VerificationWorks™ envBuilder

Application Note

Revision 1.0

Table of Contents

List of Figures

1 VerificationWorks™ *envBuilder*

1.1 Introduction

The VerificationWorks™ Environment Builder (*envBuilder*) software assists verification architects to create and maintain state-of-the-art verification environments based on popular verification methodologies, including UVM [1, 2], OVM [3], or VMM [4]. It provides a rich set of customizable templates that act as executable specifications for the desired verification environment architecture. Such environments can then be deployed efficiently across multiple projects and sites in a consistent manner. Teams can thus get a jumpstart in creating company standard and industry recommended best practices based environments, regardless of their expertise and familiarity with the verification methodologies.

1.2 Basic Flow

Figure 1-1 below shows the basic flow for *envBuilder*. Based on a set of customizable specifications of the recommended directory structure, their file contents, and a user specified argument file, *envBuilder* generates a complete verification environment shell. This generated code compiles out of the box with major simulators. The end users then add logic specific to their application. Over the life of the project, users can incrementally add newer components or add methodology updates by rerunning *envBuilder*, which in turn automatically identifies and merges in the related changes wherever possible.

envBuilder is invoked as a command line executable with an argument file. The latter contains various parameters such as the names of VIPs being used as well as how many instances of each are to be created at the top-level environment class. As mentioned earlier, *envBuilder* also provides a set of files that act as the base templates for the generated code. These template files are named as <*template_file_name.tt*>. There is one template file for each file type to be produced. E.g. uvm_env_sv.tt is a template file that is used to create the top level class to be derived from *uvm_env* base class for a UVM testbench.

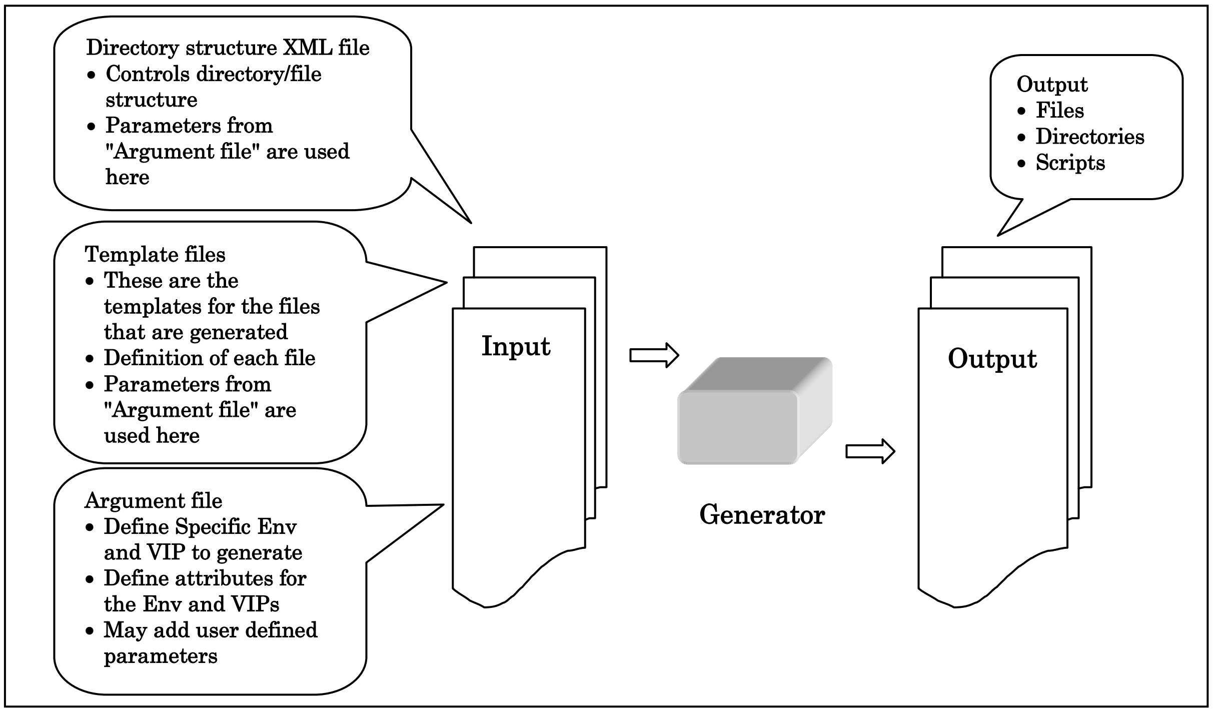
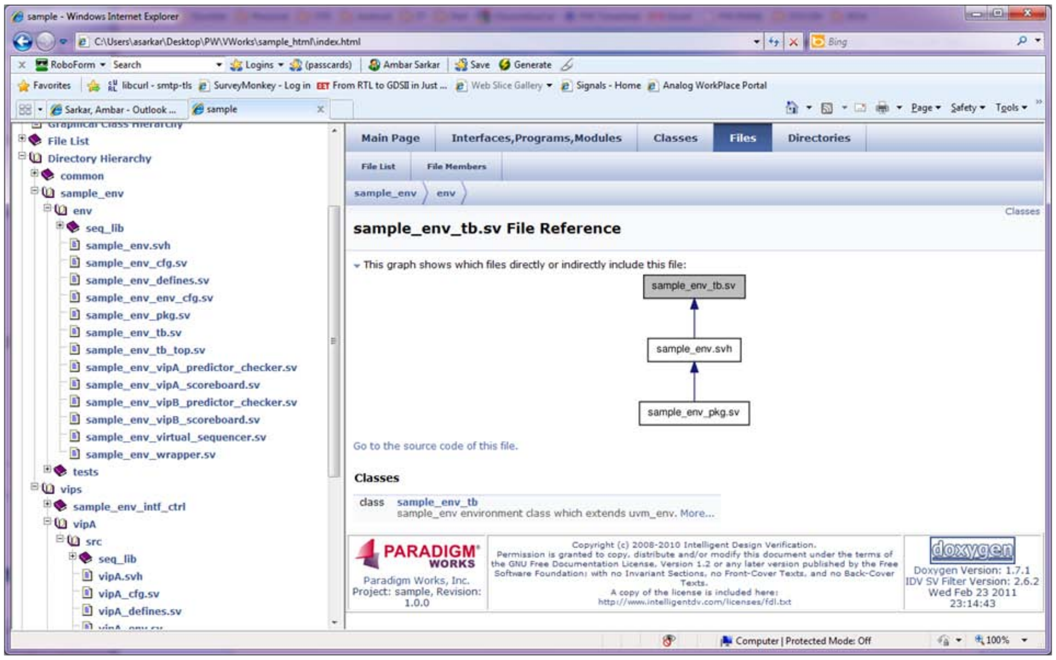

Directory structure XML file
• Controls directory/file structure
• Parameters from "Argument file" are used here
Template files
• These are the templates for the files that are generated
• Definition of each file
• Parameters from "Argument file" are used here
Argument file
• Define Specific Env and VIP to generate
• Define attributes for the Env and VIPs
• May add user defined parameters
Input
Generator
Output
Output
• Files
• Directories
• Scripts

Figure 1-1: Flow for VerificationWorks™ *envBuilder*

1.3 Fully Documented Verification Environment

The generated verification code is fully documented and stays in sync even when the verification engineer keeps adding application specific code. This is possible since the generated code is already instrumented with documentation tags using Doxygen [5]. Figure 1-2 below hows the directory hierarchy of files generated on the left pane, while the right pane shows how the files are included in a graphical fashion.



```
Graphical Class Hierarchy
File List
Directory Hierarchy
  common
  sample_env
    env
      seq_lib
      sample_env.svh
      sample_env_cfg.sv
      sample_env_defines.sv
      sample_env_env_cfg.sv
      sample_env_pkg.sv
      sample_env_tb.sv
      sample_env_tb_top.sv
      sample_env_vipA_predictor_checker.sv
      sample_env_vipA_scoreboard.sv
      sample_env_vipB_predictor_checker.sv
      sample_env_vipB_scoreboard.sv
      sample_env_virtual_sequencer.sv
      sample_env_wrapper.sv
    tests
  vips
    sample_env_intf_ctrl
    vipA
      src
        seq_lib
        vipA.svh
        vipA_cfg.sv
        vipA_defines.sv
        vipA_env.sv
```

sample_env_tb.sv File Reference

This graph shows which files directly or indirectly include this file:

Go to the source code of this file.

Classes

class **sample_env_tb**
sample_env environment class which extends uvm_env. More...

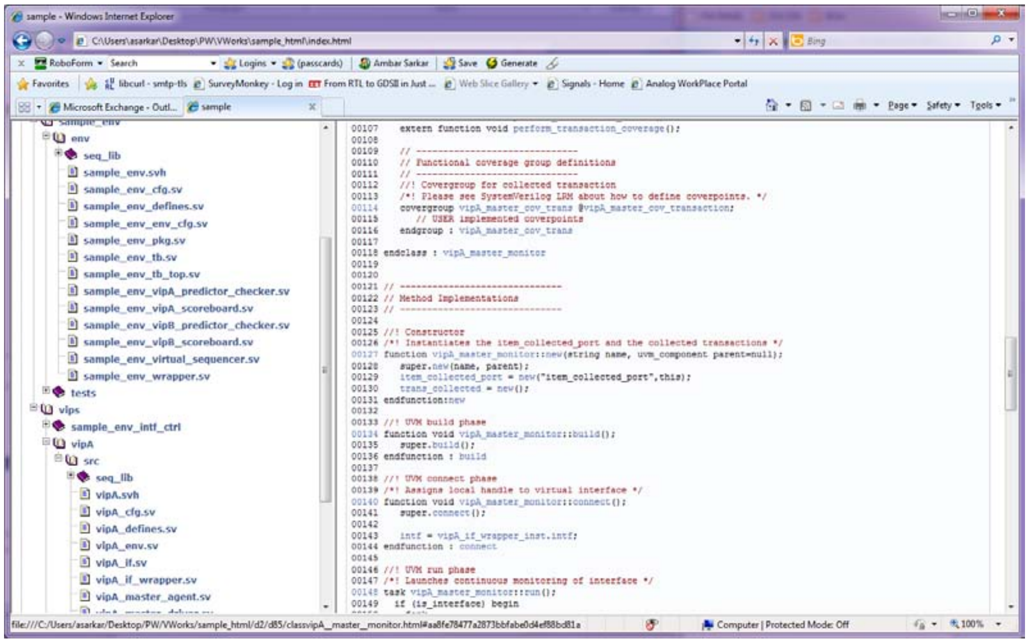
PARADIGM WORKS
Paradigm Works, Inc.
Project: sample, Revision: 1.0.0

Copyright (c) 2008-2010 Intelligent Design Verification.
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included here:
http://www.intelligentdv.com/licenses/fdl.txt

doxygen
Doxygen Version: 1.7.1
IDV SV Filter Version: 2.6.2
Wed Feb 23 2011
23:14:43

Figure 1-2: envBuilder Generated Documentation

By clicking on the appropriate links, the user can browse the generated code as shown in Figure 1-3 below.


sample - Windows Internet Explorer
C:\Users\asarkar\Desktop\PW\VWorks\sample_html\index.html
env
seq_lib
sample_env.svh
sample_env_cfg.sv
sample_env_defines.sv
sample_env_env_cfg.sv
sample_env_pkg.sv
sample_env_tb.sv
sample_env_tb_top.sv
sample_env_vipA_predictor_checker.sv
sample_env_vipA_scoreboard.sv
sample_env_vipB_predictor_checker.sv
sample_env_vipB_scoreboard.sv
sample_env_virtual_sequencer.sv
sample_env_wrapper.sv
tests
vips
sample_env_intf_ctrl
vipA
src
seq_lib
vipA.svh
vipA_cfg.sv
vipA_defines.sv
vipA_env.sv
vipA_if.sv
vipA_if_wrapper.sv
vipA_master_agent.sv
file:///C:/Users/asarkar/Desktop/PW/VWorks/sample_html/d2/d85/classvipA__master__monitor.html#aa8fe78477a2873bbfabe0d4ef88bd81a
Computer | Protected Mode: Off
100%

```
   extern function void perform_transaction_coverage();

   // ------------------------------
   // Functional coverage group definitions
   // ------------------------------
   //! Covergroup for collected transaction
   /*! Please see SystemVerilog LRM about how to define coverpoints. */
   covergroup vipA_master_cov_trans @vipA_master_cov_transaction;
      // USER implemented coverpoints
   endgroup : vipA_master_cov_trans

endclass : vipA_master_monitor


// ------------------------------
// Method Implementations
// ------------------------------

//! Constructor
/*! Instantiates the item_collected_port and the collected transactions */
function vipA_master_monitor::new(string name, uvm_component parent=null);
   super.new(name, parent);
   item_collected_port = new("item_collected_port",this);
   trans_collected = new();
endfunction:new

//! UVM build phase
function void vipA_master_monitor::build();
   super.build();
endfunction : build

//! UVM connect phase
/*! Assigns local handle to virtual interface */
function void vipA_master_monitor::connect();
   super.connect();

   intf = vipA_if_wrapper_inst.intf;
endfunction : connect

//! UVM run phase
/*! Launches continuous monitoring of interface */
task vipA_master_monitor::run();
  if (is_interface) begin
```

Figure 1-3: Browsing envBuilder Generated Code.

.

2 Supported Methodologies

envBuilder supports major verification methodologies such as UVM, OVM, or VMM. The verification architecture generated for each of these methodologies is specified as a collection of template files. Note that these templates and hence the generated output can be tweaked or completely replaced to support any methodology. These template files are collected as a directory, typically named after the methodology being followed. For example, the templates needed to generate a UVM1.0 compliant environment can be found in < installation directory>/templates/uvm_1_0_dir directory. Users can create their own template directories containing their company and/or project specific definitions. The template language being used in these files is powerful and well documented enabling users to easily customize their adopted methodology.

Following are some sample template files that can be used as is or customized further. Note that <METH> stands for OVM/UVM/VMM.

• <METH>_proj_cfg_sv.tt	Project DUT configuration class
• <METH>_proj_env_cfg_sv.tt	Project configuration class, including DUT and environment
• <METH>_vip_if_sv.tt	VIP specific interface
• <METH>_vip_mon_sv.tt	VIP specific monitor
• <METH>_vip_cfg_sv.tt	VIP specific configuration
• <METH>_vip_drv_sv.tt	VIP specific driver
• <METH>_vip_checker_sv.tt	VIP specific checker
• <METH>_vip_sv.tt	Top level VIP code

• <METH>_vip_src_list.tt	VIP specific build file
• <METH>_sample_test_sv.tt	Sample test
• <METH>_env_sv.tt	Environment
• <METH>_top_sv.tt	top.sv file in each environment
• <METH>_env_list.tt	Build file for env dir

2.1 Incorporating incremental changes

One of the key features of *envBuilder* is that it can incorporate updates to the templates into an existing environment. Since an environment may need to be continuously updated as a project progresses, a merge option is provided to add new components or incorporate any changes to the templates themselves. The user just points to the new template location, and *envBuilder* automatically identifies the changes (and updates if the option is selected) required in an existing verification environment. This is an important feature, with the templates acting as an executable specification of the verification environment that can be continuously maintained throughout the life of a project and beyond.

3 Summary

VerificationWorks™ *envBuilder* enables efficient and consistent creation of verification environments that follow popular methodologies and industry recommended practices. By providing a template driven solution to creating environments, verification and design engineers with a minimal amount of SystemVerilog or verification methodology experience can quickly contribute to the verification environment. Changes to the environment are also supported via automation, eliminating the time and risk associated with what can be cumbersome and error prone hand edits. Lastly, *envBuilder* is especially useful for teams with varying levels of experience and/or geographically distinct locations. Often the high level verification expertise within a team resides with just a few individuals, and VerificationWorks™ *envBuilder* enables these individuals to capture and propagate their expertise as an executable specification, in a scalable manner.

4 References

[1] Accellera VIP Technical Subcommittee. http://www.accellera.org/activities/vip

[2] UVM World. http://www.uvmworld.org/contributions.php

[3] OVM World. http://www.ovmworld.org/contributions.php

[4] VMM Central. http://www.vmmcentral.org/systemverilog.html

[5] Doxygen. http://www.doxygen.org

5 Company Background

Paradigm Works is a leading chip design and verification services company. The company is recognized for engineering excellence, integrity in business, and overall productivity and cost effectiveness.

We provide expert consultants and contractors both on site and offshore to assist in complex chip developments. We offer world class domain expertise (PCI Express, USB, Ethernet), application knowledge (Networking, Computing, Storage, Wireless), and leverage Paradigm Works suite of productivity accelerator software (VerificationWorks™, ReleaseWorks®, and SystemVerilog FrameWorks™) to help clients bring their innovations to market as quickly as possible.

For more information on Paradigm Works products and services call 978-824-1400 or see our web site at www.paradigm-works.com.