EE 459/500 – HDL Based Digital Design with Programmable Logic

Lecture 4
Introduction to VHDL

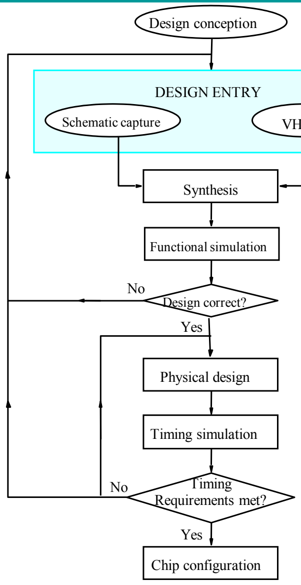
Read before class:

Chapter 2 from textbook (first part)

Outline

- VHDL Overview
- VHDL Characteristics and Concepts
- Basic VHDL modelling
  - Entity declaration
  - Architecture declaration
- Behavioural vs. Structural description in VHDL

Typical design flow



VHDL overview

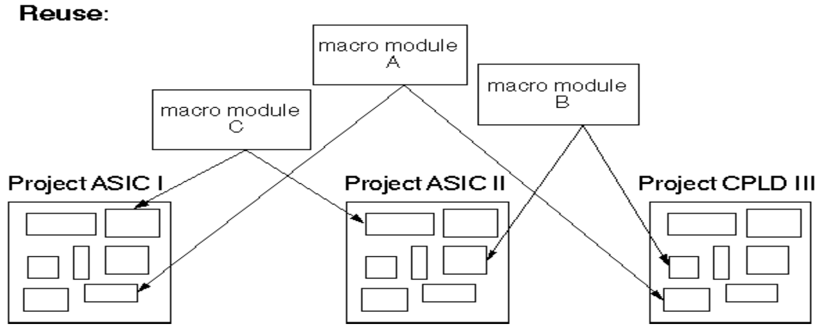
- What does VHDL stand for?
  - Very High Speed Integrated Circuit (VHSIC) Hardware Description Language
- VHDL is a formal language for specifying the behavior and structure of a digital circuit
  - Concurrent and sequential statements
  - Machine-readable specification
  - Man- and machine-readable documentation
- Initially developed under DOD auspices, later standardized as IEEE standards 1076-1987, 1076-1993, & 1076-1164 (standard logic data type)
- A concurrent language, initially aimed at simulation, later at synthesis
- Syntax similar to ADA and Pascal
- Verilog is another, equally popular, hardware description language (HDL)

Hardware Description Languages

- Both VHDL and Verilog are hardware description languages.
- They describe hardware!
- They are not software programming languages.

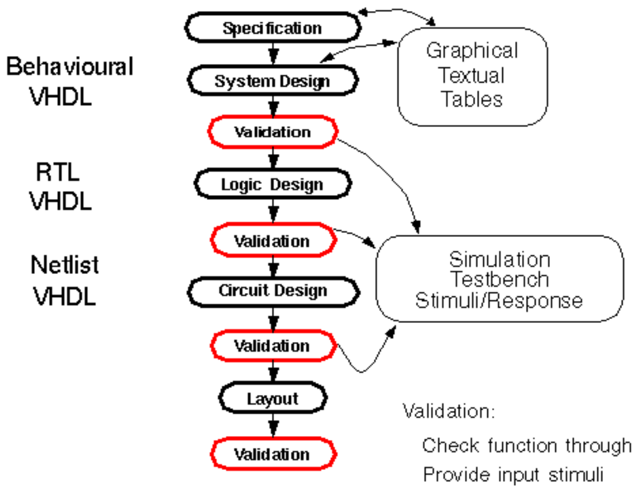
Application of HDL

Reuse:



- HDL offers design reuse capability
  - The corresponding HDL model can be reused in several designs/projects.
  - Frequently needed function blocks (macros) are collected in model libraries.

Range of use



Specification:
Description of the system requirements

System Design:
Modelling the behaviour

Logic Design:
Modelling the structure

Circuit Design:
Automatic conversion of structural description

Validation:

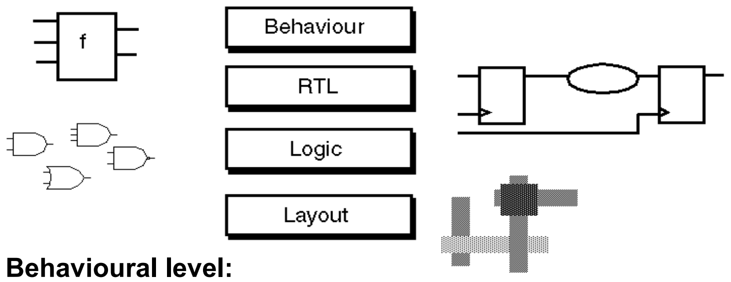
Check function through simulation

Provide input stimuli

Check expected response

Abstraction levels in Digital Design

- Abstraction - description of different parts of a system.
- Abstraction level - only the essential information is considered, nonessential information is left out.

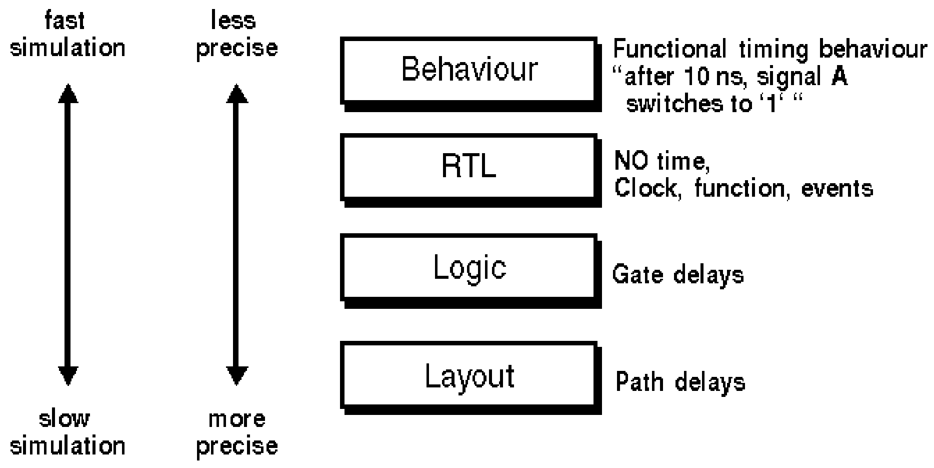


- **Behavioural level:**
  - Functional description of the design
  - Easy to describe in VHDL
  - Useful especially for simulation purposes
  - May not necessarily be synthesizable

Abstraction levels in Digital Design

- **Register transfer level (RTL):**
  - Design is divided into combinational logic and storage elements
  - Storage elements (Flip-Flops, latches, registers) are controlled by a system clock
  - Synthesizable
- **Logic level:**
  - Design is represented as a netlist of interconnected logic gates (AND, OR, NOT,...) and storage elements
- **Layout level (not really relevant to VHDL discussion)**:
  - Logic cells of target technology are placed on the chip and connections are routed
  - After layout is verified, the design is ready for the manufacturing/fabrication

Information Content of Abstraction Levels


fast simulation
slow simulation
less precise
more precise
Behaviour
Functional timing behaviour "after 10 ns, signal A switches to '1' "
RTL
NO time, Clock, function, events
Logic
Gate delays
Layout
Path delays

VHDL design unit – a quick intro

- A VHDL Design Unit consists of:
  1) **Entity declaration**
  2) **Architecture**

Entity Declaration:
Names entity and defines interfaces between entity and its environment.

```
ENTITY entity_name IS
PORT ( name_list : mode type);
END entity_name ;
```

Architecture Body:
Establishes relationship between inputs and outputs of design.

```
ARCHITECTURE body_name OF entity_name IS
-- declarative_statements
BEGIN
-- activity_statements
END body_name;
```

1) Entity Declaration

- Names entity and defines interfaces between entity and its environment.

```
entity entity-name is port (
      port-name-A:  mode  type;
      port-name-B:  mode  type;
      port-name-C:  mode  type;
      …
      );
end [entity][entity-name];
```

Port

- Each I/O signal in the entity statement is referred to as a **port**.
- A port is analogous to a pin on a schematic.
- A port is a data object.
- Can be assigned values.
- Can be used in expressions.

Mode

- The mode describes the direction in which data is transferred through a port.
- There are 4 different modes:

Mode	Description
in	Data only flows into the entity (input)
out	Data only flows out of the entity (output)
inout	Data flows into or out of the entity (bidirectional)
buffer	Used for internal feedback

Type

- VHDL is a strongly typed language
  - Data objects of different types cannot be assigned to one another without the use of a type-conversion function.
- There are two broad categories of data types:
  - Scalar - stores a single value
  - Composite - stores multiple values
- VHDL data types include:

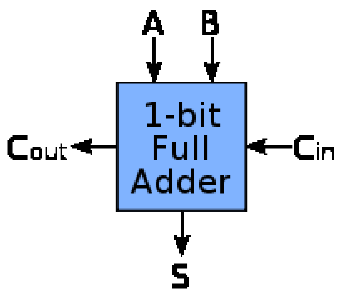
scalar	bit
	boolean
	integer
	character
	std_ulogic
	std_logic
composite	bit_vector
	string
	std_ulogic_vecto
	std_logic_vector

Type

- The most useful types for synthesis and simulation, provided by the IEEE std_logic_1164 package:
  - std_logic
  - std_ulogic
  - std_logic_vector
  - std_ulogic_vector
- **See Appendix A for difference between std_logic and std_ulogic**
- IEEE Standard Logic Types
  - Use of two-valued logic (bit and bit_vector) is generally not sufficient to simulate digital systems.
  - In addition to 0 and 1, Z (high-impedance), X (unknown), and U (uninitialized) are often used in digital system simulation.
  - The IEEE standard 1164 defines the std_logic type that has nine values:
    - 0, 1, Z, X, U, W, L, H, -

Entity Declaration - example

```
entity FULL_ADDER is
  port (
      A, B, Cin:  in  std_logic;
      S:          out  std_logic;
      Cout:       out  std_logic;
end FULL_ADDER;
```



2) Architecture Declaration

- Establishes relationship between inputs and outputs of design.

```
architecture architecture-name of entity-name is
      [declarations]
begin
      architecture body
end [architecture][architecture-name];
```

Architecture body

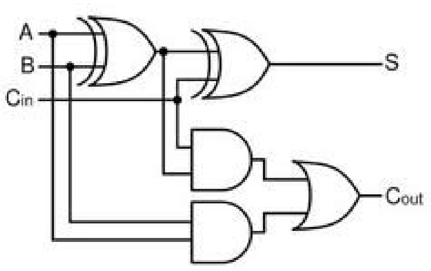
- Several different models or styles may be used in the architecture body including:
  - Behavioral
    - Dataflow
    - Algorithmic
  - Structural
- These models allow to describe the design at different levels of abstraction.

Architecture statement

- One or more architecture statements may be associated with an entity statement.
  - Only one may be referenced at a time.
- Declarations
  - Signals and components.
- Architecture body
  - Statements that describe the functionality of the design (i.e., the circuit).

Architecture Declaration – example

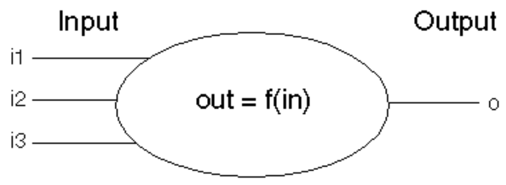
```
architecture RTL of FULL_ADDER is

begin
   S <= A xor B xor Cin;
   Cout <= (A and B) or (A and Cin) or (B and Cin);
end RTL;
```



Models/styles of description in VHDL

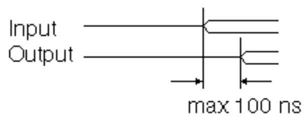
- Behavioral
  - Dataflow
  - Algorithmic
- Structural
- RTL

1) Behavioral description in VHDL



```
o <= transport i1 + i2 * i3 after 100 ns;
```

Specification:



- The function can be modelled as a simple equation (e.g., i1+i2*i3) plus a delay of 100 ns.

Behavioral description in VHDL

- Specify a set of statements to model the function, or behavior, of the design.
- Dataflow: uses concurrent statements
  - Concurrent statements:
    - Are executed at the same time; they mimic the actual hardware parallelism (processes, signal assignment)
    - Order is unimportant
- Algorithmic: uses sequential statements
  - Sequential statements:
    - Are executed in sequence (if, case, loops – while, for – assertion)
    - Order is very important

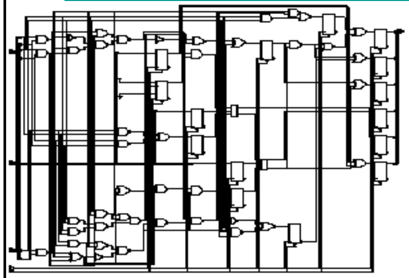
Behavioral synthesis

- Advantages
  - Easy to write HDL code; fewer lines of VHDL code
  - Useful especially for automatic generation of state machines
  - Faster simulation than RTL
- Disadvantages
  - May not be synthesizable

2) Structural description in VHDL

- Specify a set of statements to instantiate and interconnect the components necessary for the design.
- Components are defined separately.
- Signals are used to interconnect components.
- Advantages
  - Helps to describe a design hierarchically
  - Offers better control of circuit timing
  - Allows user to focus design optimization efforts on specific parts of design
- Disadvantages
  - Requires knowledge of internal structure of design
  - More VHDL code to write

Gate level in VHDL – an example/form of structural description



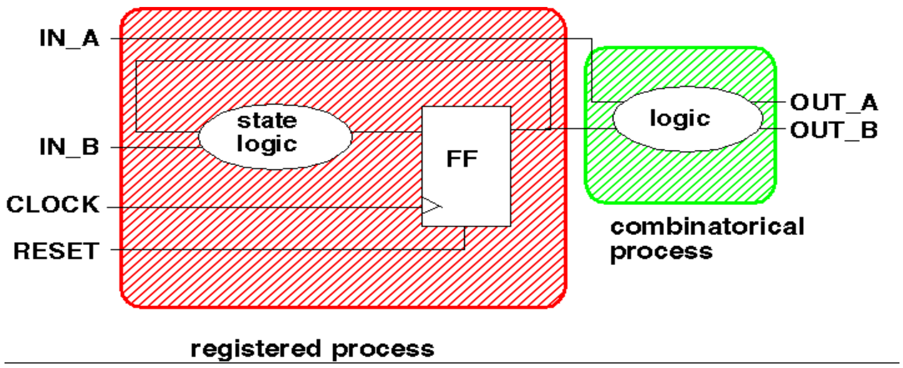
```
U86 : ND2 port map( A => n192, B => n191, Z => n188);
U87 : ND2 port map( A => I3_2, B => I2_0, Z => n175);
U88 : ND2 port map( A => I2_2, B => I3_0, Z => n173);
U89 : NR2 port map( A => mul_36_PROD_not_0,
        B => n174, Z => n185);
U90 : EN port map( A => n181, B => n182, Z => n180);
U91 : ND2 port map( A => I3_2, B => I2_1, Z => n181);
U92 : ND2 port map( A => I2_2, B => I3_1, Z => n182);
U93 : IVP port map( A => n180, Z => n192);
U94 : AO6 port map( A => n173, B => n174, C => n175,
        Z => n172);
U95 : NR2 port map( A => n174, B => n173, Z => n176);
U96 : ND2 port map( A => I3_1, B => I2_1, Z => n174);
U97 : EN port map( A => n183, B => n178,
        Z => product64_4);
U98 : ND3 port map( A => I2_2, B => I3_2, C => n174,
        Z => n183);
```

- Contains a list of the gates components (e.g., ND2, NR2, AO6).
- Each single element of the circuit (e.g., U86) is instantiated as a component (e.g., ND2) and connected to corresponding signals (n192, n191, n188).

3) RTL description in VHDL

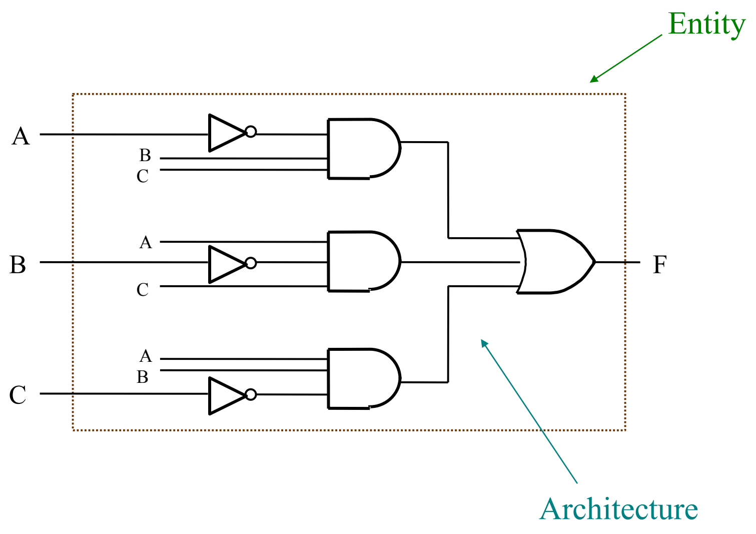
- Most realistic circuits combine a control-path or controller and a datapath to perform some computation
- In this case the description style in VHDL is closely related to the so called RTL design methodology, in which operations are specified as data manipulation and transfer among a collection of registers
- For example, the use of the FSMD model is especially recommended whenever the structure of the datapath is important
- This description style in VHDL can be regarded as a combination of behavioral and structural descriptions

FSM as an example of the simplest RTL description in VHDL



- **Functional** behaviour is modelled with registered process (clocked process) and combinational process.
- RTL VHDL code contains some sort of **structural** information in addition to the functional behaviour.

Example: simple combinational logic circuit



Example: entity

```
entity comb_logic_ckt_1 is
    Port ( A,B,C : in  STD_LOGIC;
               F : out  STD_LOGIC);
end comb_logic_ckt_1;
```

Example: architecture #1

```
architecture Boolean_Exp of comb_logic_ckt_1 is
begin

   F <= ( not(A) and B and C ) or
        ( A and not(B) and C ) or
        ( A and B and not(C) );

end Boolean_Exp;
```

Behavioral Model

Example: architecture #2

```
architecture Truth_table of comb_logic_ckt_1 is
begin

   F <=   '0'   when ( A = '0' ) and ( B = '0' ) and ( C = '0' ) else
          '0'   when ( A = '0' ) and ( B = '0' ) and ( C = '1' ) else
          '0'   when ( A = '0' ) and ( B = '1' ) and ( C = '0' ) else
          '1'   when ( A = '0' ) and ( B = '1' ) and ( C = '1' ) else
          '0'   when ( A = '1' ) and ( B = '0' ) and ( C = '0' ) else
          '1'   when ( A = '1' ) and ( B = '0' ) and ( C = '1' ) else
          '1'   when ( A = '1' ) and ( B = '1' ) and ( C = '0' ) else
          '0'   when ( A = '1' ) and ( B = '1' ) and ( C = '1' ) else
          '0' ;

end Truth_table;
```

Behavioral Model

Example: architecture #3

```
architecture Logic_gates of comb_logic_ckt_1 is
-- Component Declarations
-- components are defined in a VHDL package
component AND3
   port( inA, inB, inC: in std_logic;
         outF: out std_logic );
end component;

component OR3
   port( inA, inB, inC: in std_logic;
         outF: out std_logic );
end component;

component NOT1
   port( inA: in std_logic;
         outF: out std_logic );
end component;

-- Signal Declarations
-- used to interconnect the gates in the circuit (i.e. "wires")
signal out1, out2, out3: std_logic;
signal out4, out5, out6: std_logic;
```

Example: architecture #3 (continued)

```
begin

   NOTgate1: NOT1 port map( inA => A, outF => out1 );
   NOTgate2: NOT1 port map( B, out2 );
   NOTgate3: NOT1 port map( inA => C, outF => out3 );
   ANDgate1: AND3 port map( inA => out1, inB => B, inC => C, outF => out4 );
   ANDgate2: AND3 port map( A, out2, C, out5 );
   ANDgate3: AND3 port map( inA => A, inB => B, inC => out3, outF => out6 );
   ORgate1: OR3 port map( inA => out4, inB => out5, inC => out6, outF => F );

end Logic_gates;
```

Structural Model

VHDL Language & Syntax (General)

```
-- example of VHDL code
signal my_signal: bit;              -- an example signal
my_signal <= '0',                   -- start with '0'
             '1' after 10 ns,       -- and toggle
             '0' after 20 ns,       -- every 10 ns
             '1' after 30 ns;
```

- Signal assignment: “ **<=** “
- User defined names:
  - Letters, numbers, underscores
  - Start with a letter
  - No VHDL keyword may be used
  - Case insensitive
- List delimiter: “ **,** “
- Statements are terminated by “ **;** “ (may span multiple lines)
- Comments: “ **--** “ till end of line

VHDL Language & Syntax (Identifier)

```
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
open, register -- X : VHDL keywords
```

- Normal Identifier:
  - Letters, numbers, underscores
  - Case insensitive.
  - The first character must be a letter.
  - The last character cannot be an underscore.
  - No two consecutive underscores.
  - VHDL reserved words may not be used as identifiers.

```
\mySignal_23\ -- extended identifier
\rdy\, \RDY\, \Rdy\ -- different identifiers
\vector_&_vector\ -- legal
\last of Zout\ -- legal
\idle__state\ -- legal
\24th_signal\ -- legal
\open\, \register\ -- legal
```

- Extended Identifier:
  - Enclosed in back slashes
  - Case sensitive
  - Graphical characters allowed
  - May contain spaced and consecutive underscores.
  - VHDL keywords allowed.

Legal and illegal identifiers

- Legal Identifiers:
  - my_beautiful_signal
  - EE_459_500
  - Sel6B
- Illegal Identifiers:
  - _time_is_9am -- an identifier must start with a letter.
  - 8thsemester -- an identifier must start with a letter.
  - Homework#1 -- letter, digits, and underscore only.
  - final_ _example -- two underscore in succession not allowed
  - Entity -- keyword cannot be used as identifier
  - Time_out_ -- last character cannot be an underscore.

VHDL Reserved Words

abs	disconnect	label	package	sla
access	downto	library	port	sll
after	else	linkage	postponed	sra
alias	elsif	literal	procedure	srl
all	end	loop	process	subtype
and	entity	map	protected	then
architecture	exit	mod	pure	to
array	file	nand	range	transport
assert	for	new	record	type
attribute	function	next	register	unaffected
begin	generate	nor	reject	units
block	generic	not	rem	until
body	group	null	report	use
buffer	guarded	of	return	variable
bus	if	on	rol	wait
case	impure	open	ror	when
component	in	or	select	while
configuration	inertial	others	severity	with
constant	inout	out	shared	xnor
	is		signal	xor

VHDL information

- Recommended books on VHDL or the use of VHDL:
  - Peter J. Ashenden, The Student’s Guide to VHDL, Morgan Kaufmann.
  - Peter J. Ashenden, The Designer's Guide to VHDL, Morgan Kaufmann.
  - S. Yalamanchili, Introductory VHDL from Simulation to Synthesis, Prentice Hall.
  - P.P. Chu, RTL Hardware Design Using VHDL: Coding for Efficiency, Portability and Scalability, Wiley-Interscience, 2006.
- Useful websites – see the links provided at:
  - http://www.dejazzer.com/ee500/links.html

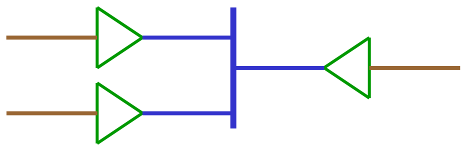
Summary

- VHDL is a hardware description language
- It has syntax (structure) and semantics (meaning)
- Entity and architecture(s) declaration
- Models/styles of description in HDL: Behavioral, Structural, RTL

Appendix A: What's the difference between `std_logic and std_ulogic`?

- The library ieee.std_logic_1164 has two types

  `std_logic` *and* `std_ulogic`
- To understand the difference, consider this circuit



  where the blue signal line has more than one driver attached to it? (*e.g.,* it's a bus)
- How do we set up our model so that the simulator knows the 'rules'?
  - which signal overrides the others

    or
  - how the signals combine together

Contending drivers

- Remember that VHDL knows nothing about the IEEE 1164 rules
  - To VHDL, the only primitive operations are those of a ‘normal’ programming language
    - addition, subtraction, etc
    - assignment
      - It does distinguish between signal and variable assignment, but only with respect to the timing of assignment of new values!
- `ieee.std_logic_1164` is *NOT* part of the VHDL standard
  - So when two `std_logic` values are applied to the same signal (i.e., wire), a VHDL simulator has to know that
    - ‘1’ overrides ‘Z’, ‘U’, ‘X’, ‘H’, ‘L’, …
    - ‘1’ and ‘0’ lead to ‘X’

    etc.

Unresolved signals

- `std_ulogic` is an unresolved type
  - It is an error to define a model in which two drivers can set the value of an unresolved signal

  because
  - there is no resolution function associated with the signal that can be invoked to determine which driver overrides the other
  - It is defined simply:

```
TYPE std_ulogic IS ('U','X','0','1','Z','W','L','H','-');
```

i.e., it is an enumerated type with possible values: `'U',` …

This says nothing about the behavior of `std_ulogic` signals
- Their behavior is encoded in the functions (`and, or,` …) that take `std_ulogic` arguments

On the other hand,

- `std_logic` is an resolved type

Unresolved signals

On the other hand,

- `std_logic` is an resolved type
  - It is defined:
    `SUBTYPE std_logic IS resolved std_ulogic;`
    Note that there is a function definition just preceding this type:

```
FUNCTION resolved( s: std_ulogic_vector )
                      RETURN std_ulogic;
SUBTYPE std_logic IS resolved std_ulogic;
```

  Thus `resolved` is a function that takes a vector of `std_ulogic` elements and returns a value of `std_ulogic` type

- This function is called a resolution function
  - It is called whenever two or more sources (signal assignments) drive a `std_logic` signal

Resolution functions

- Any resolved signal (*i.e.*, one that may be driven by two sources) is defined by a type that has a resolution function associated with it
  - A resolved type is a subtype
    - It can resolve a conflict of multiple instances of the parent type
  - The name of the resolution function immediately precedes the name of the type being resolved
  - The resolution function's
    - argument is a vector of elements of the type being resolved
      - The simulator will place the actual values to be resolved in this vector and call the resolution function

      e.g., with 3 drivers for a `std_logic` signal,
      the argument to resolved might be `('Z', 'H', '1')` which should return `'1'`
    - return value is the parent type
      - It will determine which of the values of the parent type result when the vector of signal values is applied

Implementation of resolution functions

- The simplest way to implement a resolution function uses a table

  e.g., for `std_logic`

```
TYPE std_logic_table IS ARRAY(std_ulogic,std_ulogic) OF
           std_ulogic;
CONSTANT resolution_table : std_logic_table := (
--   U    X    0    1    Z    W    L    H    -
  ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- U
  ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- X
  ( 'U', 'X', '0', 'X', '0', '0', '0', '0', '0' ), -- 0
  ( 'U', 'X', 'X', '1', '1', '1', '1', '1', '1' ), -- 1
  ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- Z
  … );
```

Implementation of resolution functions

- The function `resolved` is now very simple

```
-- On entry, s will contain the actual values being driven
-- this signal
FUNCTION resolved ( s: std_ulogic_vector )
                    RETURN std_ulogic IS
  VARIABLE result : std_logic := 'Z'; -- default, weakest
  BEGIN
  IF ( s'LENGTH = 1 ) THEN RETURN s(s'LOW);
  ELSE
    FOR k IN s'RANGE LOOP
      -- Take each signal in turn and determine the result of
      -- combining it with the previous result
      result := resolution_table(result,s(k));
    END LOOP;
  END IF;
  RETURN result;
END resolved;
```

Writing resolution functions

- *You may never need to!*
- `std_logic_1164` defines the most commonly needed one!

But,

1. You may be using `integer` types instead of `std_logic_vector` in a model of a processor for
   - convenience
   - speed in simulation
   - …

- You will need to define a resolved integer types if your model has a bus with multiple drivers in it
  - You will need to have a convention for 'disconnecting' a driver, *e.g.,* setting a driver to emit 0 when it's not driving the bus (where you would drive a `'Z'` with `std_logic`)
  - You can also explicitly disconnect a driver with VHDL's `DISCONNECT` statement

Resolution functions

2. You may have defined an abstract type
   - You (correctly) don't want to be bothered with implementation details yet
   - Your bus is a collection of signals (address, data, command, *etc*); you have a type for each one; so the bus itself is defined as a VHDL RECORD
   - The synthesizer will eventually convert it to logic for you!
   - …
   - Again you will need a resolution function
3. …

Simulation speed

- `std_ulogic` **does not have a resolution function associated with it**
  - it should use less simulation time (i.e., run faster) than `std_logic`
  - With `std_logic`, the simulator may end up checking for (or calling) the resolution function for every assignment
- **For simulation purposes `std_ulogic` is recommended wherever possible**
  - It may save simulation time
  - `std_logic` is a subtype, so it is possible to convert between them whenever necessary
  - `res_signal <= std_logic(unres_signal);`
- **However, lots of people stick to `std_logic` for different reasons (laziness?, compatibility with IPs, etc.)**