Verilog and VHDL

Martin Schoeberl

Technical University of Denmark
Embedded Systems Engineering

February 25, 2026

Overview

- VHDL and Verilog
- Interesting presentations in FSiC 2025
  - Check the Chinese OSOC project
  - OSOC stands for: One Student One Chip

Verilog Introduction

- You learned Chisel
- Verilog is the industry standard
- Used by tools as an exchange format
- Old language with a lot of quirks
- You mainly need to be able to read it
- And a few lines to connect stuff
- For serious work: use Chisel

Verilog vs SystemVerilog

- Verilog is the old standard
- SystemVerilog is an extension
- Adding a lot of features
  - 200+ keywords
- Mainly for verification (with UVM)
  - E.g., object-oriented programming
- But open-source tools do not fully support SV
- We stick to plain Verilog

Verilog Syntax

- Very C-like
  - e.g, has include file
  - defines
- Module-based
- Ports
- Wires and regs
- Always blocks
- Initial blocks
  - Only for simulation

Verilog (and VHDL) by Examples

- Will show you some examples
- First Chisel code, then Verilog (and VHDL)
- Again, we only need to be able to read some Verilog
- It is OK to use LLMs for generating Verilog or VHDL
  - Only for simulation
- You **just** need to be able to fix errors ;-)
- See also the Verilog Cheat Sheet

A Simple Component in Chisel

```
import chisel3._
import chisel3.util._

class ChiselAdder extends Module {
  val io = IO(new Bundle() {
    val a, b = Input(UInt(8.W))
    val sum = Output(UInt(8.W))
  })
  io.sum := io.a + io.b
}
```

- This is a too small component in practice

A Simple Component in Verilog

```
module adder(
  input  [7:0] a,
  input  [7:0] b,
  output [7:0] sum
);

  assign sum = a + b;

endmodule
```

A Simple Component in VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity adder is
  port (
    a, b : in unsigned(7 downto 0);
    sum : out unsigned(7 downto 0)
  );
end entity;



architecture rtl of adder is
begin
  sum <= a + b;
end architecture;
```

Using the `ChiselAdder` Component in Chisel

```
val in1 = Wire(UInt(8.W))
val in2 = Wire(UInt(8.W))
val result = Wire(UInt(8.W))

val m = Module(new ChiselAdder)
m.io.a := in1
m.io.b := in2
result := m.io.sum
```

Using the adder Component in Verilog

```
wire [7:0] in1;
wire [7:0] in2;
wire [7:0] result;

adder m(.a(in1), .b(in2), .sum(result));
```

Using the adder Component in VHDL

```
  signal in1, in2, result : unsigned(7 downto 0);

  component adder
    port (
      a, b : in unsigned(7 downto 0);
      sum : out unsigned(7 downto 0)
    );
  end component;

begin

    m: adder
        port map (
        a => in1,
        b => in2,
        sum => result
        );
```

A Register with Reset and Enable in Chisel

```
val reg = RegEnable(data, 0.U(8.W), enable)
```

A Register with Reset and Enable in Verilog

```
reg [7:0] reg_data;

always @(posedge clk) begin
  if (reset)
    reg_data <= 8'b0;
  else if (enable)
    reg_data <= data;
end
```

A Register with Reset and Enable in VHDL

```
    signal reg : std_logic_vector(7 downto 0);

begin
    process (clock)
    begin
        if rising_edge(clock) then
            if reset = '1' then
                reg <= (others => '0');
            elsif enable = '1' then
                reg <= data;
            end if;
        end if;
    end process;
```

A `switch` Statement in Chisel

```
io.out := 0.U
switch(io.sel) {
  is("b00".U) { io.out := io.in(0) }
  is("b01".U) { io.out := io.in(1) }
  is("b10".U) { io.out := io.in(2) }
  is("b11".U) { io.out := io.in(3) }
}
```

A case Statement in Verilog

```
module comb(
  input  [1:0] sel,
  input  [3:0] in,
  output reg out
);

  always @(*) begin
    case (sel)
      2'b00: out = in[0];
      2'b01: out = in[1];
      2'b10: out = in[2];
      2'b11: out = in[3];
      default: out = 1'b0;
    endcase
  end

endmodule
```

A case Statement in VHDL

```
process (sel, input)
begin
  case sel is
    when "00" => output <= input(0);
    when "01" => output <= input(1);
    when "10" => output <= input(2);
    when "11" => output <= input(3);
    when others => output <= '0';
  end case;
end process;
```

An “if...else if...else” Statement in Chisel

```
when (io.c1) {
  io.out := io.in1
} .elsewhen (io.c2) {
  io.out := io.in2
} .otherwise {
  io.out := io.in3
}
```

An “if...else if...else” Statement in Verilog

```
always @(*) begin
  if (c1)
    out = in1;
  else if (c2)
    out = in2;
  else
    out = in3;
end
```

An “if...else if...else” Statement in VHDL

```
process(c1, c2, in1, in2, in3)
begin
  if c1 = '1' then
    output <= in1;
  elsif c2 = '1' then
    output <= in2;
  else
    output <= in3;
  end if;
end process;
```

Advanced Features in Chisel

- Object-oriented code for Hardware
- Functional programming for generators
- Bundles with directions

Summary

- Hardware can be described in any HDL
- The abstraction is different
- SystemVerilog is not very well supported
- VHDL is dying (although there is a lot of legacy code)
- You might need to read (and write) some Verilog code
- Should be able to pick it up on the job
- Digital design is independent of the language