How to Disable Printing in UVM Utility Macros in Single Field Control

TechRounder PDF Edition

Live article:
https://www.techrounder.com/how-to/how-to-disable-printing-in-uvm-utility-macros-in-single-field-control/

By Vipin PG | Published January 1, 2025 | Updated January 4, 2026 | Format: Guide | 4 min read

Quick answer

To disable printing for specific fields in UVM objects, use the uvm_print_override mechanism to selectively suppress output based on runtime conditions.

Effective verification in Universal Verification Methodology (UVM) often involves balancing detailed logging and optimal simulation performance. UVM utility macros like ‘uvm_info‘, ‘uvm_warning‘, and ‘uvm_error‘ are vital for debugging, but excessive printing can degrade simulation efficiency. This guide explores the nuances of selectively disabling printing for specific fields in UVM objects, enhancing both clarity and performance.

Why Control Printing in UVM?

Print statements in UVM are indispensable for monitoring testbench behavior, debugging, and tracking the flow of execution. However, excessive or unmanaged logging can lead to several challenges:

Performance Degradation

- Excessive logging increases simulation runtime, particularly in complex scenarios.
- It can overload the simulator’s processing capabilities, slowing down overall performance.

Log Clutter

- Overuse of ‘print‘ statements generates bloated log files.
- Essential information becomes buried in irrelevant details, complicating analysis.

Memory Issues

- Printing large volumes of data consumes significant memory resources.
- This can lead to crashes or sluggish performance in resource-intensive simulations.

Analytical Difficulties

- Navigating through cluttered logs to extract meaningful data is tedious.
- Debugging becomes inefficient when key events are obscured by unnecessary messages.

Overview of UVM Utility Macros

UVM utility macros streamline several verification tasks by:

- Logging Messages : ‘uvm_info‘ , ‘uvm_warning‘ , and ‘uvm_error‘ categorize and manage output based on severity levels.
- Setting Up Environments : Automating repetitive testbench setup tasks.
- Managing Data Structures : Simplifying handling of complex data.

These macros are essential for debugging but require careful management to prevent the pitfalls of excessive logging.

Mechanisms for Disabling Printing in UVM

The ‘uvm_print_override‘ Mechanism

‘uvm_print_override‘ allows fine-grained control over which fields of a UVM object are printed. By overriding the print behavior for specific fields, you can:

- Focus on relevant data.
- Suppress unnecessary output.

Example Implementation

```
class my_class extends uvm_object;
rand int data;
rand bit valid;
function new(string name = "my_class");
super.new(name);
endfunction
// Override the print function for the ’data’ field.
function void print(uvm_printer printer);
super.print(printer);
uvm_print_override(this, "data", this.print_data);
endfunction
// Function to determine if the ’data’ field should be printed.
function bit print_data(uvm_printer printer);
return valid; // Only print if ’valid’ is true.
endfunction
endclass
```

Here:

- ‘uvm_print_override‘ is used to customize the print behavior for the ‘data‘ field.
- The ‘print_data‘ function determines printing eligibility based on the ‘valid‘ field.

Benefits

- Selective Output : Suppress irrelevant fields, focusing on critical data.
- Performance Optimization : Reduce logging overhead during simulation.

Strategies for Efficient Printing in UVM

Using Severity Levels

UVM macros categorize messages into severity levels:

- ‘uvm_info‘ : General information about testbench operation.
- ‘uvm_warning‘ : Alerts for potential issues that don’t halt simulation.
- ‘uvm_error‘ : Critical errors requiring immediate attention.

Advantages:

- Structured output facilitates filtering and prioritization.
- Helps maintain concise and meaningful logs.

Conditional Printing with ‘uvm_print_override‘

You can dynamically control printing behavior based on runtime conditions.

Example:

```
uvm_print_override(my_field, "my_field", ‘uvm_print_override_type::field_override,
‘uvm_print_override_condition::always);
```

This snippet enforces field-specific conditional printing rules.

Advanced Techniques for Printing in UVM

Leveraging ‘uvm_report_object‘ and ‘uvm_report_server‘

These classes extend UVM’s logging capabilities:

- ‘uvm_report_object‘ : Configures reporting levels and manages output.
- ‘uvm_report_server‘ : Directs logs to specific destinations like files or databases.

Capabilities:

- Filter messages based on attributes such as severity or component.
- Redirect output to customized locations.

Creating Custom Printing Functions

Custom functions offer flexibility to:

- Format messages according to specific needs.
- Include additional metadata like timestamps.
- Generate reports in specialized formats.

Implementation:

- Define a custom function for your printing requirements.
- Register it using ‘uvm_report_server::set_report_func‘ .

Best Practices for UVM Printing

To optimize debugging and simulation performance:

Minimize Print Statements

- Focus on Essentials : Print only critical data to avoid redundancy.
- Use Conditional Statements : Enable printing only during debugging or specific scenarios.

Utilize Logging Levels

- Adjust verbosity based on the current phase of debugging.
- Keep normal execution logs minimal to improve performance.

Adopt Selective Printing

- Combine ‘uvm_print_override‘ with conditional checks for precise control.
- Enable field-specific printing based on runtime parameters or events.

Optimize Debugging Practices

- Targeted Debugging : Focus on specific issues instead of logging everything.
- Print Traces : Log event sequences to identify root causes effectively.

Common Questions About UVM Printing

How do I disable printing for all fields in a UVM object?

Use ‘uvm_set_report_verbosity‘ to adjust verbosity globally. Setting it to ‘UVM_NONE‘ disables all printing.

Can I use ‘uvm_print_override‘ for nested UVM objects?

Yes, specify the full path to the nested field for granular control.

What are the advantages of conditional statements over ‘uvm_print_override‘?

Conditional statements offer greater flexibility, allowing dynamic runtime decisions. However, they can increase code complexity.

Conclusion

Efficient control over printing in UVM utility macros is crucial for maintaining a high-performance simulation environment. Techniques like ‘uvm_print_override‘, conditional printing, and leveraging advanced UVM classes enable precise, efficient, and manageable logging. By adopting these strategies, you can enhance the debugging process, optimize performance, and keep logs clean and concise.