Verilog HDL:
Compiler Directives

Pravin Zode

Outline

- Timescale, define and undef
- Conditional Compilation (ifdef, ifndef, else, endif)
- File Inclusion (include)
- Other Directives (resetall, line)

Introduction

- Compiler directives control how the Verilog compiler interprets and compiles the code
- They do not synthesize into hardware; they help in debugging, simulation, and configuration
- Typically prefixed with **(backtick)** (e.g.,define, `ifdef)
- **Key Benefits**:
  - Improves code readability and modularity
  - Helps in simulation control and conditional compilation
  - Reduces redundant code and eases debugging

Timescale

- `timescale` directive is a compiler directive that specifies, Time unit and precision
- Must appear before module declaration
- If you don’t specify timescale, some simulators default to 1ns / 1ps
- # delays in the code are scaled according to time unit.
  - Example: #10 with 1ns / 1ps → 10ns, but with 1us / 1ns → 10µs.
- Timescale can also be defined in header files and included in multiple modules for consistency

Macro Definition (define)

define - Defining Macros

- Used to create text-based macros
- Helps in defining constants and reusable expressions

```
`define PI 3.14159
`define WIDTH 8
```

```
1    `define ADD_OP 4'b0000
2    always @(*) begin
3      if (opcode == `ADD_OP)
4        result = a + b;
5    end
```

Example : Macro & Parameter

```
1   module adder #(parameter DATA_WIDTH = 8) (
2       input [DATA_WIDTH-1:0] a, b,
3       output [DATA_WIDTH-1:0] sum
4   );
5   assign sum = a + b;
6   endmodule
```

```
1   `define DATA_WIDTH 8
2   module adder (
3       input [`DATA_WIDTH-1:0] a, b,
4       output [`DATA_WIDTH-1:0] sum
5   );
6   assign sum = a + b;
7   endmodule
```

Macro Definition (undef)

undef - Removing Macros

- Removes a previously defined macro
- Syntax: undef MACRO_NAME`

```
`undef CLK_PERIOD

`undef ADD_OP
```

Conditional Compilation

- Enables debugging modes, feature toggling, and configurable code sections
- Allows different sections of code to be compiled based on macro definitions
- Directives:
  - ifdef - Checks if a macro is defined
  - ifndef - Checks if a macro is not defined
  - else - Provides an alternative block of code
  - endif - Ends the conditional directive.

Example : ifdef

```
1   `define DEBUG
2   module test;
3     `ifdef DEBUG
4       initial $display("Debug Mode Enabled");
5     `else
6       initial $display("Debug Mode Disabled");
7     `endif
8   endmodule
```

Example : ifndef

```
1  `ifndef MODE
2      `define MODE 1
3  `endif
4
5  module test;
6    initial begin
7      `ifndef MODE
8        $display("MODE is not defined");
9      `else
10       $display("MODE is defined with value: %d", `MODE);
11     `endif
12   end
13 endmodule
```

File Inclusion (include)

- Used to include external files into the current Verilog file
- Helps in modularizing code and reusing definitions across multiple files
- Syntax: `include "filename.v"

Example: File Inclusion (include)

```
// config.v - Configuration file
`define DATA_WIDTH 8
`define ENABLE_FEATURE
```

```
`timescale 1ns/1ps
`include "config.v"

module main;
  reg [`DATA_WIDTH-1:0] data; // Using the macro for width

  initial begin
    data = 8'b10101010;

    `ifdef ENABLE_FEATURE
      $display("Feature is enabled. Data: %b", data);
    `else
      $display("Feature is disabled.");
    `endif

    #10 $finish;
  end
endmodule
```

Verilog Header Files (*.vh)

- In Verilog, a header file is a separate file containing commonly used definitions such as:
  - Macros
  - Parameters
  - Constants
  - Include guards
- The header file does not contain module definitions but is included in other Verilog files using the **`include** directive.

Use of Verilog Header Files (*.vh)

- **Reusability:** Define constants, macros, or parameters once and reuse across multiple modules
- **Maintainability:** Easier to change a constant in one place rather than updating multiple modules
- **Code organization:** Keeps modules clean by separating definitions and parameters.

Use of Verilog Header Files (*.vh)

```
// defs.vh
`define WIDTH 4
`define MAX_VALUE 15
```

```
`timescale 1ns/1ps
`include "defs.vh"  // Include the header file

module counter(
    input clk,
    input reset,
    output reg [`WIDTH-1:0] count
);

    always @(posedge clk or posedge reset) begin
        if(reset)
            count <= 0;
        else if(count < `MAX_VALUE)
            count <= count + 1;
        else
            count <= 0;
    end
endmodule
```

Resetall directive

- Resets all compiler directive settings to default

```
1   `timescale 1ns/1ps
2   `define WIDTH 8
3   module example_resetall;
4     reg [`WIDTH-1:0] data;
5     initial begin
6       data = 8'hFF;
7       #10;
8       `resetall  // Resets all compiler directives
9     end
10  endmodule
```

line directive

- Controls line number information in error messages
- Example: `line 100 "source.v"

```
1   `line 200 "testbench.v"
2   module testbench;
3     initial begin
4       $display("This is a testbench");
5     end
6   endmodule
```

Summary

- `` `define `` and `` `undef `` create and remove macros
- Conditional Compilation (`ifdef, ifndef, else, endif`) allows feature toggling
- include enables modular programming
- timescale defines simulation time control
- Other directives (`resetall, line`) help in fine-tuning the compilation process.

Thank you !

Happy Learning