

---

# EEL 4783: HDL in Digital System Design

## Lecture 2+: Verilog vs. VHDL

Prof. Mingjie Lin



# VHDL & Verilog

---

**They are Hardware description languages.**

**They are each a notation to describe the behavioral and structural aspects of an electronic digital circuit.**

# VHDL Background

---

## **VHSIC Hardware Description Language.**

**VHSIC is an abbreviation for Very High Speed Integrated Circuit.**

**Developed by the department of defense (1981)**

In 1986 rights were given to IEEE

Became a standard and published in 1987

Revised standard we know now published in 1993 (VHDL 1076-1993) regulated by VHDL international (VI)

# VHDL

---

Uses top-down approach to partition design into small blocks 'components'

Entity: describes interface signals & basic building blocks

Architecture: describes behavior, each entity can have multiple Architectures

Configuration: sort of parts list for a design, which behavior to use for each entity.

Package: toolbox used to build design

# Verilog Background

---

Developed by Gateway Design Automation (1980)

Later acquired by Cadence Design(1989) who made it public in 1990

Became a standardized in 1995 by IEEE (Std 1364) regulated by Open Verilog International (OVI)

# VERILOG

---

Verilog only has one building block

Module: modules connect through their port similarly as in VHDL

Usually there is only one module per file.

A top level invokes instances of other modules.

Modules can be specified behaviorally or structurally.

- Behavioral specification defines behavior of digital system
- Structural specification defines hierarchical interconnection of sub modules

# Similarities

---

These languages have taken designers from low level detail to much higher level of abstraction.

In 2000 VI & OVI merged into [Accellera](#)

Simulation & synthesis are the two main kinds of tools which operate on the VHDL & Verilog languages.

They are not a toolset or methodology they are each a different language.

However toolsets and methodologies are essential for their effective use.

---

# Differences?

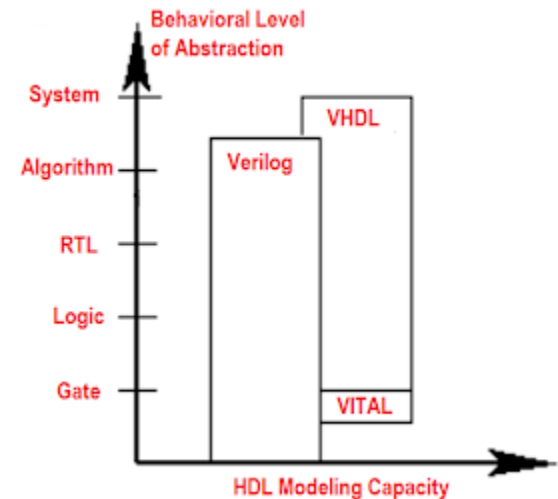
---

There are not many differences as to the capabilities of each.

The choice of which one to use is often based in personal preference & other issues such as availability of tools & commercial terms.

VHDL is “harder” to learn ADA-like.

Verilog is “easier” to learn C-like.





# Market analysis

---

According to Gary Smith, EDA Analyst at Dataquest, he says that although Verilog is dominating the market, it is going into the use of a mix of Verilog and VHDL. (report of march 2000)

But really:

As mentioned above, VHDL has many different complex data types and users can also define many other complex data types. This also makes VHDL more verbose than Verilog since Verilog only has 2 major data types and user-defined data types are not allowed in Verilog.

# Verilog vs. VHDL

---

- Verilog is an alternative language to VHDL for specifying RTL for logic synthesis
- VHDL similar to Ada programming language in syntax
- Verilog similar to C/Pascal programming language
- VHDL more popular with European companies, Verilog more popular with US companies.
- VHDL more 'verbose' than Verilog.
- Verilog and VHDL do RTL modeling equally well.

# VHDL vs. Verilog: Process Block

---

## *Process Block*

VHDL:

```
    process (siga, sigb)
    begin
        .....
    end;
```

Verilog:

```
    always @(siga or sigb)
    begin
        ....
    end
```

Both used to specify blocks  
of logic with multiple  
inputs/outputs

# VHDL vs. Verilog: Signal Assignment

---

VHDL:

```
signal a, b, c, d: std_logic;  
  
begin  
  
    a <= b and c;  
    d <= (c or b) xor (not (a) and b);  
end;
```

VERILOG:

```
wire a,b,c,d;  
  
assign a = b & c;  
assign d = (c | b) ^ (~a & b);
```

Declarations for  
one-bit wires are  
optional.

Logical operators  
same as in C.

# VHDL vs. Verilog: Interface Declaration

---

## VHDL:

```
entity mux is
  port ( a,b, s: in std_logic;
         y : out std_logic);

architecture a of mux is
begin
  .....
end;
```

## VERILOG:

```
module mux (a,b,s,y);
  input a,b,s;
  output y;

  ....
endmodule
```

# VHDL vs. Verilog: Busses

---

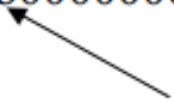
VHDL:

```
signal a,c: std_logic_vector(7 downto 0);  
begin  
    a(3 downto 0) <= c (7 downto 4);  
    c(0) <= '0';  
    c<= "00001010";  
  
end;
```

Verilog:

```
wire [7:0] ;  
begin  
    assign a[3:0] = b[7:4];  
    assign a[0] <= 0;  
    assign a = 'b0000000;  
  
end;
```

Value specified in binary.



# VHDL vs. Verilog: Busses

---

VHDL:

```
signal a,c: std_logic_vector(7 downto 0);
begin
    a(3 downto 0) <= c (7 downto 4);
    c(0) <= '0';
    c<= "00001010";

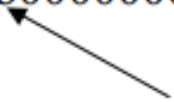
end;
```

Verilog:

```
wire [7:0] ;
begin
    assign a[3:0] = b[7:4];
    assign a[0] <= 0;
    assign a = 'b0000000;

end;
```

Value specified in binary.



# Verilog Vs. VHDL

---

- Verilog and VHDL are equivalent for RTL modeling (code that will be synthesized).
- For high level behavioral modeling, VHDL is better
  - Verilog does not have ability to define new data types
  - Other missing features for high level modeling
- Verilog has built-in gate level and transistor level primitives
  - Verilog much better than VHDL at below the RTL level.
- Bottom Line: You should know both!!!!!!



# EDA Tools

---

## Electronic Design Automation Tools

Looking at the market trend and we have to look at tools that is not just specific to one language but that can integrate both languages.

Out of all the different tools I've seen model SIM by [Model inc](#), is the one that should be used both [Altera](#) & [Xilinx](#) recommend it, and it's free for download from their websites.

- [Model inc](#) university program

# EDA Tools

---

Altera & Xilinx both have their own design environment Max + Plus II & ISE respectively.

They each have their own board to use with programs. The boards vary a lot in prices.

I ended using Altera's environment & board (kit) which can be purchased at [Altera](#) for \$150

# Simulation of counters

---

Demonstration of simple verilog & vhdl

- **LIBRARY IEEE;**
- **USE IEEE.STD\_LOGIC\_1164.ALL;**
- **USE IEEE.STD\_LOGIC\_ARITH.ALL;**
- **USE IEEE.STD\_LOGIC\_UNSIGNED.ALL;**

---

- **ENTITY Counter1 IS**
- **PORT(**
- **Clock, Reset,UPDOWN     : IN   STD\_LOGIC;**
- **Max\_count   : IN    STD\_LOGIC\_VECTOR(7 downto 0);**
- **Count       : OUT  STD\_LOGIC\_VECTOR(7 downto 0)**
- **);**
- **END Counter1;**
  
- **ARCHITECTURE behaviour OF Counter1 IS**
- **SIGNAL internal\_count    : STD\_LOGIC\_VECTOR(7 downto 0);**
- **BEGIN**
- **Count <= internal\_count;**
- **PROCESS(Reset,Clock)**
- **BEGIN**
- **IF reset='0' THEN**
- **internal\_count<="00000000";**
- **ELSIF clock 'EVENT AND clock='0' THEN**
- **IF updown='0' THEN**
- **IF internal\_count<Max\_count THEN**
- **internal\_count<=internal\_count+1;**
- **ELSE**
- **internal\_count<="00000000";**
- **END IF;**
- **ELSIF updown='1' THEN**
- **IF "00000000"<internal\_count THEN**
- **internal\_count<=internal\_count-1;**
- **ELSE**
- **internal\_count<=Max\_count;**
- **END IF;**
- **END IF;**
- **END IF;**
- **END PROCESS;**
- **END behaviour;**

```
module counter2 (updown,clock,reset,MaxCount,Count);
output[7:0] Count;
input[7:0] MaxCount;
input clock, reset, updown;
reg[7:0] Cnt;

assign Count=Cnt;

always @ (negedge clock or negedge reset)
begin
if(~reset)
Cnt=8'b0000_0000;
else if(updown)
if (Cnt<MaxCount)
Cnt=Count+1;
else
Cnt=8'b0000_0000;
else if(~updown)
if (8'b0000_0000<Cnt)
Cnt=Cnt-1;
else
Cnt=MaxCount;
end
endmodule
```