



SPECADOR

Documentation Generator

- For SystemVerilog, Verilog, Verilog-AMS, e, and VHDL

Well organized documentation in no time

```

Instances
  apb_subsystem_top : apb_subsystem_top
  ↳ i_apb_subsystem : apb_subsystem_0
    ↳ i_oc_uart0 : uart_top #(uart_data_width(32), .uart_addr_width(5))
    ↳ i_oc_uart1 : uart_top #(uart_data_width(32), .uart_addr_width(5))

Submodules
  uart_top #(uart_data_width(32), .uart_addr_width(5))
  ↳ dbg : uart_debug_if
  ↳ regs : uart_regs
  ↳ wb_interface : uart_wb
  
```

Flow Diagram of uart_top

BENEFITS

- Generates well organized and effective documentation, even from source code without comments
- Integrates easily into existing development flows, allowing design and verification groups to automate the documentation process
- Keeps the generated documentation in sync with the source code, saving time and reducing maintenance costs
- Enhances IP packaging
- Simplifies and encourages documenting the source code

OVERVIEW

Specador is a tool that automatically generates accurate HTML and PDF documentation from the source code. It works in batch mode (command line) and uses dedicated language compilers.

The tool enables your design and verification engineers to effortlessly generate and maintain proper and well-organized documentation. Users can generate meaningful documentation of a design or verification environment even from poorly documented source code. This is because Specador compiles the code and understands the project structure. Thus it can generate for example cross-linked class inheritance trees, design hierarchies, and diagrams.

Specador can be integrated easily into existing development flows. The documentation is always in sync with the source code, thus eliminating meticulous and problematic tasks such as maintaining diagrams or updating lists of ports or function arguments to reflect the current version.

Specador is especially useful for packaging IPs, either for the IP providers or for those using an IP-oriented flow in their company.

Specador recognizes the Markdown syntax, which can be used to format the documentation as desired, in order to enhance the readability of the HTML and PDF output.

```

virtual function string get_full_name() [source]
Returns the full hierarchical name of this object. The default implementation is the same as get_name, as uvm_object do not inherently possess hierarchy.

Objects possessing hierarchy, such as uvm_components, override the default implementation. Other objects might be associated with component hierarchy but are not themselves components. For example, uvm_sequence #(REQ:RESP) classes are typically associated with a uvm_sequence #(REQ:RESP). In this case, it is useful to override get_full_name to return the sequence's full name concatenated with the sequence's name. This provides the sequence a full context, which is useful when debugging. Get_full_name

virtual function int get_inst_id() [source]
Returns the object's unique, numeric instance identifier. Get inst_id

static function int get_inst_count() [source]
Returns the current value of the instance counter, which represents the total number of uvm_object based objects that have been allocated in simulation. The instance counter is used to form a unique numeric instance identifier. Get inst_count

static function uvm_object_wrapper get_type() [source]
Returns the type-proxy (wrapped) for this object. The uvm_factory's type-based override and creation methods take arguments of uvm_object_wrapper. This method, if implemented, can be used as convenient means of supplying those arguments.

The default implementation of this method produces an error and returns null. To enable use of this method, a user's subtype must implement a version that returns the subtype's wrapper.

For example:

class req extends uvm_object
  typedef uvm_object_wrapper #(const) type_id;
  static function type_id get_type();
  returns type_id::get();
endfunction
endclass

Then, to use:

factory.set_type_override(const::get_type(), subtype::get_type());

This function is implemented by the uvm_*_utils macros, if employed. Get type

virtual function uvm_object_wrapper get_object_type() [source]
Returns the type-proxy (wrapped) for this object. The uvm_factory's type-based override and creation methods take
  
```

Users have the ability to search in the documentation generated with Specador

Emac API Specification

Search docs

OVERVIEW DOCUMENTATION

Description

Features

Programming

Timing

API DOCUMENTATION

Design Topics

Modules

- Module cts_buffer
- Module dif_metastable
- Module difnxt1
- Module dram_001
- Module gate_clock_cell_g
- Module ip_async_fifo_g
- Module ip_emac_top
- Module ip_gate_clock_g
- Module ip_host_clk_mng_g
- Module ip_mac_big_endian
- Module ip_mac_cfg_hash_g
- Module ip_mac_dram_001
- Module ip_mac_dram_002
- Module ip_mac_dram_003
- Module ip_mac_dram_004
- Module ip_mac_fc_dec_g
- Module ip_mac_fc_gen_g
- Module ip_mac_host_if
- Module ip_mac_hostif_arb
- Module ip_mac_hostif_rx
- Module ip_mac_hostif_tx
- Module ip_mac_hostif_top
- Module ip_mac_hostif_tx
- Module ip_mac_hostif_bds
- Module ip_mac_mdio_g
- Module ip_mac_regs_bank

/ Modules / Module ip_mac_mdio_g

Module ip_mac_mdio_g

```

logic  pi_reset      po_master_mdio reg
logic  pi_clk        po_master_ont  reg
logic  pi_master_mdio po_mdio_err  reg
logic[15:0] pi_mdio_wdata po_mdio_rdata reg[15:0]
logic  pi_mdio_start po_mdio_done  reg
logic  pi_mdio_rnw
logic  pi_mdio_daddr
logic[4:0] pi_mdio_raddr
  
```

Block Diagram of ip_mac_mdio_g

Overview

The EMAC has a master management interface, which is used to access the PHY configuration registers. Depending on the configuration, the EMAC will either read the values from the PHY registers (auto-configuration mode) or set the values according to the programmed values (programming mode). The MDIO (po_master_mdio/pi_master_mdio) and MDC (po_master_mdc) ports are used to serially write and read management interface registers. A 2.5 MHz clock waveform must be provided to the MDC port on this interface.

Every management read/write instruction frame contains the following fields:

- PRE, Preamble - The EMAC management interface generates a full 32-bit preamble
- ST, Start of Frame (A '01' pattern indicates the start of frame)
- OP, Operation Code - A read instruction is indicated by '01', while a write instruction is indicated by '10'
- DEVAD, Device Address - A five-bit device address follows the opcode, with the most significant bit transmitted first.
- REGAD, Register Address - A five-bit register address follows the DEVAD field, with the most significant bit transmitted first. This field is used to access the management registers of the PHY.
- TA, Turnaround - The next two bit times are used to avoid contention on the MDIO port during a read transaction. For a read transaction, the PHY device and the system MDIO should be in tri-state for the first cycle of turnaround. The PHY device drives zero during the second cycle of the turnaround. For write transactions, the EMAC should drive 1 during the first cycle of the turnaround and zero during the second cycle.
- Data - The last 16 bits of the frame are the actual data bits. For a write operation, these bits are sent to the PHY device. For a read operation, the PHY device drives these bits. In both cases, the most significant bit is transmitted first.
- Idle - This indicates a high-impedance state of the MDIO line. The default state of the MDIO signal is high impedance with a pull-up resistor.

Name	Direction	Type	Description
pi_reset	input	wire logic	Global reset (asynchronous reset)

Constructors

```
function new(string name) [source]
```

Creates a new uvm_object with the given instance name. If name is not supplied, the object is unnamed. New

Functions

```
function void reseed() [source]
```

Calls random on the object to reseed the object using the UVM seeding mechanism, which sets the seed based on type name and instance name instead of based on instance position in a thread.

If the use_uvm_seeding static variable is set to 0, then reseed() does not perform any function. Reseed

```
virtual function void set_name(string name) [source]
```

Sets the instance name of this object, overwriting any previously given name. Set_name

```
virtual function string get_name() [source]
```

Returns the name of the object, as provided by the name argument in the new constructor or set_name method. Get_name

Documentation generated with Specador

FEATURES

- Language awareness** – the documentation is organized by language-specific concepts, including both relationship and structural information. For example, there are dedicated categories for packages, classes, modules, entities, or structs and one can easily explore the class inheritance, design hierarchy, or entity architectures
- Cross-linked documentation** – allows users to easily jump from the function documentation to one of its arguments or from the module definition to one of its submodules
- Hyperlinked diagrams** – Specador automatically generates diagrams such as Schematic and State Machine HDL Diagrams, and Inheritance and Collaboration UML Class Diagrams. The diagrams are hyperlinked and as such, one can click on a class in a diagram and jump to the chapter where it is documented or on a submodule in a schematic to jump to the submodule chapter
- Bitfield diagrams** – Specador automatically generates bitfield diagrams by parsing packed data types or the UVM register configurations methods
- Waveform diagrams** – Specador automatically generates waveforms specified using the popular open-source WaveDrom format
- Customized diagrams** – Specador can take directives in comments, feed them to an external script/tool for processing, and include the generated results in the documentation
- Review-oriented sections** – Specador generates sections that aggregate information, for example coverage and checking aspects
- Quick search** – users can easily search through the generated documentation
- Enhanced readability of the HTML and PDF output** – because Specador recognizes the Markdown syntax, users can beautify the HTML and PDF output by using attributes such as bold, italic, and lists
- Documentation control** – allows users to control what documentation they generate, for example, by filtering out entire packages or private APIs
- Additional embedded or linked documentation** – users can easily embed external documentation, add additional menus in the table of contents, add links to MS Word, PDF documents, or extra screenshots, and integrate with other documentation frameworks
- Scalability** – Specador automatically creates cross-links to pre-generated documentation of other IPs or projects

INTEGRATED SOLUTION

Specador Documentation Generator Specador Documentation Generator is closely integrated with the other design and verification products available from AMIQ EDA, including **DVT Eclipse IDE** and **DVT IDE for VS Code**.



TECHNICAL SUPPORT

The technical support team is available to promptly answer your questions, provide you with training, and work with you to determine your needs.

Your requirements and feedback are important. Whether you are looking for technical support or new features to improve your design and verification flow, AMIQ's technical support team strives to answer your requests in a timely manner.

CONTACT AMIQ

SUPPORT & EVALUATION
support@amiq.com

SALES
sales@amiq.com

WEBSITES
www.dvtclipse.com / www.amiq.com

Copyright 2024 AMIQ EDA S.R.L.
All rights reserved.

The information contained herein is
subject to change without notice.

SPE-0224-A4