnected rather than the operation of each component
- Ability to define a list of components
- Defining signals to interconnect these components
- Distinguish between multiple copies of same component using labels

```

architecture HA_structure of Half_Adder is
  component XOR2
    port (X, Y: in Bit; Z: out Bit);
  end component;
  component AND2
    port (L, M: in Bit; N: out Bit);
  end component;
begin
  X1: XOR2 port map (A,B,SUM);
  A1: AND2 port map (A,B,CARRY);
end HA_structure;
  
```

Behavioral

- Uses the process construct in VHDL

```
architecture HA_algorithmic of Half_Adder is
begin
  process (a,b)
    variable s:bit_vector(1 to 2);
    variable num: integer range 0 to 2:=0;
    begin
      s:=a&b;
      for i in 1 to 2 loop
        if s(i) = '1' then num:=num+1; end if;
      end loop;
      case num is
        when 0 => (carry, sum) <= bit_vector("00");
        when 1 => (carry, sum) <= bit_vector("01");
        when 2 => (carry, sum) <= bit_vector("10");
      end case;
    end process
  end HA_algorithmic;
```

Dataflow

- Convenient for illustrating asynchronous and concurrent events, where delays represent actual hardware component delays
- Realistic way of modeling hardware dependencies and concurrencies

```
architecture HA_dataflow of Half_Adder is
begin
  sum <= a xor b after 8 ns;
  carry <= a and b after 4ns;
end HA_dataflow;
```

Mixed

- Combination of other styles
- Uses structural, concurrent and sequential statements

```
architecture FA_mixed of Full_Adder is
  component XOR2
    port (p1, p2:in bit; pz:out bit);
  end component;
  signal s1 : bit;
  begin
    X1:XOR2 port map (a, b, s1); -- structural
    process (a, b, cin) -- behavioral
      variable t1, t2, t3 : bit;
      begin
        t1:=a and b;
        t2:=b and cin;
        t3:=a and cin;
        cout <= t1 or t2 or t3;
      end process;
      sum <= s1 or cin after 4 ns; -- dataflow
    end FA_mixed;
```

Sequential Statements

- Execute serially inside a process
 - Process statement
 - Wait statement
 - Variable assignment
 - Assertion statement
 - If statement
 - Case statement
 - Loop statement
 - Null statement
 - Next statement
 - Exit statement
 - Function call
 - Procedure call
 - Report statement

Concurrent Statements

- Execute in simultaneous fashion - parallel
 - Block
 - Concurrent signal assignment
 - Concurrent procedure call
 - Concurrent assertion
 - Generate

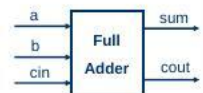
Structural Modeling – Example

- Ability to define a list of components
- Defining signals to interconnect these components
- Distinguish between multiple copies of same component using labels

2-bit adder from Full adder

- Full adder entity design –

```
entity FullAdder is
  port(a, b, cin: in bit; cout, sum: out bit);
end FullAdder;
```



2-bit Adder

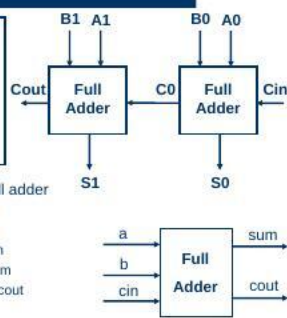
```
entity Adder2 is
  port (A, B: in bit_vector(1 downto 0);
        Cin: in bit;
        S: out bit_vector(1 downto 0);
        Cout: out bit);
end Adder2;
```

For first Full adder

- A0 -> a
- B0 -> b
- Cin -> cin
- S0 -> sum
- C0 -> cout

For second Full adder

- A1 -> a
- B1 -> b
- C0 -> cin
- S1 -> sum
- Cout -> cout



2-bit Adder (Contd..)

architecture addertest of Adder2 is

component FullAdder

port (a, b, cin: in bit; cout, sum: out bit);

end component;

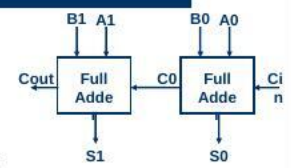
signal C0: bit;

begin

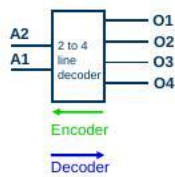
T1: FullAdder **port map** (A(0), B(0), Cin, S(0), C0);

T2: FullAdder **port map** (A(1), B(1), C0, S(1), Cout);

end addertest;



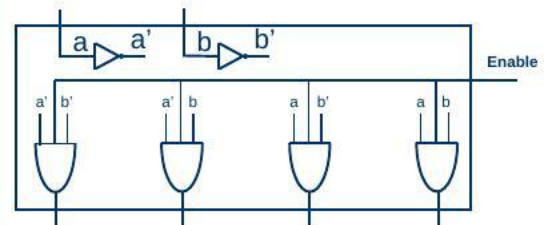
Encoders / Decoders



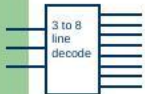
Input (A2A1)	O1	O2	O3	O4
00	1	0	0	0
01	0	1	0	0
10	0	0	1	0
11	0	0	0	1

A2'A1'

Decoder

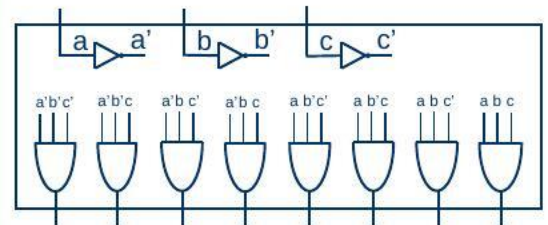


Encoders / Decoders



B2B1B0	O1	O2	O3	O4	O5	O6	O7	O8
000	1	0	0	0	0	0	0	0
001	0	1	0	0	0	0	0	0
010	0	0	1	0	0	0	0	0
011	0	0	0	1	0	0	0	0
100	0	0	0	0	1	0	0	0
101	0	0	0	0	0	1	0	0
110	0	0	0	0	0	0	1	0
111	0	0	0	0	0	0	0	1

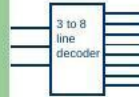
Decoder



Hierarchical Design

- Decoders
 - used in memories to select row to read or write
 - used to select one of $2N$ blocks to activate
- 3-8 Decoder using 2-4 Decoders
 - Design a 2-4 Decoder
 - Circuit design and Output waveform
 - Create a symbol
 - Design a 3-8 decoder
 - Circuit Design and Output Waveforms

Encoders / Decoders



B ₃ B ₂ B ₁	O1	O2	O3	O4	O5	O6	O7	O8
000	1	0	0	0	0	0	0	0
001	0	1	0	0	0	0	0	0
010	0	0	1	0	0	0	0	0
011	0	0	0	1	0	0	0	0
100	0	0	0	0	1	0	0	0
101	0	0	0	0	0	1	0	0
110	0	0	0	0	0	0	1	0
111	0	0	0	0	0	0	0	1